THESIS / THÈSE

DOCTOR OF SCIENCES

Guiding Agile Methods Customization
the AMQuICk Framework

Ayed, Hajer

Award date:
2018

Awarding institution:
University of Namur

Link to publication

Guiding Agile Methods Customization: the AMQuICk Framework

Hajer Ayed

A dissertation submitted in fulfillment of the requirements of the
degree of Doctor of Sciences
Discipline: Computer Science

October, 2018

Imprimé en Belgique

Doctoral Committee

Prof. Monique Snoeck (External Reviewer)

Faculty of Economics and Business
KU Leuven

Prof. Geert Poels (External Reviewer)

Faculty of Economics and Business Administration
Ghent University

Prof. Ivan Jureta (Internal Reviewer)

Faculty of Economics, Social Sciences and Business Administration
University of Namur

Prof. Anthony Cleve (Internal Reviewer)

Faculty of Computer Science
University of Namur

Prof. Benoît Vanderose (Internal Reviewer)

Faculty of Computer Science
University of Namur

Prof. Naji Habra (Advisor)

Faculty of Computer Science
University of Namur

Prof. Vincent Englebert (Chair)

Faculty of Computer Science
University of Namur

Abstract

In today's dynamic market environments, producing high quality software rapidly and efficiently is crucial. In order to allow fast and reliable development processes, several agile methodologies emerged in the late 90's and some steadily received growing attention from software practitioners regarding the number of positive experience reports and success stories. However, their implementation is still challenging in several contexts. Therefore, the companies willing to implement agility out of the "sweet-spot" context are seeking for more structured and systematic guidance to situationally adapt their agile practices.

Provided the aforementioned, we designed in this thesis the AMQuICk framework which is intended to be used by experienced software development teams, agile facilitators and/or consultants as a guide to contextualize their development practices.

The framework is composed of a customization life-cycle built upon the Quality Improvement Paradigm (QIP). Its core artifact consists of a metamodel for authoring agile building blocks called AMQuICk Essence. This metamodel incorporates the necessary elements for structuring an agile repository of practices (a kind of an experience factory), a context model and a customization knowledge base that can be represented in the form of decisional matrices. Additional operational tools of the framework are facilitation tools (to be used by the facilitator and the agile team): the AMQuICk Backlog and the AMQuICk Capitalization Workshop. The framework has been built iteratively following the Design Science Research methodology. Several case studies where necessary to evaluate its artifacts iteratively.

Key Words

Agile Software Development, Situational Method Engineering, Software Process Improvement

Résumé

De part la dynamique actuelle du marché du logiciel, il est de nos jours essentiel de produire des logiciels de haute qualité rapidement et efficacement. Depuis la fin des années 90, plusieurs méthodologies de développement agile ont émergé et certaines sont devenues de plus en plus populaires compte tenu du nombre de retours d'expériences positifs des entreprises. Cependant, la mise en oeuvre de ces méthodes reste difficile dans plusieurs contextes. Par conséquent, les entreprises désireuses d'implémenter un processus de développement plus agile dans un contexte "non propice" sont dans le besoin d'approches plus structurées et systématiques pour adapter leurs pratiques.

Dans le cadre de cette thèse, nous proposons le framework *AMQuICk* comme solution à la contextualisation des méthodes agiles. Celui-ci est destiné à être utilisé par les équipes de développement, les facilitateurs et/ou consultants Agile comme outil d'aide à l'adaptation (customization) des pratiques de développement.

Le framework est composé d'un cycle de vie de customization basé sur le paradigme QIP (Quality Improvement Paradigm) d'amélioration continue de la qualité des processus organisationnels. Son principal artefact consiste en un méta-modèle de description des pratiques et composants d'une méthode agile nommé AMQuICk Essence. Ce méta-modèle intègre les éléments nécessaires à la mise en place d'un référentiel de pratiques, un modèle de contexte et une base de connaissances de customization pouvant être représentée sous la forme de matrices de décision. Le framework propose également des outils de facilitation (à utiliser par le facilitateur et l'équipe agile): un backlog d'amélioration continue et un outil de capitalization. Le framework a été construit de manière itérative en suivant la méthodologie "Design Science Research". Plusieurs études de cas ont été nécessaires pour évaluer les artefacts développés de manière itérative.

Mots clés

Méthodes de Développement Agile, Ingénierie des Méthodes Situationnelles, Amélioration des Processus de Développement

Dedication

To the people I love the most,
my beloved parents,
my sisters,
my husband,
and all my family and friends,
without whom none of my success would be possible

x

Acknowledgments

First and foremost, my praises and thanks goes to God who blessed me with guidance, good health, a loving family and good friends who always supported me and without whom none of my success would have been possible.

I would like to express my sincere gratitude to Prof. Naji Habra for providing me the opportunity to work on an interesting research area. I thank him for his patience, his motivation and his support and guidance over the years. Throughout his supervision of my work, he did not act as an advisor only but also as a mentor who encouraged me and continuously supported me to address the encountered challenges in a more serene and positive state of mind. I will never forget the lessons learned from such a fine negotiator: Be calm, flexible and focus on the win-win. I will also never forget that the student has surpassed the mentor when we participated to the ICSE conference in Hyderabad.

I also want to thank Prof. Benoît Vanderose for the quality of our close collaboration. I am more than thankful for his precious support and for his continuous guidance and encouragement over the years. I consider myself very fortunate to have been initiated to research by such a skillful and passionate researcher. I would like to congratulate him, once again, for his position as a Professor. I will never say it enough: You more than deserve it!

I also would like to thank the members of my doctoral committee Prof. Monique Snoeck, Prof. Geert Poels, Prof. Ivan Jureta, Prof. Anthony Cleve, Prof. Benoît Vanderose and Prof. Vincent Englebert for their valuable support. Their guidance and feedback were vital for the completion of this work. It was an honor and a very enriching experience to discuss my research with them.

During the course of this doctoral research, many people helped me evaluate the relevance of the proposed framework to practitioners. I would like to thank them all for their implication in this work. More precisely, I would like to thank Virginie Mathieu, Caroline Detry, and Fabian Dermine for their contribution. I would also like to thank Dr. Imran Ghani from the University of Technology in Malaysia (UTM) for all his valuable comments

Contents

Contents

Contents

List of Figures

List of Figures

List of Tables

List of definitions

Chapter 1

Introduction

A decades-long goal has been to systematize software development in a repeatable and predictable way in order to improve the likelihood of delivering projects within a predefined timeframe. However, as a response to today's competitive market, the process of developing software systems has evolved considerably. More precisely, the inability of the conventional development paradigm to meet the new challenges of fast software delivery and quick change management, gave birth to a more modern envisioning of software development which emphasizes flexibility, leanness and human-focus.

The agile software development paradigm [Beck et al., 2001] was formalized in this context and became mainstream in the software development community. Several methods such as Extreme Programming (XP) [Beck and Andres, 2004], Scrum [Schwaber, 1995], Feature Driven Development (FDD) [Palmer and Felsing, 2001], OpenUP [Kroll and MacIsaac, 2006], DevOps [Httermann, 2012] and others appeared and are steadily gaining worldwide popularity.

However, even though the benefits of agile methods have been proved by successful implementations and experience reports, and despite the abundance of agile methods to choose from, organizations aspiring for agility are commonly confronted to several context-related challenges: no upper management sponsorship, lack of customer involvement, lack of team empowerment, a traditional organization in silos, resistance to change, etc.

Therefore, an increasing number of practitioners argue for a contextual approach to agile methods implementation: practices, deliverables, activities and any other process aspect should be properly adjusted to better accommodate the team's specific context and needs [Hoda et al., 2010]. This idea may well be established since the 90's with Situational Method Engineering (SME) [Henderson-Sellers and Ralyté, 2010], but when it comes to practice, organizations aspiring for agile methods customization are thrown in their track because of the lack of guidance approaches and decision-making tools. The newly released frameworks for scaling agility such as the Disciplined Agile Delivery (DAD) [Ambler and Lines, 2012], the Scaled Agile Framework

(SAFe) [Leffingwell, 2013], the Large-Scale Scrum (LeSS) [Larman, 2008] and the Spotify scaling model [Kniberg and Ivarsson, 2012] demonstrate a shift to a more context aware implementation of agile methods but still do not provide the necessary systematic guidance for practitioners since they rely on the expertise of coaches and facilitators.

The agile literature, as explained by [Dybå and Dingsøyr, 2008], provide a broad picture of adaptation experiences reported by agile community practitioners and researchers but most of them are hardly reusable since they lack of structure and because they are often based on the experts intuitive reasoning or on the teams' intrinsic non-quantified knowledge, i.e., the adaptation decisions are neither documented nor structured nor automated.

Regarding the aforementioned discussion, this research work aims at designing a practical guidance approach for practitioners going through agile methods implementation in challenging contexts.

1.1 Contribution

Taking all the above into consideration, the main contribution of this research work is:

> The design of a framework that provides the systematic and pragmatic guidance to organizations and teams for implementing context-specific agile methods.

The Agile Methods Quality Integrated Customization framework (AMQuICk) relies on techniques inherited from various fields of software engineering such as Software Process Improvement and Situational Method Engineering [Henderson-Sellers and Ralyté, 2010].

More precisely, it is composed of a customization life-cycle built upon the Quality Improvement Paradigm (QIP) cycle [Basili, 1985]. Its core artifact consists of a metamodel for authoring agile building blocks called *AMQuICk Essence*. This metamodel incorporates the necessary elements for structuring an *agile repository of practices* (a kind of an experience factory), a *context model* and a *customization knowledge base* (see Figure 5.1). Additional operational tools are the *AMQuICk Backlog* and the *Capitalization Workshop*.

AMQuICk is intended to be used by experienced agile facilitators or consultants as a guide for implementing situation-specific agile methods, especially

in the context of small and medium-sized enterprises transitioning to agile software development or that still have to mature their agile implementation.

1.2 **Outline**

The dissertation is organized as follows.

Part I addresses the general background of our research work. It encompasses the following chapters:

- **Chapter 2**: The chapter discusses the differences between disciplined and agile methods for software development and argues the need to provide practitioners with a more structured guidance when agile methods are to be implemented situationally.

- **Chapter 3**: The chapter presents and classifies the existing guiding approaches into four categories: *maturity-based*, *contingency factor*, *method engineering* and *experience-based* approaches. The chapter concludes this part of the dissertation with the identification of three research perspectives: *Situational Method Engineering*, *Context Study*, *Customization and Capitalization* and formulates the research questions that we are going to investigate in this research work.

Part II details the research methodology that was used to build the approach and provides an overview on the proposed approach. It encompasses the following chapters:

- **Chapter 4**: The chapter presents a review of the potential research methodologies that could have been used to conduct our research and presents the final constructive methodology that we applied to progressively design our approach.

- **Chapter 5**: The chapter provides an overview on the approach including its theoretical foundations, life-cycle and main artifacts.

Part III investigates the first research perspective of this dissertation which is aimed at exploring the design of an agile Situational Method Engineering (SME) approach. It encompasses the following chapters:

- **Chapter 6**: The chapter discusses the issues underlying the implementation of a SME methodology in an agile context, explores and compares the existing SME languages and formulates our proposal for the AMQuICk framework core.

3

- **Chapter 7**: The chapter discusses the design of AMQuICk Essence, the cornerstone metamodel of the AMQuICk framework.

Part IV discusses the contextual challenges that agile practitioners may face through a number of exploratory case studies. It encompasses the following chapters:

- **Chapter 9**: The chapter studies a transformation experience in an IT department of the Public Walloon Service (SPW) in Belgium.

- **Chapter 10**: The chapter investigates the contextual challenges that practitioners face when developing e-government services using an agile software development method.

- **Chapter 11**: The chapter investigates how the European and Asian cultural backgrounds may impact agile methods implementation.

Part V generalizes the learnings of the exploratory studies and proposes an extension to the AMQuICk framework accordingly. It encompasses the following chapters:

- **Chapter 12**: The chapter describes the theoretical process to be followed by practitioners to instantiate a suitable agile method and presents the AMQuICk customization matrix, a core AMQuICk artifact used to represent the practitioners expertise. It also presents some facilitation tools proposed as a guidance to help practitioners easily capitalize their customization knwoledge at an organizational level.

- **Chapter 13**: The chapter discusses the usability of the proposed AMQuICk artifacts based on illustrative examples and on a complete case study.

Finally, Part VI summarizes the research outcomes and discusses the framework limitations and future research perspectives and concludes this research work.

1.3 Publications

Most of the contributions presented in this thesis were published as peer-reviewed publications:

- Hajer Ayed, Benoit Vanderose, and Naji Habra. "A metamodel-based approach for customizing and assessing agile methods." In the Proceedings of the Eighth International Conference on the Quality of

Information and Communications Technology (QUATIC'12), pp. 66-74. IEEE, 2012.

- Hajer Ayed, Naji Habra, and Benoit Vanderose. "AMQuICk: A measurement-based framework for agile methods customisation." In the Proceedings of the Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA'13), pp. 71-80. IEEE, 2013.

- Hajer Ayed, Benoit Vanderose, and Naji Habra. "Supported approach for agile methods adaptation: An adoption study." In the Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering (RCoSE '14), the 36th International Conference on Software Engineering (ICSE'14), pp. 36-41. ACM, 2014.

- Benoit Vanderose, Hajer Ayed, and Naji Habra. "Implementing a model-driven and iterative quality assessment life-cycle: a case study." Electronic Communications of the EASST, 2014.

- Benoit Vanderose and Hajer Ayed. "Software Quality in an Increasingly Agile World". ERCIM News, Software Quality Issue, 2014.

- Hajer Ayed, Benoit Vanderose, and Naji Habra. "A Context-Driven Approach for Guiding Agile Adoption: The AMQuICk Framework." In the Proceedings of the Tenth International Conference on Software Engineering Advances (ICSEA'15). IARIA, 2015.

- Hajer Ayed, Benoit Vanderose, and Naji Habra. "Agile cultural challenges in Europe and Asia: insights from practitioners." In the Proceedings of the 39th International Conference of Software Engineering: Software Engineering in Practice Track (ICSE-SEIP'17). IEEE/ACM, 2017.

- Anthony Simonofski, Hajer Ayed, Benoit Vanderose, and Monique Snoeck. "From Traditional to Agile E-Government Service Development: Starting from Practitioners' Challenges." In the Proceedings of the Americas Conference on Information Systems (AMCIS'18). Association for Information Systems (AIS), 2018.

Part I.

Problem Statement

In this part of the dissertation, we address the general background of the doctoral research. More precisely, chapter 2 discusses the differences between disciplined and agile methods for software development and argues the need to provide practitioners with a more structured guidance when agile methods are to be implemented situationally.

Chapter 3 presents and classifies the existing guiding approaches into four categories: maturity-based, contingency factor, method engineering and experience-based approaches. Then, regarding the benefits and limitations of these approaches, the main research problem is decomposed into more specific research questions.

Chapter 2

Research Background

Disciplined Vs. Agile Software Development

In this chapter, we discuss the differences between disciplined and agile methods for software development and argue the need to provide practitioners with more structured guidance when agile methods are to be implemented situationally.

First of all, Section 2.1 recalls that forty years ago, the software crisis paved the way for disciplined development methods which recommend *repeatability*, *stability* and *high assurance.* Up-front planning, extensive documentation and predefined verification and validation strategies support these goals. Process improvement also focuses on predictability and stability by increasing process capability through standardization, measurement and control. Formal models and standards are used to define, tailor, assess and improve processes. These include reference process maturity models such as the [CMMI, 2006] and the [ISO/IEC 33001, 2015] and a number of organizational continuous improvement frameworks such as the QIP [Basili and Caldiera, 1995].

However, another crisis still occurred in the nineties, enlightening the need for a more flexible and lightweight development which should be more human-centric. Several methods appeared forming a new paradigm called the Agile Software Development. They advocate *simplicity* and *responsiveness* to change. Iterative and incremental development, focus on business value, close customer involvement, effective collaboration, and light-weight planning and documentation support these goals. Within this context, we explore in Section 2.2 what agility means and present the most influential agile methods. We then examine the elements and success factors of software process improvement in an Agile environment and discuss whether the reference process improvement models and frameworks are applicable for agile software development projects.

9

Section 2.3 then summarizes the contrast between disciplined and agile software development and explores the existence of hybrid approaches. It also discusses the need for specific process improvement models dedicated to agile software development which would help teams to achieve an adapted or customized agile method.

2.1 Disciplined Software Development

2.1.1 Defining Discipline

In most professions (e.g., art, sport, engineering, etc.), discipline is the foundation of any successful endeavor. The term "discipline" refers to the establishment of a system of rules of conduct or a code of behavior and to the practice of training people to obey these rules[1]. Accordingly, in software engineering, *"a process is disciplined when it specifies the set of rules that would result in behavior consistent with those rules"* [Zahran, 1998]. This set of rules corresponds to all established development practices and discipline refers to the degree to which these are effectively applied.

[Humphrey, 2006] defines software process discipline as the *"fidelity with which a defined development process is followed"* and from the perspective of the team specifies that *"highly disciplined teams thoroughly plan their work, rigorously follow their processes and consistently gather and analyze their data. Undisciplined teams, however, will be less thorough in some or all these areas [...]. (They) generally also ignore their data and continue making intuitive decisions.".* Thus, process discipline means common compliance with established processes and team self-control.

For a process to be effective, the definition of a set of rules to follow is not sufficient. People must be informed about it and if necessary trained to it. Process knowledge must be disseminated to those who are supposed to perform and participate in its activities [Zahran, 1998]. A disciplined process is a mature process that is defined, trained, enforced and followed. [Humphrey, 2006] adds another attribute, that a disciplined process is continuously improving.

Historically disciplined software development appeared with the rise of the complex aerospace systems and system engineering [Schlager, 1956]. A series of standards such as the military standards [MIL-STD-1521A, 1976] and [DOD-STD-2167A, 1988] were introduced to guide the software

---

[1]Oxford dictionary of English online

development endeavor. This continued in the 90's with standards such as the [ISO/IEC 12207, 1995] and the Capability Maturity Model [CMM, 1993] which emphasize process standardization. At the organization level, a set of processes, activities, and tasks are adopted from a specific standard and declared as a condition of efficiency. At the project level, a team tailors the standard process in accordance with contractually established criteria and guidelines.

Based on the aforementioned discussion, we propose the following definition:

Definition 2.1 (Disciplined Software Development)**.** *The systematic and formal engineering approach to software development that conforms to established processes in moving software through a plan-driven set of work-specific phases from requirements to finished code (adapted from [Boehm and Turner, 2003]).*

2.1.2 Characteristics

Based on concepts drawn from system engineering [Schlager, 1956], disciplined methods of software development promote a systematic engineering approach which, according to [Boehm and Turner, 2003], is characterized by the following features:

- Repeatability and Predictability;
- Thorough planning;
- Upfront and extensive documentation;
- Well-defined and standardized process management.

The meaning of each of these characteristics is detailed in the following sections.

2.1.2.1 Repeatability and Predictability

Repeatability means *"doing the same thing in the same way to produce the same results"* [Highsmith, 2009]. Predictability means *"obtaining the same expected results"*. Repeatable processes reduce variability through measurement and constant process control and correction with the objective of achieving predictable results.

The term of repeatability originated from the manufacturing vocabulary, where results of processes are well defined, i.e., all the characteristics and

information about the product are known in advance. *"When a process had consistent inputs, then defined outputs would be produced. Repeatable means that the conversion of inputs to outputs can be replicated with little variation."* [Highsmith, 2009].

Implementing repeatable processes to achieve predictability has been the goal of many software-intensive organizations which follow disciplined software development. Repeatability ensure comparability between projects and rationalizes the management decisions. It provides the basis for objective measurement of performances and makes it easier to verify that the project process can deliver what was specified in the beginning.

2.1.2.2 Thorough planning

As explained in the previous section, disciplined methods promote predictability. A good mean of achieving this quality is to fix the constraints of the project, i.e., the scope, time and cost and to plan thoroughly the work to be done. Planning is done as a prelude to the entire project and at a detailed level: the work is broken down into phases and tasks which may be sequential or overlapping, each task is broken into its composite activities, estimates are done and finally concurrencies and dependencies are identified to provide schedules and delivery times. The plan predicts not only the schedule and budget for construction but also the task delegation for the development team.

To achieve this target, the original tendency of disciplined methodologies was to view the development cycle as sequential steps. Linear models such as the waterfall model and the V-model were followed (see Section 2.1.3.1). More recently, incremental and evolutionary models have been adopted but still with thorough and heavy documentation. These include the incremental and the spiral model (see Section 2.1.3.1).

2.1.2.3 Upfront and extensive documentation

A main activity in the disciplined approach to software development is the big design upfront process which is common in industry and other engineering disciplines. However, documenting the software product upfront presupposes that it is possible to gather all requirements prior to writing any code. This strategy of documenting the software has the advantage of bringing more predictability which is very important in some contexts such as life critical systems.

The documentation of disciplined methods is also characterized by completeness. Several design documents are produced targeting to thoroughly detail the customer expectations, in order to avoid as much ambiguity as possible. This also ensures a maximum of traceability of the requirements, design and code which facilitates the software maintenance effort.

2.1.2.4 Well-defined and standardized process management

Process-supported environments emerged in the 80's with the work of influential researchers such as [Osterweil, 1987] which states that *"Software Processes are Software Too"*. This idea forms the basis of a discipline of software development (on top of the discipline of application software development) which *"yield highly useful insights on software process requirements, process architectures, process change management, process families, and process asset libraries with reusable and composable process components"* [Boehm, 2006].

For a disciplined process to be effective, a software process infrastructure, i.e., *"a systematic application of software processes and software life cycle models across an organization"* [Bourque et al., 2014], should be enabled at an organizational level. This is also known as *process institutionalization*. [Zahran, 1998] specifies that an effective process infrastructure must cover three important aspects: (1) process definition, (2) process training and support, and (3) process monitoring and enforcement [Zahran, 1998].

Process definition refers to the specification and documentation of the process at the organizational level. Such specification should be visible to everyone in the organization. It generally include detailed plans, activities, workflows, roles, and workproducts descriptions.
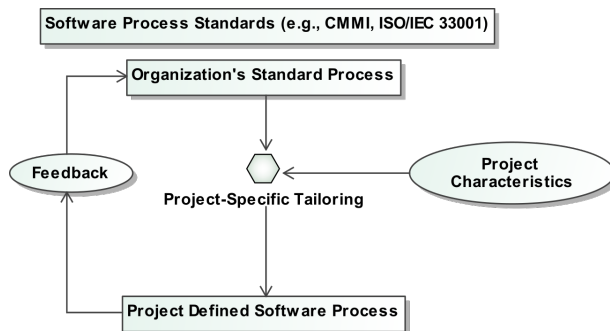
Figure 2.1.: Tailoring the organization's standard process [Zahran, 1998]

The standard definition may be tuned or tailored down to be used in different projects with specific needs (see Figure 2.1). This mechanism is known as *Process tailoring* (see Figure 2.1). Several formalisms may be used. These include [SPEM, 2008a] and the [ISO/IEC 24744, 2007] (these will be discussed in details in Chapter 6).

Process training is related to an important aspect of disciplined software development, which is consistency. In fact, a software process cannot be enacted without the active support of the people who will use it. Thus, the practitioners are trained to the philosophy of disciplined software development and to follow the standard process consistently in order to avoid non-uniformity which is considered as a source of degradation of the process performance.

Process monitoring refers to the need of inspecting the process activities in order to ensure their conformance. *Enforcement* consists in a set of activities and procedures aiming to uncover and correct the sources of non-conformance.

2.1.3 Approaches Overview

In the reminder of this section, we provide an overview of the prominent disciplined methods which embody the characteristics presented in Section 2.1.2

2.1.3.1 SDLC Models

As explained in Section 2.1.2, in a DSD environment, the development process usually follow a sequential workflow, where plans are established rigorously. These models are commonly known as Software Development Life-Cycle (SDLC) models. They appeared in the 70's as a will of more discipline after the so-called software crisis [Naur and Randall, 1968]. They aim to assess the chronological evolution of a software product or system by describing in a simplistic way the fundamental phases and deliverables of the development process. They represent only partial information about the software process, e.g., they do not show the roles of the people involved in the described activities nor describe the documents and detailed work products to be released.

The SDLC models include the Waterfall Model [Royce, 1970], the V-model [Forsberg and Mooz, 1991] and the spiral model [Boehm, 1988]. These are synthesized in the following:

Waterfall Model

The waterfall model was introduced by [Royce, 1970], specifically in the context of spacecraft mission software design. It is very simple to understand and use.

In the waterfall model, the software life-cycle is displayed as a linear sequence of activities moving along a horizontal line, punctuated by major reviews (system requirements review, preliminary design review, etc.). Each phase must be completed fully before the next

Figure 2.2.: The waterfall model

one can begin, with limited possibility of going backwards. This model is risky and invites failure since the testing phase occurs only at the end of the development cycle. If the product fails to satisfy the various external constraints, then a major redesign is required.

V-Model

The V-model (validation and verification model) is a variation of the waterfall model with a focus on build and test activities which was first introduced in [Forsberg and Mooz, 1991]. Its "V" shape (shown in Figure 2.3) was inspired by the [DOD-STD-2167A, 1988] standard which describes a model integrating two parallel waterfall cycles for hardware and software development.

The V-model starts with the same sequence of phases of the waterfall model and adds verification and validation activities. It is shaped as a "V" starting with user needs on the upper left and ending with a user-validated system on the upper right. The definition of tests starts in parallel of the preliminary phases of analysis, specification and design.

This model best shows the disciplined way of testing. The verification and validation activities of the V-model are thoroughly planned and are driven by a set of formal test plans which are derived from complete and precise model-based requirements. An independent team of testers identify for each requirement the possible test scenarios. The V-model as it was originally
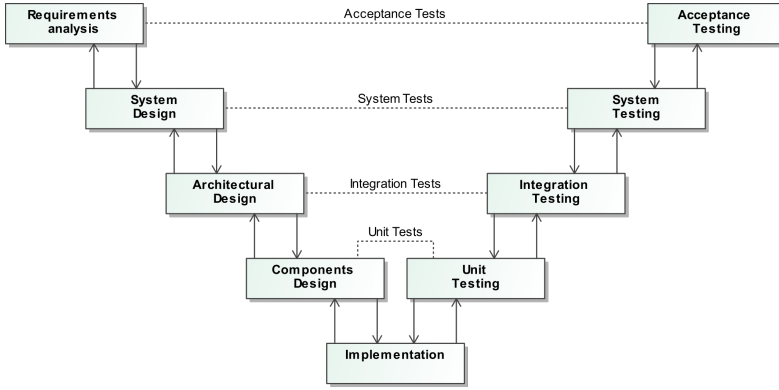
Figure 2.3.: The V-model of software development

described conforms to the [MIL-STD-1521A, 1976] standard and is therefore very prescriptive in terms of documents to produce during the software project life cycle. It demonstrates how much documentation is important in DSD methods.

Spiral Model

The spiral model [Boehm, 1988] is an incremental model to software development that focuses on the reduction of risks at each increment. It describes *"a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk"* [Boehm and Hansen, 2001].

As shown in Figure 2.4, the development of the product passes through cycles forming a spiral, each cycle is composed of 4 phases: (1) objectives, alternatives and constraints definition, i.e., solution exploring by identifying the design alternatives, the reusable components, the cost and schedule constraints, etc.,
(2) risks identification, evaluation and resolution using prototyping, simulation, benchmarking, questionnaires, analytic modeling, etc.,
(3) implementation, validation and verification of the selected solutions,
(4) planning of the next increment.

At the end of each cycle, a block of functionalities or a version of the product is released. The radial dimension represents the cumulative cost incurred in accomplishing the steps to date; the angular dimension represents the progress made in completing each step.
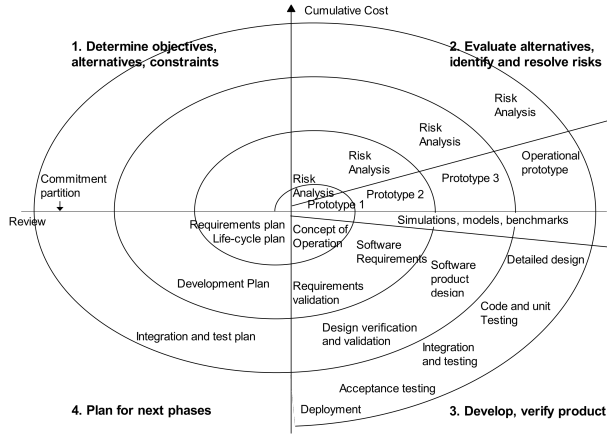
Figure 2.4.: The spiral model [Boehm, 1988]

The spiral model in particular influenced the disciplined standards of software development such as the [DOD-STD-2167A, 1988] which evolved and incorporated evolutionary strategies in requirements acquisition and product development [Boehm and Hansen, 2001]. It also paved the way for agile methods.

2.1.3.2 Military Methods (DOD)

As explained in Section 2.1.1, disciplined software development appeared with the rise of complex aerospace systems and system engineering [Schlager, 1956]. Military standards and methods were introduced to guide the software development endeavor in Mission-Critical systems.

The [DOD-STD-2167A, 1988] Standard, titled the Defense Systems Software Development, best exemplify this type of methods. It is a document-driven approach that specifies a large number of deliverables and follows a waterfall life-cycle model (see Section 2.1.3.1). The standard also provide an overview of the sequential reviews and audits required in the [MIL-STD-1521A, 1976].

2.1.3.3 PSP and TSP

The Personal Software Process (PSP) is a structured method or framework to be followed by an individual programmer. It has been developed by [Humphrey, 2000] as a result of applying the Capability Maturity Model for Software (SW-CMM) [SW-CMMI, 2002] to the smallest organization possible: an organization of a single individual. It consists basically in guidelines and procedures to help the programmer structure and control its way of working. Another purpose of the framework is to help him leverage its own skills and performance by following more discipline.

The framework rely on several documents: scripts, forms, templates, standards, and checklists. A script describes ordered tasks that the engineer should go through to complete a process step. Forms are used to guide thorough data collection. They serve for example for recording bugs or how much time the individual spend on development activities.

The Team Software Process (TSP) is a method that complements the PSP by providing a structure for self-directed teams. It helps them to create and own their processes, produce their own plans and make their own commitments. The objective of the TSP is to put software professionals in charge of their work and to make them feel personally responsible for the quality of the products they produce [Davis and Mullaney, 2003]. It has also been developed by [Humphrey, 2006].

Like the PSP, TSP also rely on scripts, forms, and standards organized in two components: team building and team management. The scripts lay out detailed steps to guide the teams through the project monitoring. Standards are used to help the engineers share common values and coding rules. For example, the TSP includes a standard for defects types and quality criteria that is used to produce the defect recording log.

2.1.3.4 RUP

The Rational Unified Process (RUP) is a process framework developed in the late 90's by Rational Software, a division of IBM. RUP is not a single concrete process but an adaptable framework for instantiating well-defined iterative and incremental processes. Despite this iterative and incremental nature which usually characterizes agile methods (see Section 2.2.2), we classify RUP as a disciplined approach since it is highly focused on process standardization and guidance which conforms to the DSD characteristic detailed in Section 2.1.2.4.

RUP provides development organizations with industry-standard best practices to help them define their standard organizational processes. With such focus on process discipline, it targets to produce high quality software deliverables within a predictable schedule and budget.

The RUP life-cycle model can be described in terms of two dimensions as shown in Figure 2.5. The horizontal axis represents *time* and shows the dynamic aspect of the process as it is enacted, i.e., how the project lifetime is decomposed into phases and iterations. The vertical axis represents *content* and shows the static aspect of the process, i.e., how it is described in terms of software development disciplines (which are basically a high-level view of workproducts and activities). The model shows that the focus on development disciplines vary over the project lifetime. For example, more business modeling is done during the early iterations of the inception phase and more implementation and test is done during the later iterations.

RUP is architecture-centric and is based on the Unified Modeling Language(UML). It defines a set of reusable process components and guidelines to help organizations define their standard processes. These organizational processes are managed in a highly disciplined way, following a top-down approach of defining, deploying and tailoring them.
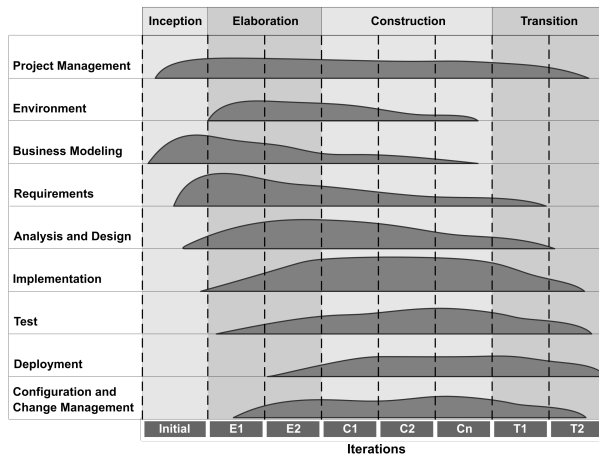
Figure 2.5.: The RUP Life-cycle

2.1.4 Software Quality Management

In DSD, the software quality management system mainly relies on formal Software Quality Assurance (SQA), which can be defined as: *"The planned and systematic approach to the evaluation of the quality of and adherence to software product standards, processes and procedures."* [Agarwal et al., 2007].

Quality assurance activities can be split into two categories. The first category aims to prevent the defects that would arise during the development, by defining a standard way of working, checkpoints and reviews. This is better known as process assessment and improvement (see Section 2.1.5).

The second category focuses on defect detection into the completed product and is better known as Quality Control. Typically, a dedicated quality team is in charge of establishing a quality plan for each project, agreeing it with the customers and making it visible to all projects' stakeholders. The plan would identify, along industry standard lines, the quality goals for the given project (e.g., the percentage and aggregated number of defects that the customer would tolerate per release) and how those goals would be achieved (e.g., inspection, testing, etc.). Once the software is fully built and functioning, the quality plan serves as a reference for quality stakeholders to verify the produced solutions and to track the reasons for poor quality.

In order to be able to thoroughly define these quality goals, smaller and easier-to-assess quality attributes are needed. This constitutes a key idea in disciplined quality management. Several quality models and standards provide such a characterization of software quality, e.g, McCall's model [McCall et al., 1977], Boehm's model [Boehm, 1981], FURPS [Grady and Caswell, 1987], Dromey's model [Dromey, 1995], [ISO/IEC 9126, 2001] model, etc. These models explicitly define the set of inherent characteristics that are further refined into subsequent indicators, which are quantitatively accessible thanks to specific metrics. Using this structured way of defining quality goals, well-defined and standardized SQA plans can be established.

Many disciplined methods rely on these models and provide rules, tools, and heuristics to measure the product's internal attributes such as modularity, reuse, coupling, cohesiveness, redundancy and hierarchy and external attributes such as understandability and maintainability [Berki et al., 2007].

This "objective" or "rationalized" view of quality is originally taught in the works of influential quality standards such as the [ISO/IEC 9126, 2001] which states that : *"Software quality is the degree to which the software product satisfies stated and implied needs when used under specified conditions."* where "stated needs" means those needs that are specified as requirements

by the customer in a contract and "implied needs" refers to the needs identified by the supplier company.

This view of quality is often criticized. For example, [Juran, 1998] claims that because of this standardized view of quality, organizations were stampeded into going after a set of quality standards instead of going after improvement at a revolutionary rate. He recommends a more flexible view of quality which adapts to the customer expectations.

2.1.5 Software Process Improvement

Software Process Improvement (SPI), i.e., *the definition of a series of actions in order to change and optimize existing processes and to achieve gains in product quality and productivity*, is a major concern in disciplined software development environments.

As explained in Section 2.1.2.4, discipline encourages process infrastructure. An emphasis is therefore put on the improvement of processes in terms of compliance and capability, i.e., *the inherent ability of process to produce planned results*. As the capability is improved, the process becomes predictable and measurable which would lead to a better control of the major causes of poor quality. By steadily improving the process capability, an organization and its standard processes are said to mature.

The typical mechanisms of a disciplined process improvement framework are depicted in Figure 2.6.
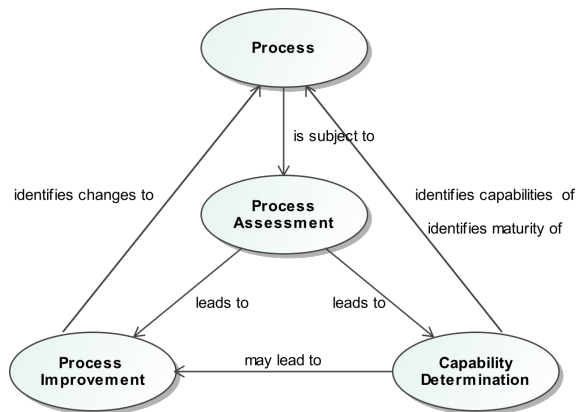
Figure 2.6.: The elements of a disciplined SPI framework

Software process improvement begins with an assessment of the software process on the basis of a benchmark model (e.g., [ISO/IEC 12207, 1995] and [ISO/IEC 9000, 2005]) or using a measurement-based approach. Benchmarking requires to compare each specific process to be assessed to the requirements described in the benchmark model to ensure that it meets a set of basic process quality goals and that the software is implemented in an efficient manner. Measurement-based assessment provides means for defining measurable goals to evaluate the effectiveness of the used processes and understand the effects of implemented improvements [Pfleeger and Rombach, 1994]. A wide range of measurement methods provides quantitative mechanisms to support measurement-based process assessment. These include the Goal-Question-Metric (GQM) method [Basili et al., 1994a]. The output of process assessment is a set of areas to improve and a capability determination.

SPI requires significant expertise, time, and money [Boehm and Turner, 2003]. Therefore, the SPI initatives are generally encompassed in an organizational strategy. Usually, a collection of specialists, known as the process group, facilitate the definition, documentation, maintenance and improvement of organizational software processes. Moreover, the infrastructure for disciplined process management includes process asset libraries and rely on process frameworks such as the [ISO/IEC 12207, 1995] and the [ISO/IEC 9000, 2005] and on process improvement approaches which can be categorized into two main categories: prescriptive and inductive.

2.1.5.1 Prescriptive Approaches

Prescriptive approaches take an approach based on a set of best practices that has proven successful in other organizations. They aim at providing organizations with a reference or a normative model that is assumed to be the best way of developing software. It is the model against which the organization processes can be assessed and improved.

By using this model, it becomes possible to identify the "maturity" of the organization and propose relevant improvements. Basically, the idea is to examine the process against the reference model and to evaluate its capability to develop software effectively, i.e., to accomplish a set of predefined process goals. A capability level is attributed regarding this evaluation and a set of improvements are recommended to help the organization upgrade the attributed capability level. Examples of this type of models include the Software Process Improvement Capability dEtermination (SPICE) [ISO/IEC 15504, 2004] (actually revised by the [ISO/IEC 330xx, 2015] series of stan-

dards), the Capability Maturity Model Integration(CMMI) [CMMI, 2006], Trillium [April and Coallier, 1995], Bootstrap [Kuvaja, 1995] and so forth.

In the reminder of this section, we present some of these prescriptive frameworks:

SPICE

The [ISO/IEC 33001, 2015] describe a set of standards for software process assessment and improvement. It relies on the principle of process institutionalization. In fact, a process is defined as : *"an implemented process that is managed and tailored from the organization's set of standard processes according to tailoring guidelines"* [ISO/IEC 33001, 2015].
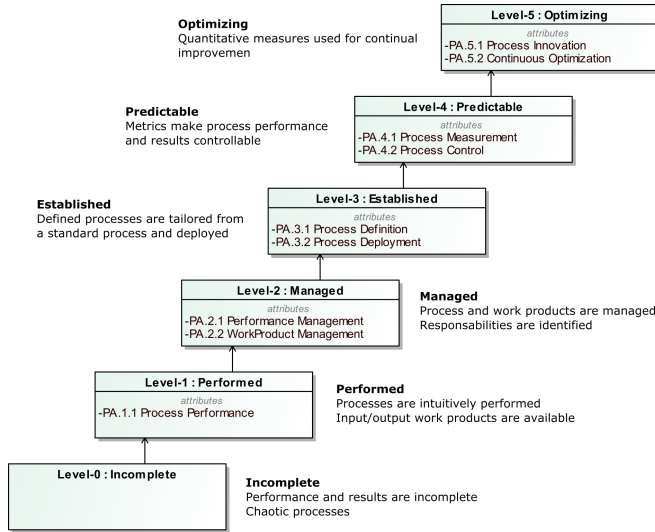
Figure 2.7.: [ISO/IEC 33001, 2015] process capability levels

SPICE proposes to assess the process against a reference model (an example is shown in Figure 2.7). A process reference model consists in *"capability levels which in turn consist of the process attributes and further consist of generic practices. [It is the model] against which the assessors can place the evidence that they collect during their assessment, so that the assessors can give an overall determination of the organization's capabilities for delivering products (software, systems, and IT services)"* [ISO/IEC 15504, 2004]. In other terms, the model fixes the scope of the assessment by precisely describing the process entities to be evaluated. Each process attribute describes a specific

aspect of the overall assessment. The standard specifies that different models may be used to support the conduct of an assessment but these should be mapped to a relevant reference model from the [ISO/IEC 15504, 2004] using a measurement framework specified in the [ISO/IEC 15504-2, 2004].

The assessment output consists of a set of process attribute ratings for each process assessed, termed the process profile, and the process capability achieved by that process. SPICE defines an ordinal scale for evaluating process capability, based upon six-defined capability levels: (0) incomplete, (1) performed, (2) managed, (3) established, (4) predictable and (5) optimizing.

CMMI

The Capability Maturity Model Integration (CMMI) is a framework of best practices designed to guide process improvement across a project, a division or an entire organization. The area of interest of CMMI addressing product and service development is called [CMMI-DEV, 2010]. It *"deals with the capability of software organizations to consistently and predictably produce high quality products"* [Humphrey, 1993].

The underlying principles of CMMI are similar to those of SPICE. The maturity of an organization process is also assessed against are reference model. It defines the following maturity levels for processes: (1) Initial, (2) Managed, (3) Defined, (4) Quantitatively Managed and (5) Optimizing.
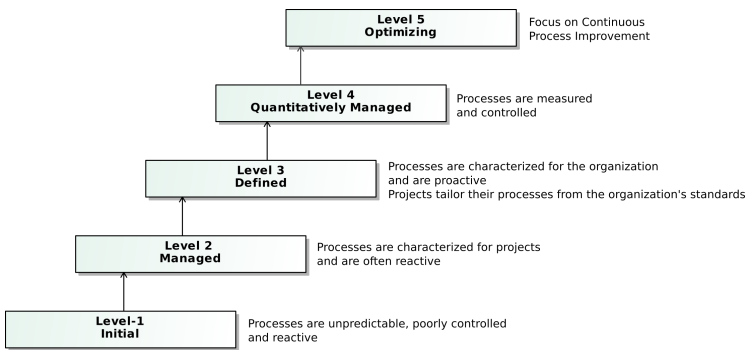
Figure 2.8.: [CMMI, 2006] process maturity levels

The CMMI has two representations: continuous and staged. The continuous representation is designed to allow the organizations focus on specific process

areas so they can primarily address the urgent improvement needs and specific goals. Maturity levels are available for each process area. The staged representation provide a standard improvement path. An overall maturity level for the organizational process is provided.

According to CMMI, SPI typically involves management steering committees implementing strategies, process groups facilitating and managing the SPI activities, and process action teams defining and implementing the improvement. The role of the practitioners is defined as performing the process.

2.1.5.2 Inductive Approaches

Inductive approaches are based on a thorough analysis of the current practices, on the identification of specific improvement goals with regards to the most critical issues and on the management of improvement activities supported by measurement [Van Solingen and Berghout, 1999].

Such approaches consists of organizational models for continuous quality improvement that are possibly supplemented by goal-driven measurement techniques. They provide systematic procedures to conduct the improvement activities in a cyclic and ongoing way and support measurement on the improvement goals. Examples of these models are Six Sigma [Harry, 1998], IDEAL [McFeeley, 1996] and the Quality Improvement Paradigm (QIP) [Basili, 1985] which is supplemented by the Goal-Question-Metric (GQM) paradigm for handling the measurement aspects.

In the reminder of this section, we present some of these inductive frameworks:

IDEAL

IDEAL provides a disciplined long-term improvement strategy for organizational processes. It consists of five phases: (I) Initiating: preparing the basis of successful improvement, (D) Diagnosing: analyzing the current and desired states developing recommendations for improvement, (E) Establishing: planning the improvement actions, (A) Acting: implementing the improvement actions, (L) Learning: learning from the experience and propose future directions for improvement. The IDEAL cycle is shown in Figure 2.9.

25
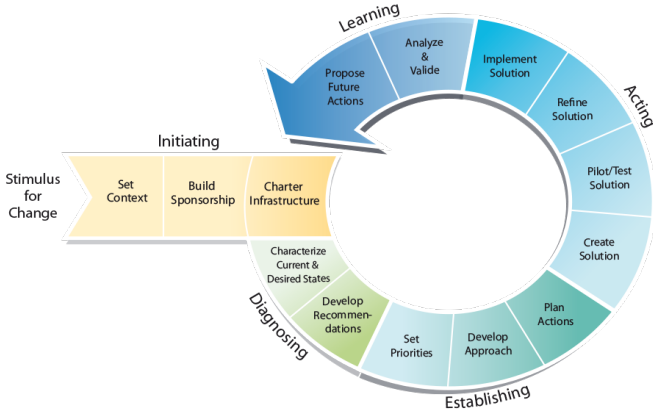
Figure 2.9.: IDEAL model [McFeeley, 1996]

QIP

The Quality Improvement Paradigm (QIP) originates from the quality paradigms of the manufacturing such as the Shewart-Deming Plan-Do-Chech-Act cycle (PDCA) and the Total Quality Management (TQM) [Ishikawa, 1976]. It is a basic methodological device of a software process infrastructure system called the experience factory [Basili et al., 1994b]. It is based on the principle that software discipline is evolutionary and experimental. The process improvement is done on the basis of the project and environmental characteristics and specific goals.

The QIP cycle consists of six steps (see Figure 2.10): (1) Characterize the environment, (2) Set Quantifiable Goals on the basis of the initial characterization and of the capabilities of the organization, (3) Choose the Process on basis of the characterization of the environment and of the organizational goals, (4) Execute the Process and provide project feedback, (5) Analyze the data gathered from each project and make recommendations for the future, (6) Package the experience in form of updated models or structured knowledge

These steps are performed at two different levels, forming a corporate learning cycle and a project learning cycle. The corporate cycle has the purpose of initiating the process for each project, analyzing and comparing the process execution data with the available knowledge and accumulating reusable experience. The project learning cycle ensures the feedback provided to
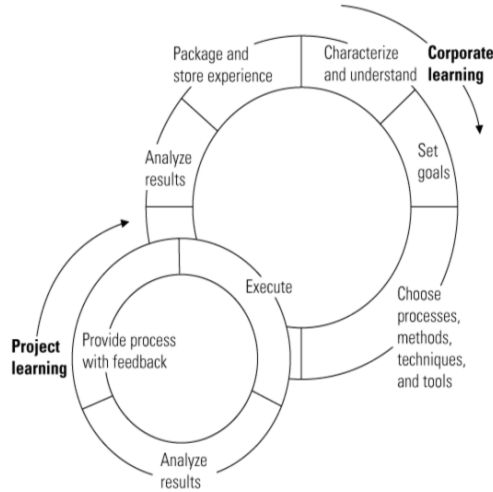
Figure 2.10.: QIP model [Basili and Caldiera, 1995]

the project during the execution phase. The improvement of the process basically rely on the analysis of the quantitative indicators of the project.

GQM

Within the QIP, software measurement is considered as a crucial component to capture experiences and retrieve knowledge on the development practices. To collect such information, measurements activities are integrated to the software development process [Van Solingen and Berghout, 1999]. The Goal-Question-Metric (GQM) [Basili et al., 1994a] is the mechanism used by the QIP for defining measurable improvement goals.

GQM relies on the concept of goal-oriented software measurement, i.e., it assumes that any software measurement should be only performed towards a clearly stated purpose. At a conceptual level, the approach describes a purposeful way to define measurement goals specific to the organization and its projects. At an operational level, a set of relevant questions are used to characterize the way a specific goal is to be assessed or achieved. Finally, at a quantitative level, a set of data is associated with each question in order to answer to it (see Figure 2.11).

The measurement goals represent corporate or project improvement goals and are defined using five dimensions: object of the study, purpose, quality

focus, point of view and environment of use. The goals are then transformed into activities that can be measured during the execution of the development process. The result of the application of the GQM approach is a measurement program targeting a particular set of issues and a set of rules for the interpretation of the measurement data. The GQM method defines the metrics from a top-down perspective and interprets the measurement data bottom-up as shown in Figure 2.11.
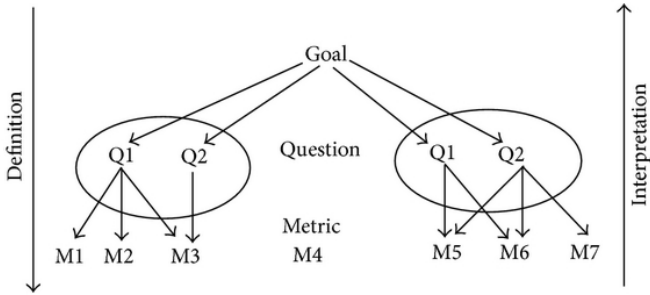
Figure 2.11.: The Goal-Question-Metric paradigm [Basili et al., 1994a]

2.2 Agile Software Development

2.2.1 Defining Agility

The term "agile" means "*characterized by quickness, lightness, and ease of movement; nimble*" [2]. Agility is therefore the ability of the software process to respond effectively to change. It was first used in the field of flexible manufacturing and later was introduced in the software community to designate lightweight methods for producing software products rapidly.

---

[2]The American heritage dictionary of the English language, 5th edition

Table 2.1.: The values and principles of the Agile Alliance

The Agile manifesto defines four values for characterizing Agile Software Development. These values indicate preferences and not alternatives encouraging the focus on certain aspects of the software development life-cycle but not eliminating others. That is, while the items on the right are important, the items on the left are even more. The 4 values are :

- *V1. Individuals and interactions* over *processes and tools*

- *V2. Working software* over *comprehensive documentation*

- *V3. Customer collaboration* over *contract negotiation*

- *V4. Responding to change* over *following a plan*

The Agile Alliance also defined twelve principles to describe the fundamentals to which and agile software process must conform. For each of the principles (AP01 to AP12) we state to which value the principle relates.

- *AP01. Focus on User / Customer:* The user/customer satisfaction should have the highest priority. This is achieved through their involvement in the development process and the early and continuous delivery of valuable software. (V3)

- *AP02. Welcome change:* Changes even late in development should be tolerated. Agile processes harness change for the customer's competitive advantage. (V4)

- *AP03. Frequent deliveries of valuable software* : Working software should be delivered frequently to the customer. Deliveries should happen, from a couple of weeks to a couple of months, with a preference to the shorter timescale. (V2)

- *AP04. Business/IT collaboration:* Business people (i.e., managers, business domain experts, administration, etc.) and developers must collaborate toward one common goal on a *daily* basis throughout the project life. (V2)

- *AP05. Team motivation and empowerment:* Motivated individuals are the

secret of successful projects and hence projects should be build around them : they should be given the adequate environment that support their needs and trust them to get the job done. (V1)

- *AP06. Face-to-face communication:* The manifesto states that this is the most efficient and effective way of conveying information to and within a development team. (V1)

- *AP07. Focus on working software:* The progress of the software developed should be tracked based on working software. Only working, integrated and tested features can be used to measure the progress. (V2)

- *AP08. Sustainable pace:* Agile processes promote a sustainable pace through the development life-cycle : The sponsors, developers, and users should be able to maintain a constant pace indefinitely. (V1)

- *AP09. Technical excellence:* Continuous attention to technical excellence and good design enhances agility. Having a high product and design models quality allows for easy maintenance and later changes, thus making a project more agile. (V2)

- *AP10. Focus on simplicity* : Simplicity is defined by the Agile Alliance as "the art of maximizing the amount of work not done". Going straight to the point is an essential value in the agile paradigm. It avoids time and resource losses. (V4)

- *AP11. Self-organizing and cross-functional teams:* A group of motivated individuals (AP05), who work together toward a goal (AP04), have the ability and authority to take decisions and readily adapt to changing demands. The manifesto says that best architectures, requirements, and designs emerge from cross-functional teams (i.e., involving all competences internally). (V1)

- *AP12. Regular adjustment:* At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. (V4)

According to [Boehm and Turner, 2003], agility is to be perceived as the counterpart of discipline. He specifies that *"when discipline ingrains and strengthens, agility releases and invents"*. It drives innovation by releasing development teams from the heavyweight process chains and by encouraging them to adopt an adaptive rather than a predictive mindset.

The first Agile methods emerged in the mid-1990s to address the challenges inherent to disciplined software development. They were initially called lightweight to distinguish them from heavyweight methodologies. Later, a group of seventeen software engineering practitioners formed the Agile Software Development (ASD) Alliance in February, 2001. The group concretely managed to define a consensual manifesto of four values and twelve principles for encouraging better ways of developing software [Beck et al., 2001]. These values and principles reported in Tab. 2.1 became one of the most referenced characterization of agile software development.

Lean thinking, a management philosophy derived from the Toyota Production System, provides an alternative foundation to agile software development known as the Lean software development. Tab. 2.2 summarizes the seven key principles of lean software development as defined by [Poppendieck and Poppendieck, 2003].

The Lean foundation share several principles with those stated by the Agile Alliance. The two software development paradigms have many in common and the underlying principles can be mapped together as proposed in [Petersen, 2010]. For example, AP03 (frequent delivery) and LP04 (deliver as fast as possible) are similar principles since they both target faster response time. Similarly, both Agile and Lean aim to improve the customer service, to manage the product quality continuously and to empower the teams. They are therefore sometimes considered as interchangeable.

The main difference between the two methodologies resides in their target. Lean is designed to eliminate waste and thus improve the operational efficiency of a manufacturing-like continuous workflow. However, Agile is designed to continuously increase the business value and thus improve effectiveness of the what is delivered to the customer. This is done by executing tasks over a short time frame called *iteration* and by frequently involving the customer in the development process. The iterative and incremental development is therefore a pillar of ASD (see Section 2.2.2).

It is important to note that the aforementioned values and principles provide a solid foundation of the agile movement but they are relative, not absolute. They represent the philosophy that development teams would attempt to

Table 2.2.: The principles of Lean Software Development

Lean is based on seven principles which are:

- **LP01. Eliminate waste** : Eliminating anything that does not add value to a product (i.e., value as perceived by the customer) is a basic concept in Lean Thinking. Seven types of waste were identified in manufacturing and transported to software development in Poppendieck and Poppendieck [2003]. The seven wastes of Software Development are identified as follows: Partially Done Work, Extra Processes, Extra Features, Task Switching, Waiting/Delays, Motion/Handovers, Defects.

- **LP02. Amplify Learning:** : Software development is a problem-solving process that requires intensive knowledge and where learning is essential during the whole development life-cycle. Learning includes a better understanding of the customer needs, architecture solutions, testing strategies, and so forth.

- **LP03. Decide as late as possible:** Software development is a process that involves variability and uncertainty. Just like the most economic markets that face uncertainty, the development team should consider options and defer commitment until the future is easier to predict. Thus, the decision-making is based on facts and not on speculation.

- **LP04. Deliver as fast as possible:** Lean has a strong focus on short cycle times, i.e. delivering a potentially

amount of features (determined by the team) on a regular basis. This helps the team to stay focused on continuously adding value.

- **LP05. Empower the team:** When equipped with adequate expertise and leadership, empowered team will make better technical and process decisions than anyone can make for them. Just like in lean manufacturing, software teams use pull techniques(i.e., an agreement to deliver increasingly refined versions of working software) and local signalling mechanisms (i.e., visible charts, daily meetings, frequent integration, etc.) so team members know what needs to be done.

- **LP06. Build integrity in:** : This is about to find and fix defects at the moment they occur. Inspecting after the fact, and queuing up defects to be fixed at some time in the future, isn't as effective. This principle encompass to integrate early and often. Finding defects during the final verification means that the development process is defective.

- **LP07. See the whole** : When improving the process of software development the whole value-stream needs to be considered end to end. The focus should be on the overall system performance rather than the individual and specialized contributions, which is likely to result in sub-optimization.

have and not a state of accomplishment. They can be implemented differently using several methods and practices (see Section 2.2.3).

Based on the aforementioned discussion, we propose the following definition of Agile Software Development which will be our reference in the remaining of the document.

Definition 2.2 (Agile Software Development)**.** *A group of lightweight approaches to software development which apply a level of flexibility into the*

delivery of the finished product. They are iterative and incremental and follow a set of values and principles, as expressed in the Agile Manifesto [Beck et al., 2001].

2.2.2 Characteristics

The agile alliance manifesto provide a solid foundation of the agile movement but more precise definitions were proposed to characterize what agility is.

[Lindvall et al., 2002] specify that the primary goal of agile software development is fast delivery, i.e., reducing the process cycle time. This goal would be realized through the following process characteristics: *iterative* (i.e., based on repetitive time-frames called *iterations*), *incremental* (i.e., each new product release add business-value and is usable), *self-organizing* (i.e., the team members are empowered, they have the ability and authority to take decisions) and *emergent* (i.e., each increment reduce the risks of failure of the project).

From the practitioners point view, different characteristics can be identified. For example, these can be found in [Beck and Andres, 2004] and [Highsmith, 2009]. We summarize them as follows: *lightweight* (i.e., the process contains only what is needed to create value for the customer), *efficient* (i.e., the process delivers a product with as less bugs as possible), *low-risk* (i.e., the risks are identified and managed early in the life-cycle), *reliable* (i.e., team members figure out ways to consistently achieve a given goal even though the inputs vary dramatically), *empirical* (i.e., the process is actively improved according to past experiences), *fun-way* (i.e., the team should enjoy the way of working).

Regarding the aforementioned review, we suggest to summarize the essential characteristics of agile methods as the following:

- *Evolutionary*,

- *Lightweight*,

- *Adaptative*,

- *Emergent*,

- *Collaborative*

2.2.2.1 Evolutionary

ASD methods are characterized by evolutionary delivery models. These include Iterative and Incremental Development and Continuous Development (a characteristic of Lean). Iterative and Incremental Development (IID) (see Figure 2.12) was created as a response to inefficiencies and problems found in the disciplined SDLC models explored in Section 2.1.3.1. The basic idea behind this approach is to develop a software through repeated cycles (iterations) and in smaller portions at a time (increments), so that developers can learn from earlier parts or versions of the system. The software problem will therefore be solved through successive approximations of the solution. Therefore, Agile is said to be empirical, i.e., based on successive experiences.

An iterative process indicates that the development is realized through successive time-boxed cycles or iterations. During each iteration, a set of defined activities are repeated.

An incremental process indicates that a smaller portion of the problem is resolved at a time. The agile approach subdivide the specific system into small subsystems. With each new release, the team implement new valuable functionalities and deliver a fully tested and usable subsystem.
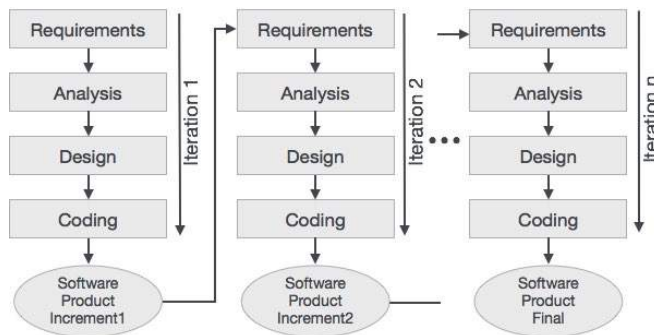
Figure 2.12.: The iterative and incremental development model

Usually, a maximum of functionalities are identified of the start of the project. They are refined throughout iterations and are broken down and built incrementally. Each set of functionalities passes through the design, implementation and testing phases before their release. The product is considered as finished when all functionalities are released and satisfy all the requirements.

2.2.2.2 Lightweight

Lightweight means that the agile process contains only what is needed to create value for the customer. This implies minimizing the work to be done in the development process (e.g., documentation, meetings, requirements etc.) in order to increase efficiency.

This characteristic is about identifying and achieving what is "just barely good enough or sufficient". For example, agile methods tells as to think about how much and which kinds of documents are needed and when they need to be written (see V2 in Table 2.1). Therefore, agile teams usually capture requirements at a high level and detail them just in time for each functionality to be developed. The rationale for this is to minimize the time spent on anything that does not actually form part of the working software.

2.2.2.3 Adaptative

ASD methods assume that changes are inevitable. Thus, rather than spending efforts trying to reduce variability, an agile team should manage to handle it efficiently. This characteristic implies that the process, requirements, planning, code and the design/architecture must be allowed to change to the advantage of the customer. This is in accordance with the AP02 principle (see Table 2.1). To achieve adaptability, agile methods actively involve users so they establish, prioritize and validate requirements.

2.2.2.4 Emergent

Emergence means that the processes and product boundaries are recognized during the project rather than predefined. The requirements emerges as the project knowledge is updated thanks to the close collaboration with the customer. The process emerges as the effectiveness of the work performed and methods used is estimated. The quality of the product emerges from the continuous handling of bugs and refactoring of code.

2.2.2.5 Collaborative

Collaboration simply means to work jointly rather than independently to accomplish a task. This is an essential characteristic in all agile methods. It implies four things : *continuous feedback*, *self-organization*, *communication* and *motivation*. Firstly, an agile team creates the environment for continuous feedback. This especially includes the feedback to the business

representatives and the customer. Secondly, an agile team is self-organized and empowered, i.e., the whole team is responsible for taking the decisions and for commitment. Thirdly, communication is an important aspect for achieving good collaboration. Face-to-face conversations and visual informative tools are encouraged. Finally, the team's motivation should be retained all-through the project life-cycle. Impediments should not derail the creative impulses of the team. Motivation is encouraged by partnership and self-directness.

2.2.3 Approaches Overview

Several agile software development methods have been developed over the two past decades. Most of them describe lightweight development process models with few prescriptive guidance which make them flexible and applicable to different agile software development projects.

However, some agile frameworks tend to be more prescriptive regarding their implementation at an organizational level. These produce well-defined process models with a perceived focus on control. We will refer to this category of agile methods as *hybrid agile-disciplined methods.*

In this section, the following agile methods are selected based on the characterization proposed in Section 2.2.2: eXtreme Programming (XP), Scrum, Feature Driven Development (FDD), Agile Modeling, Devops, Lean Software/kanban, RUP and DSDM atern.

2.2.3.1 Extreme Programming

EXtreme Programming (XP) is a popular agile method that aims to produce higher quality software, and to provide greater working experience for the development team in environments where requirements change frequently [Beck, 1999]. It relies on a set of commonsense development principles and practices which were formalized and aligned to form a novel methodology. The word 'extreme' comes from taking these principles and practices to extreme levels. XP is hence recognized as one of the most specific of the agile frameworks regarding appropriate engineering practices for software development [Agile Alliance, 2012].

As a type of ASD, it advocates frequent *"releases"* in short development cycles, which are intended to introduce checkpoints at which the customer requirements can be validated or adapted.
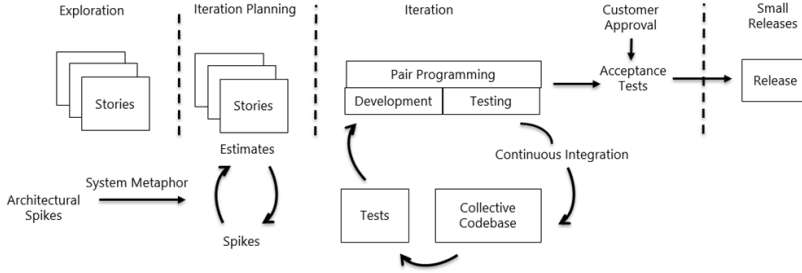
Figure 2.13.: XP process

The life-cycle of XP (see Figure 2.13) consists of exploration phase and 4 repetitive phases: iteration planning, iteration, customer approval and release. The exploration phase aims at understanding the project and describing the desired results as a set of *"user stories"* (a high-level definition of requirements, expressed in terms of business value and understandable by the customers). During the exploration phase, the team conducts *"spikes"* on any particular aspect of the project that seem to be risky, specifically on the architecture of the product to be developed. Once the high-level boundaries of the project are identified, the entire team gets together to create a release plan that everyone feels is reasonable. Then the team launches into a series of weekly iterations. At the beginning of each iteration, the team (including the customer) gets together to decide which stories will be realized during that week. Then the team breaks those stories into smaller tasks which can be affected to individuals or pairs. At the end of the week, the team and the customer review the state of the achievement and decide whether the project should continue or be released.

XP recommends several engineering practices such as pair-programming, refactoring, unit testing and collective ownership.

2.2.3.2 Scrum

Scrum [Schwaber, 1995] has emerged as one of the most popular agile software development methods [VersionOne, 2017]. It provides an iterative and incremental process framework that cuts through complexity by applying simple mechanisms such as requirements privatization, continual feedback, close collaboration, self-organization and activities time-boxing.

Scrum relies on an "inspect and adapt" process, i.e., the software is delivered in increments called "*sprints*" (usually 2-4 weeks iterations) (see Figure 2.14). Each sprint is required to deliver a potentially shippable product increment. This means that at the end of each sprint, the team has to produce a coded, tested and usable piece of software.
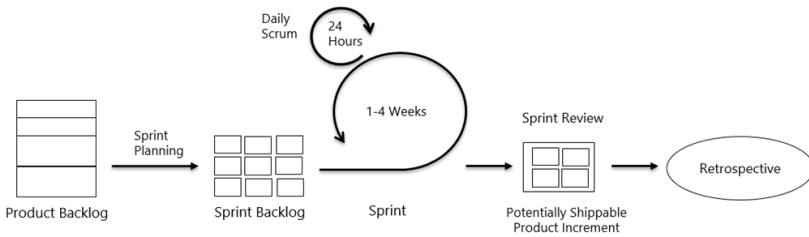
Figure 2.14.: Scrum process

Each sprint starts with a planning meeting and ends with a review of the product and a sprint retrospective. A sprint planning is a time-boxed meeting which could last up to 4 hours for a one-month sprint. It is dedicated to develop detailed plans for the Sprint and during which the exact functionalities to be implemented are discussed in details and decomposed into small tasks. Similarly to XP, the functionalities are expressed in terms of User Stories. The stakeholders of a project assist to this meeting to help the Scrum team understand the complexity of the requirements. During the sprints, the Scrum team meets daily for a 15-minutes stand-up meeting to inform about the work states and potential impediments. At the end of each sprint, the Scrum team organizes a review meeting to show the set of work accomplished during the sprint. Right after the review, a sprint retrospective is held to look for process improvement opportunities.

Scrum prescribes only 3 important roles, the product owner, the Scrum Master and the Scrum Team. The product owner is the project's key stakeholder in charge of the business. He manages the project vision and makes sure to share it with the team. He is responsible for prioritizing the list of features for the product. The Scrum master is responsible for ensuring a Scrum team applies the values and practices of Scrum. The Scrum Team represents the cross-disciplinary people in charge of developing the product. A Scrum team does not include any of the traditional software engineering roles such as programmer, designer, tester or architect. Everyone on the team works together to complete the tasks they have collectively committed to complete within a sprint.

2.2.3.3 Feature-Driven Development

Feature-Driven Development (FDD) [Palmer and Felsing, 2001] is a method
that applies the principle of Model-Driven Development (MDD) in the agile
way. Basically, the idea is to create "just barely good enough" models (see
Section 2.2.2.2) and to refine them each time a new client-valued functionality
(feature) is to be developed. The code is written based on the refinement of
the corresponding feature models.

The method consists of five basic phases, namely, the development of an
overall model, the building of a feature list, the planning by feature, the
designing by feature, and the building by feature (see Figure 2.15).
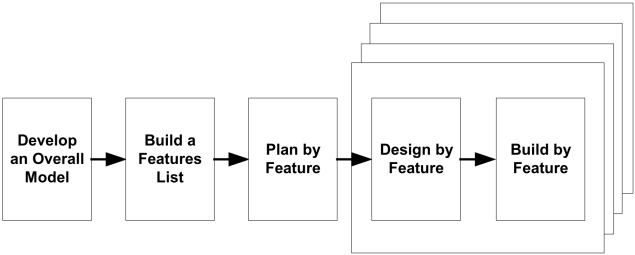
Figure 2.15.: FDD process

When the development of an overall model begins, the scope, context and
the requirements of the system are acknowledged [Palmer and Felsing, 2001].
The business stakeholders or domain experts conduct a meeting known as
a "domain walkthrough", where they provide the team with a high-level
description of the domain. As a result, an overall domain model is established.
More detailed walkthroughs can be conducted to detail the different domain
areas.

Then, based on the resulting domain model(s) and the existing requirements
documentation, a comprehensive features list is build. Features should be
detailed enough to take less than two weeks to complete. Once this list is
completed, the next step is to produce the development plan. Based on the
features priorities and dependencies, feature sets are created and assigned
to feature teams. The features are assigned as classes to individuals (class
owners).

Then features are designed and build iteratively. A design package is created
for each feature. Detailed diagrams are created and the overall model is
refined. Next, the class and method structures are written or generated

and a design inspection is held. Each class owner code its set of classes. After unit testing and successful code inspection, the completed feature is integrated to the main build.

2.2.3.4 Agile Modeling

Agile Modeling (AM) is an agile method which similarly to FDD focuses on the modeling and implementation phases. It implements the concept of Agile Model Driven Development (AMDD) (see Figure 2.16 and Table 2.3) using a collection of values, principles, and practices for modeling software.

The basic idea of AM is to consider that models only need to be good enough to reach the next release. The method propose to start by an initial envisioning phase, typically during the first week of a project, to identify the scope of the system to be developed (high-level requirements modeling) and a likely architecture for addressing it.

Then, similarly to any scrum and xp, the team selects and estimates the highest priority work to be developed during the iteration. During the iterations, the Just In Time (JIT) Modeling practice is applied. This consists in organizing modeling sessions involving a few people to discuss an issue while sketching on paper or a whiteboard. This is done whenever its needed and only for few minutes. Since it only focuses on the modeling activities, AM is to be used as a supplement to other agile methods such Scrum, XP or FDD.
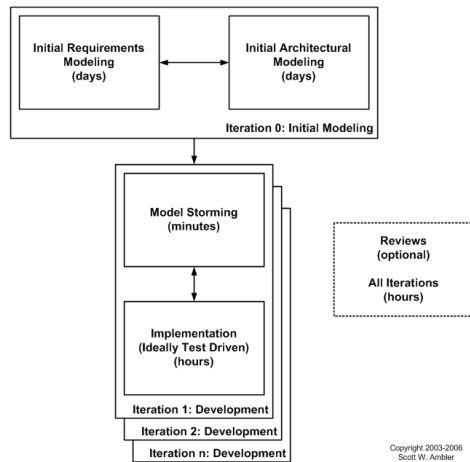
Figure 2.16.: Agile Modeling process

2.2.3.5 DevOps

DevOps [Httermann, 2012] (a clipped compound of "Development" and "Operations") is an agile method that integrates the development, deployment, and IT operation processes in order to deliver value more frequently, rapidly and reliably. It encompasses on a set of practices to improve the culture, measurement, automation and collaboration between all the team members

who are involved in creating, delivering, and monitoring software [Microsoft: DEV212x, 2017].

DevOps implementation can include the definition of the series of tools (a toolchain) used at various stages of the lifecycle (see Figure 2.17). The DevOps process starts by a planning meeting which bring together the business stakeholders, software engineers and IT operation professionals in order to plan the important activities of the project and define the metrics and indicators. Then they choose a small slice of full feature to develop so they can see how it will be received by the customer. Then using a continuous delivery pipeline, the operation team creates a script to configure a new server, the testers provision the test environment using the same script and the developers code the software. This is done iteratively until the development of the whole features.
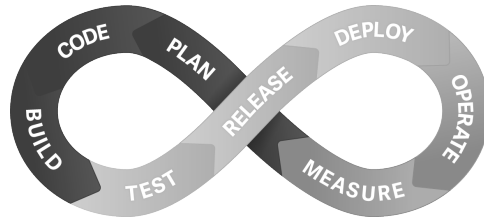
Figure 2.17.: DevOps process

2.2.3.6 Lean and Kanban

Lean software development [Poppendieck and Poppendieck, 2003] is a practice-based method that implements the set of principles and values presented in Table 2.2. Examples of the Lean practices (more commonly called tools in the lean words) include: seeing waste, value stream mapping, set-based development, pull systems queuing theory, cost of delay models, refactoring and test-driven development.

One famous tool of Lean is Kanban which in Japanese means *visual sign* or *placard*. It brings the concept of pull scheduling to enable the mechanism of "just-in-time" to software development. Using Kanban, a software process is perceived as a pipeline or workflow of feature requests passing through a number of workstations or stages (e.g., analyze the requirements, develop the code, test and deploy). This workflow is visualized using a kanban board populated with features or user stories (see Figure 2.18). At each stage, a Work In Prgress (WIP) limit can be defined. Usually, the velocity is

measured in terms of cycle times (i.e., the time that a features passes inside the pipeline).
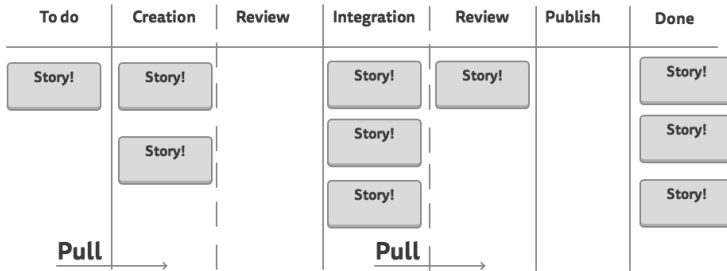
Figure 2.18.: An example of a kanban board

2.2.3.7 AUP and DAD

A lighter version of RUP (see Section 2.1.3.4) is the Agile Unified Process (AUP) [Ambler, 2006] and its simpler open source version, Open Unified Process (OpenUP) [Kroll and MacIsaac, 2006]. AUP applies a set of agile practices such as Test Driven Development (TDD), Agile Modeling (AM) (see Section 2.2.3.4) and refactoring. Recently, the AUP was superseded by the Disciplined Agile Delivery (DAD) Framework [Ambler and Lines, 2012], a a context-sensitive agile process decision framework.

DAD aims at guiding organizations in the implementation and tailoring of agile methods in a more disciplined way. It recognizes that one process size does not fit all situations and provides contextual advices regarding the alternatives of agile solution delivery. It does not recommend a unique set of practices but rather selects the most suitable from various agile methods such as Scrum, XP, AUP, Kanban, Lean and several other methods.

2.2.3.8 DSDM Atern

DSDM (Dynamic Systems Development Method) Atern [DSDM Consortium, 2008] is an approach to project management and solution delivery that embraces the principles of agile development with some discipline. It brings together values from agile development (good communication, business involvement, transparency) and from disciplined approaches (control and quality focus).

It proposes a process constituted of two sequential phases followed by three iterative phases (see Figure 2.19): feasibility study, business study, functional model iteration, design and build iteration and implementation iteration. This process is supplemented with a framework of control with appropriate guidance to ensure that the project manager is tracking and reporting the right things, and doing so in a manner which involves all the appropriate stakeholders.
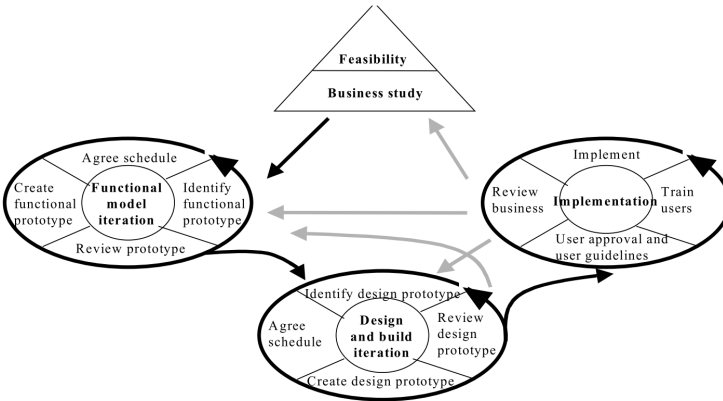
Figure 2.19.: DSDM process

2.2.4 Software Quality Management

In general terms, emergent trends in quality management tend to move to *continuous and holistic* (i.e., a matter of the whole organization) *quality assurance.* This idea seems to have matured with the Total Quality Management (TQM) approach [Kanji, 1990] which consists on the implementation of organization-wide efforts to continuously improve its ability to deliver high-quality products and services to customers. Quality is not only a matter of a dedicated quality assurance team which will control the quality of the software once the development is finished. Everyone's continuous commitment is essential to achieve it.

The agile perception of quality management falls within this perspective. In fact, since agile teams incrementally deliver working software, SQA activities are assumed to be engaged during each iteration, ensuring that both functional and structural quality of the system are addressed on regular basis. This is known as *emergent quality* or *built-in quality* [Poppendieck

and Poppendieck, 2003]. Moreover, agile teams embrace the "whole team" or "cross-functional team" principle which implies that everyone on the team is focused on the delivery of quality software. The notion of separating the quality assurance team disappears entirely. Software developers, testers, and quality-assurance personnel collaborate together on daily basis. Their roles can be interchanged or merged.

An essential role for ensuring the system functional quality is the customer itself. Quality in Agile is perceived as a *"constantly moving target based on the customer's actual experience with the product and that can be measured against his requirements (i.e., stated or unstated, conscious or merely sensed, technically operational or entirely subjective)"* [Feigenbaum, 1999].

Another essential characteristic of agile QA is taught by the first statement of the agile manifesto (see Table 2.1) which states that those who embrace agility values *"individuals and interactions over processes and tools (V1)"*. This implies that the QA process should be kept as lightweight as possible. Therefore, agile methods discourage the establishment of predefined quality plans and rather prescribe to simply do whatever is necessary at any moment to satisfy a customer's requests. This include the implementation of a set of practices such as refactoring, Test-Driven Development (TDD), Agile Model Driven Development (AMDD), pair programming or continuous integration [Ambler, 2005]. Tab. 2.3 gives an overview of some agile practices inherently related to software quality enhancement.

Technical Debt is a core concept which best demonstrate agile QA. It is a metaphor aiming to communicate about the need for refactoring the source code and its architecture. The metaphor consists in comparing code quality improvement tasks to financial debt: *"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite"* [Cunningham, 1993]. The metaphor explains that sometimes we need to develop rapidly and with few care about quality issues and that we should later improve.

2.2.5 Software Process Improvement

When considering the relationship between agile software development and SPI, there are one value and two principles in the agile manifesto (see Table 2.1) that deserve attention: (V1) the valuing of the interaction between individuals over processes, (AP11) the focus on simplicity and (AP12) the principle that encourages regular feedback and adjustment of the team's way of working. These highlight the importance of continuous improvement in

Table 2.3.: Overview of Agile quality assurance practices (adapted from [Ambler, 2005])

Practice	Description
Refactoring	The basic idea of refactoring is to make small changes to existing source code in order to improve the design, making it easier to understand and to modify. The code behaviour should not be affected. Resulting code is of higher quality.
Test-Driven Development (TDD)	This practice combines Refactoring with Test First Development (TFD) : Create a test, Run a test, adjust the functional code to make it pass the new test.
Agile Model-Driven Development (AMDD)	The approach consists of an initial requirements envisioning to understand the scope and the business domain followed by iterative model-driven development. In each development iteration, the team members perform Just-In-Time (JIT) model storming to quickly explore in detail a specific issue before they implement it (see Figure 2.16).
Acceptance Testing	An acceptance test is a formal description of the behavior of a software product, generally expressed as a business example or a usage scenario. They generally complete functional specification. Usually customers or domain experts are involved in the creation of acceptance tests.
Pair-programming	This practice consists of two developers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the "driver", the other, also actively involved in the programming task but focusing more on overall direction is the "navigator".
Continuous Integration	This practice has two objectives: (1) minimize the duration and effort required by each integration episode (2) be able to deliver at any moment a working version of the product, suitable for release.
On-site Customer	This practice consists of having the customer representative inside the development team, working with them on clarifying the requirements.

an agile environment and suggest that the actions to improve the process should be taken within ongoing projects by the practitioners themselves and

should be kept simple enough to not impede their interactions and effective collaboration.

From the perspective of the agile methods, very few recommendations about the process improvement endeavor are provided. In fact, most of the methods (see Section 2.2.3) suggest a simple meeting usually called "retrospective" or "reflection workshop" during which the improvement actions are decided. Therefore, several practitioners and researchers argue that agile teams need more support for conducting such improvement efficiently and systematically, instead of chaotically [Abbas et al., 2010; Salo, 2006; Ringstad et al., 2011; Sidky, 2007].

For this purpose, a set of studies has questioned the applicability of the traditional SPI frameworks in an agile environment. The prescriptive frameworks or process maturity models (see Section 2.1.5.1), precisely CMMI, has been particularly investigated to answer the following question: "How agile processes could be merged with standard industrial process models without impeding agility or undermining the organizational investment on maturing their processes?" [Boehm and Turner, 2005].

Such studies reveal little that would prevent teams concerned by process maturity appraisal from using agile methods. For example, [Anderson, 2005], [Fritzsche and Keil, 2007] and [Turner and Jain, 2002] demonstrate a possible level of agreement between CMMI and Agile and warn that some common CMMI practices need to be adjusted or discarded since they contradict the values and principles of agility. Consequently, it has bee shown that projects that use agile methods with certain adjustments can achieve CMMI level 2 or even 3. To achieve more agility with CMMI, [Glazer et al., 2008] argues that the key is to consider the list of CMMI practices as advisory or indicative. The organization is free to propose alternative practices and appropriate evidence.

Many agile proponents such as [Williams and Cockburn, 2003] argue that organizations using agility are less interested in standard process maturity models. This may be explained by the fact that such models support the ideology of creating a universal and repeatable software development process whereas it is considered as defective in Agile.

The inductive frameworks (see Section 2.1.5.2) such as the QIP and the IDEAL model are based on understanding the context and real needs of an organization. Therefore, they may be considered as better candidates to discuss agile process improvement. Researches such as [Sidky, 2007] and [Pikkarainen et al., 2005] provided a detailed discussion of how such models can be implemented in an agile environment. They argue that such models

are adapted for organizations seeking to implement a systematic approach for managing their agile process improvement life-cycle.

Yet, two central differences between the traditional inductive SPI models and agile SPI can be identified. Firstly, the origin of the improvement goals is traditionally the organizational level. However, the improvement initiatives in an agile project lies on the self-organizing team. Secondly, the improvement goals are traditionally defined based on the knowledge on finished projects while the immediate focus of process improvement in agile is the ongoing project [Salo and Abrahamsson, 2005; Qumer and Henderson-Sellers, 2008].

Regarding the previous discussion, we can summarize the underlying differences between the requirements of an SPI initiative in a disciplined vs. an agile environment as shown in Table 2.4.

Firstly, the table shows that the ideologies of agile software development emphasize the need for process adaptation. In Agile, the concept of universal and repeatable processes is considered defective and it has been proposed that every situation calls for a specific agile methodology [Boehm and Turner, 2003]. Secondly, the improvement actions should be decided within ongoing projects and by the team. Such fundamental differences advocate the need to define new SPI methods for the agile context. Such methods would support organization in the process of systematic selection and deployment of new agile practices and for adapting them to suit the organizational context.

Table 2.4.: Disciplined vs. Agile SPI (adapted from [Salo and Abrahamsson, 2007])

	DSD and SPI	**ASD and SPI**
Process definition	Standardized, measurement-based and Repeatable	Context-specific and adaptable
Process focus	Predictability and high-assurance	Responsiveness and rapidity
Process control	Organizational level	Team level
Knowledge transfer	Document-based	Face-to-face communication
SPI focus	Improvement of organizational processes (future projects)	Continuous improvement of daily working practices (ongoing projects)

2.3 Research Scope

In the previous two sections, we presented a contrasted view of discipline and agility which we summarize in Table 2.5. Disciplined methods can be classified as predictive, process-focused, plan-driven and document-driven whereas agile methods can be classified as adaptive and people-oriented. It is worth mentioning that this comparison was purposely pushed to the extremes. In reality, several software development methods mix disciplined and agile characteristics. The distance between both approaches is actually a spectrum of various methods which balance rigor and flexibility at different rates [Boehm and Turner, 2003].

Table 2.5.: Discipline vs Agility (adapted from [Nerur et al., 2005])

	Disciplined Development	Agile Development
Fundamental assumptions	Systems are fully specifiable, predictable and can be built through meticulous and extensive planning	Systems are highly adaptable and can be built through iterative cycles by small teams using the principles of continuous improvement
Development model	Linear life-cycle model	Evolutionary-delivery model
Management style	Command and control	Collaborative leadership
Knowledge management	Explicit (document driven)	Tacit (people oriented)
Communication	Formal	Informal
Quality Management	Heavy planning and strict control. Late and heavy testing	Continuous control of requirements, design and solutions. Continuous testing
Process Management	Standardized and compliance-based	Light-weight and based on the team experience

We also presented in Section 2.2.3 the main agile methods that have seen extensive application in companies: Extreme Programming (XP) [Beck and Andres, 2004], Scrum [Schwaber, 1995], Lean Software [Poppendieck, 2007], DevOps and others. All these methods offer many tangible benefits over disciplined methods (e.g., improved time-to-market, productivity, software quality and time efficiency) which make them the actual appealing solution for better software development. However, organizations and teams currently face major obstacles when trying to apply agile methods out of the "sweet-spot", that is, the ideal conditions in which agile values are most likely to

succeed [Nerur et al., 2005; Marchenko and Abrahamsson, 2008; Boehm and Turner, 2005; Abrahamsson et al., 2009].

Typically, agile methods are found to be adapted for small, self-organizing and collocated teams of motivated developers with at least one co-located customer [Boehm and Turner, 2003]. Their implementation, although desirable, is not trivial in large-scale organizations with several business initiatives and multiple projects being executed in parallel [Reifer et al., 2003; Nerur et al., 2005]. Therefore, it has been suggested that organizations with such complex software development settings need guidance approaches for supporting them in the deployment of agile methods [Qumer, 2010; Sidky, 2007]. A number of practical scaling frameworks have been recently developed. The most prominent consist of the following: Scaled Agile Framework (SAFe) [Leffingwell, 2016], Large Scale Scrum (LeSS) [Larman and Vodde, 2016], Disciplined Agile Delivery (DAD) [Ambler and Lines, 2012] and the Spotify Scaling method [Kniberg and Ivarsson, 2012]. Each of these frameworks draws from the "first-generation" agile and lean methods (see Section 2.2.3) and extends them with a set of practices intended to help large scale organizations in resolving challenges associated with team siz e, customer involvement, project constraints, business case approval and partners and end-users relationships [Alqudah and Razali, 2016].

More generally, several circumstances other than the complex settings of large-scale organizations may influence the deployment of agile methods. Therefore, it has been suggested that agile methods need to be *"tailored"* or *"customized"* according to the specific context of their application [Cockburn, 2004a; Boehm and Turner, 2003]. *Agile methods customization* is an important topic that have been investigated in several research work [Qumer, 2010; Sidky, 2007; Alqudah and Razali, 2017; Ayed et al., 2014, 2017; Simonofski et al., 2018].

However, although this idea is commonly accepted, the way methods should be customized is still unclear. Practitioners and researchers reported valuable experience reports on agile methods customization [Fitzgerald et al., 2003; Layman et al., 2006; Cao et al., 2004] but these are hardly exploitable. Indeed, they are often based on the intuitive, intrinsic and non-quantified knowledge of agile experts and most are too narrowly linked to a specific situation. Hence, they cannot provide general and practical guidance that can be systematically reused by other practitioners.

Regarding the aforementioned discussion, the following issue is formulated:

> **Main Identified Issue:** What practical guidance can be provided to organizations and teams to systematically and impar-

tially support them in the implementation of agile methods to their specific contexts?

As previously mentioned, this research question is closely linked to the issue of *agile methods customization*. Moreover, three more issues are connected to the question: agile methods *adoption*, *assessment*, *improvement* (see Figure 2.20).
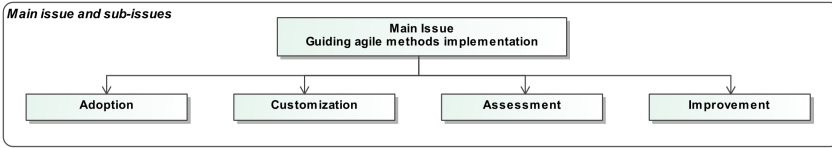
Figure 2.20.: Identified issues

In the reminder of this section, we provide details on each of these related issues. Customization is highlighted since it is the main research issue we target to investigate.

2.3.1 Customization

As previously mentioned, methods *customization* or *tailoring* refers to the adaptation of software development methods to accommodate the specific context. We report in the following some of the relevant topics of research related to this issue:

- **Customization approaches for Agile:** The customization of software methods is *"commonplace in the vast majority of software development projects and organizations"* [Fitzgerald et al., 2006]. Empirical studies such as [Fitzgerald, 2000] show that method use in software practice is rather limited. Only few developers rigorously adhere to method prescriptions. Rather, most tend to intuitively and spontaneously combine parts from different methods and discard all what they judge unnecessary.

  Several approaches such as *"method engineering"* (see Section 3.3) have been proposed for guiding and automating methods customization in disciplined process-supported environments (see Section 2.1.2.4 and Figure 2.1). However, although agile methods customization is recognized as important, few guidance specific to the agile context

has been proposed and therefore, more research is still to be done in this area. The suitability of disciplined approaches for agile methods customization should be questioned since too much discipline may lead to what [Fitzgerald et al., 2006] call *"goal displacement"* where team members *"lose sight of the fact that their primary goal is the product"*.

- **Customization factors:** Specific factors from the development context (e.g., system criticality, requirements variability, leadership style, team size, etc.) determine the suitability of a particular software development method and influence the way it should be customized. These characteristics are commonly known in literature as *"contingency factors"* or *"tailoring criteria"*. Research studies on this aspect of agile methods customization target to answer the following questions: What are the relevant criteria to be considered in the customization of agile methods? What are dependencies between different criteria and how should certain criteria influence the customization of agile methods? [Kalus and Kuhrmann, 2013a; Campanelli, 2016].

- **Customization knowledge:** Another topic of interest concerns the capture of the customization knowledge, that is, the implications do the customization factors have. Systematic approaches to customization usually store this knowledge in the form of guidelines or actions stored as a predefined catalog of customization knowledge [Kalus and Kuhrmann, 2013b; Patel et al., 2004]. Several questions should drive the constitution of such a catalog: What relevant information should be provided? How to structure this information? Who is responsible of creating and evolving the catalog? These questions also need to be considered in the design of a systematic approach for agile methods customization.

2.3.2 Adoption

Adopting agile software development methods is a wide and complex organizational change that usually impacts several aspects of the organization (e.g., its structure, culture, management practices, produced artifacts, technologies in use, etc.).

Software organizations face several issues related to agility adoption. These can be synthesized as follows:

- **Readiness to adopt agile methods:** Understanding the readiness of organizations and individuals (i.e, ability and readability to engage with the agile philosophy) is crucial to minimize the adoption risks and

avoid failure. An organization has to be aware of the blocking factors so that an enlightened decision on whether or not to proceed with agile adoption can be made. Moreover, it's crucial to carefully study the transformation strategies.

This question has been studied in several research work. In particular, generic and systematic models to assess the readiness of organizations to adopt agile methods have been proposed [Boehm and Turner, 2003; Sidky, 2007; Nerur et al., 2005]. Such models undeniably provide valuable insights regarding the risks of adopting agile methods in certain circumstances, but yet still neglected. Indeed, regarding the growing pressure to embrace agility, several organizations decide to move to agile software development without much consideration.

- **Effective and Efficient implementation of agile practices:** Another critical issue to be considered is the effectiveness (i.e., doing the right things) and the efficiency (i.e., doing things right) of the adopted agile methods.

Indeed, experience shows that agile practices are often superficially understood and misapplied. [Gregory et al., 2015; Simonofski et al., 2018] refers to this challenge as "Shallow Adoption" where methods are followed mechanically without deep knowledge. This may be explained by the following:

1. the non-prescriptive nature of agile methods makes them open to different interpretations,

2. the key information on agile practices is limited, unstructured and scattered which is highly confusing,

3. the information on agile experiences may be misleading since usually more success stories than failures are available, and

4. the terminologies regarding the key concepts of agile software development is often confusing: usually each method uses its own terminology, thus, two different terms may designate the same practice. (see Section 2.2.3).

2.3.3 Assessment

Once an agile process is implemented, its capability to satisfy the quality goals and eventually its maturity should be assessed. As discussed in Section 2.2.5, agile methods include few process assessment and improvement mechanisms

which are purposely made lightweight. Thus, the underlying challenges are often reported in the agile literature:

- **Quality assessment:** There seems to be a general feeling in the agile community that if you follow thoroughly all the practices associated with your chosen method then by definition you are delivering quality (see Section 2.2.4. Some practitioners take this hypothesis for granted without real investigation of the question. Several research studies investigated the question but the topic still controversial between those who state that agile practices not necessarily impacts the quality of systems and those who report undeniable benefits.

  Moreover, the relationship between the agile process quality and the resulted software product quality is not straightforward and often depends on the specific context of the project and organization. In fact, one agile method may lead to success when applied to a particular project and to failure under different circumstances.

- **Agility assessment:** Several research work propose to empirically assess the suitability of an existing or an about-to-be-adopted methodology using questionnaires or specific metrics. Many of the assessment approaches evaluate methods with regards to their respect to agile principles and values and identify accordingly an agile capability or maturity level (Similarly to the CMMI (see Section 2.1.5.1)).

  For example, [Qumer, 2010] (see Section 3.3.1.1) proposes an agility measurement index intended to be used by agile consultants to determine the degree of agility of individual practices (lower levels), methods, teams or organizations (upper levels). According to the obtained agility score, a capability level (from basic to advanced) is determined.

  Also, several self-assessment tools proposed by practitioners, typically agile consultants, can be found in literature [3].

2.3.4 Improvement

We argued in Section 2.2.5, that once adopted, agile methods require on-going sustain and improvement. However, methods for supporting teams and organizations in doing so are lacking. This issue is explained in the following:

---

[3]`https://www.benlinders.com/tools/agile-self-assessments/`

- **SPI models for Agile:** In Section 2.2.5, we discussed the applicability of the traditional SPI models in an agile environment and argued the need to define new SPI methods for the agile context. However, few research on this topic can been found in literature [Salo, 2006; Sidky, 2007; Ringstad et al., 2011]. Several questions still need to be investigated: How to efficiently improve teamwork in agile software development? How to integrate the agile SPI activities of individual project teams with traditional organizational SPI activities?

2.4 Summary

In this chapter, we provided a comparison between disciplined and agile software developement and argued the benefits of balancing both approaches and situationally implementing agility.

The terminology of discipline is adopted from [Boehm and Turner, 2003]. It refers to a systematic engineering approach to software that carefully defines repeatable and manageable processes. Discipline implies a high concern of completeness of the documentation and plans so that thorough verification of each development step can be accomplished after the fact. In the contrary, the agile approach promise to be more lightweight by valuing close collaboration, working software, changeability and customer satisfaction.

The chapter also formulates the main research problem that we investigate in this research, i.e., how to assist organizations and teams through the situational implementation of agile methods? Chapter 3 presents and classifies the existing guiding approaches that target this issue.

Chapter 3

Related Work and Research Questions

We discussed in Chapter 2 how agile software development meet some current expectations of modern software organizations. However, implementing agile methods is not straightforward. Organizations face major challenges during the adoption, tailoring, assessment and improvement of agile methods (see Section 2.3). Consequently, they ask for guidance and assistance on how to implement agility in their particular cases.

In this chapter, we review and classify the existing research initiatives for guiding the situational implementation of agile methods. The classification is done regarding the techniques used: maturity-based approaches, contingency factor approaches, method engineering approaches and experience-based approaches. Since no such categorization can be found in literature (only the contingency factor and method engineering approaches are reported), this chapter may serve as a starting point for researchers investigating the problematic of situational agile methods implementation.

The existing approaches are presented in Sections 3.1, 3.2, 3.3 and 3.4. Their strengths and weaknesses are discussed in the light of the agile fundamental principles and values shown in Table 2.1 and the agile SPI requirements shown in Table 2.4.

In Section 3.5, we provide a synthesis on the identified approaches, discuss the challenges that are still to be solved (which are not yet well investigated in literature) and discuss the opportunity of reusing and integrating techniques from disciplined process tailoring and improvement, namely SME and QIP.

Finally, regarding the identified research gaps, we formulate in Section 3.5 the thesis research questions.

3.1 Maturity-based Approaches

A maturity model presents *"an evolutionary progress in the demonstration of a specific ability or in the accomplishment of a target from an initial to a desired or normally occurring end stage"* [Leppänen, 2013]. There exist several maturity models for improving the organizational software development process in a prescriptive way (see Section 2.1.5.1). These are intended to institutionalize a collection of predefined practices and to control their consistent implementation so as to increase the profitability of the concerned organization or team and to ensure a certain level of maturity as compared to competitors.

As explained in Section 2.2.5, implementing maturity models in an agile environment is challenging since they may impede responsiveness, rapidity, team-focus and continuous improvement of the team's daily practices.
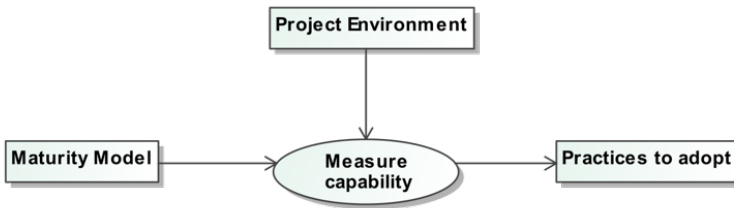
Figure 3.1.: Maturity-based approach

However, with the ever growing interest towards agile methods, several proposals for agile process maturity models came out in the last decade [Buglione, 2011]. Such models are intended to assess the level of adoption and maturity of agile practices/values inside an organization/team and to guide the setting up of agile practices in an organization/team following a structured and progressive approach (see Figure 3.1). These are discussed in the following section.

3.1.1 Existing Research

The agile maturity models are used to : (1) assess the current state of agility adoption (capability determination), (2) increase the adoption level, (3)

support the progressive implementation of practices, and (4) further the deployment of agile values and principles.

In the following sections, we report some of the most influential maturity-based approaches.

3.1.1.1 Structured Framework to Agile Adoption

The agile adoption framework [Sidky, 2007] (shown in Figure 3.2) is a structured and repeatable approach designed to guide and assist agile adoption efforts. Its main components consist in a 4-stages process of adoption steps and a measurement index (see Figure 3.2).
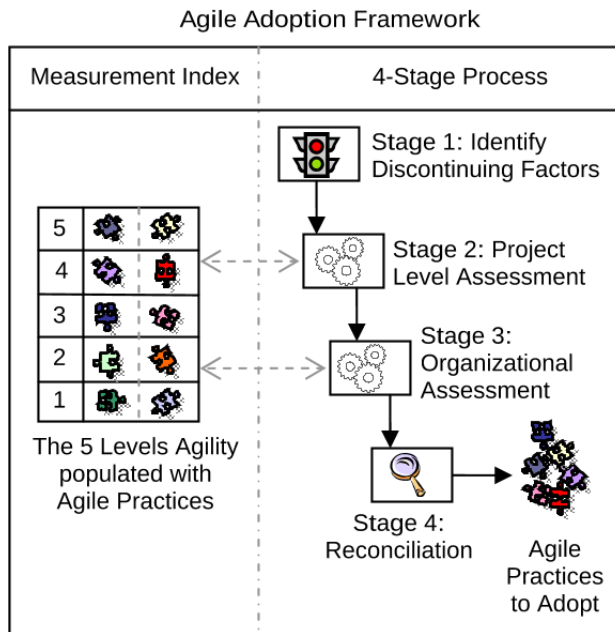
Figure 3.2.: The Agile adoption framework

The first stage of the adoption process aims at verifying whether organizations are ready to adopt agile software development. The second and third stages uses the Sidky Agile Measurement Index (SAMI) to assess the project's and organizational agility. The last stage suggests a final set of Agile practices for organizations to adopt by reconciling any differences between the targeted Agile Levels identified in Stage 2 and Stage 3.

The SAMI consists on a scale to be used by agile coaches to determine the agile potential of a project or organization (i.e., the degree to which that entity can adopt agile practices). The higher the agility level, the more practices can the organization adopt. The approach defines 5 levels of agility and each introduces an important quality or characteristic (e.g., collaboration) into the organization to help it become more agile. A set of agile practices lead to the realization of the respective level quality.

The Agile adoption framework provides structured steps to assist organizations increasing their agility level progressively. A fundamental issue with this approach is that the agility level is determined by the ability of an organization to adopt agile practices rather than its ability to implement the values and principles of the Agile manifesto. Several practitioners inform us about this problem and argue that an organization may be ready to select some agile practices or configure them without having a real agile mindset.

Another critical limitation of the framework is that it does not propose means to *"configure"* practices. Rather, for each level, it suggests a set of practices that might be suitable without any configuration. Additionally, it follows a predefined roadmap in suggesting agile practices preventing teams from controlling their own way of working.

3.1.1.2 Agile Maturity Model (AMM)

The Agile Maturity Model (AMM) [Patel and Ramachandran, 2009] is a SPI model for agile software development teams. It proposes a staged representation which shows how the agile processes should mature from an initial or ad-hoc level to a level showing continuous improvement based on the agile principles and practices. The model defines 5 maturity levels similar to those defined by the [CMMI, 2006]: initial, explored, defined, improved and sustained. Each maturity level is decomposed into a set of Key Process Areas (KPAs) that indicate the areas an agile team should focus on to improve. It also proposes assessment questionnaires in order to assess the team capability scores for each KPA and to recommend improvements based on best agile practices knowledge.

The model also suggests a simple questionnaire to evaluate the suitability of agile methods to the project context. This assessment brings three possible results based on the answers supplied by the team: (1) ready to adopt Agile, (2) ready to adopt Agile but with improvement of recommended areas, and (3) not ready for Agile (see figure 3.3).

The AMM has the advantage of being team-focused, clear and understandable. It is designed to enable new agile teams to estimate easily whether agile
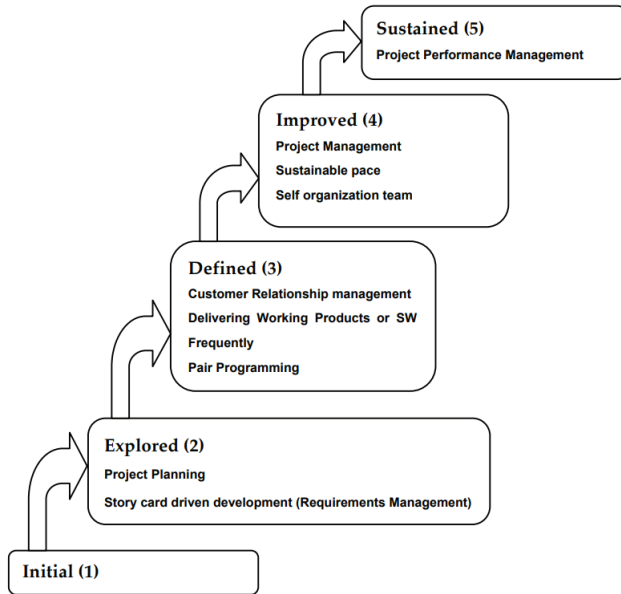
Figure 3.3.: The Agile Maturity Model [Patel and Ramachandran, 2009]

methods will fulfill their needs and to help them improve their practices based on a a simple staged maturity model. The main motivation for the staged representation of improvement models, is to allow for comparison between organizations or teams.

However, the fundamental limitation its inflexibility in prioritizing process improvements areas. In fact, the model proposes a predefined improvement path which may not be feasible in practice. Moreover, the team is not involved intimately in the overall improvement process which may compromise team self-organization and empowerment. Another limitation of the model resides in the recommendation of best practices without considering if they are feasible or not. The possible configuration of the recommended practices is also not considered.

3.1.1.3 Agility Adoption and Improvement Model (AAIM)

The Agility Adoption and Improvement Model (AAIM) [Qumer et al., 2007] is a generic agile maturity model proposed as part of the Agile Software Solution Framework (ASSF) (see Section 3.1). AAIM is to be used at an

organizational-level as a gradual road map for leveraging the implementation of agile values and principles.

The model proposes 6 agile maturity levels (from level 1 to level 6), organized into three blocks: agile-prompt, agile-crux and agile-apex. The agile-prompt introduces level 1: agile infancy. The agile-crux consists of the core of the AAIM levels, level 2: agile initial, level 3: agile realization and level 4: agile value. Finally, the agile-apex block presents level 5: agile smart and level 6: agile progress (see Figure 3.4).

At each block, the degree of agility of a software development process is measured quantitatively by using an agility measurement index known as the 4-DAT [A.Qumer and Henderson-Sellers, 2007] (see Section 3.1). Each stage specifies goals that must be achieved to attain a particular agility value. The relevant practices to be incorporated to the method can be derived from the current or targeted maturity level.
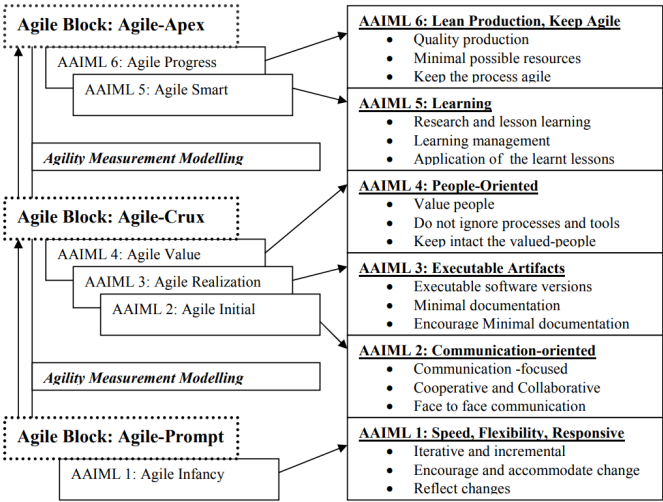
Figure 3.4.: The Agility Adoption and Improvement Model [Qumer et al., 2007]

The model results from an ethnography study of several industrial case studies on the adoption and assessment of agile methods and from the feedback of both researchers and practitioners. However, its relevance, specifically at the team level, still have to be demonstrated. In fact, the

model informs about a set of goals to achieve at each level without providing practical guidelines on how to do so.

3.1.2 Limitations

The maturity-approaches presented in Section 3.1.1 and others such as the ThoughWorks Agile Maturity Model [Humble and Russell, 2009] and the Scrum Maturity Model (SMM) [Yin et al., 2011], propose CMMI-staged like models for maturing agility in organizations (see Section 2.1.5.1). In other terms, they all provide organizations with a predefined and proved improvement paths and summarize agility implementation in a simple form: a single maturity-level rating.

Such models are found to be very disciplined and unadapted for agile environments [Qumer, 2010]. Indeed, as explained in Section 2.2.5, Agile SPI models should focus on the continuous improvement of daily work practices of the team. However, these maturity-based models rely on predefined introduction of practices, preventing teams from controlling their own way of working. We argue that a goal-oriented continuous model (inspired by the CMMI continuous representation) to be used by teams would be much more adapted for agile methods. However, to our knowledge, few and yet immature research initiatives such as [Packlick, 2007] proposed this kind of maturity models.

3.2 Contingency Factor Approaches

The contingency factor approaches, also known as situational approaches, is inspired by a management theory that determines the most appropriate organizational structure or design (e.g., the management style) based on environmental circumstances. The approach basically describes causal relationships between a collection of independent contingency variables characterizing the organizational specific situation and a set of response variables representing the organizational or managerial actions taken in response.

In other terms, it represents the 'if and then' approach to management where 'if' represents the independent variable characterizing a specific situation and 'then' represents the dependent organizational action to be taken in that situation. It assumes that the organizational effectiveness depends on the appropriate matching of contingency factors with organizational designs [Zeithaml et al., 1988].

Similarly, the contingency factor approach to software development methods suggests that specific factors from the development context (e.g., system criticality, requirements uncertainty level, leadership style, etc.) should be used to identify the most appropriate development method. Based on contingency factors, the most appropriate method can be selected from those available on the market, in the published literature or in a portfolio of organizational methods (see Figure 3.5).
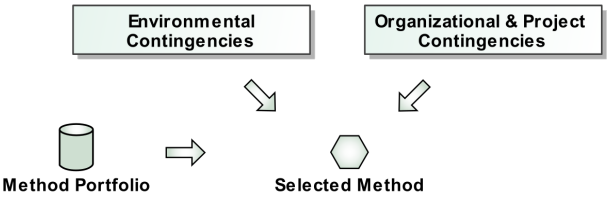
Figure 3.5.: Contingency factor approach

Several contingency factor approaches has been proposed to guide organizations through the selection of suitable agile methods and practices. These are discussed in the following section.

3.2.1 Existing Research

Several guidance approaches relying on the principle of contingency factors have been proposed. They can be classified in two sets according to their purpose of use: agile adoption (i.e., verifying the suitability of agile methods) and agile tailoring (i.e., adapting existing methods and practices to make them feasible under specific conditions). These two sets of existing studies are detailed in the following sections.

3.2.1.1 Contingency approaches for agile adoption

Research approaches aiming to assess the suitability of agile methods have concluded that there exist critical contingency factors influencing the decision to select agile or plan-driven methods in a particular project situation.

In an influential research, [Boehm and Turner, 2003] described five critical factors: the project's size, criticality, dynamism, personnel, and culture. For
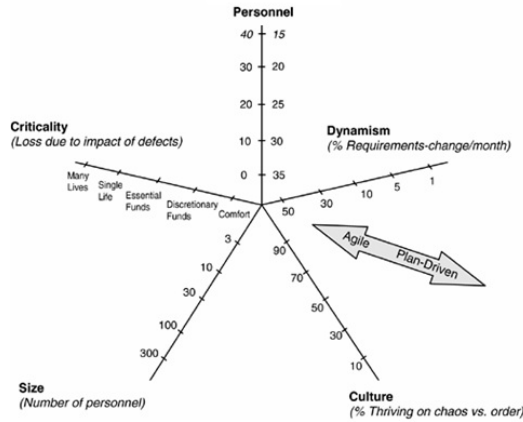
Figure 3.6.: [Boehm and Turner, 2003] approach

example, the approach suggests that agile development is well-matched for small products and teams where people feel empowered. By rating a project along each of the five factors and visualizing them in a radar chart as shown in Figure 3.6, one can determine the most suitable development approach: agile or plan-driven. If all the ratings are near to the center, an agile method is suggested and if they are near the periphery, a plan-driven approach would be more convenient. Furthermore, the approach warns about the risks of adopting agile or plan-driven in a particular situation and provide additional material on how such risks can be managed by balancing agility and discipline.

Similar research studies and industrial initiatives have identified more factors to assess the suitability of agile methods [Ahimbisibwe et al., 2015; Nerur et al., 2005; Khan and Beg, 2013; Griffiths, 2007].

There also exist few formal approaches that tackled the problem of assessing the suitability of agile methods based on contingency factors. For example, [Mikulėnas and Butleris, 2010] proposes an approach for evaluating the suitability of agile methods using an Analytic Hierarchy Process (AHP) [Saaty, 1988].

3.2.1.2 Contingency approaches for agile tailoring

There exist valuable studies that investigate the contingency factors influencing agile methods adaptation or tailoring [Kalus and Kuhrmann, 2013a;

Campanelli, 2016; Alqudah and Razali, 2017]. Usually, this studies rely on authors own expertise or on the review of experts' knowledge. The common research methodology consists in: (1) reviewing the previous work to systematically identify the crucial factors and (2) possibly conducting an additional survey and a multivariate factor analysis to keep only relevant factors.

The drawback of such studies is that they identify an isolated set of contingency factors without necessarily providing answers on how to exploit them in practice.

More practical contingency factor approaches provide precise guidance on which methods and practices are suitable for a specific context. For example, [Cockburn, 2004b] which introduces the Crystal agile family of methods suggests to instantiate the characteristics of these methods based on two contingency factors: system criticality and team size. The approach proposes a set of configured methods from which project managers can choose (see Figure 3.7).
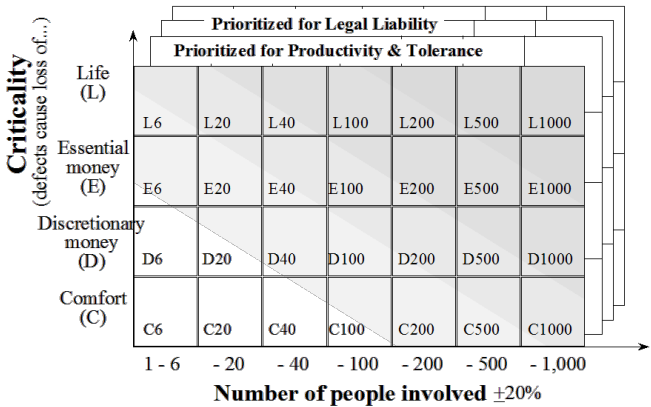
Figure 3.7.: Crystal Family of agile methods [Cockburn, 2004b]

Another practical approach is proposed by [Saleh, 2013]. The approach comes up with a model that supports the selection of agile practices suitable for developing a given software project. In order to do so, the model brings together contingency factors and data mining techniques.

Firstly, using the k-means method and the Euclidian distance measurement, the research suggests to form clusters of agile practices in terms of their support to certain abilities (e.g., support of team distribution, large iterations,

acceptance testing, etc.). The practices belonging to the same cluster are considered as comparable and practices from different clusters are considered as complementary. Secondly, the approach is supplemented by a cost function to provide the development team with feasible information about the selected practices: learning difficulty level, adoption difficulty level and tools to be used. Finally, the approach uses a rule-based decision system on top of the cost function to provide the team with recommendations. The inputs of this system are the project's contingency factors and the sets of practices. The output consists in a list of recommended agile practices (see Figure 3.8).

A similar approach is proposed in [El-Said et al., 2009]. It proposes a mathematical model to act as a tailoring tool relying on fuzzy quantification of experienced agile methods.

The idea of providing agile teams with practical guidance for agile process configuration using contingency factors has also been discussed in [Kruchten, 2013] which proposes eight key contextual factors. The study concludes that it would be interesting for future research to provide an agile team with a kind of recommending system: *"a tool to which we would provide values [of the contingency factors], that would give an indication of which practices are usable, which are not, which would require adaptation or special consideration, as the starting point for a process configuration"*. This idea was considered as a particularly interesting lead for solution for our research work (see Section 3.5).

3.2.2 Limitations

According to [Zeithaml et al., 1988], the main challenges of the contingency factor research consist in:

- identifying the critical contingency variables that distinguish between development contexts,

- grouping similar contexts based on these contingency variables, and

- determining the most effective development method in each major group.

Although the contingency factor approaches presented earlier met these challenges, they are still criticized because of their deductive nature [Fitzgerald et al., 2006]. Indeed, the contingency factor approach preconfigures the customization decisions and limits the set of methods. The team members have no control on the customization process, i.e., they only should be able to understand and execute the recommended method or practices.
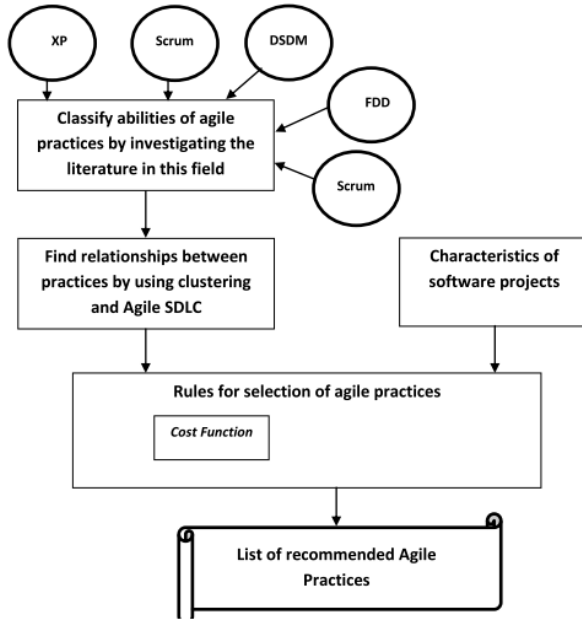
Figure 3.8.: [Saleh, 2013] approach

Another fundamental issue with the contingency factor approach in practice is that *"an organization is expected to have a range of methods available to developers who are expected to be familiar with each and merely choose the most appropriate one depending on the contingencies of the situation"* [Fitzgerald et al., 2003]. In practice, such an expectation is not realistic since close familiarity with even one method is not at all common among developers.

3.3 Method Engineering Approaches

The need for systematic and standardized techniques to select or construct situation-specific methods has led to the emergence of the so-called Method Engineering (ME) approaches. ME can be defined as *"the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems"* [Brinkkemper, 1996].

A core theme of this discipline is Situational Method Engineering (SME), which aims at designing development methods tuned to the situation of the
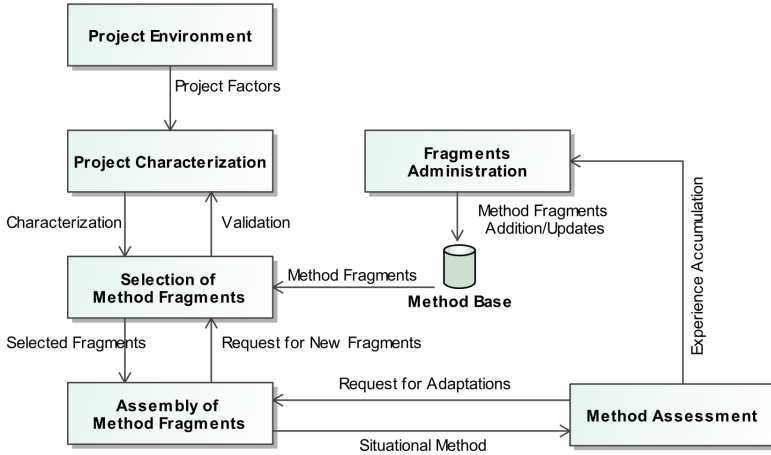
Figure 3.9.: Method Engineering Approach [Brinkkemper, 1996]

project and the organization at hand. Designing a situation-specific method is time-consuming and costly. Hence, it is not efficient to build situation specific methods from scratch. Rather, a project-specific method is created by selecting, tailoring and assembling appropriate method building blocks. These building blocks are called fragments or chunks and are stored and retrievable from a so-called method base [Harmsen et al., 1994]. Usually, standardized metamodeling constitutes the preferred specification formalism for method fragments. The rationale of using metamodels is to ensure an objective and unbiased description of method fragments and to allow methods comparison in a sound and scientifically verifiable way [Brinkkemper, 1996].

The building blocks are assembled into a situational method using the configuration process depicted in Figure 3.9. The starting point of this process is the evolving project environment which can be characterized using a set of contingency factors, such as the project constraints, the technical expertise of the development team, the managerial model and the organizational structure. These factors are input of the process of selecting the appropriate method fragments from the method base. The set of selected fragments are then assembled into a consistent method regarding a set of rules and constraints. This step allows to remove inconsistencies regarding

fragments granularity, arrangement (e.g., how fragments are sequenced) and dependencies.

ME approaches have been used for constructing appropriate agile methods. We explore the existing research of tailoring agile methods using method engineering techniques in the following section.

3.3.1 Existing Research

In a systematic literature review, [Campanelli, 2016] state that method engineering is a primary choice regarding agile methods tailoring. This may be explained by the fact that method engineering is an established research area. Specifically, ME techniques are used in disciplined software development environments to help organizations tailor down their standard methods to make them project-specific (see Section 2.1.2.4).

The applicability of such techniques to adapt agile methods and/or to enable situation-specific selection of agile practices has been questioned in a number of research studies. However, we were able to identify only few mature approaches which consider the distinguishing features of agile methods. These are discussed in the following sections.

3.3.1.1 Agile Software Solution Framework (ASSF)

The ASSF is designed to assist organizations in assessing the degree of agility they require and in tailoring and improving their agile or hybrid methods [Qumer, 2010]. To do so, the framework relies describes a theoretical toolkit that implements the paradigm of assembly-based method engineering. More precisely, it describes the following components (see Figure 3.10):

- Knowledge Base: the component that provides means for the construction of method fragments, for example using the standard [ISO/IEC 24744, 2007] metamodel (see Section 6.2.2). Basically, the fragments contain basic knowledge of software methods: people, tools, process and product features. They also encompass additional features such as agility degree, abstraction level, business value, business policy, business rules and legal aspects. For each agile project, the agile team, with the help of an agile coach, may select appropriate agile fragments from the knowledge base.

- Process Composer: the tool that allows to compose the fragments into a consistent method

- Agility Calculator (also known as 4-DAT analysis tool) [A.Qumer and Henderson-Sellers, 2007]: the component is designed to measure the degree of agility of either a process fragment or the whole composed process. It examines software methods from four dimensions: method scope, agility characterization (with 5 key attributes: flexibility, speed, leanness, learning and responsiveness), characterization of agile values (based on those proposed in the agile manifesto [Beck et al., 2001]) and software process characterization. The agility degree of practices and processes is evaluated quantitatively and qualitatively. In both cases, this characterization relies on experts judgment, i.e., "the decision-makers need to include their own, often subjective, weightings to any evaluation of the most appropriate agile method [or practice]" [Qumer and Henderson-Sellers, 2008]

- Knowledge Transformer: the component that transforms the knowledge about method fragments to a usable format so they can be published or visualized.
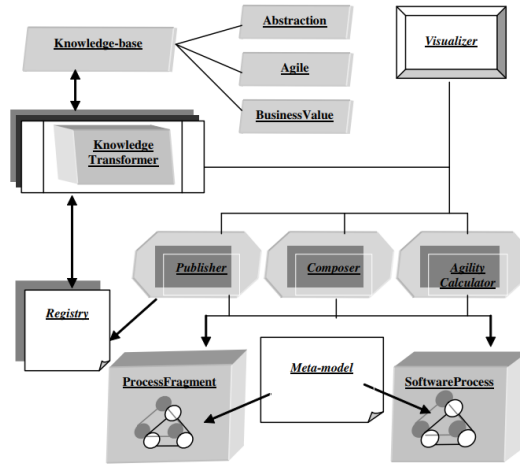
Figure 3.10.: The main components of the ASSF [Qumer and Henderson-Sellers, 2008]

The ASSF also include maturity-based reasoning (see Section 3.1). In fact, it suggests to select the relevant agile components based on an Agility Adoption and Improvement Model (AAIM) [Qumer et al., 2007]. This means that the relevant agile parts and practices are incorporated to the method to be

followed based on the current or targeted maturity level of an organization or team.

The ASSF is highly prescriptive, i.e., it describes a high-level systematic approach to assist organizations, and managers through the effort of agility adoption. For example, it does not provide a definite formalism to structure the knowledge base. Rather it suggests to reuse one of the existing metamodels and provides generic guidelines about the elements that should be added to document relevant agile fragments. The suitability or applicability of such metamodels in an agile environment is not really questioned.

The definition of agility in this framework is also questionable, specifically when it is used to measure the agility of individual practices. In fact, it's arguable that a low or high degree of agility of a unique practice is not necessarily informative since the agility of an agile method or process is achieved by the synergy and harmonization of several practices [Fitzgerald et al., 2006].

For example, the agility degree of the XP practice "sustainable pace" (previously known as "40h-week") scores badly (one-degree of agility out of the 5 identified by the framework) [Qumer and Henderson-Sellers, 2006]. However, the practice is heavily stressed by XP advocates because several studies show that during overtime, a drop in cognitive abilities may result in increasing mistakes and thus in accumulating technical debt. On the contrary, a sustainable pace may result in better quality and better customer satisfaction [Mann and Maurer, 2005]. Regarding the 4-DAT evaluation of sustainable pace, it may seem not very interesting to an agile team to comply to that practice notwithstanding that it has such impacts on quality.

3.3.1.2 Partial Agile Methods Adaptation Framework (PAMAk)

Few process metamodels has been described in the literature to support agile methods adaptation. Actually, most of the existing approaches that rely on method engineering do not specifically define a metamodel suitable for agile methods. rather reuse the existing process metamodels as proposed by the ASSF (see Section 3.3.1.1).

The PAMAk framework describes a formal approach for partial agile methods adaptation [Mikulėnas et al., 2011]. It defines a specific metamodel that reuses some features from existing process metamodels and integrates concepts which are directly or indirectly related to agile suitability assessment or adaptation. Using this metamodel, it is possible to decompose agile methods into elements or fragments. By composing, merging or generalizing these fragments, one can define new composite patterns. These new patterns and

the basic fragments can then be selected and coupled to construct a suitable agile method to be implemented in a specific context (see Figure 3.11). The elements of the metamodel can be of different abstraction levels and be related to agility requirements or specific application areas.
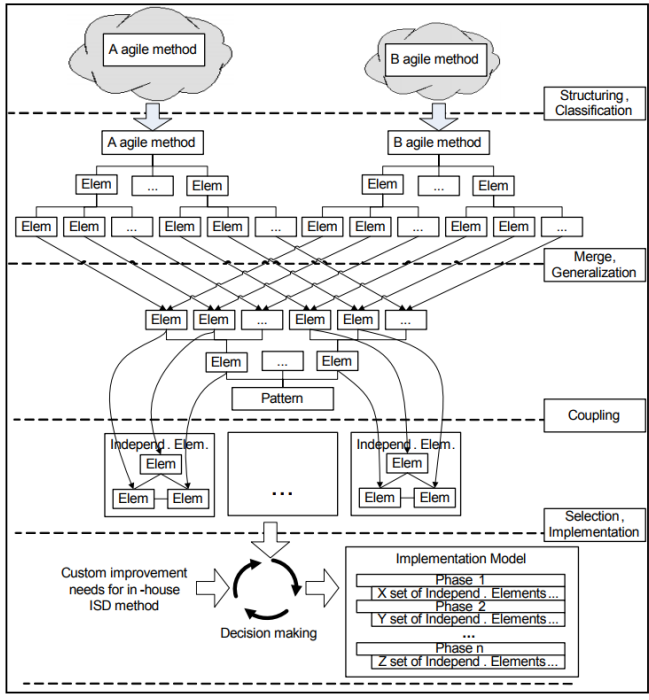
Figure 3.11.: Overview of the PAMAk Framework

The added value of the proposed metamodel resides in such concepts as agility requirements, levels of element adaptation, application areas and prioritization criteria. These are to be used to facilitate decision making when building an agile implementation model.

From the end user's point of view, the framework facilitates the process of element selection. For example, when the project manager identifies the need to improve application areas such as "code quality", the framework would suggest corresponding elements such as "Test Driven Development". However, the definition of this decision mechanism is unclear. It is not enabled explicitly using defined selection rules, it is rather assumed by the framework user regarding the existing relations between the elements.

Furthermore, the framework does not enable mechanisms for configuring practices which are presumed as unsuitable.

3.3.2 Limitations

Contingency factor and ME approaches are somehow comparable. In fact, contingency factors are essential in both approaches. In the first case, the contingency factors are used to select a preconfigured method from an available library. In the second case, a new method is engineered from the ground up using existing method fragments [Brinkkemper, 1996].

However, while agile ME approaches resolve the problem of methods pre-configuration, they still not grant the team members with enough process control. Indeed, in most of the agile ME approaches, it is left to the judgment of a method engineer to ascertain whether the agile method fragment is appropriate or not, perhaps with the help of a decision-making tool [Qumer and Henderson-Sellers, 2006]. When the tailoring of agile methods is done this way, developers lose control on their own way of working and learn little about each tailoring effort as they progress from one project to the next. This leads to a decrease of process lightness and to a lack of capitalization on improvement initiatives conducted at the project level.

This challenge has been discussed in [Conboy and Fitzgerald, 2010] which suggest that the involvement of all developers in the decision of implementing and tailoring an agile method would be beneficial. In particular, it would help to manage the potential negative consequences of introducing non-desirable practices (e.g., resistance to change, team frustration, etc.).

However, in order to be able to involve the team in the process of engineering a suitable method, it is critical to ensure the familiarity of developers with the selected method and the alternative agile methods and fragments that could eventually be used. It is also essential to ensure that the developers have sufficient knowledge about informed methods tailoring, i.e., that they are able to pursue an unbiased and informed decision process.

Another drawback of such approaches is that it is usually unclear whether they rely on a top-down or bottom-up assembly process (see Section 6.1.3). The two approaches describes different strategies on how to tackle the construction of a suitable method. When a top-down approach is used, a method outline or skeleton supplemented with some tailoring guidelines is defined. Then a number of fragments is chosen and added to the situational method. The choice of the method outline and the guidelines limits the number of possible fragments to be inserted. When a bottom-up assembly

approach is used, a situational agile method is built block-by-block. The team would start form small to big. The final method is obtained after a number of increments.

Additionally, most of the proposed agile method engineering approaches suggest to meticulously document an appropriate agile method model without questioning the utility of having such a model for the development team. They also rely on the traditional method engineering languages and metamodels, without questioning their adaptability in an agile environment. Traditional method engineering metamodels are very prescriptive. They provide static and operational semantics defined in terms of the abstract syntax.

3.4 Experience-based Approaches

The overwhelming majority of empirical studies on agile methods deployment are single or multiple-case studies which consist in in-depth and detailed examination of a specific situation over a period of time [Fitzgerald et al., 2003; Cao et al., 2004; Layman et al., 2004]. These studies have the advantage of being inspiring since they directly report the practitioners knowledge and experience.

However, [Dybå and Dingsøyr, 2008] argue that a major challenge consists in increasing their quality in terms of:

- *rigor*: do the studies rely on a thorough and appropriate research methods?,

- *credibility*: are the findings well-presented and meaningful?, and

- *relevance*: how useful are the findings to the software industry?.

Specifically, the studies are found to lack of *credibility* since the customization decisions are reported in a narrative way and are based on opinions and intuitive reasonings. Also, the strength of *evidence* is often found to be very low since findings are presented as very specific to a particular situation at a particular time. Consequently, it makes it difficult to offer specific advice to industry [Dybå and Dingsøyr, 2008].

Recognition of these limitations led some researchers to consider more structured and systematic experience-based approaches to agile customization (see Figure 3.12).
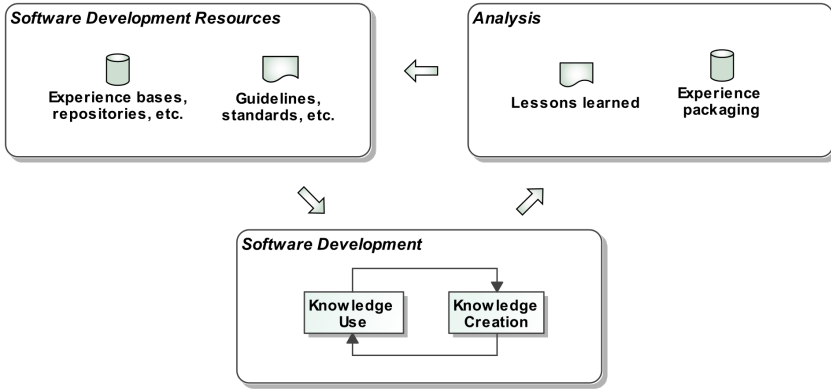
Figure 3.12.: Experience-Based Approach

Experience-based approaches to software methods improvement rely on two core principles:

1. organizational learning: the cyclic and systematic steps to continuously improve organizational processes (see Section 2.1.5.2), and

2. experience reuse: the collection and reuse of lessons learned from past projects. This requires a systematic and empirical methodology to collect and structure learnings (for example as causal relationships between context factors and process outcomes [Henninger, 2003]).

The most concrete example of an experience-based approach is the Basili Experience Factory [Basili et al., 1994b; Basili and Caldiera, 1995]. The basic methodological device of the approach is the QIP paradigm presented in Section 2.1.5.2. It relies on the notion that *"improving the software process and product requires the continual accumulation of evaluated experiences (learning) in a form that can be effectively understood and modified (experience models) into a repository of integrated experience models (experience base) that can be accessed and modified to meet the needs of the current project (reuse)"* [Basili et al., 1994b].

Very few examples of agile experience-based approaches can be found in the literature. These are presented in the next Section.

3.4.1 Existing Research

Only few approaches compatible with the idea of experience-based tailoring are available [Salo and Abrahamsson, 2007; Chau and Maurer, 2004; Amescua et al., 2010; Krasteva et al., 2010] and almost none that rely on a unified and complete experience and knowledge-driven process exists.

[Salo and Abrahamsson, 2007] proposed a relevant framework that relies on the Quality Improvement Paradigm (QIP) (see Section 2.1.5.2). The framework adapts the QIP life-cycle to support the systematic selection, tailoring and deployment of new agile practices.

The approach argues that a constant collaboration between the project teams and organizational level is a key success of agile implementation. Its major contribution consists in an iterative improvement process called Post-Iteration Workshop (PIW) which:

- provides project teams with a mechanism to tailor the deployed and the existing software development practices during the ongoing project

- provides the organizational level with mechanisms for gaining systematic and rapid feedback from the process improvement of projects.

The PIW process consists of six steps to be executed by the team after each development iteration: (a) preparation, (b) experience collection, (c) planning of improvement actions, (d) piloting, (e) follow-up and validation, and (f) storing (see Figure 3.13). Suitable practices are identified by implementing the improvement actions in the ongoing project and iteratively evaluating their usefulness with available metrics and experience data.
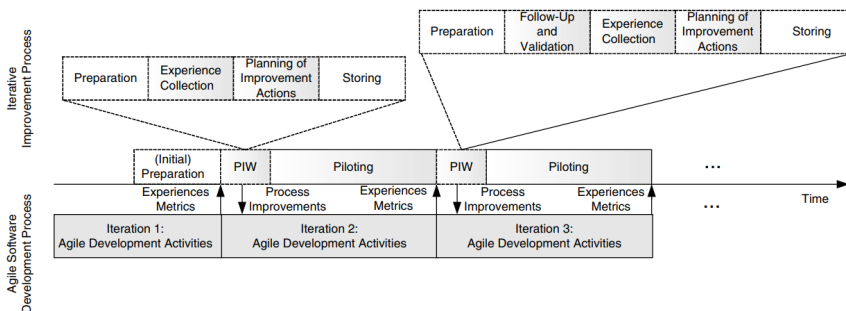
Figure 3.13.: Steps and information flows of the PIW [Pikkarainen et al., 2005]

Another approach worth citing is reported in [Amescua et al., 2010]. The approach is aimed at process learning and provides a knowledge-based process asset libraries (PALs) (a kind of an experience factory) to store agile best practices. However, although the approach details how the process knowledge should be stored, utilized and reused in context (using tailoring guidelines), it doesn't provide teams with means to grow this knowledge by documenting their newly experienced practices nor provide means to propose configurations to the process assets.

3.4.2 Limitations

Although the interest of the [Salo and Abrahamsson, 2007] framework has been validated in some industrial cases, the authors specify that more specific procedures are needed to support the systematic selection and deployment of new agile practices as well as for tailoring them to suit organizational needs [Salo and Abrahamsson, 2007].

Indeed, the framework theoretically describes the steps to follow for an experience-based improvement of agile practices but, apart from the PIW tool, it doesn't prescribe means on how to implement such a process inside an organization. Precisely, the way that the improvement and tailoring knowledge is to be stored is unclear, i.e., the structure of the experience factory is not detailed.

Moreover, the way that the development context impacts agile methods implementation is not elicited which makes it difficult to exploit the generated knowledge to guide other teams outside of the organization.

[Amescua et al., 2010] also do not focus on the customization knowledge. Indeed, the research details the structure and functionality that a knowledge repository should have and only overlooks the concept of tailoring guidelines. It does not clearly specify how methods are contextualized and how suggestions can be automatically provided to project teams. Additionally, the approach doesn't allow teams to document and integrate their improvements and newly introduced practices which contradict the agile process improvement mindset (see Section 2.1.5).

3.5 Discussion and Research Questions

The review of the approaches for supporting situational agile methods deployment, allows us to structure the existing knowledge and to identify

several gaps that still need to be addressed. These are synthesized in Table 3.1.

Table 3.1.: Review of situational agile implementation approaches

	Characteristics
Maturity-based	(–) Disciplined, prescriptive
	(–) Teams have no control on customization
	(–) No configuration of practices
Contingency Factor	(+) Clear elicitation of context factors and customization guidelines
	(–) Pre-configuration of customization guidelines
	(–) Teams have no control on customization
	(–) Limited set of custom methods to recommend
Method Engineering	(+) Richer description of custom methods (using composition of method components)
	(–) Teams have no control on customization
	(–) Ambiguity regarding components assembly
	(–) Unadapted method description languages
Experience-based	(+) Improvements and customization driven by the team
	(–) Few and immature approaches
	(–) Unclear elicitation of context factors and customization guidelines

Maturity-based approaches were found to be useful for organizations seeking a straightforward roadmap to transition to agility or attempting to improve their agile maturity at an organizational level. However, unlike the contingency factor and method engineering approaches, maturity models under-consider the customization aspects.

One marked feature of both contingency factor and method engineering research is that they help to structure the customization decisions. However, *"they have been largely deductive in nature, employing theoretical and conceptual arguments to suggest how methods should be customized"* [Fitzgerald et al., 2006]. Little is known regarding their practical application in real software development practice.

Generally speaking, in the software field, practice is often ahead of research, and it has been argued that much can be learned from examining good practice of past experiences. Therefore, we examined if there exist experience-based approaches that report and capitalize on the practitioners knowledge but were able to find only few and yet immature approaches.

Given the above, we considered investigating the use and customization of agile methods in actual practice and in providing an integrated approach for practitioners. Specifically, we argue that providing a better customization method implies bridging the gap between structured and knowledge-based approaches.

For this purpose, 4 research questions were progressively formulated and refined over the course of this research, following an exploratory research cycle (see Chapter 4). For more convenience, we summarize them in the following sections.

3.5.1 SME Perspective

We argued in Section 3.3.2 that Situational Method Engineering (SME) is a disciplined and structured approach that have the advantage to help structuring the key concepts of agile methods and their customization knowledge. However, applying it in an agile context raises several issues such as the impediment of team process control.

This highlights the need for a specific approach for engineering situation-specific agile methods and practices. Specifically, we argue that such an approach should provide a better method engineering formalism (e.g., simplified syntax, efficient graphical notation, etc.) and easier ways to exploit the customized method at the team level.

For guiding the construction of this specific SME approach, we propose the following research questions:

RQ1.a: How to use a situational method engineering approach for the customization of agile methods?

RQ1.b: Which methodological elements are relevant for creating situational specific methods and for structuring agile methods knowledge?

These two questions are mainly investigated in Chapters 6, 7 and 8.

3.5.2 Context Study Perspective

In order to be able to support agile practitioners with more objective guidance, we argue that it would be relevant to understand and characterize the context thoroughly. Indeed, a thorough understanding of the context leads to accurate, precise and reusable customization decisions.

The agile contingency factor approaches that we investigated in section 3.2 best elicit the context thanks to the concept of "contingency factors". Nonetheless, we argued in Section 3.2.2 that they usually rely on a subset of predefined factors and customization decisions and that few is known about their practical application. Therefore, in this research, we target the study of different contexts in practice and propose a customization approach with an extensible representation of development contexts.

> **RQ2:** What specific contexts influence the implementation of agile methods and how these can be characterized ?

This question is mainly investigated in Chapters 9, 10 and 11.

3.5.3 Customization and Capitalization Perspective

Another important gap that we aim to investigate is the characterization of the customization knowledge. The contingency factor approaches rely on a predefined catalog of customization guidelines and actions which are not extensible and often not precise enough.

Moreover, they are often considered as too theoretical, i.e., not relying on the agile team expertise and learnings. We argue that experience-based approaches provide a promising solution to handle this issue provided a fine-grained characterization of the teams' customization decisions.

The following question drive our proposal for a relevant customization catalog:

> **RQ3:** How to define reusable and fine-grained customization guidelines ?

Another problem with situational agile adoption approaches consists in their inflexibility. Once a practice or method is suggested and selected, it often stays unchangeable. These do not provide any mechanism for evolving the practice and allowing the team to communicate their adaptations. In contrast, we discussed earlier that an agile practice enacted by a team should

be continuously adapted to better fit their specific. This knowledge is highly valuable and constitutes a yet neglected aspect in situational approaches.

The following questions drive our proposal of a tool to support a project-team to systematically capture their improvement and capitalize their learnings at an organizational level.

> **RQ4:** How can an agile team continuously improve a customized method and capitalize its context-related learnings?

These questions are mainly investigates in Chapters 12 and 13.

Part II.

AMQuICk Framework

In this part of the dissertation, we detail the research methodology that was used to design our approach called AMQuICk Framework and provides an overview of it.

More precisely, Chapter 4 presents a review of the potential research methodologies that could have been used to conduct our research and presents the final constructive methodology that we applied and which based on the Design Science Research methodology.

Chapter 5 provides an overview on the approach including its theoretical foundations, life-cycle and main artifacts.

Chapter 4

Framework Design

The main research questions and motivation for the development of a new integrated framework for guiding agile methods customization have been proposed in the previous two chapters. To guide our research for necessary answers and to ensure rigor and reliability of results, a coherent research method should be decided.

This chapter presents a review of the potential research methodological choices that could have been used to conduct our research (see Section 4.1) and the final constructive research methodology that we applied to progressively design the AMQuICk framework (see Section 4.2). The latter section also presents the research studies which we conducted with the aim of assessing the validity of the framework.

4.1 Review of Research Methodologies

The selection of a research methodology and its research instruments depends on a set of aspects such as the goal of the research, the nature of the research topic and the availability of data sources [Benbasat, 1984]. Regarding the goal of this research (i.e., the development of a new framework) and its exploratory nature (in the way it explores the solution space), a qualitative and constructive research approach seem to be adequate.

This section provides a brief review of different research methodologies that could have been considered and presents our motivation for choosing a design science that includes case study and grounded theory research as part of its rigor cycle.

Survey Research

Survey research is a retrospective study of a situation that allows to gather information from a large group of people, referred to as a population, with

the purpose of establishing theoretical relevance. It usually requires the formulation of a clear research question, a careful sampling of a representative subset of the population and rigorous statistical analysis. Questionnaires are the primary data collection source. Surveys are especially well-suited for capturing what is happening broadly over large groups of projects when a particular method, tool, or technique is employed [Sjoberg et al., 2007].

This research was driven by a motivation to explore agile methods customization in practice and to progressively construct a practical solution to be recommend to practitioners. At the time this research project was initiated, no clear question was formulated (see definition 2.3) and the framework was not yet designed. Therefore, survey research was not a suitable option for conducting this research.

Case Studies

Case studies are one of the most common qualitative research methods in information system and software engineering research. They are usually used to test, generate or describe a theory or to assist in the understanding of a complex phenomenon [Benbasat, 1984; Dresch et al., 2015]. Specifically, they allow SE researchers to study and evaluate the implementation of methods and tools in an industrial setting.

Case study research enables to empirically study a contemporary phenomenon within its natural setting, precisely *"when the boundaries between phenomenon and context are not clearly evident"* [Yin, 2017]. Case studies can be either single or multiple. They are well well-suited when 'how' or 'why' questions are being asked and when the investigator has little or no control over events [Yin, 2017].

According to [Robson and McCartan, 2016], case studies can also be categorized as:

- Exploratory: finding out what is happening, seeking new insights, generating ideas and hypotheses for new research or initiating a change,

- Descriptive: portraying a situation or a phenomenon,

- Explanatory: seeking an explanation of a situation or a problem, mostly but not necessary in the form of a causal relationship.

These different ways to conduct case study research makes it possible to apply it for many situations and combine it with other methods. In this research, case studies can be useful in exploring different agile contexts (i.e., to answer RQ2 and RQ3) and in testing the proposed framework (see Section 4.2.2).

Action research

Action research designates a spectrum of constructive case studies where *"theory emerges in the process of changing"* [Cunningham, 1997]. Indeed, this research methodology aims at resolving a collective problem by intervening, observing the process of change and exploiting learnings. It usually targets the challenges of a specific organization and involves participants and representatives of the situation researched in a cooperative and participatory way [Thiollent, 2011; Dresch et al., 2015].

A proper conduction of action research follows a cyclical process where actions are planned and implemented to address a certain aspect of the research problem at each iteration. For example, [Susman and Evered, 1978] describes an iterative process of five main stages: (1) diagnosing, (2) action planning, (3) action taking, (4) evaluating, and (5) specifying learning (see Figure 4.1).
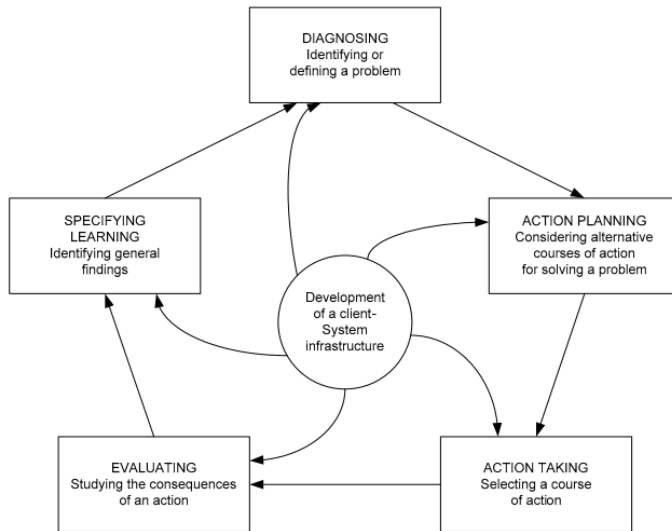
Figure 4.1.: Action Research Cycle proposed by [Susman and Evered, 1978]

Following such cycles, action research produces practical solutions to real problems with opportunities for a theory to emerge.

Despite of the constructive and practical nature of action research, it was not considered as a primary methodology for this research because of two

major drawbacks. First, the nature of action research implies that results are situation-specific and second, it requires a specific research setting including a longterm industrial collaboration where the researcher have an active role. However, the methodology can be useful in refining or evaluating a designed artifact.

Grounded Theory

Similarly to Action Research, grounded theory is also a constructive research approach but with more focus on theory generation. Key features of grounded theory are the iterative and systematic collection, coding and validation of data that may help to describe a phenomenon of interest [Abdel-Fattah, 2015]. Data are captured through the observation of people, their interactions, their the main concerns and how they go about resolving them.

Grounded theory was not retained as the primarily research methodology since it does not target to design solutions to practical problems. However, similarly to case studies, it can be used as a secondary methodology to explore the problem of agile customization in a specific situation (see Section 4.2.2).

Design Science Research

Design science research is a relatively new methodology in information systems research that emerged to reduce the gap between theory and practice [Dresch et al., 2015; Wieringa, 2010; Hevner and Chatterjee, 2010]. It advocates the concept of *"exploration through design"*, a prescriptive research style that goes beyond descriptions, explanations and predictions. Indeed, design science research seeks to *"(i) explore new solution alternatives to solve problems, (ii) to explain this explorative process, and (iii) to improve the problem-solving process"* [Holmström et al., 2009]. The solution obtained from the conduction of design science research is not necessarily optimal. The target is a workable and satisfactory solution to the investigated problem that must be capable of generalization to a certain class of problems (midrange or substantive theoretical relevance) [Dresch et al., 2015; Holmström et al., 2009].

Design Science research was selected as the method for conducting this work because it focuses both on building practical solutions and adding value to existing theoretical knowledge. Indeed, on the one hand, the methodology allows us to adopt a reality-observer perspective with the purpose of exploring, describing and explaining contextual implementations of agile methods. On the other hand, based on this newly generated theoretical knowledge, we can actually intervene by proposing an adequate solution. Another reason

that motivates this choice is that the methodology is highly relevant in information systems research since it highlights the role of artifacts and deals with the lack of professional relevance.

A detailed description of how Design Science Research was applied in this research is provided in the following section.

4.2 Design Science Methodology for Building AMQuICk

[Hevner, 2007] describes design science research as an embodiment of three closely related cycles of activities: relevance cycle, rigor cycle and a central design cycle (see Figure 4.2).
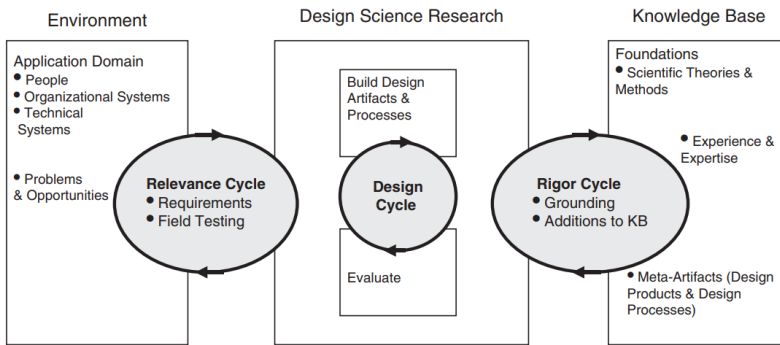
Figure 4.2.: Design Science Research Cycle as defined by [Hevner, 2007]

The Relevance Cycle initiates the design science research by eliciting the requirements for the research, the opportunity/problem to be addressed and the acceptance criteria for the solution to be designed. This is done through an exploration of the domain or research environment. The output of the design science research activities must be confronted with this environment for study or evaluation in the application domain. This is known as field testing and can be conducted using an adequate technology transfer methodology such as action research (see Section 4.1). The result of field testing will determine whether the new artifact is of satisfactory quality and whether additional iterations of the relevance cycle are necessary to refine the research requirements.

The Rigor Cycle connects past knowledge to the design science activities of the research project. It retrieves from the scientific knowledge base any appropriate research construct (e.g., theory, method, experiences and existing artifacts) that would help design or evaluate the solution artifacts. It is important to thoroughly research and reference such knowledge in order to ensure that the designed artifacts are research contributions. The research knowledge base can be extended with any relevant output of the research design activities: documentation of new artifacts, report of the field testing of artifacts, report of actual experiences in implementing the generated artifacts or in performing the research.

The Design Cycle is central to any design science research. It iterates between the core activities of building and evaluating the solution artifacts [Hevner, 2007]. According to [Simon, 1996], this process consists in generating design alternatives against the research requirements until having a satisfactory solution. Usually, multiple design iterations are necessary until achieving relevant artifacts that can enrich the research knowledge base.

Following this representation, we provide an overview of the design science methodology applied to conduct this research in Figure 4.3. Furthermore, we explain in Table 4.1 how the key aspects of design science research were addressed using the checklist introduced by [Hevner and Chatterjee, 2010]. The next section details the 3 iterations that were necessary to design the AMQuICk framework.

4.2.1 Design Iterations

Iteration 1: Situational Method Engineering Perspective

This first iteration aimed at exploring the Situational Method Engineering (SME) perspective (see Section 3.5.1) and at designing a first version of the AMQuICk metamodel for agile methods customization. To achieve this goal, we identified as part of the *Relevance Cycle* the requirements for an agile SME approach (see Chapter 6) which we later refined against the evaluation of the metamodel. Then as part of the *Rigor Cycle*, we reviewed the existing SME languages and confronted them to the requirements identified earlier. This allowed us to confirm the need for a specific SME language for agile and to justify the use of "Essence DSL". Finally, the metamodel was instantiated to create the AMQuICk repository of practices (see Chapter 8).

Table 4.1.: Design science research checklist

What are the design requirements?	The research aims at designing a practical agile guidance approach that can be provided to organizations and teams to systematically and impartially support them in the implementation and customization of agile methods. Requirements were incubated after a literature review and refined with exploratory case studies. These may be synthesized as follows: • well-structured and configurable agile building blocks, • practical characterization of customization knowledge, • continuous storage and reuse of the team experiences.
What is the created artifact and how is it represented?	An integrated framework for agile methods customization that includes an agile methods metamodel at its core, a repository of practices, an agile knowledge base, a practices improvement tool and a capitalization tool (see Chapter 5).
What design processes will be used to build/refine the artifact?	Exploratory industrial case studies, literature reviews and grounded theory were used to build and refine the artifacts (see elements in red in Figure 4.3 and Section 4.2.2).
What theories support the artifact design and the design process?	Several theories (both descriptive and explanatory) were grounded from the research knowledge base to support the artifact design (see elements in light blue in Figure 4.3). These mainly include the QIP (see Section 2.1.5.2), ME approaches (see Chapter 6), primary agile customization studies and the Hofstede model (see Chapter 11).

Table 4.1.: Design science research checklist (continued)

What evaluations/improvement are performed during the internal design cycles?	Part III presents the incubation of the AMQuICk framework. Its iterative refinement, i.e., the extension of its core metamodel and the design of the practical artifacts, is performed based on the case studies learnings (see Chapters 9, 10 and 11). These case studies are also used to illustrate the usability of the proposed solution.
How is the artifact introduced into the application environment and how is it field tested?	The relevance of the AMQuICk framework was discussed with agile experts using semi-guided interviews. This is reported in Chapter 14. Field testing was conducted partially. Specifically, the improvement and customization tools were designed and tested on a master students project. This is reported in Chapter 13.
What new knowledge is added to the knowledge base and in what form (e.g., peer-reviewed literature, meta-artifacts, new theory, new method)?	A new conceptual framework is developed during the course of this research. The framework is composed of the following artifacts: AMQuICk core metamodel, a repository of agile practices, and a collection of customization matrices (customization knwoledge base). No such knowledge has been yet discussed in literature. Also, other value-added results are the case studies reported in Chapters 9, 10 and 11 which investigate the context-related issues of agile methods implementation. These were published in peer-reviewed conferences.
Has the research question been satisfactorily addressed?	The 4 research questions were addressed iteratively. The evaluation mainly relies on the interview of few experts and on a number of case studies. According to [Holmström et al., 2009], such a subjective evaluation may demonstrate mid-range theoretical relevance. More intersubjective evaluation in multiple contexts would be necessary to strengthen the evaluation.
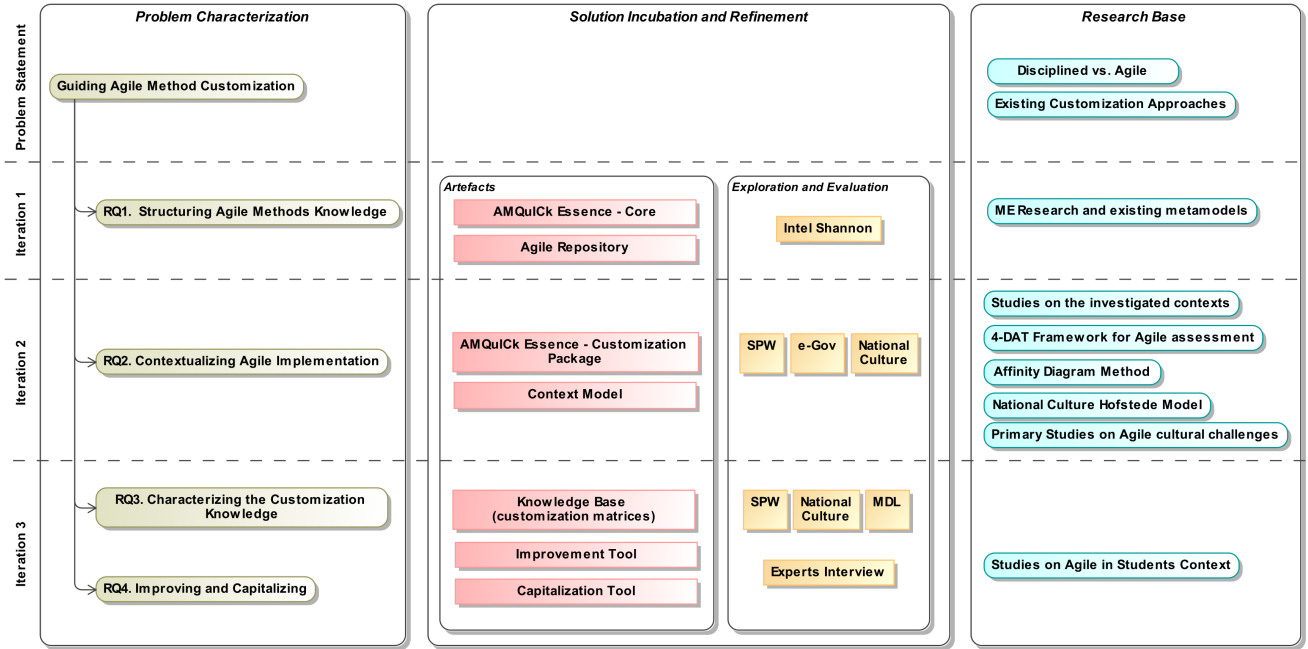
Figure 4.3.: Research Methodology

Iteration 2: Context Study Perspective

With regard to the evaluation of the AMQuICk Essence metamodel, we identified the need for a better characterization of the customization knowledge and formulated the second research question (see Section 3.5.2). The design cycle encompasses exploratory studies that we conducted to investigate the challenges of agile implementation in different contexts and to retrieve customization decisions made by participants (see Chapter 9, 10 and 11). As part of the rigor cycle, each exploratory research starts by a review of primary studies that are connected to the specific context being investigated.

Iteration 3: Customization and Capitalization Perspective

The goal of this iteration is to focus on the collection, structuring and capitalization of the customization knowledge in practice. As part of the *Relevance Cycle*, the third and fourth research questions were defined based on the results of the context study perspective. Lessons learned from these case studies allowed us to enrich the AMQuICk metamodel to characterize the context and customization aspects. We were also able to design a practical representation of customization decisions in the form of matrices intended to be directly used by practitioners. The usability and relevance of these features is evaluated based on a number of illustrations in Chapter 13 and on the feedback of agile experts' (see Chapter 14). This iteration also discusses the design of an improvement and a capitalization tool to be used by teams in practice. The usability of this tool is evaluated in the context of a students' project (see Chapter 13).

4.2.2 Exploration and Evaluation

Several exploratory and evaluative case studies and illustrations were used to support this research. These are briefly reported in the following subsections.

SPW Case Study: Agile in a Transitional Context (Exploratory)

This study aims at investigating the challenges of implementing agile methods in a transitional situation. The investigated organization is an public IT service that has a well-established tradition of waterfall "siloed" development. The management of the organization was interested to smoothly transition to agility so we conducted a context study to understand their organizational

and project-level challenges and to prepare a transition plan which later allowed us to design the AMQuICk life-cycle. Besides the context study, a customization of Scrum was proposed and discussed with a pilot project-team during a project retrospective using the affinity diagram methodology. The study is detailed in Chapter 9.

E-Gov Case Study: Agile e-Government Service Development (Exploratory)

Regarding the previously presented case study, implementing agile in a public service seem to be particularly challenging. In this case study, we further investigated the inherent challenges that practitioners face when developing e-government services using agile practices. To do so, we organized 3 focus groups with representatives of the public service from different decisional/executive levels, identified and categorized the most commonly reported challenges. We then analyzed their impact on the principles of the agile manifesto (see Table 2.1).

The study contributes in several ways to the existing body of knowledge on agile methods. Indeed, to our knowledge, almost no studies can be found on the agile development of e-government services. The results could therefore serve as guidelines for practitioners when implementing agile methods in an e-government setting. The study is detailed in Chapter 10.

Culture Case Study: Agile Cultural Challenges (Exploratory)

One of the threats to validity of the previous study is that the results are narrowly dependent on the Belgian context. Indeed, there exist some research studies that tend to show that cultural factors should not be disregarded when evaluating the efficiency and/or relevance of software engineering practices. The question was also briefly discussed by few agile experts but, to our knowledge, no in-depth studies have been conducted.

Therefore, in this study we investigate the impact of the cultural background of people (i.e., their national culture) on the implementation of agile practices in 3 culturally diverse contexts: Belgium, Malaysia and Singapore. The outcome is a functional set of hypotheses about the potential relationships between cultural traits and the application of agile practices.

The previous two studies allowed us to initiate the idea of customization decision matrices. The study is detailed in Chapter 11.

MDL Case Study: Agility applied on a students' project (Exploratory and Evaluative)

This study aims at proposing a solution for the new requirements formulated in the second iteration. Indeed, the interview with experts confirmed the need to operationalize the framework for the team and to provide them with a mean for capitalizing their expertise at an organizational level. This would be done using simple and visual facilitation tools in order not to impede the agile mindset.

Prototypes of such tools were designed and used in the context of a software engineering capstone course. The latter consists on a development project that simulates a professional environment and where a hybrid Scrum method (i.e., a mix of Scrum, XP, RUP and Kanban practices) is implemented as the framework for managing students' projects and developing the required software.

The study allowed us to design iteratively a goal-driven and continuous improvement tool that student groups used to customize their methodology throughout the sprints according to their specific needs. Moreover, we designed and tested a capitalization activity to be used during the final project retrospective in order to collect the team expertise on using and customizing agile practices. The study is detailed in Chapter 13.

Experts Interviews (Evaluative)

To evaluate the AMQuICk Life-cycle, the context model and the customization knowledge base, we conducted semi-structured interviews with agile experts from different organizations. The overall approach was discussed in details with them and we collected data on their context factors and on the significant adaptation decisions they have undertaken sofar. This study allowed us to demonstrate the usability of the decision matrices, to evaluate the relevance of the framework and to collect the improvement opportunities necessary to operationalize it in real-word situations. The study is detailed in Chapter 14.

Chapter 5

Framework Overview

AMQuICk Theoretical foundations, Life-cycle and Artifacts

As previously discussed, while many organizations are interested in adopting agile methods suitable to their contexts, there is little available guidance on how to do so. This research aims at designing a practical guidance approach to support teams and organizations in implementing situation-specific agile methods. This chapter presents an overview of the approach including its theoretical foundations, life-cycle and main artifacts.

5.1 Claim

An inherent characteristic of agile methods is *adaptability*. Thus, agile methods advocate the frequent adjustment of the development process using retrospectives which, as we argued earlier, may not be sufficient for teams and organizations with complex settings. Indeed, the business goals, the project settings, the people, and the environment in which an organization works create a unique context that may be very challenging. For example, not all teams can commit to *time-boxed iterations* as a way of managing their work. In some situations, a *flow based development* is more adequate. When teams lack relevant agile expertise, they may not consider such customization decisions and would rather implement practices they learned about without much consideration.

Research initiatives for guiding agile methods customization (see Chapter 3) have shown several limitations. Specifically, they have been largely prescriptive, employing predefined and theoretical rules to recommend what a team should better do. Based on the detailed review of the existing approaches and on the lessons learned from the case studies conducted in this research (see Part IV), we claim that success factors of agile methods customization rely on:

1. clear and sufficient structuring of agile building blocks,

2. accurate elicitation of the context factors and customization knowledge,

3. empowering the team to continuously improve their customized method, and

4. continuous capitalization and reuse of team experience.

The proposed approach builds on existing research knowledge to fulfill those requirements.

Firstly, the concept of agile building blocks is inherited from the discipline of SME that advocates the composition of situational development methods using reusable components. We argue that documenting the building blocks of agile methods in a well-structured repository is particularly beneficial for agile practitioners from the same organization or a large-scale community. Indeed, as previously discussed, the experience shows that agile practices are often misapplied because the key information about their efficient application is limited, unstructured and scattered.

Secondly, the approach relies on inductive and experience-based process improvement. Precisely, it adapts the QIP approach and its organizational learning cycle (see Section 5.3.2) to provide systematic steps and guidelines to organizations and teams in order to help them customize their agile practices.

A set of principles also establish the theoretical foundation of AMQuICk. These are presented in the next Section.

5.2 Founding Principles

The following principles will guide the design of the AMQuICk framework.

5.2.1 Flexible Customization

A systematic process for customizing software methods requires a certain balance between control and flexibility. At one extreme, a highly controlled customization approach allows each project to choose its methodology from one of several rigid methodologies. On the other, a situational method is assembled using building blocks that may be of different granularity [Odell, 1996; Harmsen et al., 1994]. A less flexible approach selects and tunes a method outline. The two latter strategies necessitate the use of Computer

Assisted Method Engineering (CAME) tools and the expertise of a method engineer.

AMQuICk advocates for a flexible customization strategy by reusing the concept of building blocks but unlike the traditional method engineering approaches, provide the team with more responsibility in such a process. Indeed, the team should never be provided with rigid methods designed by a methodologist or automatically generated using a CAME tool. The method facilitator in collaboration with the team should define their preferred way of working in terms of practices. Customization rules are used to recommend, prevent the use of or propose configurations for the desired practices. Adopting such a strategy intends to not impede the team empowerment. [Tomasini and Kearns, 2012] refers to this as "the art of balancing freedom with guidance".

5.2.2 Growing Customization Knowledge

The concepts of knowledge and experiences capitalization and reuse have been studied and applied for several purposes such as industrial processes adaptation [Llamas et al., 2016], process learning improvement [Amescua et al., 2010] and software quality improvement [Basili and Caldiera, 1995]. In this research, we use it for the purpose of customizing agile methods to the specific context.

A distinction should be made between customization experiences and customization knowledge. According to [Amescua et al., 2010; Dalkir, 2013] knowledge is the subjective interpretation of individual values, perceptions, and experience. When past experiences are analyzed, information in the form of lessons learned, guidelines, rules, etc. is implied and knowledge is obtained.

[Dalkir, 2013] states that creating, sharing, and applying knowledge as well as feeding the valuable lessons learned and best practices into corporate memory foster continued organizational learning.

This thesis relies on this statement. It assumes that as the implementation of agile grows in a particular organization and as more experiences are collected from different project teams, the knowledge on where practices do or do not work grows. This implies that the definition of the customization knowledge should be flexible. Also, the newly identified context-related challenges should be cataloged in the knowledge base.

5.2.3 Shared Mental Model

Shared mental models are defined as *"knowledge structures held by members of a team that enable them to form accurate explanations and expectations for the task, and, in turn, to coordinate their actions and adapt their behavior to demands of the task and other team members"* [Cannon-Bowers and Converse, 1993].

Software engineering is a highly collaborative activity. The success of complex projects depends above all on the coordinated activity of a team of individuals who have different degree of expertise in particular software engineering areas. Several studies have found that shared mental models are necessary to facilitate information sharing between developers, reconcile conflicts between client representatives and software development project leaders, and improve the overall quality of software.

[Yu and Petter, 2014] particularly demonstrates the value of agile practices in improving shared mental models (i.e. shared understanding) among developers and customers in software development teams. Specifically, it argues that some practices help to develop a shared understanding of the product development tasks (taskwork mental models) and other practices create a shared mental models about team processes and interactions (teamwork mental models). For example, it argues that the *on-site customer* practice improves the development of taskwork mental models since it offers greater opportunities for the team to understand the customer needs. Moreover, the study suggests that when agile practices are not well understood or clearly conveyed to teams, shared understanding is affected and the adaptation of methods is made more difficult.

This fundamental aspect should be taken into account when developing a guidance approach for agile methods adaptation, i.e., the proposed approach should investigate means for improving the team knowledge sharing. One of the principles that motivated the development of AMQuICk is the shared mental theory. Specifically, the repository of agile practices and building blocks is intended to help the team having a shared metal model regarding their development process (see Chapter 8). Actioning and visualizing the method building blocks in the team working space using practice and the improvement board are other techniques that we investigated and that may improve the shared understanding of agile practices(see Chapter 12).

5.2.4 Continuous, Bottom-up and Goal-driven Improvement

Agile software development methods are people-oriented . Developers are enthusiastic and creative problem solvers by nature [Wynekoop and Walz, 2000]. Therefore, we assume that if they are provided with a practical goal-oriented method, they would be motivated to reflect on and customize their own way of working and to increase their application of agile values and principles.

Moreover, it has been discussed that developers have a strong sense of personal worth [Wynekoop and Walz, 2000]. We argue that a bottom-up improvement approach would be in adequacy with this personality trait since it would enable them make their own decisions, disseminate their knowledge and experiences in trying new things and give input to managerial decisions.

The AMQuICk approach highlights the importance of capitalizing on the team process knowledge and provides simple means to help the team communicate about their experimented practices and improvements. Specifically, the improvement backlog helps to fulfill this principle.

The customization of the methods in AMQuICk should also rely on a goal-driven strategy for collecting improvement needs. Goals are defined using the SMART criteria:

- Specific (S): Clear and non-ambiguous definition of goals: what, where and how?

- Measurable (M): The measurement provides feedback about the goal implementation and should inform when the goal is achieved

- Achievable (A): In order to insure that the goal is achievable, it should be assigned to individuals or groups (the drivers)

- Realistic (R): Goals should be achievable in a time frame

- Time-bound (T): Time frame is crucial and should be aggressive, yet realistic

Additionally, we argue that a collaborative improvement goals identification is necessary. This is done using a collaborative activity in which the team members negotiate the improvement goals and the underlying actions to be taken.

Team level improvement also implies *process self assessment*. Self-assessment is key in performing agile software process assessment. Its success lies on

its low cost, good accessibility and ownership of the result. The approach encourages team self assessment of the customized method.

5.3 The AMQuICk Framework

To address the issues referenced in Section 3.5, this research proposes the Agile Methods Quality Integrated Customization Framework (AMQuICk) for systematically guiding the implementation of situation-specific agile methods.

The approach was first introduced in [Ayed et al., 2012]. Its incorporates ideas from recent research and several different paradigms and principles (see Sections 5.1 and 5.2). It encompasses a number of key methodology characteristics that have not been discussed in previous publications.

An overview of the research contributions is provided in Figure 5.1.
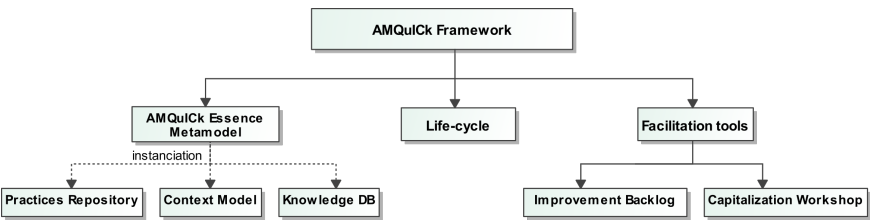
Figure 5.1.: AMQuICk Framework Contributions (An Overview Diagram)

The approach is based on Situational Method Engineering (SME) (see Section 3.3), on the Quality Improvement Paradigm (QIP) [Basili and Caldiera, 1995] and on the paradigm of reuse-based organizational learning, which is a software engineering approach to collect and reuse past experiences with the goal of constantly improving the development process (see Section 2.1.5.2).

AMQuICk describes a theoretical approach for continuously assisting agile practitioners, i.e., during the identification of the challenging context factors, the customization of the method, throughout its enactment and during the capitalization of agile experiences and adjustment of the customization knowledge.

It provides a way to structure the team individual customization knowledge in decisional matrices. These contain information regarding the challenging

context factors and how the applied practices were perceived in the presence of such context factors.

The team experience would be capitalized at the organizational level. Based on a set of experiences, within the same organization or across different organizations, an integrated (or aggregated) customization knowledge would be implied. Moreover, the collection of customization matrices would be exploitable to automatically generate a set of recommendations, including information on how practices should be configured to make them suitable.

Finally, in accordance with the agile paradigm, the method initially designed should not remain unchanged: it should be continuously adjusted. At the process level, AMQuICk describes a goal-driven improvement and capitalization tools to enable this evolution.

The next sections discuss in further details the context of use of the framework, the different stages of its life-cycle and the artifacts necessary to operationalize it.

5.3.1 Context of Use

AMQuICk is intended to be used by experienced agile facilitators or consultants as a guide for implementing situation-specific agile methods, especially in the context of small and medium-sized enterprises transitioning to agile software development or that still have to mature their agile implementation.

The agile facilitator is the primary user of the approach. It is responsible for:

- initiating the approach implementation,

- conducting the context study in order to identify the discontinuing and challenging factors and to prioritize the agility quality factors,

- populating the repository of agile practices,

- formalizing the customization knowledge based on the experiences of project-teams and possibly with the help of a projects portfolio manager and decision-making tools.

Another important role involved in the AMQuICk life-cycle is the development team. Firstly, it actively collaborates in studying the context and

101

choosing the suitable practices. Secondly, supported by the agility facilitator or consultant, the team is responsible for determining the relevant improvement actions and for storing its experience.

AMQuICk is not to be viewed as an imposed and rigid step-by-step process. Rather it should be considered as a flexible guidance approach that facilitates the decision making when agile practices should be implemented in challenging contexts. The approach outcomes are informative, i.e., a team is not forced to adhere to recommendations. The approach can also be used for learning purposes since teams can use it to grow their agile knowledge by learning from each other experiences.

Other artifacts than those suggested in this research may support the deployment of such an approach. Precisely, agile consultant can use the proposed models to generate their own situation-specific checklists or instrument such as assessment models or context models. It would also be particularly interesting to use a decision-making tool to automate the generation of customization guidelines based on the experiences of teams from different companies (community-level). The relevance of such a tool is discussed in the future works of the thesis (see Chapter 14).

5.3.2 Life-cycle

The AMQuICk life-cycle, depicted in Figure 5.2, consists of 3 levels:

- **Organizational / Project Management Level :** This level is where the organizational learning occurs. Its core activities consist of: the context analysis, the recommendation of adapted practices and design of custom methods, the continuous improvement of the deployed method, the capitalization of the project-team experiences and generation and reuse of the customization knowledge.

- **Process Level :** This level is where the process learning occurs. Its core activities (plan, do, check and adapt) are related to the execution and continuous refinement of the custom method throughout the software development iterations.

- **Product Level :** This level represents the development iterations. It is intimately related to the Process Level. Indeed, it has been acknowledged that the software solution influences the evolution of the process and vis-versa. Any change monitored on the products may result in a revision of the process for the next development or maintenance iteration and any change in the process may have impacts

on the produced artifacts. This level is out of the scope of this research but we have briefly discussed the importance of this notion of process-product co-evolution in [Ayed et al., 2013].
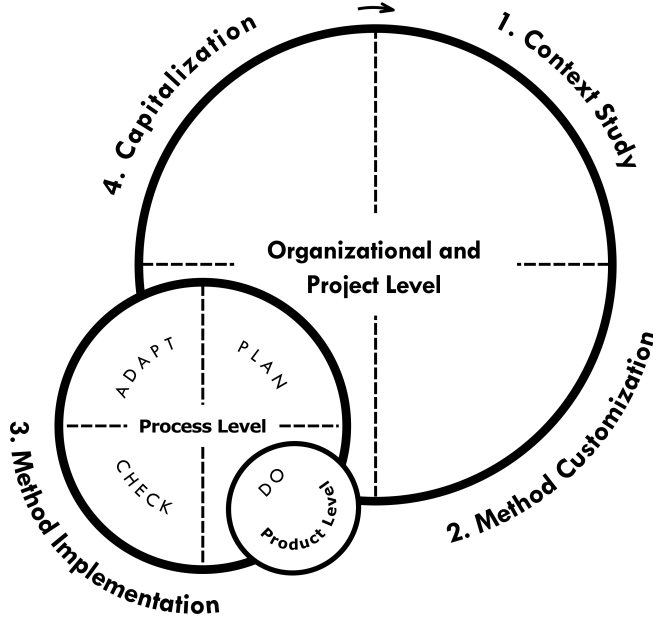
Figure 5.2.: AMQuICk Framework Life-cycle

The suggested framework and its steps are designed to comply to the continuous improvement ideology of the QIP. However, the steps of QIP were not all considered to be compatible with the requirements of a guiding approach to agile methods implementation mainly because the improvement decisions in the QIP are planned and driven by the organizational level whereas the agile team plays a fundamental role in the customization and improvement of its way of working. Table 5.1 shows how the steps of the QIP map with the steps of AMQuICk.

The main cycle of AMQuICk consists of the following 4 steps: (1) Context Study, (2) Customization, (3) Implementation and (4) Capitalization. These are detailed in the following sections.

Table 5.1.: Mapping between AMQuICk and QIP steps

QIP Steps		AMQuICk Steps	
1. Characterize and Understand	1. Context Study	1.a. Assessing the situational context	
2. Set Goals for Improvement		1.b. Identifying challenging context factors	
		1.c. prioritizing agility goals	
3. Choose Process	2. Customization	2.a. Select preferred practices	
		2.b. Gather Knowledge from Similar Contexts	
		2.c. Recommend Configurations	
4. Execute	4.1. Execute	3. Implementation	3.a. Plan
	4.2. Analyze		3.b. Do
	4.3. Feedback		3.c. Check
			3.d. Adapt
5. Analyze Results	4. Capitalization	4.a. Store Experience	
6. Package Experience		4.b. Update Customization Knowledge	

5.3.2.1 Context Study

This step aims at understanding, analyzing and characterizing the context. Interviews, GQM-based diagnosis, and risk-assessment tools can be used during this step. It results consists of the following tasks:

(1.a.) assessing the situational context,

(1.b.) identifying challenging context factors and

(1.c.) prioritizing the agility goals (process quality goals).

5.3.2.2 Customization

Provided the challenging context factors and the initial process quality requirements, the suitability of the preferred agile practices can be assessed. This step consists of 3 main tasks:

(2.a.) selecting preferred practices,

(2.b.) gathering knowledge from similar contexts (retrieving adequate customization matrices), and

(2.c.) recommending configurations.

The output of the customization step is a set of recommendations, including information on whether practices should be selected, discarded or configured. When practices need to be configured, a set of possible configurations may be suggested.

This step also verifies whether the selected practices can be integrated and suggests to remove inconsistent combinations.

5.3.2.3 Implementation

The implementation step is about:

(3.a.) the planning of practice-focused and goal-driven improvement actions (Plan),

(3.b.) the execution of the situation-specific method designed in the previous step (Do),

(3.c.) checking the effectiveness of improvement actions (Check),

(3.d.) when an improvement action is satisfying enough, the method is adjusted accordingly for the next iterations (Adapt).

The output of this stage is a stored history of improvement actions and a list of the adjustments decided by the team.

5.3.2.4 Capitalization

Future incoming projects have to be able to profit from the gained experience in implementing methods and practices. This step consists of the following:

(4.a.) Storing the team experience in the form of new practices or practice configurations, common pitfalls or guidelines on practices execution, tools, techniques, check-lists, etc.

(4.b.) Encoding / updating the customization knowledge by the agile facilitator, coach or consultant.

5.3.3 Framework Artifacts

AMQuICk steps are operationalized using different artifacts (see Figure 5.3).

Its core artifact consists of a metamodel for authoring agile building blocks called *AMQuICk Essence*. This metamodel incorporates the necessary elements for structuring an *agile repository of practices* (a kind of an experience factory), a *context model* and a *customization knowledge base* (see Figure 5.1). Additional operational tools are the *AMQuICk Backlog* and the *Capitalization Workshop*.

The following sections presents briefly each of these artifacts.
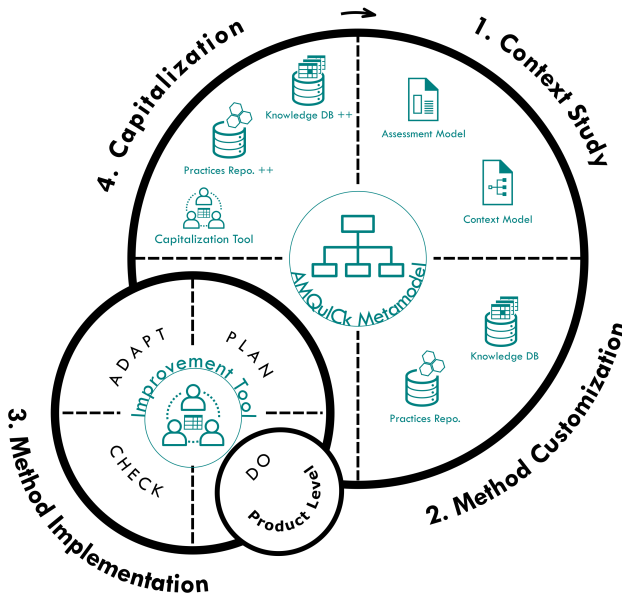
Figure 5.3.: AMQuICk Components

5.3.3.1 AMQuICk Essence Metamodel

The described approach defines a metamodel for agile methods customization. The metamodel relies on the Essence Domain Specific Language (DSL) for methods engineering and therefore we named it *AMQuICk Essence*. The metamodel allows to design well-structured and uniform agile practices using

a set of building blocks. Besides, it includes specific packages for modeling situational contexts, customization knowledge, and improvement actions.

The metamodel has evolved from its first version published in [Ayed et al., 2012]. The original metamodel separates the constructs of a software method into structural aspects (what do we produce?) and behavioral or process related aspects (what we do, when and how?). Its current version focuses more on practice-based methods authoring. It also supports methods agility, i.e., it allows practices and methods to be configured and refined during a project to reflect the team experience. The choice of building the AMQuICk metamodel upon Essence is further explained in Chapter 6.

As illustrated in Figure 5.4, our approach complies to the Meta-Object Facility (MOF) architecture [ISO/IEC 19502, 2005]. At the M2-level lies the required metamodel with its different packages. The figure purposely separates the context package from the method content package. The former is a major contribution of this work while the latter takes advantage of the preexisting Essence DSL to define the abstract syntax of agile methods. At the M1 level, method building blocks, method models and context models are designed. A specific context element may call for a specific method building block. This relationship is guaranteed by customization guidelines derived from the body of knowledge of agile experts and stored in a knowledge base. The M0-level is concerned with the collection of new data regarding the context and the methods and practices enactment. Further details on this architecture are provided in Chapter 7.

5.3.3.2 Context Model

The definition of the context is crucial in the formalization of adaptation decisions. At the context modeling step, AMQuICk aims to provide a better formalization of the context, so that accurate and fine-grained customization guidelines can be formulated.

Agile facilitators or coaches which are part of the team (e.g., scrum masters) decide about the context factors that will be used to formalize the customization guidelines and procedure to assess them. An assessment model (typically a questionnaire) may be used to get accurate indicators. During the customization step, team members contribute to the assessment of the project's context (instantiation of the context model).
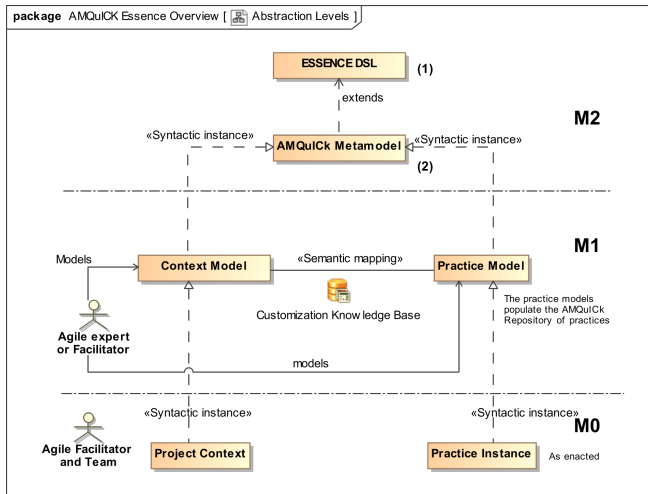
Figure 5.4.: AMQuICk Metamodeling levels

5.3.3.3 Practices Repository

This repository stores the building blocks intended to be reused for authoring agile practices. These are documented by agile experts or practitioners. It is structured according to AMQuICk Essence. The organization may consider storing the key agile practices reported by the community in their local repository. At the start of the project, the agile facilitator with the team will select the convenient and desirable practices. They can also document the experimented practices as they become established. The repository encompasses documentation of methods, practices organized as families, individual practices and practices building blocks.

The methods and practices documented within this repository represent the proven organizational knowledge. It can also be useful in training new practitioners joining the organization.

This artifact is further discussed in Chapter 8.

5.3.3.4 Knowledge Base

In order to support the long-term vision of assisted customization of agile methods, it is crucial to complement existing approaches with more objective

and systematic guidance. AMQuICk proposes to structure such guidance using customization matrices stored in an organizational or inter-organizational knowledge base.

The knowledge base should be continuously expanded and modified by practitioners as they apply agile practices. The collection of matrices would be exploitable to automatically generate recommendations for practitioners willing to customize agile practices to their specific context.

This artifact is further discussed in Chapter 12.

5.3.3.5 Improvement and Capitalization Tools

The AMQuICk improvement tool is a team-level visual tool to be used for guiding a goal-driven improvement of the selected practices. The improvement backlog practice operationalizes elements from the metamodel using an expressive graphical syntax and practice cards.

The AMQuICk capitalization tool is a kind of a post-mortem activity animated by the methodology facilitator. It is suggested as part of the capitalization step to facilitate the collection of the team experience regarding the implementation of practices in context.

These artifacts are further discussed in Chapter 12.

Part III.

SME Perspective

This part of the dissertation investigates the first research perspective of the dissertation which is aimed at exploring the design of an agile Situational Method Engineering (SME) approach.

More precisely, Chapter 6 discusses the issues underlying the implementation of a SME methodology in an agile context, explores and compares the existing SME languages and formulates our proposal for the AMQuICk framework core.

Then, Chapter 7 discusses the initial design of AMQuICk Essence, the cornerstone metamodel of the AMQuICk framework.

Chapter 6

Agile Method Engineering

AMQuICk relies on Situational Method Engineering (SME) (see Section 3.3) which aims at developing software methodologies that fit the specific situation of the project at hand. The rationale for SME is that a project-specific method is created by selecting, tailoring and assembling appropriate building blocks that we call *method components* designed using a method description language. This enables an unbiased description of method components that can be systematically reused across projects.

However, as we previously mentioned in Section 3.3.2, applying SME in an agile context raises several questions. In Section 6.1, we discuss these questions in further details and propose a list of criteria that an agile method engineering approach should meet. Then, in Section 6.2, we explore and compare the existing SME languages in order to study their suitability for the purpose of engineering agile methods. The comparison of these languages is done with regards to the list of criteria identified in Section 6.1. Finally, we justify in Section 6.3 the decision to rely on [Essence, 2014] as a basis for the AMQuICk framework.

6.1 Requirements for an Agile Method Engineering Approach

We identified in Section 3.3.2 some of the issues underlying the implementation of method engineering in an agile context and argued the need for an agile specific method engineering approach. In the following sections, we describe in more details these issues and formulate the specific criteria that such an approach should have.

6.1.1 Method Engineer Role

Method engineering approaches separate between the method modeling stakeholders and method enactment (endeavor-level) stakeholders. At the method domain, the method engineer role is in charge of specifying the contingency factor values, retrieving the relevant components from a method repository and composing them into situational methods (see Figure 3.9) usually with the help of a Computer Aided Method Engineering (CAME) tool. At the endeavor level, the team just has to know what the activities are and in which order should they execute them.

However, since agile methods encourage teams to take the responsibility for the evolution and adjustment of their own practices (see Table 2.1), agile method engineering approaches have to rethink the role of the method engineer [Fitzgerald et al., 2006]. Specifically, they should support teams to take a more active role in the maintenance and evolution of their methods. The usage of method repositories and CAME tools has to be redefined so that the retrieval, population and improvement of method components directly involve practitioners.

6.1.2 Description of Method Components

As noted earlier, method engineering represents a software method not as a single indivisible entity but as a collection of small elements called method components or fragments. The latter basically consist of method building blocks descriptions from a product perspective (what to produce?) and a process perspective (how to do it?). However, since agile methods consist of values, principles and practices, an agile method engineering approach should consider this specificity.

Another point of interest should be considered when implementing a method engineering approach in an agile context: method components usually do not include the information regarding the method evolution during the lifetime of the project. This is problematic since agile methods are living methods where practices change overtime (see Table 2.1). To be consistent with this agile principle, a method engineering approach should include tangible guidance to help teams continuously inspect and adapt their methods.

The granularity layer of method components, introduced by [Brinkkemper et al., 1999], should also be considered when designing any SME approach. The coarser granularity level covers a complete method and the finer level, for example, covers a specific type of a workproduct to be delivered. As noted in

Section 2.2.3, most of the agile methods describe best practices, roles, main activities and provide some details on the essential workproducts to deliver without much prescription. Therefore, agile method components should be of average abstraction. This would make the construction of agile methods easier while maintaining a correct level of flexibility of customization. In fact, the layer of granularity presents a trade-off between complexity and flexibility [Becker et al., 2007]. In other words, if the granularity level of method components is low (coarse-grained), it is less complex to define situation-specific methods but users would have less flexibility in customizing fine-grained parts of methods and vis versa.

6.1.3 Construction of Situational Methods

Depending on the method engineering approach, method components can be of different granularity, i.e., they can represent a single activity or workproduct but they can also describe a complete method model. Hence, the process of constructing situation-specific methods can start with: (1) a set of atomic method components which must be assembled, or (2) an existing method which has to be configured-down or extended. In the first case (1), situational methods are composed from the selection of a set of components that originate from different methods and which are stored in some repository. In the second case (2), the starting of the SME approach is a base method that is divided into components which are examined according to the project circumstances and extended, removed or configured if necessary [Becker et al., 2007].

We present in the following sections the techniques used by each strategy and discuss their applicability for situational agile method engineering.

Bottom-up Strategy

The bottom-up strategy also known as assembly-based or reuse-based relies on the selection and combination of reusable pieces of methods following formal assembly rules of four different types [Bajec et al., 2007]: (a) Process flow rules that define conditional transitions between activities in the process view of the method, (b) Structure rules to constrain the link between method elements of any type (not just activities), (c) Completeness rules to help check whether the created method includes all required components and, (d) Consistency rules to ensure that the project-specific method is constituted of a consistent selection of components.

The first two types of rules (a) and (b) put constraints on associations between method components and the last two types (c) and (d) assure that each constructed method is complete and consistent. As an example, the following assembly rules can be defined by an organization to design a consistent project-specific method (adapted from [Bajec et al., 2007]):

R1. If [the process is in a decision node regarding prototyping] and [the domain is new or the client requires system prototyping] then [include the activity *"Develop prototype of the system"*]. (Process flow rule)

R2. If [the process is in the activity *"Develop prototype of the system"* and the time frame for producing the prototype is short] then [*"develop the prototype of the system"* using the *"Balsamiq"* tool[1] (Structure rule)

R3. Each activity except the last one must have at least one successor activity (Completeness rule)

R4. Each activity must be linked with exactly one role (Completeness rule)

R5. The tool *"Attlassian Jira"*[2] is dependent of the *"Issue tracking"* activity (Consistency rule)

The method engineering research has mainly focused on the assembly-based techniques. However, we argue that such techniques are not adapted for an agile method engineering approach because of the following:

- Agile methods are practice-based whereas the assembly-based techniques focus on the process view of methods.

- The assembly-based techniques are found to be costly in terms of use since they require the expertise of a method engineer and the support of composition guidelines [Becker et al., 2007]. This wouldn't be feasible in an agile context since the focus on the method customization should be kept as lightweight as possible.

- The assembly-based techniques are found to be relatively costly in terms of preparation since they should specify constraints which restrain all possible combinations between components [Becker et al., 2007].

For these reasons, we argue that if such techniques are to be applied in an agile context, they should be simplified so that only the inconsistencies regarding practice associations are verified. For instance, the composition of two exclusive practices such as *"relative story point estimation"* and *"time estimation"* should not be allowed.

---

[1] https://balsamiq.com/
[2] https://jira.atlassian.com/

Top-down Strategy

In the top-down strategy also known as extension-based, the method engineer adapts a reference method model using the following approaches [Becker et al., 2007; Bajec et al., 2007]:

- Configuration: Certain elements from the base method are identified as configuration points. They may be removed, extended or modified through matching predefined configuration rules that refer to specific project situations,

- Instantiation: Methods components are adapted by selecting specific values from predefined possible occurrences,

- Specialization: Method components are modified or added using a reference metamodel or method pattern.

The configuration approach requires lots of preparation since rules should be predefined and method components annotated according to these rules. Similarly, the instantiation approach require consequent preparation effort since it is necessary to designate the adaptable parts of the method repository and the domain of valid values. This makes both mechanisms costly to implement. However, the cost of preparation of the specialization approach is low since it only requires the definition of a well-formed metamodel to be used by a method engineer.

The cost of use of these approaches also should be considered. In fact, the guidance in the case of configuration is high and consequently it is easier to use. Instantiation provides possible values to choose from but no guidance on what values to choose in which situation. Therefore, it is more difficult for a method engineer to use it. Specialization is also costly to use since no situational guidance is provided [Becker et al., 2007]

We argue that because of its low cost of use, the configuration mechanism would be the most adapted to use by agile practitioners. However, the configuration approach should not only rely on predefined rules if it is to allow the agile team to contribute to the customization process as we judged necessary in Section 6.1.1. Besides, depending on the degree of flexibility that we wish to provide to an agile team, instantiation and specialization can also be considered. For example, if they are allowed to introduce any new method component to their specific method, then specialization from a metamodel or a pattern is a solution. However, if the organization wishes to constrain the set of possible variations, then the situational method can be constructed using instantiation.

6.1.4 Summary

To synthesize, we argued in the previous sections that the following criteria should be considered for an agile method engineering approach:

- **Use**: The approach should rethink the role of a method engineer and allow practitioners to directly customize their own way of working. Most of the agile methods define the role of a method facilitator, a member of the team with that manages the development process. It has the responsibility of verifying that the team comply to the defined method, to remove any potential impediments and to help a team to self-organize and make changes quickly. We therefore argue naturly that the role of method engineer should be replaced by the role of a method facilitator. Moreover, a method engineering approach should be easy to use. A top-down strategy to method construction is therefore privileged.

- **Definition of constructs**: The method description language should be practice-focused and of a relatively high-level of abstraction since too much details makes the process of method customization very complex.

- **Scope**: The approach should focus on the method domain as well as the method enactment and continuous improvement

- **Top-down strategy**: The approach should focus more on the top-down mechanisms for method customization, since their cost of use is lower. This implies the customization process to start by a method base that is to be configured down.

In the following section, we will review the existing method engineering languages and compare them with regards to the above list of criteria.

6.2 Existing Method Engineering Approaches

Using a SME approach, a specific method is created by selecting, tailoring and assembling appropriate components from a method repository [Harmsen et al., 1994] (see Section 3.3). The description of methods and method components usually conform to standardized languages called Method engineering languages. The most commonly used formalism to describe method engineering languages is metamodels. The latter provide the necessary method guidance for describing the structural aspects of software methods such as work products, roles, development practices, resources to use and so

on. Moreover, they capture the information regarding the behavioral aspects of software methods (the process-view) such as the sequence of activities, the typical life-cycle for producing a specific workproduct and so on.

Several software method engineering approaches have been defined in the last decades and each of them proposes to rely on a specific metamodel. Section 6.2 discusses the most influential approaches, namely, the Open Process Framework (OPF) [OPF, 2009], the Software Engineering Metamodel for Development Methodologies (SEMDM) [ISO/IEC 24744, 2007], the Software Process Engineering Metamodel (SPEM) [SPEM, 2008a] and the kernel and language for software engineering methods [Essence, 2014].

In the following sections, these approaches are presented and discussed according to the criteria presented in Section 6.1.4. Section 6.2.5 provides a synthesis and comparison between the reviewed languages and Section 6.3 questions their suitability to agile environments and argues the choice to rely upon the Essence language for the design of the AMQuICk customization metamodel.

6.2.1 OPF

The OPEN Process Framework (OPF) [OPF, 2009] consists of a standard framework intended to provide an extensible and tailorable process environment to help organizations in the process of creating and configuring project-specific processes.

The framework comes with an extensible repository of reusable method components documented as hierarchical linked web pages, including construction and usage guidelines. Every component in the repository is an "instance of" some class in a process metamodel specified in the Meta Object Facility (MOF) language (see Section 7.1).

A method engineer is in charge of constructing project-specific processes or organizational standards by selecting appropriate and cost effective components. He is also responsible for documenting, searching, retrieving, and tailoring the method components from the OPF method repository. Figure 6.1 shows the standard core classes of method components and the general relationships between them as described in the OPF metamodel. The major metaclasses consist of *WorkProduct* (the process components that are produced during the project), *Producer* (the roles or individuals in charge of the creation of the work products) and *WorkUnit* (the operations performed by producers to develop the work products). These core metaclasses are supplemented by two other top level classes: *Stage* (the
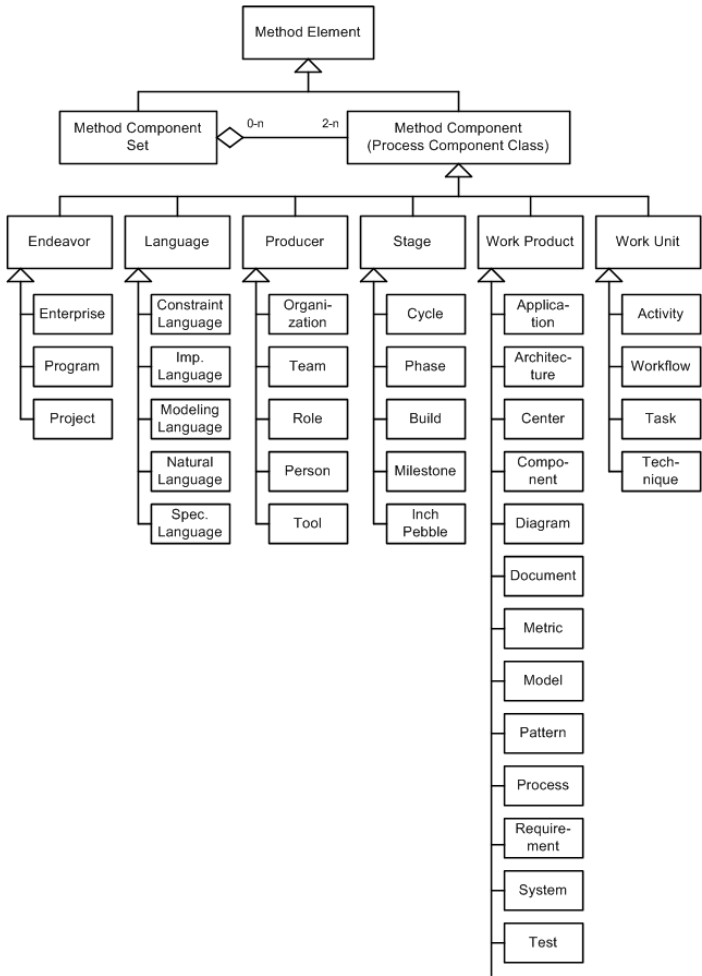
Figure 6.1.: OPEN Process Framework (OPF) core elements

time intervals organizing the work units) and *Language* (the components representing the formalisms used to document work products).

The OPF components are of fine granularity. For instance, several components aim to describe what work is to be done at different granularity levels: Workunit (the supertype), Activity, Workflow, Task and Technique. Activities are at a coarser granularity than Tasks although both describe

what work is to be done. Techniques state how the Task is to be undertaken [Henderson-Sellers et al., 2008].

It is also worth noting that the OPF framework mainly concentrates on the method domain layer at the expense of the endeavor layer and specifically on the product view. It does not consider elements to support process enactment nor describes mechanisms to configure or assemble method components.

6.2.2 ISO/IEC 24744

[ISO/IEC 24744, 2007] is an international standard which describes a formal framework for the definition and extension of software development methods. The core element of the standard is the Software Engineering Metamodel for Development Methodologies (SEMDM) which serves as a formal base for supporting method engineers while authoring and extending software methods. The metamodel includes three major aspects: the process to follow, the products to use and generate and the people and tools to involve. Figure 6.2 depicts the core classes of the metamodel.
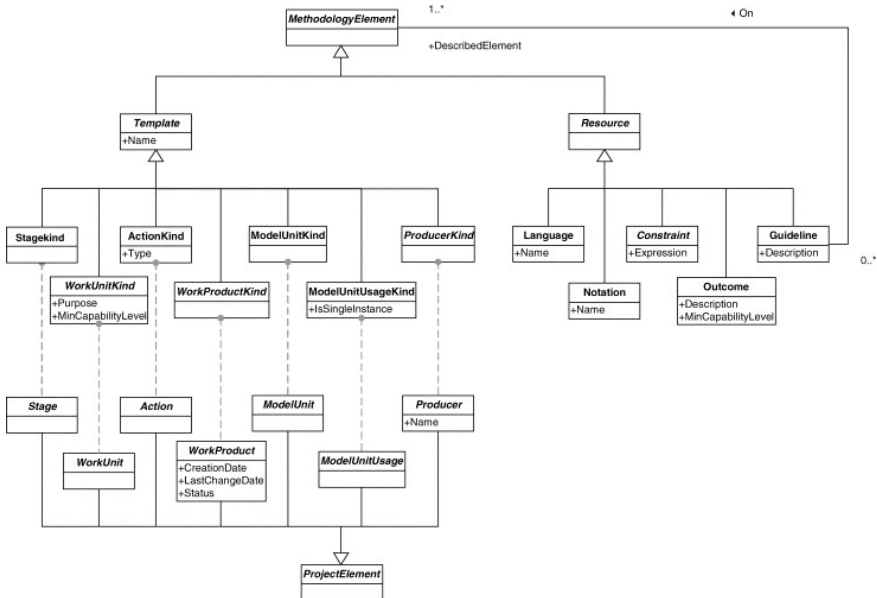
Figure 6.2.: ISO/IEC 24744 core elements

Unlike other method engineering metamodels that only define entities in the method domain, the ISO/IEC 24744 metamodel contains both classes to design a method and classes to capture its endeavor (enactment level). From the practical viewpoint, this is realized thanks to dual-layer modeling (i.e., the description of two modeling layers: method description and method enactment) incorporating powertype patterns and claobjects [Henderson-Sellers and Gonzalez-Perez, 2005b]. This *"already breaks with commonly known and established (MOF-based) modeling paradigms"* [Kuhrmann et al., 2013].

For example, SEMDM includes the class TaskKind and the claobject Task. Task represents an actual task as performed at the endeavor level. TaskKind, on the other hand, represents a kind of task as documented in a methodology. Task has attributes such as StartTime or Duration. TaskKind has attributes such as Name or Purpose. Obviously, every task "is-of" a particular task kind. This is shown in the metamodel by pairing Task and TaskKind into the powertype pattern Task/*Kind, meaning that the TaskKind class (the powertype) partitions the Task class (the partitioned type) (see Figure 6.3).
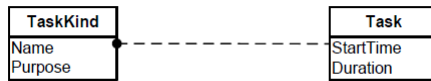
Figure 6.3.: ISO/IEC 24744 - Powertype pattern formed by the *Task* and *TaskKind* classes

The metamodel also integrates constructs for assessing the process at the endeavor level (based on metrics analysis) but do not provide means for capturing the tailoring guidance.

6.2.3 SPEM

The Software and Systems Process Engineering Meta-model [SPEM, 2008a] is a formal metamodeling language designed by the OMG to represent concrete software development processes and a family of related development processes.

SPEM is specified as a standalone metamodel in the MOF language (see Section 7.1) as well as a UML profile (an XMI schema to be integrated to
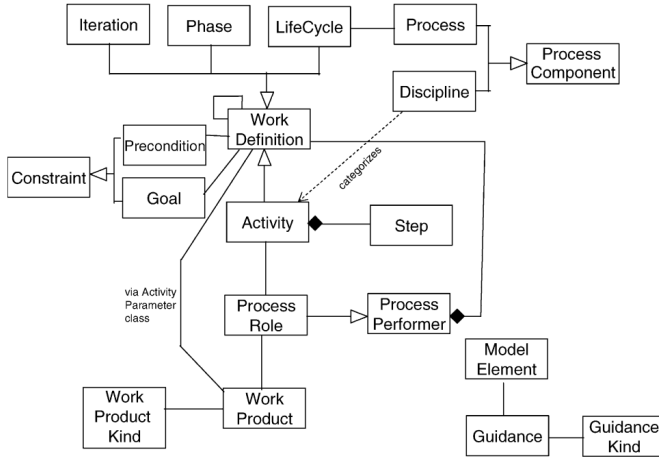
Figure 6.4.: SPEM core elements [Henderson-Sellers and Gonzalez-Perez, 2005a]

UML modeling tools) [SPEM, 2008b]. It comes with a high-level representation of methodology elements and thus can be used to specify multiple software development approaches including waterfall, iterative, incremental evolutionary and agile.

The basic idea of SPEM is that a development process model can be described in terms of few abstract entities basically: *"Process Role"*, *"Activity"* and *"WorkProduct"* (see Figure 6.4). The process roles are active entities that perform operations called activities on concrete, tangible entities called workproducts.

A key separation of concerns in SPEM regards the distinction between the process structural view and dynamic view including temporal informations. In practice, the process entities are separated by the two special package types: a *"MethodContent Package"* and a *"Process Package"* (see figure 6.5). The *"MethodContent Package"* defines reusable method content elements (chunks) such as definition of *roles*, *workproducts*, *tasks* and *guidance*. The *"Process Package"* reuses the methods content elements to create end-to-end processes including temporal sequences such as iterations and phases. It also defines when tasks are to be performed via activity sequences definition and work breakdown structures. It provides a rich set of customization attributes to specify the temporal guidance for process elements. Such elements allow

to generate customized project plans (e.g., with a user-defined number of iterations).
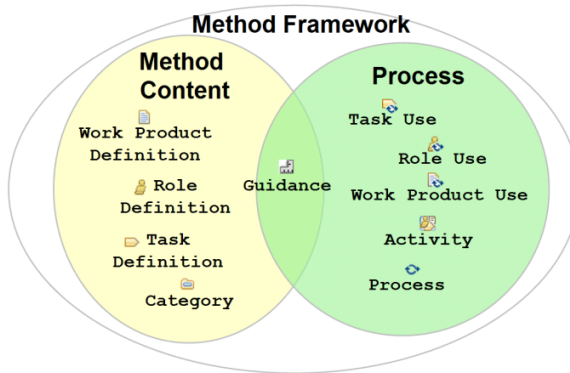
Figure 6.5.: SPEM - Key entities defined in the "MethodContent Package" and "Process Package"

As part of the *"Method Plugin"* package, SPEM describes a set of elements to address the concern of scaling and tailoring methods. The package defines extensibility and variability mechanisms for method content and processes. It provides more flexibility in defining different variants of method fragments and by allowing them to be plugged-in on demand, thus creating situational content only when it is required. An overview of the package constructs is shown in Figure 6.6.

SPEM was largely supported by process authoring tools such as the Rational Process Workbench (RPW)[3] and later the Eclipse Process Framework (EPF)[4]. It was also supported by a number of plugins integrated to UML modeling tools such as Enterprise Architect[5] or Magic Draw[6].

6.2.4 Essence

Essence [Essence, 2015] is a Domain Specific Language (DSL) for engineering software methods developed as part of the SEMAT (Software Engineering

---

[3]`https://www.ibm.com/software/rational`
[4]`https://www.eclipse.org/epf/`
[5]`http://sparxsystems.com/resources/developers/spem_profile.html`
[6]`https://www.nomagic.com/product-addons/no-cost-add-ons/spem-plugin`
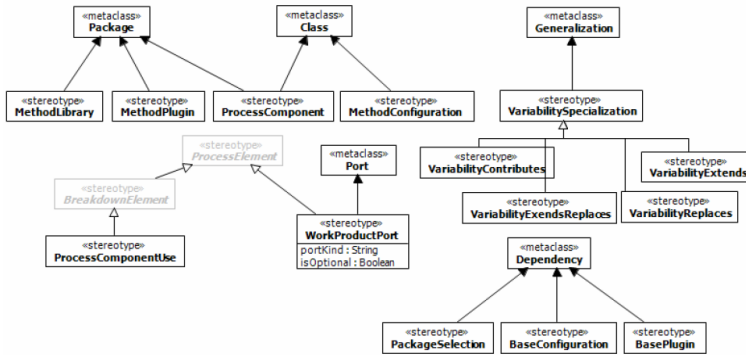
Figure 6.6.: SPEM - Key entities of the "MethodPlugin Package"

Method And Theory) initiative[7]. It is intended to support method engineers as well as practitioners. An interesting point about Essence is that it is practice-oriented rather than activity-oriented. Indeed, a method is defined as a *"composition of practices (at the desired level of abstraction) forming a description of how an endeavor is performed.".* Therefore, Essence promises a better support for the definition of agile methods and their enactment comparing to SPEM [SPEM, 2008a]. However, unlike in SPEM, Essence does not provide means to guide method assessment and customization. It does not recognize the contextual factors faced by organizations and of individual development teams [Ambler and Agile, 2010].

The Essence specification introduces a method architecture that distinguishes 3 core element Groups formed by the composition of basic elements: *kernels*, *practices* and *methods* (see Figure 6.7 and Figure 6.9)
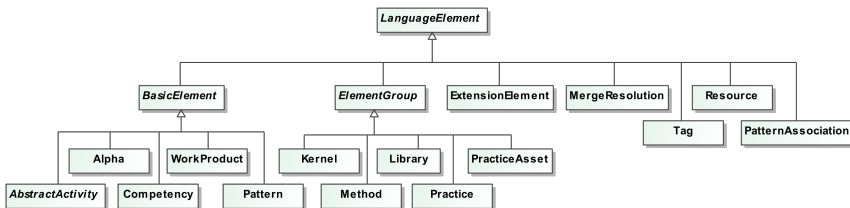
Figure 6.7.: Essence core elements

---

[7] http://semat.org/home

A *kernel* provides a light-weight model of the essential aspects of software engineering (see Figure 6.8). It provides the necessary abstract elements to be used when instantiating any Software Engineering (SE) method: the things we always have (alphas) (e.g., stakeholders, requirements, development team, etc.), the things we always do (abstract activities) (e.g., understand stakeholders needs, understand the requirements, test the system, track progress, etc.), and the skills we always need (competencies) (e.g., analysis, development, testing, leadership, etc.) in order to conduct software engineering endeavors. The kernel elements are organized into 3 areas of concerns: customer, solution and endeavor.
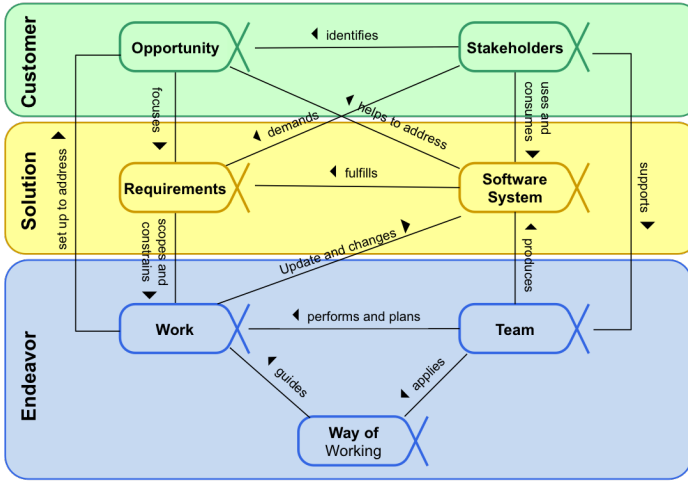
Figure 6.8.: The essence kernel alphas

A *practice* is the concrete guidance of how to handle a specific aspect of a software engineering endeavor (e.g., *user stories [Agile Alliance, 2012]*). Practices may extend the software engineering *kernel* to provide specific guidance on how to form a consistent *method*. A *method* is a composition of practices. Unlike the conventional method engineering languages, in Essence, methods are not just descriptions for developers to read, they are dynamic so they can effectively support their day-to-day activities. This characteristic makes Essence more suitable for an agile team [Essence, 2014].

Figure 6.10 shows an informal conceptual overview of the main language elements and their most important associations using the graphical syntax of the Essence specification. The elements centered in the figure, i.e. *alpha*, *abstract activity* , *workproduct*, *competency* and *pattern*, provide the abstract
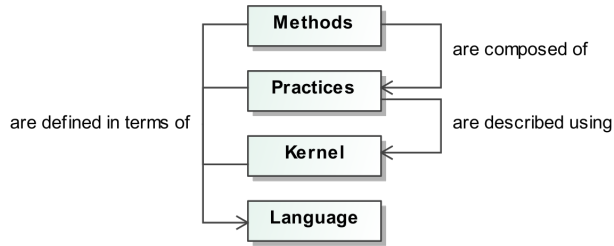
Figure 6.9.: Essence DSL Core Elements

and essential things to do, things to work with and things to know in software engineering endeavors. They are used to represent both the structural and dynamic semantics of software practices and to describe the contents of any software engineering kernel.
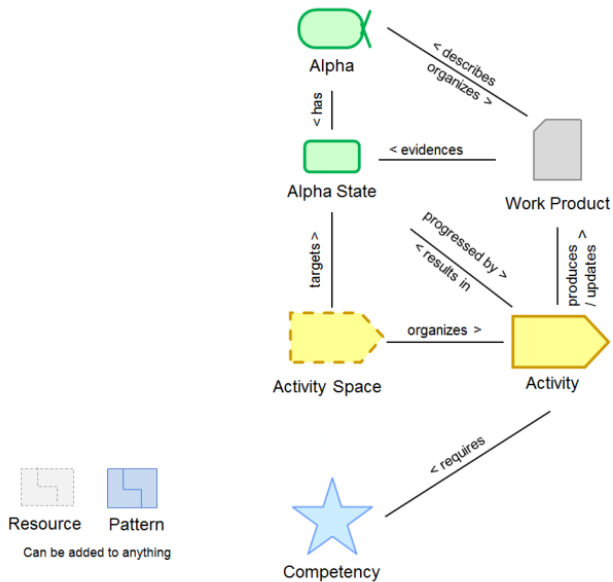
Figure 6.10.: Essence DSL - Conceptual overview

The *Alpha* (an acronym for Abstract Level Product Health Attribute) represents an essential element for making methods actionable. They are used to visualize the achievement of objectives at different level of concerns (e.g., solution, team work, customer, etc.). Alphas are provided with evolving *states* that are further detailed into *checkpoints*. Using these simple mechanisms, the project accomplishments can be simply controlled. Accomplishments may also be visualized using the graphical syntax of Essence and physical cards. Figure 6.11 shows examples of a physical *alpha state cards* (colored according to the area of concern) with a specific current *state* and a *checklist* of what should be done to achieve the state.
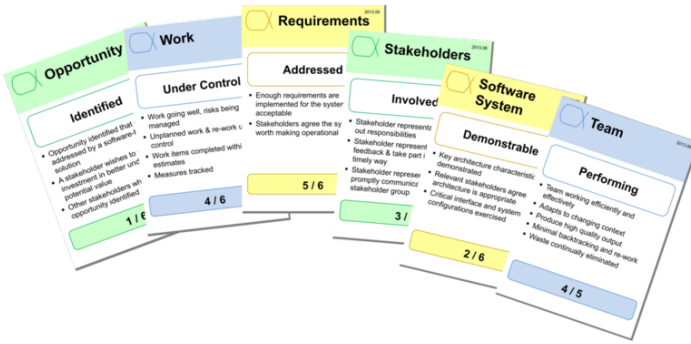
Figure 6.11.: Essence DSL - Examples of alpha state cards

The Essence users are provided with a Practice Workbench[8] for composing, authoring and managing software development practices and methods (see Figure 6.12).

6.2.5 Comparison

A number of method engineering metamodels have been proposed in Situational Method Engineering research. In the previous sections, we have critically examined four of these: the OPEN Process Framework (OPF) [SPEM, 2008a], the Software Engineering Metamodel for Development Methodologies (SEMDM) [ISO/IEC 24744, 2007], the Software Process Engineering Metamodel (SPEM) [SPEM, 2008a], and the Kernel and Language for Software Engineering Methods (Essence) [Essence, 2015]. For each metamodel, we discussed its specification, usability, constructs granularity, tool support and whether it integrates method assessment and customization guidance.

---

[8]https://www.ivarjacobson.com/esswork-practice-workbench
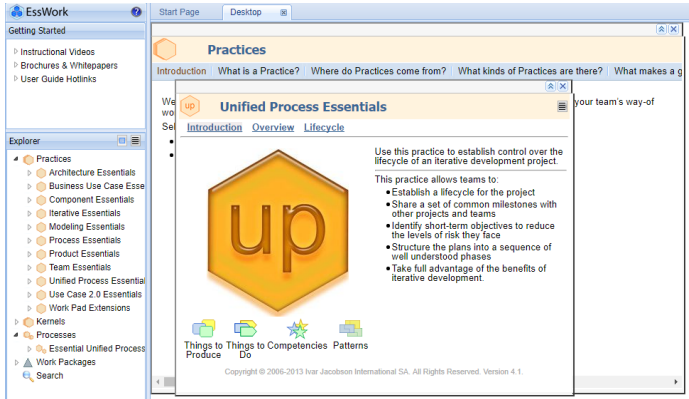
Figure 6.12.: EssWork Practice Workbench.

A comparative analysis is shown in Table 6.1. First, this comparison shows the particularity of the ISO 24744 language [ISO/IEC 24744, 2007] regarding the formalism. In fact, it relies on a dual-layer formalism using powertypes and claobjects to separate the method constructs from their application. The other languages combine all the method constructs in a single layer corresponding to MOF M1 level (see Section 7.1). Another difference regarding the formalism consists in the fact that Essence uses a BNF-like syntax to express different constraints regarding the semantic of language constructs.

Another noteworthy difference relates to the abstraction level (i.e., the degree of details incorporated into language constructs). OPF, ISO 24744 and SPEM highly detail the set of language constructs whereas ESSENCE keep them at coarse granularity. Details are to be added by languages users thanks to specific constructs such *extension elements* and *kernel*. As discussed in Section 6.1, this might reduce the complexity of implementing and using a method engineering approach relying on Essence.

In terms of usability, it appears that all the investigated languages mandate specialists, namely, method engineers to author methods and associated elements such as activities, workproducts and producers. For instance, to describe an agile method such as Scrum, the method engineer would define practices such as *"sprint"*, *"user stories"* and *"relative estimation"*, activities such as *"daily scrum"* and *"sprint review"* and workproducts such as *"burndown chart"*, *"product backlog"* and *"sprint goal"*. At the endeavor level, software developers use the described methods: they apply practices,

hold activities, and create effective software products. Except in the case of Essence, developers have almost no control on the method description, configuration and improvement. Indeed, when using Essence, they still have the possibility to visually bring the method into action and to automatically track the goals achievement.

Regarding the method assessment, it appears that both ISO 24744 and SPEM define a number of measurement-based concepts for assessing the quality of the designed method. Their definition of the quality assessment is therefore rather disciplined (see Section 2.1.5). In the other hand, the assessment of a method designed in Essence is ensured using a continuous tracking of *Alpha* states which is more in line with the agile principles (see Section 2.2.5).

Regarding the aforementioned discussion, we have chosen to reuse the metamodel defined by the Essence DSL. This choice is justified on the one hand by the definition of method constructs in Essence (coarse granularity and practice-based) and on the other hand by its focus on methods use (methods constructed using Essence are meant to be actionable at the team level). The selection is also explained by the fact that Essence *"supports methods agility, meaning that practices and methods can be refined and modified during a project to reflect experience and changing needs"* [Essence, 2015].

In the next section, we discuss the need to extend it to fit the requirements discussed in Section 6.1.1.

6.3 Proposal for the AMQuICk Metamodel

Regarding the requirements for an agile method engineering approach discussed in Section 6.1 and the comparison of method engineering languages provided in Section 6.2, we argued that Essence is the most suitable SME language for the AMQuICk metamodel.

However, the Essence metamodel should be adapted and enriched since it lacks the necessary knowledge for context-driven customization. Specifically, the following features should be considered:

- Define the necessary element for structuring agile methods components at a relatively high-level of abstraction since too much details makes the customization process more complex.

Table 6.1.: Comparison of situational method engineering languages

Characteristic	OPF	ISO 24744	SPEM	ESSENCE
Reference	[OPF, 2009]	[ISO/IEC 24744, 2007]	[SPEM, 2008a]	[Essence, 2014]
Last Release	December, 2001	December, 2014	Version 2.0, April, 2008	Version 1.1, December, 2015
Specification	MOF-based metamodel	Powertype-based metamodel	MOF-based metamodel UML 2.0 profile	MOF-based metamodel UML 2.0 profile BNF-style Textual Notation
Scope	Method domain	Method domain Endeavor	Method domain Endeavor	Endeavor (Methods are actionable and trackable)
Utilization	Method Engineer	Method Engineer	Method Engineer	Method Engineer and Team
Constructs Granularity	Fine-grained	Fine-grained	Fine-grained	Coarse-grained
Practice-based (vs activity-based)	✗	✗	✗	✓

Continued in next page ...

Table 6.1.: Comparison of situational method engineering languages (continued)

Characteristic	OPF	ISO 24744	SPEM	ESSENCE
Lightweight assessment guidance	✗	✗ Disciplined Measurement concepts	✗	✓ Means to track progress using *Alpha* and *Alpha State*
Customization guidance	✗	✗	✓ Concepts for modeling configurability, variability and tailoring needs	✗
Graphical Notation	✗	✓	✓	✓
Tool Support	Repository of OPF components	✗	RPW [1] EPF [2] UML modeling plugins [3],[4]	EssWork Practice Workbench [5]

- Define the necessary elements for structuring an agile experiences factory that would inspire teams, enhance inter-teams learning and facilitate the creation of customized methods at a project level.

- Define the context in which practices will be applied: the context attributes determine the applicability of practices as well as how they should be tailored.

- Extend the graphical syntax to be used by team members in order to support the documentation of practices and to visualize the team experience easier.

Chapter 7

AMQuICk Essence Core

Foundation and Practice Content Packages

As explained in Chapter 5, the metamodel for agile methods customization that we call AMQuICk Essence is a cornerstone of the AMQuICk approach. First, it is designed to facilitate the construction of contextualized agile methods, i.e., it specifies the concepts, rules and relationships required for authoring agile method components and for combining them into context-specific methods. Secondly, it is also to be used as a structure for the repository of reusable practice components. Lastly, it allows to record the customization knowledge base of agile methods (see Chapter 12). The resulting models contain the building blocks of a software method and the knowledge about how these should be combined and customized.

The AMQuICk Essence metamodel has been incrementally developed and refined through 4 iterative design cycles. As a result of the first design cycle, we present in this chapter the core of the metamodel which describes the necessary elements to author agile practices.

Section 7.1 describes how the core elements of AMQuICk Essence fit into the Meta-Object Facility (MOF) architecture. Section 7.2 explains how the core elements of the metamodel are structured into packages. Sections 7.3 and 7.4 focus on the specification of the core elements allowing to author methods, practices and practice repositories. Section 7.5 demonstrates the usage of the designed constructs. Finally, we discuss in Section 7.6 the opportunities of refining AMQuICk Essence by extending it with the evidential knowledge according to practices objectives and context requisites.

7.1 **Specification**

In order to explicit the concepts required for authoring and customizing agile methods, our approach relies on metamodeling, i.e., it defines a metamodel to express the abstract syntax and the structural relationships between method elements. The metamodel is based upon the metamodel supporting the Essence language for methods engineering [Essence, 2014] (see Section 6.2.4). We therefore call it *"AMQuICk Essence"*.
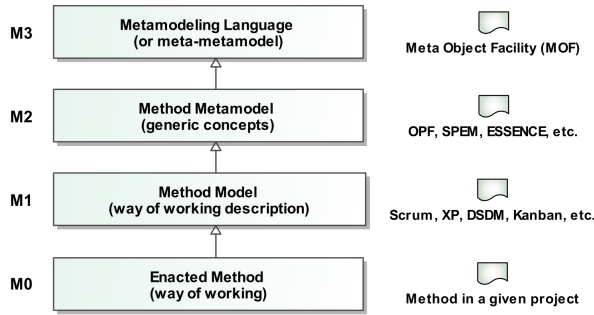
Figure 7.1.: The Meta-Object Facility (MOF) layers

It also fits into a standard specification technique, the Meta-Object Facility (MOF) [MOF, 2011]. The Meta-Object Facility is the conceptual architecture defined by the Object Management Group (OMG) for supporting Model-Driven Architecture (MDA). It is designed as a four layered architecture shown in Figure 7.1. The four levels (from M0 to M3) allow the representation of the software method constructs at different abstraction levels:

- Level 0 - Method Enactment/Endeavor Level: the concrete elements of the running development methodology (e.g., the run-time instance of a practice applied by the team)

- Level 1 - Method Level: the elements specifying the method model (e.g., the description of a specific practice such as pair-programming or test driven development)

- Level 2 - Method Metamodel Level: the metamodel describing the types of method constructs (e.g., the description of the practice construct type)

- Level 3 - Meta-metamodel: the universal specification language allowing the definition of any domain-specific metamodel, including

the different constructs used for expressing this specification, like a "meta-class" or a "binary directed relationship".

People working on an endeavor (e.g., a specific software development project) are concerned by the *Enactment/Endeavor Level*. They make use of methodologies, tools and so forth, which are all defined in the *Method Level*. These two levels is all what the software development team is concerned with. The other pair of levels, i.e., the *Method Metamodel Level* and the *Meta-metamodel Level*, is of interest to methodologists, method facilitators, and tool builders [Tran et al., 2009].

Figure 7.2 shows how the core elements of the AMQuICk framework match this conceptual architecture. AMQuICk Essence logically fits at the M2-level. It includes the domain-specific constructs (meta-classes) representing the semantic of agile methods customization.
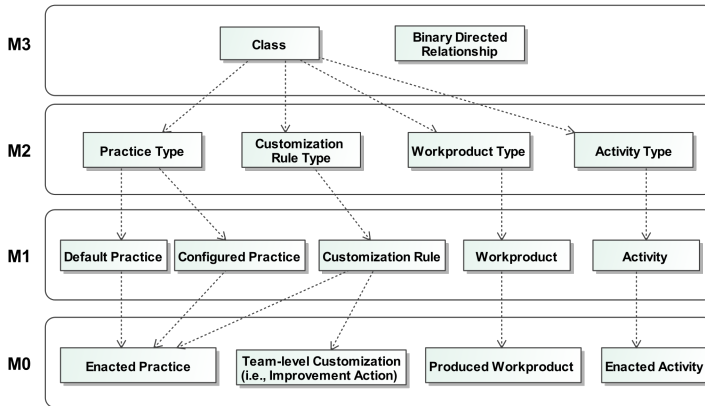
Figure 7.2.: AMQuICk Essence Levels

Agile methods and their constitutive elements belong to models at the M1-level (method definition level). These models are instances of the AMQuICk Essence Metamodel. This means that the agile facilitator would instantiate practices and associated agile building blocks at the M1-level. For instance, to describe an agile method like *Scrum*, the method facilitator would define practices such as *Sprint Planning* and *Daily Scrum*, activities such as *Sprint Planning Meeting* and work products such as *Sprint Backlog* at M1-level. He also would document any known or experienced practice configurations.

A team implementing a default or a configured version of *Scrum* in a specific project would be working at the M0-level (method enactment level). The project team would hold different instances of *Sprints Planning*, and *Daily Scrums* and produce multiple *Sprints backlogs*.

AMQuICK Essence is consistent with the Essence metamodel, i.e., a mapping or a transformation between the two metamodels is possible. However, the Essence metamodel is made very abstract, it is complemented by a set of invariants using the Object Constraint Language (OCL) syntax in order to express correctness constraints on the metamodel [Essence, 2015].

In the design of the AMQuICk Essence, we made the choice of expressing as much constraints as possible using the UML diagrammatic notation to facilitate its comprehension. This design choice was possible because the range of constraints on the constructs of the metamodel is known and manageable. Only few constraints could not be expressed otherwise than by the use of OCL.

7.2 Structure

AMQuICk Essence consists of methodological elements that are used for more than just simply generating method descriptions for teams. These are defined into 4 packages:

- **Foundation:** inherits the generic language constructs from Essence.

- **Practice Content:** contains the definition of agile methods constructs, namely the practices and associated aspects such as *activities*, *roles*, *workproducts* and *resources*.

- **Customization:** captures the constructs necessary to elicit the context and customization knowledge.

The following sections present the packages investigated during the first design iteration, i.e., the Foundation and Practice Content Packages.

7.3 Foundation Package

The Foundation package describes all the base elements, including abstract super classes, necessary to form a baseline foundation for AMQuICk Essence. The core elements of this package and their relationships are inherited and slightly adapted from the Essence metamodel (see Figure 7.3). A detailed
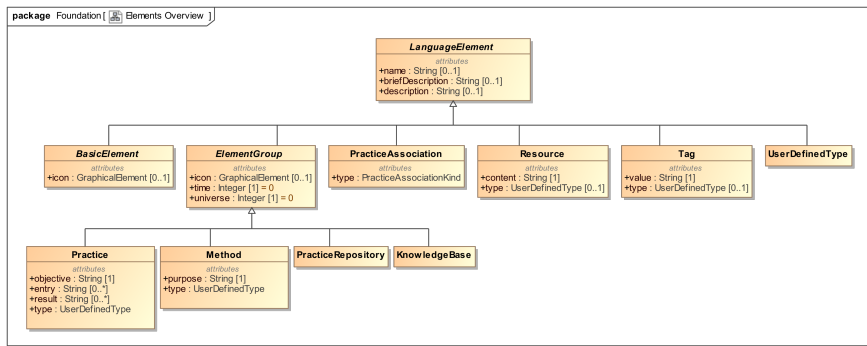
Figure 7.3.: Foundation package - Core elements

definition of each of the foundation package constructs (meta-classes) is provided below.

7.3.1 LanguageElement

Definition 7.1 (Language Element). *An abstract superclass for any AMQuICk Essence concept required for constructing and customizing software development methods (adapted from [Essence, 2014]).*
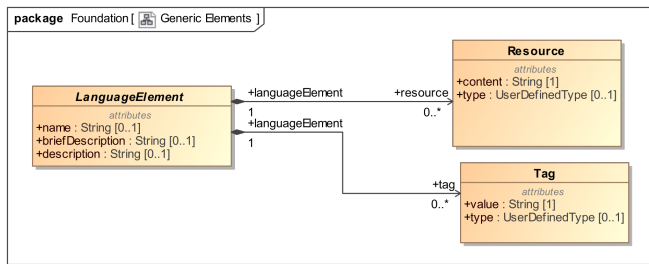
Figure 7.4.: Foundation package - LanguageElement associations

A *Language Element* is the root for all basic elements, element groups and other generic elements. It defines all the concepts required for creating composite entities. Any language element may be related to *Tags* and *Resources* (see Figure 7.4 and Table 7.1).

Table 7.1.: Directional associations owned by a LanguageElement construct type

Association	Multiplicity	Description
tag : Tag	[0..*]	Tags used in labeling the language element.
resource : Resource	[0..*]	Resources used to associate additional informations to the language element.

7.3.2 BasicElement

Basic elements are constituted of the essential building blocks to be used in authoring and customizing agile practices. Basically, they capture the elemental and essential things to do, things to work with and the knowledge of how to do things. Basic elements can be defined as follows:

Definition 7.2 (Basic Element)**.** *An abstract superclass for all the elemental concepts that may be used for constructing software practices.*

Basic elements represent the small configurable parts of practices that teams interact with and adjust to create new practice configurations. Its subtypes consist of *Activity*, *Workproduct*, *RoleUse*, *Role*, *Competency*, *Measure* and *Criterion* (see Section 7.4 and Figure 7.11). The Basic Element construct type is directly inherited from Essence and is identified by a *name*, *icon*, a *briefDescription* and a *description* (see Table 7.2).

Table 7.2.: Attributes characterizing a *Basic Element* construct type

Attribute	Type	Description
name	String	The name of the basic element.
icon	Graphical Element	The graphical representation of the basic element.
briefDescription	String	A short and concise description of the basic element.
description	String	A more detailed description of the element that eventually contains rich formatting informations.

The icon attribute provides a graphical syntax for practice components. It is of type *GraphicalElement*, a data type introduced by the Essence DSL. Using this attribute, it is possible to instantiate a full graphical syntax for method components that may be useful for making methods actionable. In

Section 8.4, we discuss the usefulness of such graphical facilities to design practice cards which can be visualized in the workspace of an agile team and used for stimulating retrospectives.

7.3.3 ElementGroup

Definition 7.3 (Element Group). *An abstract superclass for all the composite concepts of AMQuICk Essence.*

ElementGroup is a generic Essence concept that is used to design meaningful collections of elements that belong together for some purpose. An element group owns or refers other language elements.

As shown in Figure 7.3, it is the supertype for *Method*, *Practice*, *PracticeRepository* and *KnowledgeBase* (the latter will be discussed in Chapter 12). Similarly to basic elements, an Element Group is characterized by a *name*, *icon*, a *briefDescription* and a *description*. It is also characterized by two additional attributes, time and universe (see details in the next paragraph), that allow to capture element group versions.

An Element group may own or refer one or many basic elements (see Figure 7.5).
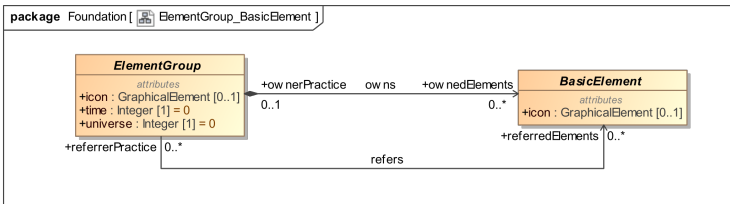
Figure 7.5.: Foundation package - ElementGroup and Basic Element associations

Meaning of Time and Universe

Agility promotes evolving software processes that are aligned with changing business objectives. To capture the evolution of agile practices, methods and even practice repositories overtime, we define the *time* attribute of Type *Integer*. This attribute allows to capture not only the current version on an Element Group but also its history of configurations. The default time value

BASE_TIME $= 0$ represents the very first configuration of an element before any change is introduced at a specific point of time.
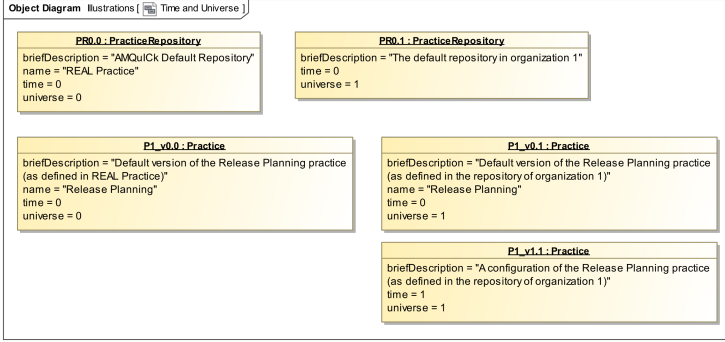
Figure 7.6.: Foundation package - Usage of time and universe attributes

In same manner that every object in AMQuICk Essence models is associated to a time point, it is also associated to a *universe*. The reasoning behind this concept is that the same agile component can exist in different parallel versions depending on the organization in which it is implemented. In the metamodel, we represent this concept as an attribute of *Element Group* of Type *Integer*. The default universe value BASE_UNIVERSE $= 0$ represents the default definition of the component independently from a specific organization. When a particular object is designed with the default settings (BASE_TIME, BASE_UNIVERSE), it means that it is part of the default settings of the AMQuICk framework.

An illustration of element groups, demonstrating the usage of time and universe attributes is provided in Figure 7.6.

7.3.4 Method

Definition 7.4 (Method). *A composition of a set of configured practices (at a desired level of abstraction) forming a description of the practitioners' way of working.*

Methods are not only specifications for developers to read. They represent their actual way of working, i.e., they describe the structural as well as the behavioral aspects of their daily work activities and things that are actually done. This is a different perspective from the conventional definition of

software methods that is found in influential metamodels such as [ISO/IEC 24744, 2007] or [SPEM, 2008a] which differentiate between a method content and its application in a process.

Table 7.3.: Attributes characterizing a *Method* construct type

Attribute	Type	Description
purpose	String	A concise description of the method goal, expressed in a simple statement. For example, a method purpose may be expressed as follows: • Enhancing the agile mindset • Scaling agility at the organizational level • Delivering value continuously • Delivering value iteratively • Experimenting and learning rapidly • Improving the product functional quality • etc. Additional information can be provided using the *description* attribute inherited from the *Element Group* construct type.
type	UserDefinedType	A user of the metamodel may define additional method types to categorize methods. For example, a method can be of the following types: project management, development, maintenance or software deployment operations.

To represent the real way of working of agile teams, a method is designed in AMQuICk Essence as a composition of a set of practices (see Figure 7.7). It has a *purpose* and may be of a specific user-defined *type* (see Table 7.3). As mentioned in [Essence, 2015], the set of practices composing a method should assure the coherence, consistency, and completeness properties (see Section 6.1.3). These properties are assured if the created method includes all required practices that entirely fulfill its purpose and if it is constituted of a consistent selection of practices (verified using the *PracticeAssociation* construct type).

At the M0 level, the method model is composed of a set of configured practices and resources used by the team to guide and support their work. Example of resources are project management tools such as Atlassian Jira[1],

---

[1] https://www.atlassian.com/software/jira

source control tools such as GitHub[2] or continuous integration tools such as Jetbrains TeamCity[3].
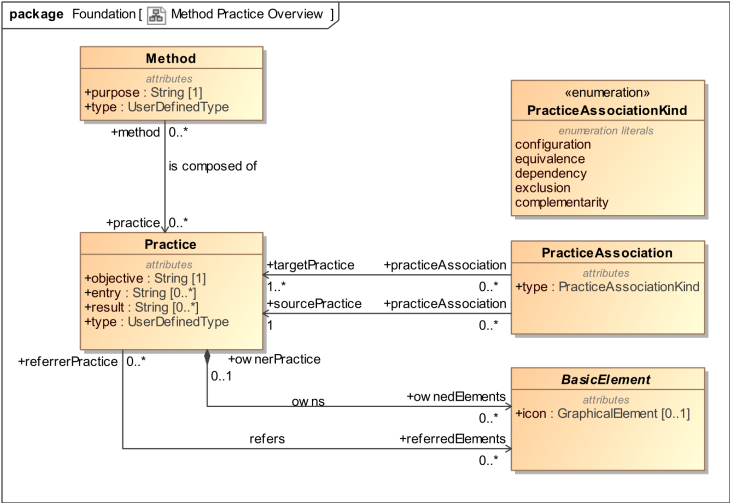
Figure 7.7.: Foundation package - Practice associations

7.3.5 Practice

A basic concept in methods authoring using AMQuICk Essence is the *Practice* construct type. It constitutes the basic building blocks and the entry point to describe any well-formed agile method. A practice is defined as follows:

Definition 7.5 (Practice)**.** *A guidance on how to handle a specific aspect of software development or teamwork. It consists of the proven, repeatable and systematic way of doing work that has a positive impact on the product or process quality (adapted from SPEM [2008a] and Essence [2014]).*

A *Practice* construct type is a derivation of an element group (see Figure 7.3). It is used to describe the guidance on how to perform a specific work, the strategy to do it and the instructions to verify that the objectives have been

---

[2] https://github.com/
[3] https://www.jetbrains.com/teamcity/

achieved. If a practice is composed of a set of activities, the achievement of its objectives is verified using the *CompletionCriterion* construct type.

The effectiveness of practices may also be verified using the *Measure* construct type. Indeed, practices can be associated with measures (see Section 7.4.6). For example, the *"continuous deployment practice"* can be associated with a set of measures such as *"deployment frequency"* and *"failed deployment percentage"* (see Figure 7.16). Such measures should be visible (i.e., referred) within a practice description. This is done using the *refers* relationship between the Practice and BasicElement construct types (see Figure 7.5).

Different types of practices can be defined by AMQuICk Essence users. Indeed, practices usually address different areas of development and teamwork. Therefore, similarly to methods, a practice construct type has a specific user-defined *type* attribute . Table 7.4 details the attributes available to characterize a practice.

A practice can be part of several methods and refers or owns basic elements (see Figure 7.7). It can be associated to other practices using the *PracticeAssociation* construct type (see Section 7.3.6). Indeed, most practices do not capture all aspects of how to perform a specific aspect of software development. Instead, a practice addresses only one perspective and need to be complemented to be effective. To achieve this, it can be associated to other complementary practices.

As earlier explained, practices may exist of parallel versions depending on the *SituationalContext* to which they are associated. For example, a *"daily meeting"* may exist in different versions in different practice repositories. Each version corresponds to a specific configuration of the practice aimed for a specific situational context.

A full example of practice authoring is shown in Figure 7.17.

7.3.6 PracticeAssociation

A practice can be associated to other practices using the *PracticeAssociation* construct type which is defined as follows:

Definition 7.6 (PracticeAssociation)**.** *A construct type used to represent a relationship or a dependency between practices.*

A practice relates a source practice to one or many target practices (see Figure 7.7). To ensure that a practice cannot be associated to itself, we define the following OCL constraint:

Table 7.4.: Attributes characterizing a *Practice* construct type

Attribute	Type	Description
objective	String	The objective that the practice pursues expressed in a short statement. Additional explanations can be provided in the description attribute which is inherited from *ElementGroup*
entry	String	A textual list of conditions or criteria required to start the execution of a practice. This attribute is to be used when we want to provide an overview of what is needed to start the practice. If a more precise list of entry conditions is to be provided, it is recommended to rather use the *EntryCriterion* construct type that accurately defines entry conditions in terms of *Workproducts*.
result	String	A textual list of conditions or criteria obtained as outputs after the execution of a practice. This attribute is to be used when we want to provide an overview of the results of a practice. If a more precise list of completion conditions is to be provided, it is recommended to rather use the *CompletionCriterion* construct type that accurately defines completion conditions in terms of *Workproducts*.
type	UserDefinedType	A user of the metamodel may define additional practice types to categorize practices. For example, a practice can be of the following types: • Development practices: composed of elements required for developing the software product, designing user interfaces, track the product versions, etc. • Teamwork practices: practices aimed at improving teamwork values such as collaboration, communication and transparency. • Organizational practices: practices aiming to apply the agile mindset in the entire organization.

```
-- A practice cannot be associated to itself
Context PracticeAssociation inv:
self.targetPractice -> forAll ( p | p <> self.
    sourcePractice )
```

Besides, to prevent a source practice from being associated with the same target practice twice, we defined the following constraint:

```
-- An association between a source practice and a target
    practice is unique
Context PracticeAssociation inv:
self.forAllInstances() -> forAll (pa1, pa2 |
pa1.sourcePractice = pa2.sourcePractice implies
(pa1.targetPractice.allInstances()-> intersection( pa2.
    targetPractice.allInstances() ))->isEmpty())
```

A practice association may be of 5 kinds: *configuration*, *equivalence*, *dependency*, *complementarity* or *exclusion* (see Figure 7.7). These association kinds are defined in the following paragraphs.

Configuration

Every practice of the software development endeavor can be configured in a way that is convenient to the team or organization. For example, the *"Sprint Planning"* practice may be configured for a distributed environment by spending some more time pre-planning for future sprints to identify interdependencies as soon as possible. Also the practice should be configured in a convenient way so the distributed teams could interact easily with each others (for instance by using videoconferencing tools). All this information is captured in a second practice to be associated with the default "Sprint Planning" practice using the *PracticeAssociation* construct type and using the *time* and *universe* attributes (see Section 7.3.3).

A practice can be configured in various ways by making a specific arrangement of practice constructs (i.e., basic elements and auxiliary language elements). To indicate that two practices configure the same default practice, a *PracticeAssociation* of kind *configuration* should be created. The sourcePractice[1] end point corresponds to the default practice and the targetPractice[0..*] end point relates the set of configured practices (see Figure 7.7).

A practice should have one and only one default configuration where BASE_TIME $= 0$. Practice configurations from the AMQuICk default repository are necessarily associated to one default practice where $(\text{BASE\_TIME}, \text{BASE\_UNIVERSE}) = (0, 0)$

Equivalence

Equivalence between two practices occurs when similar results are reached from similar entries and similar objectives. In other terms, if we define a practice P as a triple formed by an Entry (E), an Objective (O) and a Result (R): $P = (E, O, R)$, we say that $P1 = (E1, O1, R1)$ is equivalent to $P2 = (E2, O2, R2)$ if and only if :

E1 is similar to E2
O1 is similar to O2
R1 is similar to R2

Note that this similarity is determined by agile software development experts.

Two equivalent practices substitute each other. Therefore, they cannot be part of the same method. For example, the XP practice *"Planning Game"* is equivalent to the Scrum practice *"Sprint Planning"*. When both practices are selected to compose a method, only one should be retained.

Dependency

Dependency between two practices occurs when one practice requires another to be consistent. A practice $P2 = (E2, O2, R2)$ depends on another practice $P1 = (E1, O1, R1)$ if the results of P1 are required as an input for P2:

$R1 \subset E2$

For example, the *"User Story Mapping"* practice cannot be implemented if we do not implement the *"User Story"* practice to capture the requirements.

Complementarity

Complementarity designates a relationship in which two or many practices improve or emphasize each other's qualities. A practice P2 complements a practice P1, if P2 contributes with extra features to P1 in such a way to improve it.

For example, the *"Relative estimation"* practice (see Section 7.19) is complementary with *"User Story"* since it adds extra features for improving the subjective estimation of user stories.

Exclusion

Exclusion between two practices occurs when two practices cannot be implemented at the same time because they have contradictory objectives. For example, two exclusive practices can be: (*Test Driven Development* and *Test-after*) or (*Relative estimation* and *No Estimation* (recommended by Lean)). Similarly, the *Code Review* practice is exclusive with *Pair-programming* since the later implies that developers write, inspect, and change the code continuously in pairs. A code review, in contrast, involves inspecting the code later, usually when the author thinks it is ready for deployment.

In other terms, a practice $P1 = (E1, O1, R1)$ excludes a second practice $P2 = (E2, O2, R2)$ if and only if:

> O1 is contradictory with O2

Note that contradiction of two practices' objectives is determined by agile software development experts.

7.3.7 PracticeRepository

As earlier discussed, a *PracticeRepository* construct type intends to capture organizational experiences in implementing agile practices and methods. It may be defined as follows:

Definition 7.7 (Practice Repository)**.** *A container of established and configured practices and methods that provides one or more organizations with meaningful guidance for a specific area of knowledge (e.g., organizational agility).*

A practice repository stores the expertise of a unique organization or several organizations. By default, it comes with a predefined set of default practices (i.e., those documented in the default AMQuICk repository).
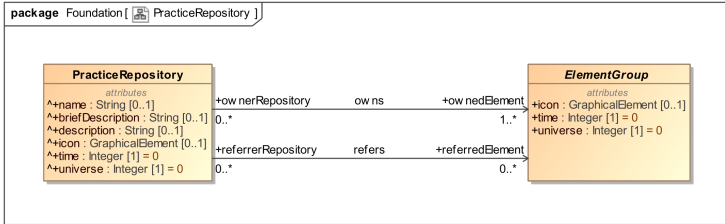
Figure 7.8.: Foundation package - PracticeRepository associations

The association of practice repository to other construct types is ensured using the *owns* and *refers* relationships to the *ElementGroup* entity (see Figure 7.8). A practice repository PR1 having a universe value u1, will own all practices or methods of the same universe. Practices or methods from a different universe are only associated as referred elements (see Section 7.3.3).

The following OCL constraints are needed to verify that a practice repository cannot be associated with other construct types than *Practice* and *Method*.

```
-- A PracticeRepository cannot be associated to itself
Context ElementGroup
inv: self.oclAsType(PracticeRepository).ownerRepository
    ->isEmpty()
inv: self.oclAsType(PracticeRepository).
    referrerRepository->isEmpty()
-- A PracticeRepository cannot be associated to a
    KnowledgeBase
Context ElementGroup
inv: self.oclAsType(KnowledgeBase).ownerRepository->
    isEmpty()
inv: self.oclAsType(KnowledgeBase).referrerRepository->
    isEmpty()
```

7.3.8 UserDefinedType

A *UserDefinedType* is used to constrain the definition of typed elements. It is directly used at the endeavor level, without an instantiation process. It is defined as follows:

Definition 7.8 (UserDefinedType)**.** *A UserDefinedType is a named type containing a description and constraints that can be used to detail typed elements (adapted from [Essence, 2014]).*

As explained in Sections 7.3.9 and 7.3.10, user defined types are to be instantiated at the M1 level to constrain the usage of resources and tags. An example of a UserDefinedType construct is provided in Figures 7.9 and 7.10.

7.3.9 Resource

In order to be able to extend the definition of method elements with external resources, the metamodel integrates a *Resource* construct type that can be defined as follows:

Definition 7.9 (Resource)**.** *An external source of information referenced by an AMQuICk Essence model (adapted from [Essence, 2014]).*

A *Resource* is used to add external informations to any modeling construct. It is directly used at the endeavor level, without an instantiation process. For instance, resources are used to reference external descriptions, templates or guidelines for methodology elements. A resource is simply characterized

by a resource type and a content(see table 7.5). It is to be associated to a unique language element (see table 7.1). The resource type is a *userDefinedType* (see Section 7.3.8). It is used to enable consistent representation and interpretation across community or inter-organizational repositories of practices [Essence, 2014]. For example, a resource type may be simply defined as an enumeration of the following values: book, research paper, experience report, video, tool, template, etc.

Table 7.5.: Attributes characterizing a *Resource* construct type

Attribute	Type	Description
type	UserDefinedType	The user defined type associated with the resource. This is used to define more specific types such as books, research papers, etc.
content	String	A reference to the content of the resource. If no type is provided, the reference is provided in any suitable way (e.g., an url, pdf, etc.)
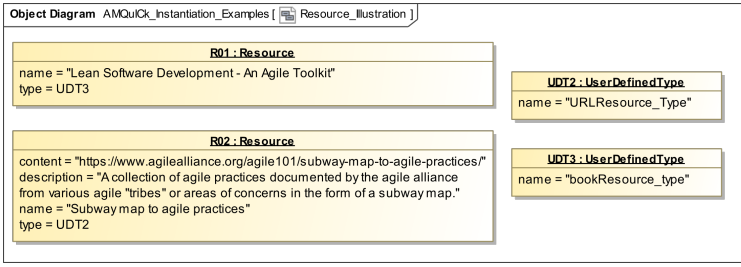
Figure 7.9.: Foundation package - Example of resources

7.3.10 Tag

A *Tag* is used to attach some labeling to a specific modeling construct. It is directly used at the endeavor level, without an instantiation process. It may for example be used to add search labels to a specific method component.

Definition 7.10 (Tag). *A label attached to a language element for the purpose of identification or to provide additional information (adapted from [Essence, 2014]).*

151

Table 7.6.: Attributes characterizing a *Tag* construct type

Attribute	Type	Description
type	UserDefinedType	The user defined type associated with a tag. This is used to define specific types version tag, search tag or area of concern tag.
content	String	A reference to the content of the tag.

Examples for tagging include author tags, version tags, and categorization into areas of concern like "team space", "customer space", "solution space" or "project management space". An example of tags definition is illustrated in Figure 7.10.
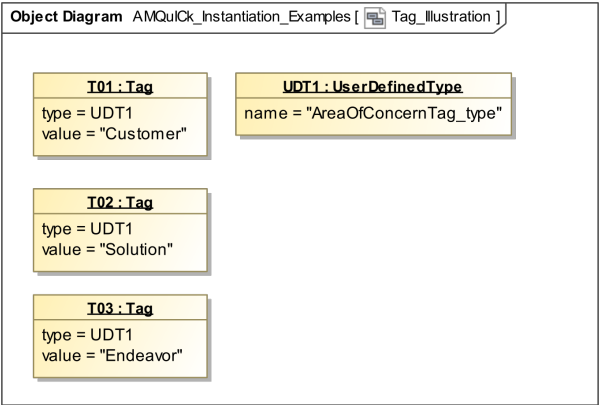
Figure 7.10.: Foundation package - Example of tags

7.4 Practice Content package

The *PracticeContent* package contains the basic elements required for authoring agile practices. The elements and their relationships allow to both represent the structural and behavioral aspects of practices. They may be composed in different ways to generate simple or rich (in expressiveness and usefulness) practice models.

Rich modeling is allowed by the metamodel since some organizations may require thorough method guidance. However, the recommendation of the

AMQuICk framework is to keep practice models as simple as possible in order to ensure a wider applicability, greater adaptability and acceptance by the agile community. Simple practice models should be perceived as a descriptive tool to share mental models and to enable continuous learning. They conceptualize what people actually do and how they do it and not what we think they should be doing [Kruchten, 2011]. Therefore, they should not be prescriptive regarding the gradual transformation of artifacts and the work breakdown into activities and tasks.

Moreover, unlike Essence, the elements of this package do not focus on tracking the team progress but rather on capturing the possible configurations of methods and practices (i.e., the specific arrangement of practice construction elements). In other terms, while Essence focuses on the dynamic of methods use (supported and enforced by tools), AMQuICk Essence targets the dynamic of methods evolution and customization.

Furthermore, AMQuICk Essence focuses on the customization of practices. Specifically, it describes other constructs outside the PracticeContent package to support this aspect (see Chapter 12).

An overview of the package elements is provided in Figure 7.11. A detailed definition of each of these constructs is provided in the following subsections.
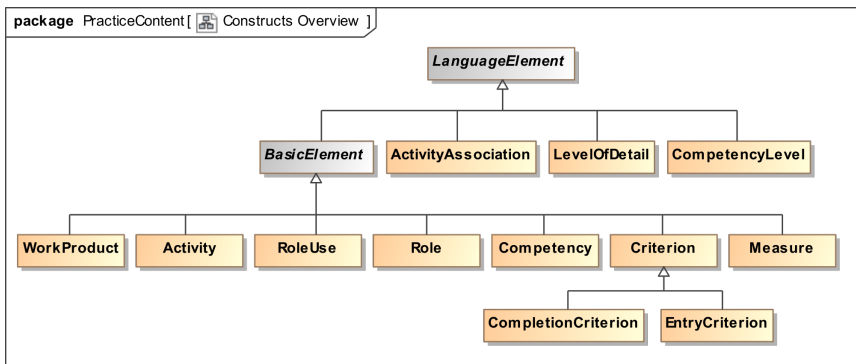
Figure 7.11.: PracticeContent package - Elements overview

7.4.1 Activity and ActivityAssociation

Activities constitute basic and essential elements to describe the dynamic behavioral of practices. Essence incorporates several constructs for capturing the dynamic behavior of methods, including *Alpha*, *Activity*, *ActivityAssociation*, *State*, *Checkpoint* and *Action*. As earlier argued, AMQuICk Essence captures such dynamic behavior in less details since it focuses on structuring practice building blocks and not on tracking the progress of work or on describing the exact breakdown structure that progressively transform product artifacts. Therefore, only the *Activity* and *ActivityAssociation* construct types were retained.

An activity constitutes the basic unit of work captured by AMQuICk Essence. It describes the work to be performed in order to contribute to or achieve the objectives of the owned practice. It is defined as follows:

Definition 7.11 (Activity)**.** *A concrete definition of the basic unit of work performed within a practice. (adapter from [SPEM, 2008a] and [Essence, 2015])*

If a rich practice model is to be created, entry and completion criteria (*Criterion* construct type) may be associated with each activity composing the practice. An activity could start if its entry criteria are fulfilled and is considered as completed if all its completion criteria are fulfilled. These criteria may be expressed using a simple textual notation or in terms of the level of detail of a workproduct (see Section 7.4.4). Indeed, the completion of an Activity may be subject to the achievement of a certain level of completeness of a workproduct.

Besides, activities may be associated with competency levels and competencies (see Section 7.4.3).

Finally, activities may be related to each other using the *ActivityAssociation* construct type (see Figure 7.12) which is defined as follows:

Definition 7.12 (ActivityAssociation)**.** *A relationship or dependency between two activities expressed by an owner practice to capture a work flow or a work breakdown structure.*

Activity associations may be of different types among which: *part-of*, *start-before-start*, *start-before-end*, *start-after-start* and *start-after-end*. For example, if the kind of the association is *part-of*, it means that the first end of the association (i.e., *activity1*) is part of the second end of the association (i.e., *activity2*). Similarly, if the association is of kind *start-after-end*, it means

that the first end of the association (i.e., *activity1*) starts after the end of the second end of the association (i.e., *activity2*) (see Figure 7.12).
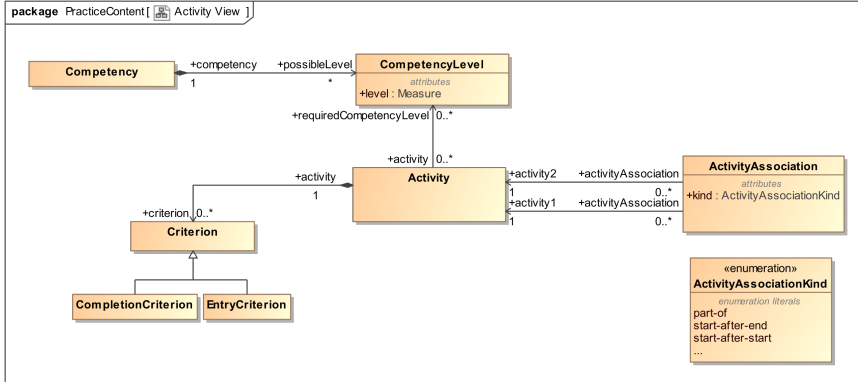
Figure 7.12.: PracticeContent package - Activity view

7.4.2 Competency and CompetencyLevel

A competency refers to the capabilities, knowledge and skills required to complete an activity successfully. It can be of different levels. For example, to perform the *"Unit Testing Activity"* the performer (i.e., a team member) is required to have a good knowledge of one automated testing framework such as JUnit[4], Mockito[5], or pytest[6].

We define Competency and CompetencyLevel as follows:

Definition 7.13 (Competency)**.** *A competency is a quantifiable capability, knowledge or skill required to perform an activity successfully.*

Definition 7.14 (CompetencyLevel)**.** *A CompetencyLevel is a measure that quantifies the capabilities required to perform the activity.*

As earlier noted, competency levels contribute to the rich modeling of practices. A systematic definition of competency levels for all activities composing a software practice is not at all recommended by AMQuICk since this could result in highly prescriptive practice models. Rather, for

---

[4]https://junit.org/junit5/
[5]http://site.mockito.org/
[6]https://docs.pytest.org/en/latest/

informative purposes, competencies can be directly attached to practices without necessarily specifying the level of expertise that is required.

7.4.3 Workproduct and LevelOfDetail

A workproduct refers to any artifact developed and produced during the project by the development team to support the overall software development process. We define it as follows:

Definition 7.15 (Workproduct). *Any artifact developed and produced during the project that delivers value at any step of a software engineering endeavor.*
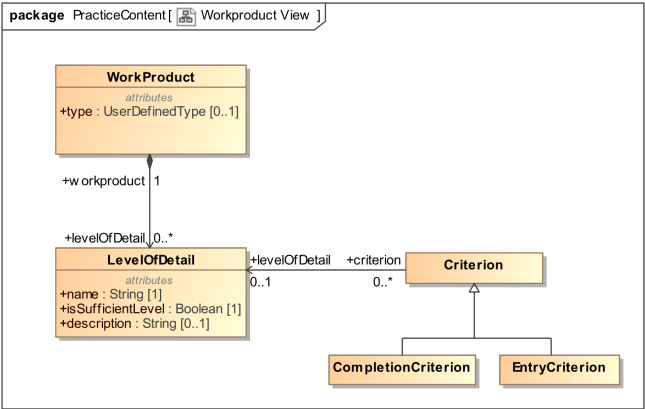
Figure 7.13.: PracticeContent package - Workproduct view

A workproduct can be of different types such as internal reports, specification documents, models, technical notes, code, tests scripts, QA plans, bug lists, deployment notes, and so on. It can also be an internal or external deliverable or a description for non-tangible product elements.

Workproducts may be described at different level of details (see Figure 7.13). The *LevelOfDetail* construct type is used to describe the quantity and granularity of information that is present in a workproduct. For example, they allow to distinguish between *"a [high-level] system architecture, a formally modeled system architecture, and an annotated system architecture which is ready for code generation"* [Essence, 2015]. It depends on the practice to define which of these detail levels is sufficient.

It is important to note that levels of detail may be used to define completion or entry criteria for practices or activities. For example, the completion criteria of a *"User Story"* practice is achieved when the produced Story is *estimated* and if it contains *acceptance criteria* (see Figure 7.17).

7.4.4 Criterion, CompletionCriterion and EntryCriterion

The *Criterion* construct type and its subtypes *CompletionCriterion* and *EntryCriterion* are used to inform about the conditions that determine whether a practice or an activity composing it is ready to be executed (entered) or to be indicated as complete. These construct Types are defined as follows:

Definition 7.16 (Criterion, CompletionCriterion and EntryCriterion). *Conditions that can be tested as True or False to determine whether a practice or an activity can be entered or is complete (adapted from [Essence, 2015])*
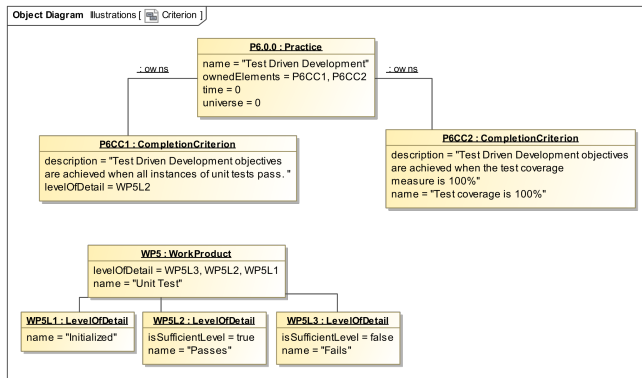
Figure 7.14.: PracticeContent package - CompletionCriterion example

A *Criterion* must be specialized by either *EntryCriterion* or *CompletionCriterion*. It may be expressed in terms of a required level of detail of a specific workproducts. For example, the completion of the *"Test Driven Development"* practice is subject to a *"is passed"* status of all instances of the workproduct "unit test" (see Figure 7.14). An activity or a practice is considered as completed or ready to start when the specific level of detail defined by the criterion is reached. A criterion may also be expressed independently from a workproduct's level of detail. Back to the *"Test Driven*

Development" practice example, another completion criterion would be the achievement of a test coverage of 100% (see Figure 7.14). Indeed, test driven development is a test first approach. Theoretically, it recommends to test every single line of code, even though in practice it happens often that developers do not test anything obvious.

It is worth to note that the systematic definition of entry and completion criteria for all activities composing a software practice is not at all recommended by AMQuICk since this could result in highly prescriptive practice models. Rather, for informative purposes, such criteria can be directly attached to practices as a mental note.

7.4.5 Role and RoleUse

Practices are performed thanks to the active participation of one or more individuals with given skills. The roles involved in the fulfillment of practices can be defined using the *Role* and *RoleUse* construct types.

A role is defined as follows:

Definition 7.17 (Role). *An individual (or a group of individuals) with a set of skills that perform(s) a work unit either directly or indirectly (i.e., creates, evaluates, iterates, or maintains).*

Definition 7.18 (RoleUse). *An association referencing a unique role to a practice and specifying how that role is involved in the fulfillment of the practice.*

Every *RoleUse* can reference only one Role and every Role can be represented by many *RoleUses* (see Figure 7.15).

A *Role* involved in a *Practice* may be referred simply or by specifying its responsibility (eg., responsible, performer, facilitator or observer) using the *RoleUse* construct type. For example, a *"Daily Scrum"* Practice is performed by team members under the supervision of a Scrum Master. The Scrum Master is then considered as a facilitator of this activity and the team as a performer role.

7.4.6 Measure

Practices may own or refer measures that contribute to their effective implementation and assessment. For example, the *Test Driven Development*
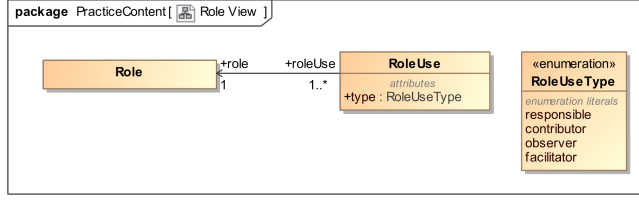
Figure 7.15.: PracticeContent package - Role View

practice can be assessed by systematically verifying the *test coverage* measure. Similarly, the *continuous code review* can be assessed by verifying the *number of pull requests per sprint.*
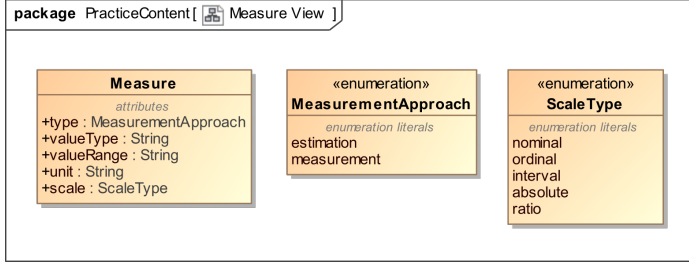
Figure 7.16.: PracticeContent package - Measurement View

The measure construct type can be defined as follows:

Definition 7.19 (Measure)**.** *The number or category assigned to an entity that can be assessed quantitatively by making a measurement.*

A measure can also be defined as a *"function whose input are software or process data and whose output is a single numerical value that can be interpreted as the degree to which the software or process possesses a given attribute that affects its quality"* (adapted from [IEEE 1061, 1998]).

Table 7.7 details the attributes that characterize a measure construct type. In earlier versions of AMQuICk Essence, we further detailed the practice measurement concepts into measurable entities and measurable attributes [Ayed et al., 2012]. Such constructs would be used in the context of measurement-based assessment of agile practices which is out of the scope of the thesis. Indeed, the main goal of the research is on structuring and customizing the

159

Table 7.7.: Attributes characterizing a *Measure* construct type (adapted from [Vanderose et al., 2012])

Attribute	Type	Description
type	MeasureApproach	Provides information of the stance of the method regarding software measurement (i.e., measurement or estimation).
valuetype	String	Provides the type associated to each value produced by the method (e.g., integer, real, string, etc.).
valueRange	String	Provides an interval of values (of the same value type) that presents the lower and upper bounds for each value produced by the measure.
unit	String	Provides the unit associated to each value produced by the measure (e.g., story points, line of codes, etc.)
scale	scaleType	Provides the scale associated to each value produced by the method (i.e., nominal, ordinal, interval, ratio or absolute)

agile building blocks and not at providing practitioners with agility assessment models. This question was however discussed in previous research results and specifically the integration with the MOCQA framework for iterative quality assessment model was considered [Vanderose et al., 2012, 2014].

7.5 Practice Authoring Examples

In this section, we illustrate how AMQuICk Essence constructs may be used to structure agile practices knowledge. It should be regarded as an early testing of AMQuICk Essence focused on the exploitability and expressiveness.

Figures 7.17, 7.19 and 7.18 show some illustrative examples to demonstrate how AMQuICk Essence constructs can be used to describe practices. The following paragraphs detail the illustrated practices: User Story (P2.0.0) , Story Mapping (P3.0.0) and Relative Estimation (P4.0.0).

User Story

User Story[7] is a core practice in agile software development that many methods recommend. It designates a practice that helps to write concise and value-driven requirements called user stories. A User Story should contain just enough information so that developers can provide a reasonable estimate for it and can develop it within one iteration. It is typically a short and a simple description of a feature told from the perspective of the user who desires the capability. They are typically written following the connextra template:

> As a *< type of user >*, I want *< some goal >* so that *< some reason >*.

Figure 7.17 shows an example of how the User Story practice can be modeled using AMQuICk Essence constructs. First, using the *RoleUse* construct type, the practice is attached to two roles :

(RU1) the *Product Owner* the business representative whose responsibility is to make sure that a *Product Backlog* of user stories exist and is consistent,

(RU2) the *Team* since user stories can be written or refined by team members.

The practice description is also associated with a set of resources that would support practitioners write good user stories:

(RES1) the connextra template, also known as the *Role-Feature-Reason*[8] template, which is the most common way for writing simple user stories,

(RES2) the Card-Conversation-Confirmation (3C's)[9] model that originates from the XP method and which describes 3 aspects to consider when writing user stories: (1) a physical *Card* that easily communicate the essential information of user stories to developers, (2) the feature captured by the user story is communicated to the team members through a *Conversation* (exchange of thoughts and opinions) and (3) the user story needs to include *Confirmation* examples and acceptance tests to check if the feature is correctly implemented.

---

[7]https://www.agilealliance.org/glossary/user-stories/
[8]https://www.agilealliance.org/glossary/role-feature/
[9]https://xprogramming.com/articles/expcardconversationconfirmation/

(RES3) the INVEST checklist describes a widely accepted set of criteria to assess the quality of a user story: **I**ndependent (of all others), **N**egotiable (should leave space for discussion), **V**aluable (must deliver value), **E**stimable (its size can be estimated), **S**mall (fine grained enough to plan, estimate and prioritized) and **T**estable (provide enough information to make the definition of tests possible).

The example of User Story authoring also shows the related workproducts and how activities may be used to capture the workflow that practitioners could follow to implement the practice. More precisely, the example describes the following activities and workproducts:

(P2AC1) Define EPICs: Usually prior to writing any user story, based on the initial *Product Vision (WP1)*, high level product features called *EPIC user stories (WP2)* are identified and are later broken down into smaller *User Stories (WP2)*.

(P2AC2) Initialize: The product owner initializes the set of user stories based on his product knowledge, the product vision and on the list of EPICs. This activity may be done in collaboration with team members.

(P2AC3) Simplify and decompose: The initial set of user stories may be simplified, composed and decomposed with respect to the INVEST criteria so that a better set of user stories is produced. This activity may be done in collaboration with team members.

(P2AC4) Define acceptance criteria: A user story is considered as ready for development if it is testable. Usually, it is recommended to define acceptance criteria (i.e., the conditions of satisfaction of the feature). Acceptance criteria provide the detailed scope of the requirement which help the team to understand the value and design test cases.

(P2AC5) Estimate: A user story is considered as ready for development if it has been estimated, usually in terms of development effort. Relative estimation is usually a widely recognized practice for estimating user stories.

Moreover, the example shows that it is possible to link practices to metrics. For example, user stories are usually associated with the following measure:

(M1) Lead Time: A measure borrowed to Lean manufacturing which represents the time elapsed between the formulation of the user story and its deployment in production.
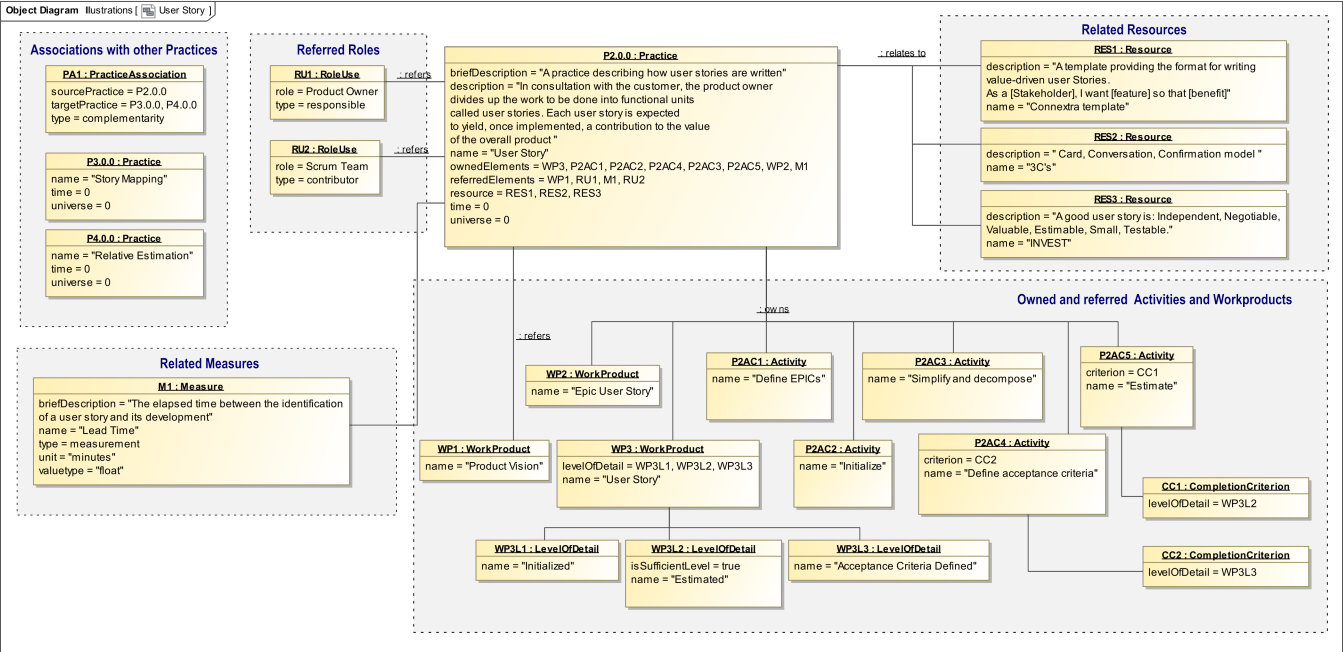
Figure 7.17.: Practice Authoring Example - User Story

Finally, the user story practice is associated with several practices. The example depicts the following association:

(PA1) The *Story Mapping* and *Relative Estimation* practices complement the *User Story* practice. Indeed, Story Mapping allows to improve the activity of collaborative collection and/or prioritization of user stories and Relative Estimation complements the practice with an estimation technique.

Story Mapping

Story Mapping[10] is a practice that provides a visual way of structuring a backlog of user stories. It helps to collect and/or refine the requirements collectively and to enhance their understanding.

The practice basically consists of an engaging exercise where all participants (product owners, method facilitators and team members) are involved in the process of ordering user stories and defining development priorities.

It generates a more structured and visual structure of user stories known as a *Story Map.* Usually, the horizontal axis of the story map sorts user stories according to product goals (business value), main product features and/or Epics. The verical axis allows to sort the user stories according to their development priority or their implementation sophistication (i.e., level of details we add for each story or feature). The practice is usually performed during the definition phase in order to support the product owner prioritize the business needs in collaboration with the team.

Figure 7.18 shows an example of how the User Story practice can be modeled using AMQuICk Essence constructs. First, the model explicits the main workproduct produced by the practice, i.e., the *Story Map* using the workproduct construct type. Then, it defines three main activities:

(P3AC1) Sort Stories: Sort stories by features, business goals, and/or EPICs.

(P3AC2) Prioritize Stories: Prioritize stories according to a specific factor (e.g., development priority or implementation sophistication).

(P3AC3) Define Minimum Valuable Product (MVP): The minimum viable product (MVP) is the minimum collection of user stories that is requirement to satisfy the customer.

---

[10]`https://www.agilealliance.org/glossary/storymap`

Object Diagram Illustrations [🔲 Story Mapping]

Associations with other Practices

PA1 : PracticeAssociation
sourcePractice = P2.0.0
targetPractice = P3.0.0, P4.0.0
type = complementarity

PA2 : PracticeAssociation
sourcePractice = P5.0.0
targetPractice = P3.0.0
type = complementarity

P2.0.0 : Practice
name = "User Story"
time = 0
universe = 0

P5.0.0 : Practice
name = "Backlog Grooming"
time = 0
universe = 0

P3.0.0 : Practice
description = "Story mapping consists of ordering user stories along two
independent dimensions. The "map" arranges stories along the
horizontal axis in rough order of priority and along the vertical
axis, in order of implementation sophistication. "
name = "Story Mapping"
ownedElements = P3AC2, P3AC3, P3AC1
referredElements = WP3
time = 0
universe = 0

Owned and referred Activities and Workproducts

: refers

: owns

WP3 : WorkProduct
name = "User Story"

WP4 : WorkProduct
name = "User Stories Map"

P3AC1 : Activity
briefDescription = "Sort
stories by features, business
goals, and/or EPICs"
name = "Sort Stories"

P3AC2 : Activity
description = "Prioritize stories
according to a specific factor
(e.g., development priority)"
name = "Prioritize stories"

P3AC3 : Activity
name = "Define MVP"

Figure 7.18.: Practice Authoring Example - Story Mapping

In addition to the association expressing complementarity between the user story and the story mapping practices (PA1), the example describes the following association:

> (PA2) The Story Mapping practice complements the *Backlog Grooming* [11] practice which consists of the ongoing or regular basis refinement of the product backlog.

Relative Estimation

Figure 7.19 shows an example of how the *Relative Estimation* practice can be instantiated.

The rational of Relative Estimation[12] is that people are not good at estimating time-based activities. Indeed, the more complex the activity is, the more difficult will be its accurate estimation in actual time units (weeks, days and hours). Therefore, agile methods recommend to estimate in relative complexity, i.e., by judging how big or complex tasks (typically user stories) are with respect to each other.

The unit of complexity that is used in this kind of estimation is irrelevant. Usually, teams user non-numerical scales (or groups) like t-shirt sizing or point estimates, typically known as *Story Points* or *Nebulous Units of Time (NUTs)*[13]:

> (M2) Story Point: this measure constitute the most common way to estimate the overall effort of implementing user stories. Usually, Fibonacci-like scales are used as point estimates, e.g., 1, 2, 3, 5, 8, 13, 21, 40, 80, 120 and infinite. The idea is that the larger the story is, the more uncertainty there is around it and the less accurate the estimate will be.

Usually, Relative Estimation relies on the following process:

> (P4AC1) Assign a relative story: The team starts by having an agreement on a reference story as one of the simplest user stories and some point value is assigned to it (e.g., 3 story points).

> (P4AC2) Compare the relative effort: The team members have a discussion regarding the effort required for implementing the user story in comparison with the reference story. They should consider its complexity, the required amount of work and the potential risks, gaps or issues that they may encounter.

---

[11]https://www.agilealliance.org/glossary/backlog-grooming
[12]https://www.agilealliance.org/glossary/relative-estimation
[13]https://www.agilealliance.org/glossary/points-estimates-in/

(P4AC3) Assign a story point: All stories are estimated according to the reference story. For instance, the team might decide that the reference story S1 is a 5 point estimates. Then, when estimating a specific story S2, it can be compared to S1: is it twice as bigger or more complex? Is it half as complex?

The Relative Estimation practice may be performed in several ways to make it as fun and lightweight as possible. For example, teams may use techniques such as *Silent Grouping*[14] or *Planning Poker*[15].

In the default instantiation of the practice, we associate it with the Planning Poker *resource* which is of type *tool*:

(RES4) Planning Poker: A playful technique to relative estimation that is to be used during the iteration planning. Every team member posses a cards play carrying numerical values corresponding to story points. After the the team discusses and fully understands the user story, each person silently picks an estimate for the story. The cards are shown when every one has decided about an adequate story point. The team then discusses the different estimation, specially when there is an estimation gap and decides about the final estimate to assign to the story.

7.6 Illustration: Intel Shannon Case Study

In order to evaluate the usefulness of AMQuICk Essence on a realistic case and to get insights on how it should be refined, we selected an industrial case study (and used it as a source of secondary data) based on the following criteria:

- Reliability: Does the method rely on a rigorous methodology for data collection ?

- Reproducibility: Does the study report generalizable knowledge or structured lessons that other practitioners can exploit?

- Focus on customization: Does the study report an industrial experience where agile methods are customized (i.e., the study should not just be a story telling on agile implementation)?

---

[14]https://systemagility.com/2011/05/22/using-silent-grouping-to-size-user-stories/
[15]https://www.agilealliance.org/glossary/poker/
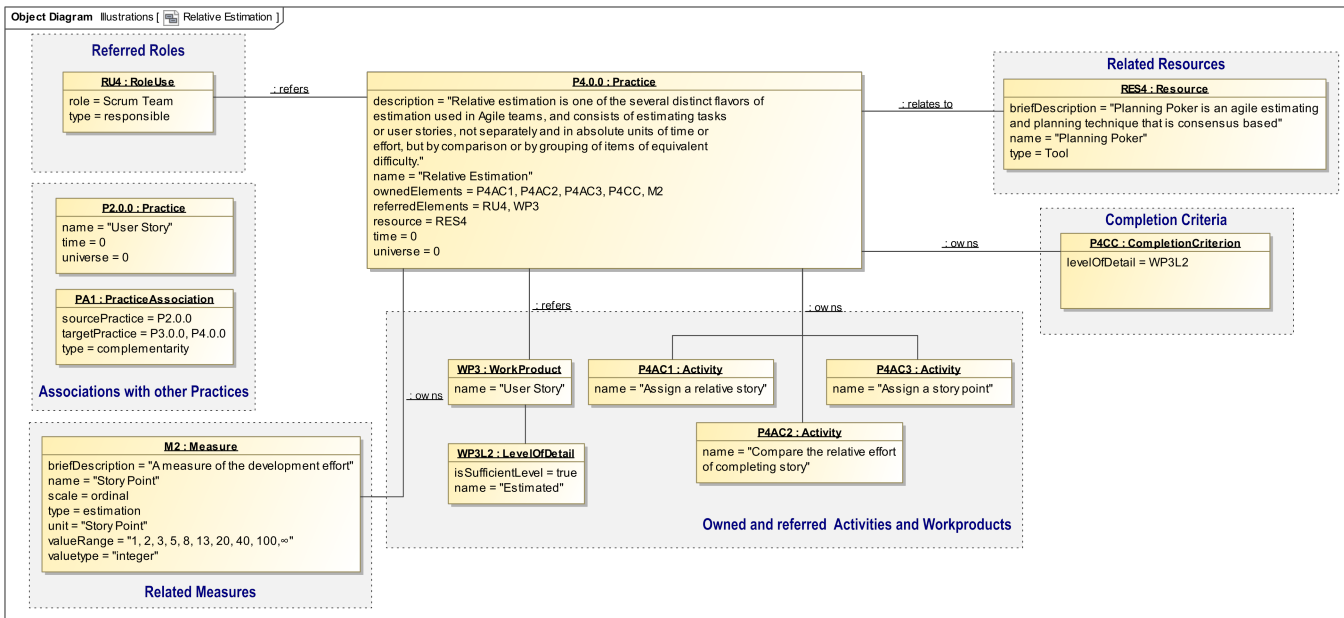
Figure 7.19.: Practice Authoring Example - Relative Estimation

- Quality of situation description: Does the study report the characteristics of the studied context? Does it highlight how these characteristics particularly impacted the enactment of practices?

The selected study reports the implementation of agile methods at Intel Shannon in Ireland [Fitzgerald et al., 2006]. More precisely, it describes how the company embraced and customized practices from XP and Scrum, to meet their specific challenges.

The study follows an interpretive and exploratory research methodology. Data on the usage of XP and Scrum are collected through series of formal and informal personal interviews, e-mails surveys, post-scrum workshops and by inspecting source code and queries of change management systems (e.g., non-commented lines of code, code defect density, release dates, etc.).

The 2-year longitudinal study was conducted on two medium-sized projects at Intel Shannon in Ireland, each aiming at the software development of one processor type. Small, experienced and distributed teams, with an average of 3 years of agile experience, worked in both projects.

The method adapted to the organization specific context (i.e., critical domain, distributed development, rapid time-to-market, etc.) consists in a combination of XP and Scrum with some configurations and home-made practices. The study describes what practices were retained, discarded or customized. It highlights the benefits of agile methods customization and reinforce the view point that agile methods, like any software method, should be tailored if they are to achieve optimum effects [Conboy and Fitzgerald, 2010].

7.6.1 XP Usage

The case study report a pragmatic implementation of XP, i.e., only a selection of practices have been implemented and those were carefully monitored to the specific context of Intel Shannon. Figure 7.20 summarizes how XP was used in the context of Intel Shannon.

Seven out of the twelve XP practices (see Section 2.2.3.1) were configured and implemented:

(XP.P1) *Pair Programming*: the practice was applied for the development of critical components and by pairs of similar profiles. It appears to be not suitable for simple tasks and for maintenance. Guidelines were formulated and shared across the teams.

Object Diagram Intel Shannon [🖫 Intel_XP configuration]

Practices that were selected / customized

Intel.XP.P5.0 : Practice
briefDescription = "Collective ownership (confined to single teams not across teams)"
name = "Collective ownership"
time = 0   Default Version
universe = 1   Intel Shannon Repository

: PracticeAssociation
sourcePractice = Intel.XP.P1.0
targetPractice = Intel.XP.P5.0
type = complementarity

Intel.XP.P1.0 : Practice
briefDescription = "Default version of pair programming"
description = "Development pairs for all features"
name = "Pair programming"
time = 0
universe = 1

: PracticeAssociation
sourcePractice = Intel.XP.P1.0
targetPractice = Intel.XP.P1.1
type = configuration

Intel.XP.P1.1 : Practice
briefDescription = "Configured version of pair programming"
description = "Development pairs for small/well understood problems"
name = "Pair programming"
resource = P1.1.R1
time = 1
universe = 1

Owned Elements

P1.1.R1 : Resource
name = "Intel PP Lessons Learned"
type = Internal Document

: relates to

Intel.XP.P3.0 : Practice
description = "Systematic refactoring at the end of iterations"
name = "Refactoring"
resource = P3.0.R1
time = 0
universe = 1

: relates to

P3.0.R1 : Resource
name = "Intel Refactoring patterns"
type = Internal Document

: PracticeAssociation
sourcePractice = Intel.XP.P2.0
targetPractice = Intel.XP.P3.0,
Intel.XP.P4.0
type = complementarity

Intel.XP.P2.0 : Practice
description = "Design on white board for each block of code and documented later"
name = "Simple Design"
time = 0
universe = 1

: Measure
briefDescription = "Baseline schedule dates vs actual dates"
name = "Accuracy to Dev. Schedule"
scale = ordinal
type = measurement
unit = "days"

: owns

Intel.XP.P4.0 : Practice
name = "Coding Standards"
resource = P4.0.R1
time = 0
universe = 0

: relates to

P4.0.R1 : Resource
briefDescription = "A document for C/C++ coding and code inspection"
name = "Intel C-coding Standard"
type = Internal Document

Intel.XP.P6.0 : Practice
briefDescription = "Unit Testing for some features
Regression tests nightly (using a test suite)
Daily smoke test (external equipment)
in the weeks of release"
name = "Testing"
ownedElements = P6.0.M1
time = 0
universe = 1

: owns

P6.0.M1 : Measure
briefDescription = "Defect count/ size of the release"
name = "Defect Density per release"
scale = ordinal
type = measurement
unit = "KLOC"
valueType = "integer"

Practices that were replaced

XP.P7.0 : Practice
description = "a co-located customer representative fully commited to work with the team"
name = "On-site Customer"
time = 0
universe = 0   Default Repository

: PracticeAssociation
sourcePractice = XP.P7.0
targetPractice = Intel.XP.P7.0
type = configuration

Intel.XP.P7.0 : Practice
description = "A domain expert available at some key stages of the iteration"
name = "Customer Proxy"
time = 0
universe = 1

XP.P8.0.0 : Practice
briefDescription = "Frequent and small releases (up to 4 week)"
name = "Small Release"
time = 0
universe = 0

: PracticeAssociation
sourcePractice = XP.P8.0.0
targetPractice = Intel.XP.P8.0
type = configuration

Intel.XP.P8.0 : Practice
briefDescription = "Initiation: No release until silicon availability
Intermediary phase: Minor Release every 4 to 6 weeks
Advance phase: Major Releases every two quarters "
name = "Release practice"
ownedElements =
time = 0
universe = 1

Figure 7.20.: The use and configuration of XP at Intel Shannon (overview)

(XP.P2) *Simple Design / Quick Design Sessions*: Design was done on white-boards before each block of code is written and then documented. Consequently, the design documents emerge in parallel with the development. The practice was as complementary with refactoring. A design pattern standard was progressively designed and used to guide teams keep a simple design and help them refactor their code.

(XP.P3) *Refactoring*: A practice that was considered as complementary with design activities. It was done systematically and at early stages.

(XP.P4) *Coding Standards*: A practice that was adapted even before the implementation of XP. A C-coding standard is frequently used by teams.

(XP.P5) *Collective Ownership*: Was practiced only to single teams and not across teams. The pair programming practice was found highly beneficial for code ownership and therefore considered as complementary.

(XP.P6) *Unit Testing*: Implemented as part of a regression test suite and all tests were run nightly. More complete tests (integration and smoke tests) are run daily during the last week of the release.

(XP.P7) *Continuous Integration*: The practice was only applied for single components developed during the iterations. Indeed, it was found to be difficult to implement for the whole project regarding its complexity and since external test equipments are necessary.

Three of the remaining XP practices were replaced by alternative practices:

(XP.P8) *On-site Customer*: Since the Intel Shannon teams work at early stages of the product development, it was not really feasible to have customers involved. Besides, the product owners which are located at the US cannot be available on daily basis. The product marketing group sometimes act as a customer proxy (they prioritize the product features based on potential revenue).

(XP.P9) *Small Releases*: The practice is not feasible regarding the complexity of the projects and since the release depends on the availability of Silicon. When the Silicon was available, the teams have a minor release every 4 to 6 weeks and major releases every two quarters.

(XP.P10) *Planning Game*: The practice was not retained since it is equivalent to the Scrum Sprint planning practice

Finally, two practices were completely discarded: *the 40-h week* and *Metaphor*. The first was found inadequate. Indeed, working hours should often be extended because of the discrepancy of time zones between Europe (between the US and Ireland). Metaphor was not used because it was found unclear and not suitable for all design situations.

7.6.2 Scrum Usage

The study reports that most of the Intel Shannon teams use Scrum (see Section 2.2.3.2). More precisely, the following practices were selected and configured:

(Scrum.P1) *Daily Meeting*: The practice was found particularly positive for visibility. Several configurations were introduced by the different teams. In its default version, the team members meet around a task post-it board. Team members arrive to the meeting with post-its for their daily tasks and discuss them with their team mates. When needed, tasks are decomposed (in Scrum this is normally done during the Sprint planning). In another configuration of the practice, the team systematically documented their meeting in the project repository. The practice was also implemented by a distributed team which combined video conferencing and an online post-it board.

(Scrum.P2) *Sprint Planning*: This practice was selected in replacement of the XP planning game. It was configured so that for each project, a two stages planning is implemented: one at the start of the project and one at the start of every Sprint. During the first planning stage (that includes team leads and some team representatives), the project is decomposed into a functional series of sprints. The overall features to be developed during the project is identified. A high level estimate of features is allocated (in working days) and the features are distributed across a number of Sprint. Some teams used the Delphi Technique [Linstone and Turoff, 1975] for estimating the features.

(Scrum.P3) *Sprint*: In Scrum, it is recommended to have time-boxed Sprints. However, in Intel Shannon this was found unsuitable because of the criticality of projects. Rather, a scope is fixed and when all the features have been developed a minor version is released. A

major version is released when all features are fully integrated, tested (regression testing) and deployed on processors.

(SCRUM.P4) *Sprint Closure*: The practice was modified to include wrap-up sessions where the teams report their lessons learned, discuss the work done and the extra tasks included, etc.
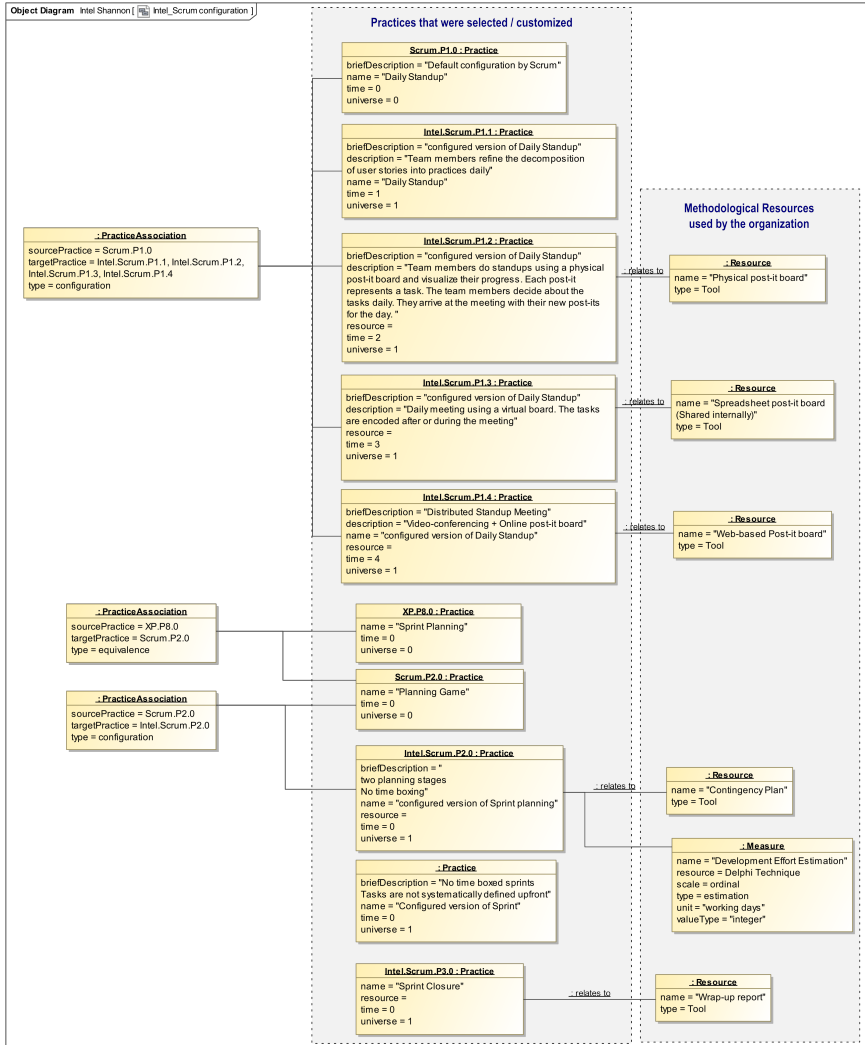
Figure 7.21.: The use and configuration of Scrum at Intel Shannon (overview)

7.6.3 Reflection on AMQuICk Essence Refinement

In the previous two sections, we were able to structure the method implemented at Intel Shannon using AMQuICk Essence elements (practices, resources, tools, measures, etc.) and captured the information regarding its evolution overtime (the different practice configurations that were experienced). However, some essential knowledge regarding the implementation and configuration of practices couldn't be represented.
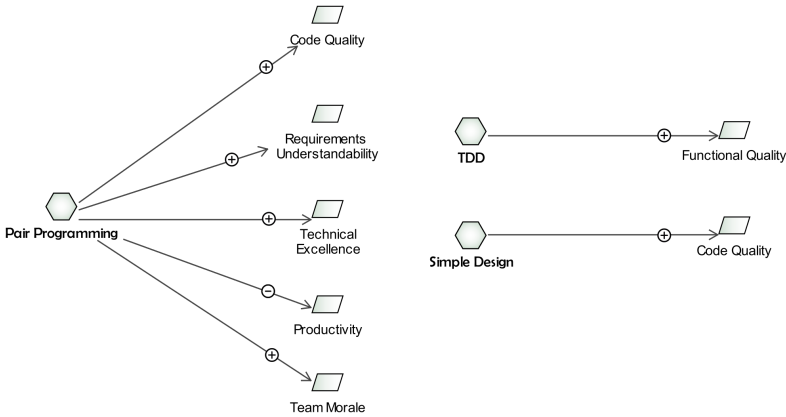
Figure 7.22.: Examples of reported relationships between adopted practices and quality goals

First, using the current version of AMQuICk Essence, we couldn't capture essential information regarding the benefits of the configuration of practices. Indeed, the study discusses the impact (assumed and empirically validated) of the introduced practices on the process and product quality (see Figure 7.22) and reveals that practices achieve their benefits only in the presence of certain context characteristics.

For example, in the context of *junior developer profiles*, pair programming was found beneficial to *code defects reduction*. Moreover, in the context of *complex features*, the practice was reported as beneficial for *requirements understandability*, *team morale* and *technical excellence*. Finally, in the context of *experienced team members* and/or *simple and repetitive tasks* (such as maintenance), pair programming was found to have a negative impact on *productivity*. This kind of information could be essential for

guiding the customization of agile methods but it is still not provided in the current version of AMQuICk Essence.

The study also reports several practice configurations for specific situational contexts. For example, a version of daily meeting was proposed for *distributed teams* and another version was proposed for contexts where *documentation of the process is important*. Again, using the current version of the metamodel, it is not possible to capture the relationship between a practice configuration and the situational context for which it is intended.

Based on the aforementioned discussion, it is clear that the AMQuICk Essence core needs to be extended in order to support the customization guidance. Chapters 9, 10 and 11 present case studies that helped us explore what kind of actual refinements were necessary and how to implement them in AMQuICk. The implemented refinements are presented in Chapter 12.

7.7 Summary

In Chapter 6, we have argued that a situational method engineering approach can be used in the context of agile software development provided a suitable metamodel and a lightweight method engineering approach.

AMQuICk Essence is a method engineering metamodel that allows a practice-based authoring of software development methods. Moreover, it describes the essential elements necessary to implement the AMQuICk framework: the practices repository and the customization knowledge base.

In this chapter, we described the foundation and practice content packages of the metamodel. At this level of the research, we focused on eliciting the essential concepts required for authoring agile practices building blocks and practice configurations. Several design choices have been decided among which:

- Practice-based method modeling: AMQuICk Essence characterizes methods as the association of a set of coherent practices. As explained in Section 6.1.2, this allows a coarse granularity level of method models and makes it easier to suggest configurations. Indeed, practices constitute the unique variability point of method models. Consequently, customization decisions (see Chapter 12) would provide high-level guidance which does not impede the team flexibility.

- Practice and methods evolution: Agile methods encourage team members to take responsibility for the evolution of their own practices.

Therefore, AMQuICk Essence introduced elements for capturing different practice configurations (the *time* and *universe* attributes and the *PracticeAssociation* element of type *configuration*). By doing so, we may track the evolution of practice implementations in one or several organizations.

The current version of the metamodel does not yet provide means to capture the actual evidence about the success or failure of practices under different project situations nor describes means to capture customization decisions. The necessary refinements are discussed in Chapter 12.

In the next chapter, we demonstrate how the current version of metamodel can be used to design the AMQuICk practices repository.

Chapter 8

AMQuICk Repository of Practices

Earlier in this part of the thesis, we discussed the existence of method engineering frameworks (and their descriptive languages or metamodels) which rely on the idea of picking up and assembling building blocks from various software methods to constitute a situational method. We then designed the AMQuICk Essence metamodel which is intended to document agile building blocks.

The approach impacts the way that agile methods customization is performed. First, practice building blocks have to be produced and maintained in an efficient way. Besides, the emphasis of agile methods on collaboration and communication implies that the practices and their constitutive blocks should be easily accessible and shared among the stakeholders. These aspects tend to increase the time and effort dedicated to the method customization and improvement. Therefore, various types of tools instantiating the metamodel should be provided to improve the usability of the approach.

This chapter provides an overview on the repository of agile practices and its practice cards modeler that may be supplied in order to ensure the usability and effectiveness of AMQuICk Essence. Sections 8.1 and 8.2 discuss the existing tool support dedicated to methodological elements authoring and discusses how to take advantage of them in the context of authoring AMQuICk practice models. Section 8.3 presents the Agilia web repository of practices that have been developed during the course of this research in order to improve the usability of the approach. Finally, Section 8.4 discusses how a dedicated graphic modeler of practice cards can be used to improve the usability of the repository.

8.1 CAME Tools and AMQuICk

As explained in [Kent, 2002], *"tooling is essential to maximize the benefits of having models, and to minimize the effort required to maintain them"*.

Computer Aided Method Engineering (CAME) tools are required essentially for authoring and storing method components and for making them available to those who are defining, implementing and managing methods in an organization.

Beside these basic editing functionalities, the support expected for such tooling range from the verification of the well-formedness of method models to the possibility to work easily on designing new method components. Moreover, such tools are intended to improve the process learning and to ensure a greater degree of independence of the team [Amescua et al., 2010].

Based on the method engineering frameworks presented in Section 6.2, a number of CAME tools including method composer tools, repositories of method fragments and method modelers have been proposed (see Section 6.2).

For instance, the Eclipse Process Framework (EPF)[1], a process engineering tool based on the SPEM specification, provides a method composer[2] and a number of libraries for some well-known methods such as Scrum, XP and OpenUP[3].

Similarly, the SEMAT Essence initiative[4], provides a set of development tools based on the Essence specification, among which the EssWork practice Workbench[5] for practices authoring and composition and the Essence Practice Library[6] which describes a number of agile essentials.

Research initiatives such as [Tran et al., 2009] and [Amescua et al., 2010] proposed extensions to existing method repositories, in order to integrate agile-specific fragments or developed their own repositories built upon non standardized process metamodels.

Such method repositories have the advantage of being more structured than some well-known industrial method bases such as the Agile alliance base[7] or the recently released Deloitte agile landscape map[8]. However, they still do not have much visibility among the agile community. [Esfahani and Yu, 2010] argues that the most likely reason for this is that complicated solutions for method engineering won't even be acknowledged by agile software companies.

---

[1] https://www.eclipse.org/epf/
[2] https://www.eclipse.org/epf/composer_dev_guide/dev_env.php
[3] http://www.eclipse.org/epf/downloads/praclib/praclib_downloads.php
[4] http://semat.org/
[5] https://www.ivarjacobson.com/esswork-practice-workbench
[6] https://practicelibrary.ivarjacobson.com/
[7] https://www.agilealliance.org/agile101/agile-glossary/
[8] http://blog.deloitte.com.au/navigating-the-agile-landscape/

Since AMQuICk relies on the Essence 1.0 specification, we considered the opportunity of reusing the EssWork Practice workbench (see Section 6.2.4), a tool for composing, authoring, collaborating and managing software development practices and methods based on the Essence specification.

However, it was found unsuitable because of the following:

- It relies on the Essence DSL and provides no way to extend it so we can model practices and practice cards using the newly introduced concepts.

- The workbench is not open source and the version actually available for practitioners only allows to visualize process fragments documented by the SEMAT initiative. When we tested the trial version, we found very little guidance on how to develop a method or practice from scratch.

- Like EPF composer, the Esswork practice workbench is a desktop-based software system tightly coupled to the Eclipse environment which makes it more difficult to scale, i.e., to make it usable by practitioners from different companies. During its testing, we experienced several technical problems such as crashing and loosing the library which is also commonly reported throught the EPF forum[9].

- It focuses on providing the team with means to track its progress in terms of *Alphas* 6.2.4 (a concept that was not retained for AMQuICk Essence since it is covered by the *Goal* construct type that we introduce later in Chapter 12).

- It does not include the actual evidence on the success or failure of practices under different project situations.

Given the aforementioned discussion, we designed a web repository of agile practices that instantiates the AMQuICk Essence specification (see Section 8.3). Unlike the published repositories presented earlier, our repository is intended to be extended by a knowledge base (see Chapter 12), i.e., the experience of other practitioners in using the different practices. This makes it more actionable since such knowledge helps teams decide on the appropriateness of their own method.

---

[9]`https://www.eclipse.org/forums/index.php/f/49/`

8.2 DSM Tools and AMQuICk

Since AMQuICk relies on a method engineering metamodel, more options were available for building the repository of agile practices and its practice cards modeler. These include metaCASE tools and Domain-Specific Modeling (DSM) frameworks [Kelly, 2004]. In the following, we argue why these tools were not retained.

MetaCASE tools are similar to CAME tools with the former focusing at the software level and the later at the method level. MetaCASE tools such as MetaEdit+ [Kelly et al., 1996] and MetaDone [Englebert and Heymans, 2007] are dedicated to the generation of language workbenches enhanced with graphical abilities on the basis of a provided metamodel. Using such tools is useful for a faster and easier generation of a graphic process modeler such as the Objecteering SPEM Modeler[10].

DSM coding frameworks such as the Eclipse Modeling Framework (EMF) and the Kevoree Modeling Framework (KMF) [Francois et al., 2014] include code generation facilities for building tools and other applications based on structured data models. They provide a powerful toolset for developers to model, structure, and reason about domain data. In our context, these frameworks may be useful for facilitating the authoring and exchange of AMQuICk domain-specific concepts with other applications (for example, method composers relying on SPEM). Exploiting KMF may also be considered for structuring, processing and analyzing the knowledge base.

However, in the current version of the framework, the reuse of DSM coding frameworks and DSM tools is found superfluous. Indeed, even though we argue the advantage of visualizing some parts of the method graphically (as graphical physical cards) for transparency and method improvement purposes (see Section 8.4), providing an agile team with a complete process model designed using a process modeler may be inconsistent with the agility characteristics (see Section 2.2.2). Therefore, supporting a whole workbench for graphically modeling the development method was never an objective of AMQuICk. Rather, the framework aims at providing a structured and lightweight representation of agile methods building blocks.

---

[10]`http://www.objecteering.com/free_addons_spem.php`

8.3 **Repository of Practices**

8.3.1 Objectives

The deployment of the AMQuICk framework requires not only the collection of contextual information (see Chapter 12) but also the documentation of comprehensive knowledge on agile practices.

Given the information available in the agile community and in the research base, AMQuICk intends to assist practitioners to determine adequate methods and practices with respect to their needs and preferences and with regards to the constraints available within a specific project.

Indeed, unlike the existing method engineering initiatives, AMQuICk suggests that organizations may take advantage of the multiple learned lessons documented in the research base or shared by practitioners in the active agile community. It assumes that when such learnings are captured, structured and disseminated (i.e., made accessible for practitioners from the same organization or across organizations), customization decisions are made easier.

The documentation of such knowledge in a structured repository can also be inspiring for project managers and method facilitators in order to come up with innovative practices.

Besides, it may provide teams with a greater autonomy in process learning and with an enhanced inter-teams communication regarding the success or failure of practices in certain circumstances.

Provided the aforementioned discussion, we designed a repository of agile practices and reusable practice components that we called Agilia with the objective to provide functionalities to maintain an up-to-date, structured and extensive knowledge on available agile practices and to relate them with the return of experiences of practitioners from various companies. Moreover, it intends to provide a user friendly access to this knowledge.

8.3.2 Overview

The Agilia repository constitutes a way to gather comprehensive knowledge on different available agile methods and practices. Relying on the open source Java Spring Boot framework[11], Hibernate and MySQL, the repository provides agile practitioners and experts with the following functionalities:

---

[11]`https://projects.spring.io/spring-boot/`

- creating/changing a practice content,

- creating new practices,

- configuring existing practices,

- browsing practices based on keywords and/or tags,

- publishing selected practices,

- reporting experiences on the usage of practices in specific contexts.

Compared to existing repositories of practices, the documentation of experience is the most valuable feature of our repository. Indeed, as explained in Chapter 12, the repository is enhanced with the content stored in the knowledge base, i.e., with the recommendations documented by practitioners. These can be suggested by novice practitioners as well as agile experts.

As shown in Figure 8.1, the repository provides a user interface to edit and consult the content of the online database of practices. A quick navigation through associated concepts is available to any practitioner willing to browse the documented practices. Authenticated users may document their learned lessons and expert users have more privileges in documenting new practices and recommendations.

Since Agilia is the default repository of AMQuICk, its content is inherited by any other organizational repository. It is intended to be continuously expanded and modified based on practitioners feedback. It can also be useful in training new practitioners.

The data it contains is structured according to AMQuICk Essence models. It also provides support for the interrogation of the database by the AMQuICk practice cards modeler (see Section 8.4). An overview of some practices documented in Agilia is provided in Table 8.1.

8.3.3 Architecture

From an architectural point of view, the key aspect of the repository resides in the way its conceptual schema has been designed. Many schema models would have been valid for designing a repository for AMQuICk. However, ensuring the compatibility with AMQuICk Essence is essential. Therefore, we designed the conceptual model for Agilia as a series of horizontal transformation (i.e., the generation of a target model from a source model while conserving the same level of detail) of AMQuICk Essence [Mens and Van Gorp, 2005] [Vanderose et al., 2012] [Vanderose and Habra, 2011].
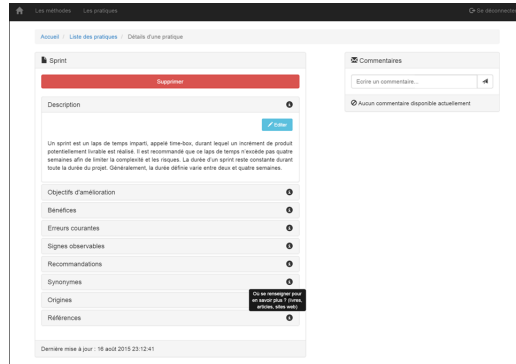
Figure 8.1.: Agilia Repository

As a result of this transformation, several elements from the metamodel have been adapted. For instance, some attributes were modified and new ones were added for practical reasons (e.g., created_by and last_updated_by were added in order to track the practitioners involved in authoring the building blocks). Also, many-to-many cardinalities were instantiated as intermediary associations:

- Practice_Method: instance of the "*owns*" directed relationship between the Method and Practice construct types

- Workproduct_Practice: instance of the "*owns*" directed relationship between the Workproduct and Practice construct types

- Metric_Practice: instance of the "*owns*" directed relationship between the Workproduct and Practice construct types

- Recommendation_Goal: instance of the "*supports*" directed relationship between the Recommendation and Goal construct types (discussed later in Chapter 12)

Another adaptation consists of the definition of 3 instances for the Resource construct type:

- Guideline: A resource associated with a method or a practice to provide external guidance that may be of the following types: book, research paper or practitioner return of experience (typically blog posts).

- Pitfall: A common pitfall (misapplication) of a practice documented by an agile expert

- Benefit: A common benefit of a practice documented by an agile expert. Usually it is expressed in terms of process goals, e.g., increased productivity, reduced development cost or reduced technical debt.

Figure 8.2 shows the overall data schema of Agilia. The code, data modeling and binding of this version of the repository were encoded from scratch. A good alternative would have been to rely on DSM frameworks (provided that they are compatible with web-based applications) which include code generation facilities for building tools and other applications based on structured data models (see Section 8.2).

8.4 Practice Modeler

8.4.1 Objectives

We argue that a more user-friendly way to exploit AMQuICk Essence models and to improve their usability is to use the card metaphor to graphically visualize the practices implemented by the team.

This idea is inspired by the Essence specification which provides users with graphical cards to visualize the most important aspects of the language with the purpose of tracking the team progress in different areas of concerns. In AMQuICk, we use this graphical facilitation technique for different purposes:

- assisting the practitioners in the improvement of their way of working (see Chapter 13), and

- contributing to the construction of a shared mental model (see Section 5.2) of the implemented method and the customization decisions made by the team.

In order to author practice cards, a modeling tool enhanced with the graphical syntax of AMQuICk Essence building blocks is provided and integrated to the repository of agile practices. This modeler aims at providing an improved usability for practice authoring and an easier navigation within practice models.

8.4.2 Overview

A practice card presents a succinct summary of the most important things practitioners need to know about a particular agile practice. It takes the

Table 8.1.: Overview of some stored practices

Practice	Brief Description	Equivalent Practices
Domain Walk-throught	A practice introduced by FDD which consists on creating domain area models during one or many collaborative activities which bring together domain experts, business representatives, architects, analysts, developers and any other involved stakeholder.	Collaborative Domain modeling
Dedicated Customer Representative	A practice that recommends the inclusion of an actual customer representative or user inside the team, available full-time to answer the questions	On-site Customer
Model Storming	A just in time modeling practice recommended by the agile modeling method. The practice aims at assisting developers drive quick design sessions.	Stand-up Design
Acceptance Test Driven Development	A development practice relying on the close collaboration between customers, developers and testers to produce acceptance tests (from the users point of view) prior to the development of the features. It may be automated or not.	Behavioral Driven Dev., Specification by Example
Test Driven Development	A practice where the development is systematically conducted through three interlaced steps: writing unit tests, coding and refactoring.	Test-first
Coding Standards	A practice that recommends to maintain a formalized set of rules and practices that developers can adhere to when writing code in order to ensure that the produced code is easily understandable, maintainable and extensible.	
User Story	A practice describing how work is divided into small functional units excepted to yield, once implemented, a contribution to the overall value of the product.	
Story Mapping	A practice that consists of ordering user stories along two independent dimensions. The "map" arranges stories along the horizontal axis in rough order of priority and along the vertical axis, in order of implementation sophistication.	
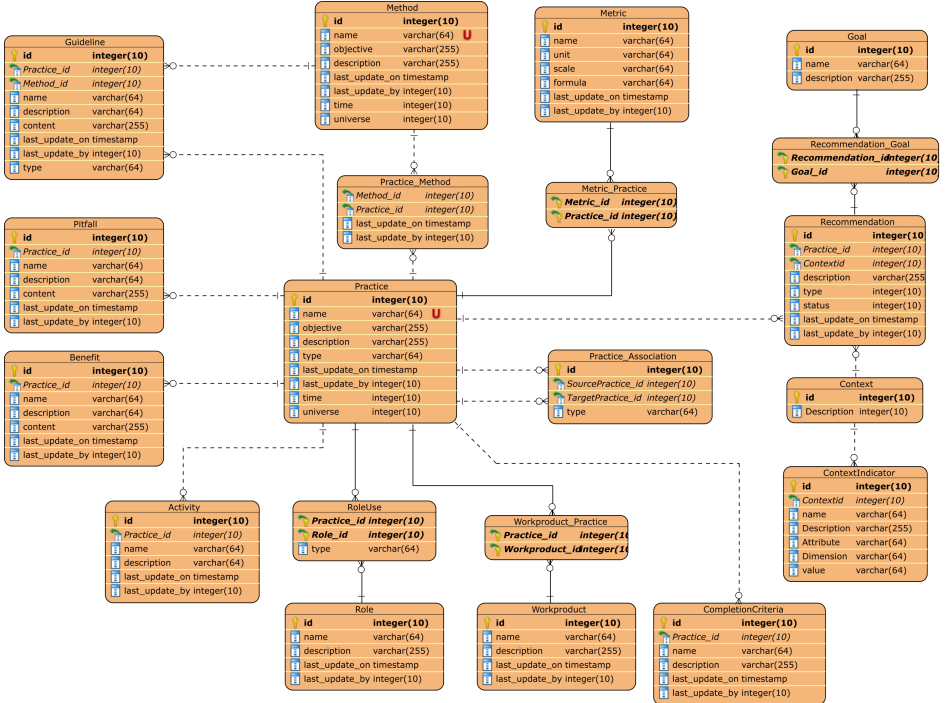
Figure 8.2.: Data Schema of AMQuIck default repository

form of a tangible and actionable card that may be printed and visualized in the team working place. Practice cards are intended to be used during the improvement and customization workshops (see Section 5.3.2 and Chapter **??**) as hands-on to drive the discussion between team members. Indeed, their handy form makes them easy to put on a table, stick on a board or an agile learning wall, handle them and reason about [Essence, 2015].

The graphical model we propose to complement the AMQuICk repository of practices is developed using Draw2d Javascript[12], a HTML5 Javascript library for the visualization and interaction with diagrams and graphs. The library was chosen because of the rich diagramming features it provides, the availability of data binding features and the possibility to export the modeled features in different formats (JSON, SVG, PNG, etc.). The choice of Javascript as a programming language is due to its high portability, especially for web-based applications and mobile devices.

The modeler includes the following features (see Figure 8.3):

- a toolbar from which elements can be dragged and dropped to add practice building blocks,

- a central canvas providing the overview on the practice model, and

- a top menu from which practice models can be loaded, registered and which contains a set of utility features for selecting and (un)grouping elements.

Since practice cards only contain an overview of the practice building blocks, a more detailed view on the modeler is ensured using dedicated property views that allow to edit the component attributes. Draw2d allows to easily read and write files to different file storage backends using a simple API. We used this file storage abilities to persist the data of the modeler to JSON files. However an integration with the repository persistence layer is still required.

As shown in Table 8.2, AMQuICk relies on a graphical notation that intends to improve the readability of practice cards. This graphical syntax has been instantiated using the icon attribute (an attribute of the BasicElement and GroupElement construct types).
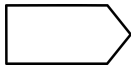
Figure 8.4 shows an example of practice authoring.

---

[12]http://www.draw2d.org/draw2d/

Table 8.2.: Graphical Syntax of core AMQuICk Essence Element

Entity	Symbol
Method	
Practice	
Activity	
Workproduct	
Role	
Competency	
Goal	
Resource	
Metric	

8.5 Summary

In this chapter, we designed the AMQuICk repository for browsing and authoring agile practices and the actual benefits it would provide to the framework users. We also designed a practice modeler to improve the usability of AMQuICk Essence models.

So far, the tooling developed to support AMQuICk implementation is comprised of the repository of agile practices, the practice modeler and the
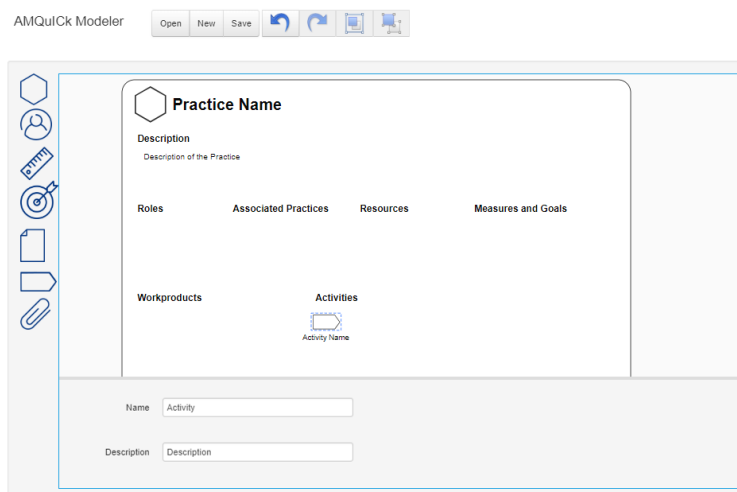
Figure 8.3.: AMQuICk practice modeler - Overview

knowledge database which documents a set of practitioners experiences and recommendations (see Chapter 12). Each of these tools focus on a specific aspect of the AMQuICk framework. Integrating them would provide a more consistent support of the overall methodology and therefore it is part of our future perspectives.

Since the repository of agile practices and the customization knowledge base already exchange compatible data (SQL data), this effort should focus on the integration between the practice modeler and the repository of practices. More precisely, the repository should be refined to provide a web service for reading and writing practice models as JSON files, thus allowing the AMQuICk model editors to consult and integrate knowledge retrieved through these web services and to feed the repository starting from the modeler. Moreover, using these web services it would be easier to integrate the AMQuICk knowledge with external methodological repositories which rely on a different structures (e.g., on the TOGAF standard[13], SPEM or any other method modeling specification). Using the KMF framework to persist AMQuICk Essence models in JSON-like files (in noSQL database) is also a good alternative to facilitate the tools integration.

In Chapter 12, we discuss the population of the knowledge base with the knowledge regarding the actual benefits demonstrated by agile method practices in specific contexts based on the return of experiences of practitioners.

---

[13]`http://www.opengroup.org/subjectareas/enterprise/togaf`
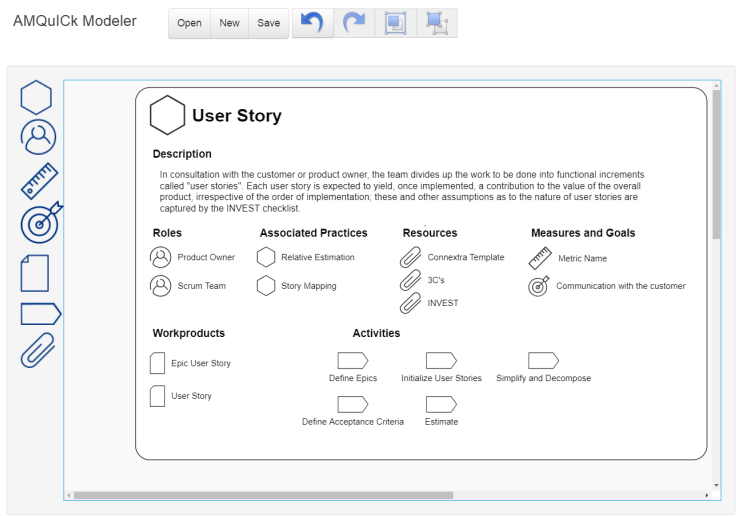
Figure 8.4.: AMQuICk practice modeler - An example of practice authoring

This knowledge allows to gain a realistic view on what each agile practice can bring and in which context and informs about the necessary requisites for the deployment of practices.

Part IV.

Context Study Perspective

In order to understand the customization challenges that agile practitioners may encounter, we conducted in the part of the dissertation three exploratory case studies. Each is discussed in one chapter as follows:

Chapter 9 studies a transformation experience in an IT department of the Public Walloon Service (SPW) in Belgium.

Chapter 10 investigates the contextual challenges that practitioners face when developing e-government services using an agile software development method.

Chapter 11 investigates how the European and Asian cultural backgrounds may impact agile methods implementation.

Chapter 9

SPW Case Study

Challenges of Implementing Agile in a Transitioning Context

Transitioning from disciplined to agile software development is a wide and complex change that may be challenged by several aspects of the organization (e.g., its structure, culture, management practices, produced artifacts, technologies in use, etc). In this chapter, we argue that in order to succeed in this transition, it's crucial to understand the organizational and project's context and to design custom methods accordingly.

As an exploratory research, the chapter studies a transformation experience in an IT department of the Public Walloon Service (SPW) in Belgium. It tends to show that the transition to agility cannot be accomplished only at the team-level without taking into account the overall context of the organization. A context-specific method must carefully align the team level context and requirements and the organizational level context.

In Section 9.1, we review some previous work that had investigated the influence of the organizational context and provide an overview on the background of the case study. In Section 9.2, we detail the research design that we applied to determine and validate the identified challenges. The challenges that emerged from the organized meetings are contextually explained and presented in Section 9.3. The results validity is then discussed in Section 9.4. Finally, in Section 9.5 we synthesize the lessons learned from this case study and explain how it contributes to the thesis.

9.1 Background

It has been acknowledged by previous studies [Fitzgerald et al., 2006, 2003] that the organizational characteristics influence the implementation of agile methods (see Chapter 3). In fact, one agile method may lead to success

when applied in a particular organization and to failure in a different settings. Therefore, understanding the organizational context is crucial to minimize the adoption risks and to make the right customization choices.

Several studies in the literature have identified the organizational context as a dimension that potentially affects the implementation of development methods [Robinson and Sharp, 2005; Siakas and Siakas, 2007; Tolfo and Wazlawick, 2008; Boehm and Turner, 2003]. They all clearly show the importance of considering the organizational context which they conceptualize in a number of different ways [Iivari and Iivari, 2011].

In this chapter, we investigate the question (i.e., the impact of the organizational context) in the IT department (D443) of the DGO3 direction[1], one of the departments of the "Public Service of Wallonia (SPW)". The DGO3 is a large organization of ~2,300 employees that manages the material and issues related to agriculture, natural resources and environment in Wallonia.

The D443 is a medium-sized IT department (84 people at the time the study was conducted). It is in charge of about a hundred software products and is composed of 5 services : Development, Development Support, Security, Exploitation and IT Help-desk. The scope of this case study is the Development service. More precisely we interviewed and collaborated with practitioners from the following business units : Project Management (PM), solutions ARchitecture (AR), Quality Assurance (QA), DEVelopment (DEV), Functional Analysis (FA) and Software Maintenance (SM) (see Annex B for more details).

The main focus of these units is on the development of business applications for the Walloon Payment Agency which heavily depends on the agricultural policy of the European Union. At the time the research was initiated, the department was involved in the development of 15 projects among which the project IDEES, the pilot project we selected to study.

The desire to adopt an Agile approach came from the team working on IDEES. The team members, seeking to improve their own working environment, began implementing some Scrum practices in 2013 in their group comprised of one business project manager, one technical project manager, 8 developers and 3 analysts. The group had no special skills in agile software development. They only read some agile literature before starting implementing Scrum.

In this context, 6 months after the initiation of the first agile experience in the department, we conducted this case study with the purpose of diagnosing the current process, identifying and contextually explaining the challenges

---

[1]The General Operational Direction for Agriculture, natural Resources and Environment: `http://environnement.wallonie.be/administration/dgo3.htm`

encountered in implementing Scrum and finally identifying in collaboration with the practitioners, some customization opportunities.

The following section further details the methodology used in this case study.

9.2 **Methodology**

This case study is exploratory and mainly qualitative. The organization was selected based on convenience sampling. Indeed, we were contacted by the project management unit of the IT service for training purposes. After preliminary meetings and regarding a common interest to methods customization, a research collaboration has been defined to further study the customization of Scrum.

The overall transition project follows the roadmap depicted in Figure 9.1. This roadmap was designed in collaboration with the projects' portfolio manager. It sets the premises for the AMQuICk framework lifecycle.
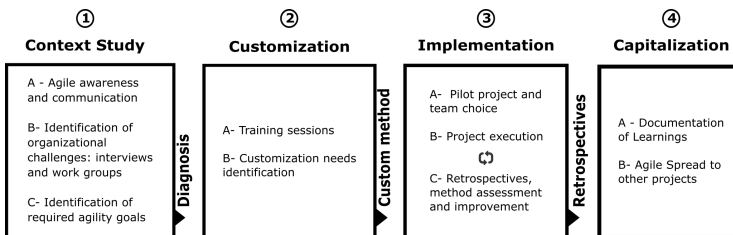
Figure 9.1.: Transition roadmap

Two transformation modes are usually reported by agile practitioners : big-bang and step-by-step. The big-bang transition mode consists of applying agile to all teams and all projects, as quickly as possible. This mode takes few time and raises less resistance to change but requires a great investment in terms of time and training. The step-by-step mode means that agile is applied in a progressive way. First, agile values are applied in a pilot team and a pilot project (not too risky but fairly representative of the organization reality). Then, the experience is capitalized and spread to other projects transversally or progressively.

Regarding the complexity of the D443 context, a progressive transformation strategy that employs frequent inspect-and-adapt cycles was advocated.

The following sections further detail the methodology followed for this case study. More details can be found in the research collaboration internal report [AYED, 2013].

9.2.1 Objectives

The main focus of the case study is to investigate the contextual challenges encountered by the organization during its transition from a traditional waterfall model to a more agile software development approach. More precisely, its intends to:

1. Study the organizational context and identify the helpful and discontinuing/challenging context factors at the organizational level,

2. Study the transition at in pilot project and identify the helpful and discontinuing/challenging context factors at the project level,

3. In collaboration with the pilot project-team practitioners, identify areas of improvement and customization opportunities.

9.2.2 Data Collection

A methodological triangulation is applied for gathering data [Denzin, 2017]. The data of the case study is collected from interviews, questionnaires and a project level retrospective.

Interviews

It has been largely acknowledged that it's crucial to spend significant time and energy building urgency and convincing people that change is beneficial [Satir and Banmen, 1991]. Moreover, it is recommended to start the transition by assessing the current process and defining the expected benefits [Tomasini and Kearns, 2012].

Therefore, after a few meetings with the IT department director and the projects portfolio manager and as part of the first transitioning step, we organized 6 interviews with unit representatives (one interview per business unit). At first, the participants introduce themselves and the researcher introduces the aim of the interview. Then, the participants are asked about their actual process (the unit responsibilities, personnel size, geographical distribution, workflow, etc.) and its strengths and weaknesses regarding to agility values. Finally, they are questioned about their expectations, i.e., the

improvement opportunities from an agile perspective. We used the same semi-structured interview guide for all the interview sessions.

The 23 practitioners that we interviewed are distributed as follows: Maintenance unit (MA): 2, Architecture unit (AR) : 3, Quality Assurance unit (QA) : 3, Development unit (DEV) :3, Project management unit (PM) : 7 and finally Functional Analysis unit (FA): 3 + 1 external consultant. Only 7 out of the 23 practitioners have already experienced agile methods: 4 developers, 2 project managers and 1 analyst.

Questionnaires

Two specific questionnaires were designed in this study. The first questionnaire[2] (see Table 9.1) is organized into 4 sections and aims at analyzing the current process in terms of agility. The second questionnaire[3] (see Table 9.2) is designed to get more insights regarding the context and the practitioners knowledge of Scrum practices and to identify the desired and/or applicable agile practices.

The two questionnaires were e-mailed to 64 persons from all the units. 47 responded to the survey, which gives a participation rate of 74%.

Project Retrospective

After a 6 months experience in implementing Scrum, we organized a retrospective meeting with the selected pilot project-team, the IDEES project. We animated the retrospective using the *affinity diagram* facilitation method. This method consists of the following steps : Record each idea on a card (post-it), look for ideas that seem to be related and finally sort cards into groups until all cards have been used [Pyzdek and Keller, 2014; Kawakita, 1975]. Causal relationships between groups are also indicated on the diagram.

At first, participants were asked to think about the following questions : "What worked well?" (opportunities for the transition) and "What did not work or what can be improved?" (challenges for the transition). Then, each practitioner silently drew his reflexions on post-its. We recommended to write down a unique idea/point per post-it and to reuse the adequate color to differentiate negative and positive points and improvement opportunities. Two affinity diagrams were built. These can be viewed in [AYED, 2013]. Table 9.4 shows the transcription of the identified groups of challenges.

---

[2]`https://ayedhajer.wufoo.com/forms/znj8luh18v8dyw/`
[3]`https://ayedhajer.wufoo.com/forms/q11mi9p60lhglme/`

The 15 team members all participated to the retrospective. An agile facilitator was also invited to join to bring insights and recommendations regarding possibly interesting practices. The latter only contributed to the discussion phase.

9.2.3 Data Analysis

To analyze the gathered data, we followed a theory triangulation strategy, i.e., more than one theoretical scheme is used for the interpretation of data: an organizational agility model [Boehm and Turner, 2003], a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis of the identified challenges [Jackson et al., 2003], and an affinity diagram [Pyzdek and Keller, 2014].

Organizational Agility Model

As previously mentioned, the organizational context can be conceptualized in different ways to study its impact on agile methods implementation.

For instance, [Robinson and Sharp, 2005], based on [Cockburn, 2004a] (see Section 3.2.1.2), looks at three organizational context dimensions: the organization's behavior, beliefs, attitudes & values, the organizational structure and the physical and temporal setting of the organization. [Siakas and Siakas, 2007] identifies an ideal organizational culture to embrace agility by relying on the [Hofstede, 2011] cultural dimensions (see Chapter 11). [Tolfo and Wazlawick, 2008] defines the following seven organizational context dimensions: innovation and risk, detail orientation, outcome orientation, people orientation, team orientation, aggressiveness, and stability. Finally, [Boehm and Turner, 2003] defines five factors to be considered when an organization adopts agile methods (see Section 3.2.1.1): (1) Personnel, (2) Dynamism (requirements variability), (3) Culture (Thriving on chaos vs. order), (4) Team size and, (5) Criticality.

Using the results of the questionnaires, we analyzed the organizational context according to the 5 factors of the [Boehm and Turner, 2003] model. This model was found the most convenient since its dimensions are easier to capture regarding the study settings. Later in Chapter 11, we used [Hofstede, 2011] to study the cultural perspective of organizations.

SWOT Analysis

Based on the collected data from the questionnaires and interviews, we summarize the benefits and challenges at the organizational level in the form of a SWOT risk analysis (Strengths, Weaknesses, Opportunities, Threats) (See Figure 9.3). The main intention of this analysis is to classify the influential factors reported by practitioners using two dimensions: favorable/unfavorable and internal/external. This analysis tend to confirm the influence of the 5 factors of the [Boehm and Turner, 2003] model and reports additional challenging factors.

Affinity Diagram

At a project level, using the retrospective results, i.e., the affinity diagrams, we identify the perceived critical context factors and reflect on the improvement and customization opportunities. The results are reported in Table are reported in Table 9.4.

9.3 **Results**

9.3.1 **Organizational Context Study**

As a preliminary understanding of the organizational context and using the results of questionnaires and interviews, we first used the [Boehm and Turner, 2003] assessment methodology to objectively visualize the risk of adopting agility in the context of the D443 (see Figure 9.2).

The results of the assessment show high risk ratings of 4 out the 5 critical context factors defined by the methodology. This strengthens the idea of careful agility adoption and customization. More precisely, the implementation of agile methods seem to be threatened by the:

- Lack of agile experience (or knowledge): 65% have no agile experience, 18% have a little experience and only 2% have good experience. Based on these results, we identified training needs and planned two half-days training sessions. with the purpose of introducing different agile methods to 25+ practitioner from the service. Practitioners demonstrated particular interest to the DSDM approach which is more adapted to large organizations with high levels of governance [Tudor and Walter, 2006].
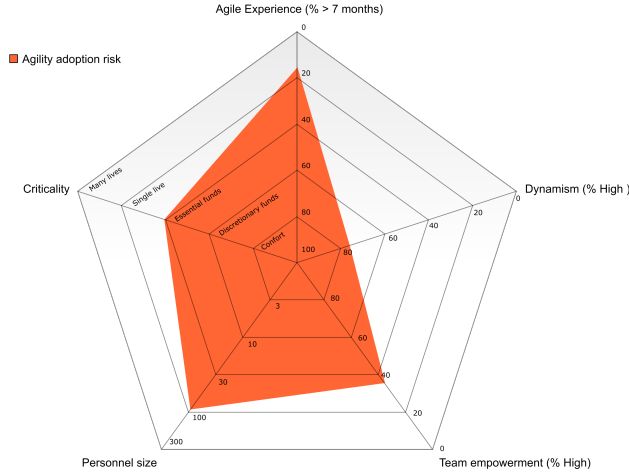
Figure 9.2.: Organizational context according to [Boehm and Turner, 2003]

- Lack of team empowerment: only 36% of practitioners believe that the team empowerment is satisfactory. Precisely, the team members reported not being involved in the project planning and in the negotiation of the scope with the business stakeholders. They are often required to stick to the initial agreement decided between the business project leader, the technical project leader and the customer representatives.

- The size of the organization: the D443 employs more than 84 people working at different projects and different levels which means that implementing agile methods at scale is more complex and would necessitate more customization. The question was discussed during interviews.

- The criticality of the domain: the D443 works on governmental projects and depends on several regulations which may have a significant impact on the implementation of agile methods. The practitioners discussed this aspect specifically during the interview session of the managers unit which report that their work depends on essential European and governmental funds and therefore regulation tend to privilege a more disciplined development process. We further investigate this question in Chapter 10.

The 5th dimension, i.e., the requirements dynamism seem however to be an opportunity to apply agility.

This preliminary analysis step allowed us to get insights regarding the challenging context factors and to plan the resolution strategies to undertake. As a second step, the data collected from Interviews (I) and the first questionnaire (Q1) are theoretically coded using a SWOT matrix (Strengths, Weaknesses, Opportunities, Threats) (see Figure 9.3). The matrix was later enriched with the factors identified during the Project Retrospective (PR) (see section 9.3.2).

Table 9.1.: Questionnaire 1 (Q1) : Current process analysis

Rate the following statements 4 : strongly agree , 3 : agree , 2 : disagree , 1 : strongly disagree, N/A	Mean	Agree (%)
Section 1. Team organization		
1.1 Teamwork composition		
1.1.1 Team members are kept together as long as possible.	3.3	79.4%
1.1.2 Each member work exclusively on the current project.	2.8	52,9%
1.1.3 Everyone required in the project is on the team.	2.5	35.2%
1.2 Team Empowerment / Collective Leadership		
1.2.1 Manager don't tell members how to achieve goals.	3	58.8%
1.2.2 Members choose tasks according to their added-value.	2.1	14.7%
1.2.3 Manager rarely change priorities within the iteration.	2.7	35.3%
1.2.4 Members choose by them-selves the tasks to do.	2.3	35.3%
1.3 Communication and transparency		
1.3.1 Meetings are enough recurrent and improve visibility.	2.8	55.8%
1.3.2 Team members discuss problems unreservedly.	3.1	70.6%
1.3.3 Inter-team communication is efficient.	2.4	41.2%
1.3.4 Documents are used to support informal communication.	2.7	53%
Section 2. Project management		
2.1 Planning		
2.1.1 The team maintains a prioritized task list.	3.1	79.4%
2.1.2 The team performs just-in-time planning.	2.8	61.8%
2.1.3 The team performs collective planning.	2.9	64.7%
2.1.4 Short iterations (max. 4 weeks.)	2.5	35.3%
2.2 Estimation and Progress tracking		
2.2.1 The effort estimation is done collectively.	2.8	54.5%
2.2.2 For each task, a done criteria is defined.	2.9	55.8%
2.2.3 The team is able to estimate and visualize the progress.	2.3	35.3%
2.2.4 The customer track the progress closely.	2.8	58.8%

Table 9.1.: Questionnaire 1 (Q1) : Current process analysis (continued)

Rate the following statements 4 : strongly agree, 3 : agree , 2 : disagree , 1 : strongly disagree, N/A	Mean	Agree (%)
Section 3. Specification and requirements analysis		
3.1 Spec. is done collectively (business & technical actors).	3.1	67.6%
3.2 High-level initial architecture is performed.	2.7	41.2%
3.3 Specifications are gradually refined.	3.1	64.7%
3.4 Team manages efficiently requirements change.	2.8	52.9%
3.5 Team can start developing with incomplete requirements.	3	67.6%
3.6 There's many requirements change per month.	3.3	76.4%
3.7 The design is performed just-in-time (at iteration start).	2.6	44.1%
3.8 Iterative conceptual modeling	2.6	47%
Section 4. Development practices		
4.1 The team uses unit testing	2	24.2%
4.2 The team uses refactoring	2.1	23.6%
4.3 The team respects code versioning	3.6	85.3%
4.4 The team members share the code (collective ownership)	2.3	38.23%
4.5 The team respects design patterns and simplicity rules	3.2	70.6%
4.6 Continuous integration	2.5	38.23%
4.7 Acceptance testing (involving customer or representatives).	2.7	45%

The analysis reveals that **strengths** (internal favorable factors) are the team internal communication (Q1 1.3.2, I), autonomy (Q1 1.2.1, I, PR) and the management buy-in (I). In fact, 57% of the respondents think that the team members are sufficiently autonomous and 70% think there's enough transparency in problem management. These two positive factors were also discussed during interviews. Team members report that the Scrum management practices they experienced so far improved the internal communication and brought greater motivation (see Section 9.3.2).

The only **opportunity** i.e., (external favorable factor) that we were able to identify is the Requirements Dynamism or variability (I, Q1 3.6). Indeed, this means that the project scope is prone to change in which case agile methods would be helpful to deal with the evolving requirements.

The **weaknesses** (i.e., internal unfavorable factors) include the lack of a suitable organizational culture. This involves the lack of process visibility, the inflexibility to change, the internal reluctance, traditional planning, etc. For example, the organizational process usually consists of long iterations, a direct consequence of the disciplined organizational culture (see Section B.2).

Table 9.2.: Questionnaire 2 (Q2) : Context analysis

---

1. Do you have experience in agile development?

No experience

0 - 6 months

7 - 12 months

More than 1 year

More than 2 years

Section 2. Rate your knowledge of the following development practices

3 : Expert , 2 : Familiar with , 1 : acknowledged , 0 : Not acknowledged

---

2.1	Release planning
2.2	Scrum daily meeting
2.3	User stories
2.4	Planning poker
2.5	Definition of Done
2.6	Sprint plan
2.7	User stories
2.8	Sprint review
2.9	Sprint retrospective
2.10	Task board, Kanban board, Post-it board
2.11	Agile Risk Board
2.12	Product Backlog
2.13	Sprint Backlog
2.14	Burn-down chart/ Burn-up chart
2.15	Unit testing
2.16	Refactoring
2.17	code versioning
2.18	Collective ownership
2.19	Continuous integration
2.19	Acceptance testing (involving customer or representatives)
...	

Section 3. Rate the following practices according to the degree of change you desire and think is realistic.

(same list of practices)

Some reluctance towards the agile philosophy was also noticed during the interviews, especially among business units that are working on many projects at the same time or in which the task realization is not collaborative (e.g., the application maintenance unit in which each member works on the

maintenance of a specific project). 70% think that the business stakeholders don't collaborate enough with the technical team members.

Another seemingly critical **weakness** is the inter-team communication. Indeed, only 41% of the respondents think that the inter-team communication is satisfactory. Moreover, the problem was largely discussed during interviews. For example, the communication between the DEV, AR and FA is seemingly conflictual. In this context, we discussed leads for solutions and proposed some practices that were later applied for improving the communication with the FA unit (see Section 9.3.2).

Questionnaire 2 reveals another important **weakness** factor that was previously discussed by the project team: the lack of knowledge of the agile practices. 65% have no agile experience, 18% have a little experience and only 2% have good experience.

A major **threat** (i.e., external unfavorable factor) is the lack of availability of the customer. 60% of the respondents believe that the customer does not track the work progress frequently and 72% would like a better implication.

STRENGTHS (INTERNAL)	WEAKNESSES (INTERNAL)
CF1: Team internal communication CF2: Team autonomy and empowerment CF3: Management Buy-in	CF4: Organizational structure and culture CF5: Inter-team communication CF6: Agile experience and skills CF7: Size (of the organization and team) CF8: Team co-location (vs distribution)
OPPORTUNITIES (EXTERNAL)	THREATS (EXTERNAL)
CF9: Requirements Dynamism	CF10: Domain criticality CF11: Customer availability CF12: Contracting model (budget, time, scope)

Figure 9.3.: SWOT Analysis

9.3.2 Project's Context Study

The retrospective results are mitigated : the participants reported an important list of issues but also undeniable benefits, especially at the team-level (e.g., better team collaboration, better visibility of the work progress, regular releases and product delivery within deadlines, quick detection of bugs, etc). These are more detailed in [AYED, 2013]. In this chapter we focus more on the challenging factors that would drive customization guidelines.

The challenges were either expressed as difficulties to satisfy specific agility goals or as challenging context factors. They are reported in Table 9.4.

One of the major challenges that was reported consists of the lack of inter-team communication. Indeed, the team reported several issues related to the communication with other business units. This factor is emphasized by the organizational structure since the processes followed by the units are not always aligned with the team iterations (see Appendix B.3). For example, the functional analysis (FA) unit is stand alone and its resources are not exclusively allocated to IDEES. Due to these circumstances, the analyst cannot always be present at the daily agile meetings and can find difficulties to deliver detailed analysis on time (i.e., at the sprint start).

The team also reported that adopting agile software development methods only at the team level is not sufficient to provide satisfactory level of agility. In fact, Scrum was applied for product development while other units were still employing the traditional waterfall-like process for planning, budgeting, testing, etc. This hybrid development approach is known as the "water-Scrum-fall" approach. The difference of agility levels between the organizational overall process and the team process is assessed in one of our previous works [Ayed et al., 2014]. The assessment shows that the degree of agility of the development sprints is satisfactory (0.6) while it is still very low in the other phases (0.36). Such a difference in agility levels contributed to the communication problems.

Other issues related to the lack of agile knowledge are reported: difficulties to plan and estimate, no clear definition of roles, etc. Many issues regarding the organizational culture are also discussed: the lack of visibility and visual facilitation tools, long meetings, overspecialization of team members (recruitment style in public organizations), etc.

Another point discussed during the retrospective is the possible customization opportunities (see Table 9.4). Each customization opportunity is expressed as a recommendation `+` , a configuration or a cautious implementation

`C` or an exclusion `–` of a specific practice. As shown in the table, the following customization opportunities were discussed:

[C1] **User Stories Estimation** `C` : The team reported difficulties to estimate the identified features in terms of story points (relative estimation). They explain this by their lack of experience (CF6) and by the internal communication issues (CF1). Moreover, the team reported difficulties to estimate because of the lack of understanding of features. This is caused by the lack of customer and business representatives availability (CF11). As a matter of fact, more attention needs to be paid to user stories estimation. Additionally to a proper consideration to the practice, some consensus-based planning techniques may be useful. The team discussed the utility of using planning pokers as a consensus estimation technique. This practice would help to integrate each one estimation of the development effort an in fine improve on-time delivery.

[C2] **Sprint planning** `C` : The team reported that the sprint planning meeting usually takes much more time than expected because of the large team size (CF7) and the criticality of the project (CF10). A solution discussed to resolve this issue is to configure the sprint planning practice by *dividing it into two meetings* as suggested by the LeSS method[4]: (1) a high level meeting which brings together team representatives and the product owner to discuss the sprint or release objectives and clarify the items to be developed. At this step no decomposition into tasks is performed. (2) a separate meeting with all team members aimed at decomposing the user stories into smaller tasks and to estimate them.

[C3] **Grooming session per release** `+` : Team members explained that there was no dedicated customer representatives to the project and discussed suitable practices to reduce the impact of this factor. Particularly, they considered to organize some feature grooming sessions (one per project release for example). The grooming session would bring together the product owner (PO), the available representatives and business stakeholders and some, or all, of the rest of team to review items on the backlog and ensure that they still are up-to-date. This would also decrease the external interventions during sprints. Doing so, the customer representative should not necessarily attend to sprint planning, and therefore the practice would be recommended in a context where the customer availability is low

---

[4]Large Scale Scrum: `https://less.works/less/framework/sprint-planning-one.html`

(CF11). In case of the D443, customer representatives work at the same time on other projects, so their availability is limited.

[C4] **Domain walk-through** `+` : Another practice that would further align the team with the customer project view without much solicitation of customer representatives with limited availability (CF11) is to allow the team to meet them at least once, when the project is launched. The idea is to start the project by a domain walk-through meeting[5] where the customer and domain experts introduce the team through the entire project. Doing so, the customer representatives are mainly involved at the start of the project and (eventually at the start of releases) with the purpose of improving the team product vision. Their implication is reduced afterwards. The team discussed the eventual usefulness of the practice regarding the domain complexity (CF10).

[C5] **Model in small increments** `−` : Small incremental modeling was also discussed by the team as a possible solution for the upfront analysis issue. However, since the organization has a traditional contracting model (CF12) and regarding the projects criticality (CF10), an important time-consuming analysis phase still have to be done at the beginning of the project. The actual process already include some kind of incremental modeling but increments are of several months (see Section B.3). Another challenging factor to the implementation of the practices is the organizational structure (isolated analysis unit) (CF4). Having small incremental modeling seem also to be unfeasible at the moment of the study but a focus group between the project portfolio manager and a number of analyst representatives was later organized to discuss the question of having smaller functional analysis increments.

[C6] **US writing workshops** `+` : Writing user stories is not an easy task for unexperienced practitioners (CF6). Agile training sessions usually include user story writing workshops[6] performed by the agile facilitator (scrum master) or a coach. Later, two user story writing sessions were organized in collaboration with the researcher and a senior functional analyst. Analysts and developer representatives participated to it.

---

[5]http://www.martinbauer.com/Articles/Content-Modelling/Step-1-Domain-Walkthrough
[6]https://www.mountaingoatsoftware.com/exclusive/story-writing-workshop-pdf-download

[C7] **Custom extension of EA** `c` : This customization to Scrum was already introduced by the analysis unit at the time the retrospective was animated. It consists of using a special extension of Enterprise Architect that allows writing user stories and attach them to use cases. The practice was appreciated but remains time consuming (work is done twice !)

[C8] **Roles ambiguity (coaching)** `+` : The lack of roles clarity seem to be caused by the structure of the organization (CF4) and the lack of agile experience and knowledge (CF6). Obviously, it may be confusing to transition from a development process were traditional titles are used (business manager, technical manager, business representative, analyst, architect, analyst developer, etc.): how will play the role of the product owner? who will play the role of the scrum master? etc. Accountability in scrum is mainly around all three roles: the PO is accountable for maximizing the value the product to be delivered, the SM is accountable improving the team velocity and improving their way of working and the team member are all accountable for delivering a product of good quality. Integrating these roles may not be easy. Coaching sessions may be enforced to integrate this concepts. Guidelines exist in the literature[7].

[C9] **Sprint-0 (technical envisioning)** `c` : The team members discussed a lack of cross-functional skills (CF6). Consequently, at some phases of the project, some individuals are overloaded while others are stuck waiting. Moreover, they reported a lack of time to decide about the best technical alternatives to employ in the project.

As a solution, more technical envisioning (a step where the team can lean, balance and choose from different technical alternatives) at the start of the project is considered and discussed. More precisely, a practice that could be systematically introduced in future projects is the so-called Sprint-0[7]. This practice may be viewed as a project before the project. It would allow the team to get time to set-up the technical environment, define the coding standard, develop some code snippets or animate technical demos. However, regarding the contracting model (CF12), the planning and the high level technological architecture are fixed with no consultation of the team. Therefore, the team thinks the practice would be beneficial but its applicability to their environment should be verified.

---

[7]`https://manifesto.co.uk/scrum-practice-sprint-zero/`

[C10] **Pair programming sessions** `+` : Similarly to the previous recommendation, pair programming sessions may also be recommended to reduce the lack of cross-functional skills.

[C11] **Parallel independent testing** `+` : In the context of the D443, a whole team approach where agile teams are totally cross functional and where all tests are done during sprints is not feasible. The main reason for this is that testers are outside of the development teams (CF4). Moreover, regarding the criticality of the domain (CF10), a sophisticated testing (e.g., security testing, production testing, investigative testing, integration testing, non-functional testing [Gregory and Crispin, 2014]) is necessary. Having the team focus on all these kinds of testing is difficult. In such a context, it is recommended to let the team focus on functional and confirmatory testing while delegating more complex testing to a parallel independent testing team.

[C12] **Scrum** `c` : Because of the lack of experience (CF6) and of the large size of the Scrum team (CF7), the interviewed team members reported that meetings tend to last more than expected (20 to 30 min for daily meetings, and 4 to 8h for sprint planning meetings). A solution that was discussed to scale up Scrum is to divide the unique scrum team to two groups. The practice is known as Scrum of Scrums and may be very useful when several teams should collaborate on different components of the same project[8].

[C13] **Retrospectives facilitation techniques** `+` : The pilot-team reported that retrospective meetings were not systematically held and that they were sometimes considered as a loss of time. In order to motivate the team, several retrospective techniques[9] are shared among the agile community and could be recommended for making the team retrospectives more motivating and efficient.

[C14] **Physical** `–` **vs Virtual** `c` **visibility tools**: The project team used to rely on the classical version of Jira Atlassian (an issue tracker) for encoding user stories and tracking progress. However, several studies demonstrate the benefits of having physical visibility tools [Perry, 2008; Sharp et al., 2008]. Switching to physical post-it boards was discussed during the project retrospective. Specifically, 2 tools were

---

[8]https://www.agilealliance.org/glossary/scrum-of-scrums/
[9]https://trello.com/b/4OBwQg57/retrospective-techniques-for-coaches-scrum-masters-and-other-facilitators

suggested: a physical task board and a portfolio alignment wall. For the next release, while maintaining their Jira task board, the team also implemented a physical task board with post-its representing user stories and tasks and swim lanes representing the current task state. This solution was not satisfying because of : (a) the size of the team (CF7), (b) information was double encoded and (c) accessibility to distributed stakeholders (CF8). Therefore, the team chose to get back to the virtual task board but using a better Jira extension, Green Hopper, actually known as Jira Agile[5] (an addon for Jira that integrates more specific agile tools such as boards, burndowns, point estimation, etc.), and reported a better satisfaction level.

9.4 Discussion

In this study, we investigated the influence of the organizational context on agile methods implementation and customization. We have been able to confirm the influence of the 5 factors of the [Boehm and Turner, 2003] model and to identify 7 more potentially critical factors. The final set of factors can be classified as shown in Table 9.3.

Table 9.3.: Identified context factors

Dimension	Factor
Organization	Management buy-in
	Organizational structure and culture
	Inter-team communication
	Agile experience and knowledge
	Organization size
Project-team	Internal communication
	Team autonomy and empowerment
	Team distribution
External Environment	Requirements dynamism
	Domain criticality
	Customer availability
	Contracting model

Table 9.4.: Retrospective - Challenges transcription

ID	Challenge (agility goal)	Freq.	Details	Possible customizations	Type
1	Planning	11	Decomposition of requirements into USs	USs estimation (use planning poker) [C1]	C
	(On-time delivery)		Self assignment of tasks	Sprint Planning [C2]	C
	(Waste elimination)		Difficulty to estimate (Requirements not well understood)		
			No respect of tasks assignment		
			Work estimation : the dev. effort not considered		
			Long sprint planning		
2	Collaboration with business	8	Weak collaboration between PO and business	Grooming session per release [C3]	+
	(Valuable software)		Weak collaboration with analysts	Domain Walk-through [C4]	+
			Team disturbed by external demands during sprints		
			External pressure		
3	Analysis	8	Thorough upfront analysis	Model in small increments [C5]	−
	(Frequent delivery)		Alignment of user stories (JIRA[5]) and use cases (UML EA [6])	US writing workshops [C6]	C
	(Valuable software)			User Story [C7]	+

[5] https://jira.atlassian.com/
[6] Enterprise Architect http://sparxsystems.com/products/ea/

Table 9.4.: Retrospective - Challenges transcription (continued)

ID	Challenge (agility goal)	Freq.	Details	Possible customizations	Type
4	Roles ambiguity	7	Unclear role definition	Coaching / Guidelines[7] [**C8**]	+
	(Role ownership)		*"Everybody is responsible for every thing and for nothing !"*		
5	Team Skills	6	Overspecialization	Sprint-0 (technical envisioning) [**C9**]	+
	(Technical excellence)		Frustration because not enough learning and knowledge sharing	Pair programming sessions [**C10**]	+
6	Testing	6	Testing workload at the end of sprints	Parallel independent testing [8] [**C11**]	+
	(Continuous testing and delivery)		No stories for identified bugs		*C*

---

[7]Example of Guidelines: `https://www.linkedin.com/pulse/how-resolve-agile-roles-conflicts-confusions-doubts-ajay/`
[8]`http://www.ambysoft.com/essays/agileTesting.html`

Table 9.4.: Retrospective - Challenges transcription (continued)

ID	Challenge (agility goal)	Freq.	Details	Possible customizations	Type
7	Project initiation	5	Late delivery of the high-level architecture	Sprint-0 (envisioning) [C9]	c
	(Frequent delivery)				
8	Meetings	5	Long daily meetings (same for two sub-projects)	Scrum of Scrums (1 meeting / subproject) [C12]	+
	(Waste elimination)		No effective retrospective	Retrospective facilitation tools [C13]	+
9	Visibility	5	Lack of visibility tools	Physical task board [C14]	−
	(Process visibility)		No clear and visible definition of priorities	Virtual task board (Jira)[C14]	c
			KPIs: are they adapted to agile projects?	Portfolio alignment wall[9] [C14]	+

Table 9.4.: Retrospective - Challenges transcription (continued)

ID	Challenge (agility goal)	Freq.	Details	Possible customizations	Type
10	Organizational structure	2	Transition to agile without assessing risks at an organizational level	–	–
11	Team Size	2	Large team size (15 p.) involving two sub-projects	Scrum of Scrums [C12]	+
12	Team Distribution	1	Different locations for project sub-groups	–	–

+	Helpful practice (based on the experience of the team)
c	Practice that needs customization or careful implementation (based on the experience of the team)
–	Practice to avoid (based on the experience of the team)

If we reuse the radar chart analogy from [Boehm and Turner, 2003], the risk of implementing agile methods in the context of the D443 can be visualized as shown in Figure 9.4. Some of these factors were already measured using the questionnaires and other were estimated for illustration purposes.
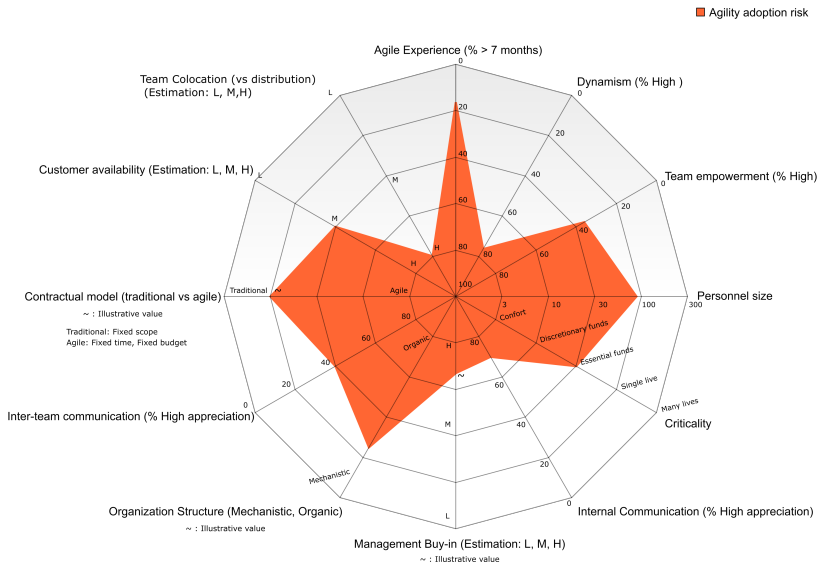
Figure 9.4.: Illustration of the 12 identified factors (adapted from [Boehm and Turner, 2003])

It is worth noting that among these factors, some are easily measurable or estimable using questionnaires for example (as we did in this chapter) while others are more difficult to capture.

Specifically, ranking the organizational structure (and culture) regarding agility is not obvious. Some research works establish the relationship between the organic organizational structure and the success of agile implementation [Overby et al., 2006]. Studies such as [Strode et al., 2009] investigate the impact of hierarchical structures on agility. This factor may also be associated to the concept of enterprise agility, i.e., the degree of adaptivity and flexibility to changes in the environment [Sherehiy et al., 2007]. Finally, some studies investigate this factor in further details. For example, [Tolfo and Wazlawick, 2008; Siakas and Siakas, 2007] use the [Hofstede, 2011] model for dimensioning the culture of the organization. For the aforementioned reasons, we further study this factor in Chapter 11.

9.5 **Lessons Learned**

This first case study allowed us to capture 12 influential context factors, some of which were previously discussed in Chapter 3. We classified these factors to 3 dimensions: organization, project-team and external environment and discussed their measurability and their impact on agile practices customization.

A direct implication for the AMQuICk framework is that the context is *multidimensional*. In other terms, it can be captured using several dimensions and factors (attributes). Moreover, the context is *measurable*, i.e., it can be measured using one or many assessment models. To measure the organizational context, we used in this study a simple opinion-based questionnaire. As earlier discussed, this may be insufficient for some composite factors such as the organizational culture. For this kind of factors, more sophisticated assessment models can be used. Several assessment models exist in the literature or are shared in the agile community[10].

The advantage of dimensioning the context and measuring it using different factors is to allow an easier and more precise definition of the customization knowledge (i.e., at a fine grained level). As a simplistic example, the set of measurable context factors can be used to represent the agile experts knowledge in the form of customization/recommendation rules as shown in the following:

$$Domain\_Criticality = High \Rightarrow RecommendPractice(Domain\_Walkthrough)$$
$$(9.1)$$

$$Customer\_Availability = Low \Rightarrow ConfigurePractice(Sprint\_Planning)$$
$$(9.2)$$

More interestingly, the measurable context factors could be used to encapsulate the knowledge of practitioners in a structured dataset. The collection of experiences would then be reused to predict adequate customizations (using a recommender system for example [Ricci et al., 2015]).

This study also tends to show that the adoption of agile methods in a non-sweet spot context should be carefully planned. The implementation of a progressive strategy seem to be beneficial. Indeed, the management was at first hesitant because of the lack of a relevant assessment regarding the benefits and risks of agility. We therefore, advocated a progressive and cautious transformation. We chose a pilot project of mid-level criticality to implement Scrum (see Figure 9.2). Despite the number of encountered

---

[10]`https://www.benlinders.com/tools/agile-self-assessments/`

issues, the pilot-team enthusiasm impacted on the way of working of two other teams that started to work in a more iterative way by including more frequent checkpoints and team meetings. Then, once the interest to agile methods raised in the service, a natural desire to gain more knowledge and to spread agile values among all the teams has followed.

The study also tend to show the importance to systematize the representation of the project-team knwoledge so it can be better disseminated at an organizational level.

With these informations in mind, we discuss in Chapter 12, the extension of AMQuICk Essence by including new concepts necessary to document agile practitioners knowledge.

Chapter 10

E-Gov Case Study

Challenges of Implementing Agile in E-Gov

In the previous chapter, we observed the implementation of a custom agile method in a transitional context, identified the challenges met by stakeholders from different units and synthesized the overall impact of the organizational context on the implementation of agile methods.

However, since we studied an IT public-sector organization, a legitimate question that needs to be investigated regards the possible influence of the sector, its intrinsic characteristics and the nature of e-government service development on agile methods implementation. Indeed, agile software development is still controversial in some circles such as the public IT sector [Mackinnon, 2004]. The main reason for this skepticism is that the public sector reality is perceived as hardly suitable for agile management structures and culture.

In this chapter, we investigate the contextual challenges that practitioners face when developing e-government services using an agile software development method. To do so, we organized three focus groups with practitioners involved in the process of implementing agile methods in different Belgian e-government organizations. The purpose of the study was to identify the challenges they faced so far in implementing agile practices to their contexts.

In Section 10.1, we provide an overview on some previous work that had investigated e-government service development using agile methods. In Section 10.2, we detail the research design we applied to determine and validate the e-government specific challenges. The challenges that emerged from the organized focus groups are contextually explained and presented in Section 10.3 and their validity is discussed in Section 10.4. Finally, in Section 10.5 we synthesize the lessons learned from this case study and explain how it contributes to the thesis.

10.1 **Background**

E-government refers to *the use of Information and Communication Technologies (ICT) by governments to deliver their services in an optimal way*. For a long time, disciplined development methods such as the traditional waterfall model and RUP (see Section 2.1.3) have been prevailing in the development of e-government services. The inherent complexity of the public sector is so important that disciplined methods with their thorough planning and standardized processes were for a long time considered as the safest way to produce e-government services.

However, new challenges of e-government (e.g., changing political strategies, willingness to deliver more valuable services and to establish better transparency between citizens, political representatives and the service development stakeholders, etc.) have contributed to make a shift in governmental strategies. Indeed, these are becoming interested in a number of new initiatives such as the open government[1] for providing a better transparency [Lee and Kwak, 2012], increasing the collaboration with citizens [Axelsson et al., 2010; Lindgren, 2014; Anthopoulos et al., 2007] or enhancing innovation [Holgersson et al., 2017].

Among these initiatives is the willingness to shift towards better development practices. In this regard, several research studies have underlined the contextual specificities of the e-government domain and their general impact on development practices.

For example, [van Velsen et al., 2009] underlined the following factors: the heterogeneous and large user group (i.e., the citizens), the complicated processes and requirements of e-government services and the crucial need for interoperability between the systems of different governmental bodies.

[Anthopoulos et al., 2016] conducted a systematic literature review to understand and classify the failure factors of e-government service development. The identified factors consists of the influence of the organizational power, the lack of political support and commitment, the lack of appropriate skills, several business management issues, the ambiguous business needs and unclear visions.

[Mergel, 2016] is another study worth citing that referred a failure project "*healthcare.gov*" to demonstrate how disciplined models may be dissatisfying. This failure has initiated calls for more agile management approaches which are expected to help e-government organizations adapt faster to political and citizen requests.

---

[1]`https://www.opengovpartnership.org/`

In order to meet such challenges, e-government organizations have already shown a particular interest in agile software development approaches. [Powner, 2012], a report published by the U.S. Government Accountability Office, identifies 32 agile practices and approaches as effective for applying in a federal environment and reports a set of recommendations regarding the application of agile methods in e-government projects. It also identifies a set of challenges encountered by governmental agencies when transitioning to agile software development. These can be synthesized into 3 perspectives:

- Team perspective: the report mentions that teams usually had difficulties in collaborating closely, transitioning to self-directed work and committing to more timely and frequent input.

- Organizational perspective: the report mentions that agencies had trouble committing staff and that agile methods do not align with federal reporting, procurement, status tracking and artifacts review practices.

- Customers perspective: the report mentions that customers did not trust iterative solutions.

This report constitutes a valuable reference for e-government practitioners. However, since it only reports challenges without explaining them contextually, it cannot be useful for deriving comprehensible recommendations for practitioners from other e-government agencies.

In this research, we collaborated with practitioners from different e-government agencies in Belgium with the purpose to elicit the general contextual factors that influence the implementation of agile methods and to provide e-government practitioners with comprehensible and reusable guidelines.

10.2 Methodology

As an exploratory methodology to find out and validate the context-specific challenges and to analyze them, we decided to follow a simplified grounded theory approach (see Section 4.1). Following this methodology, data was iteratively coded at each step of the process to identify the challenges and the correlated causal factors. Three iterations were necessary to generate enough learnings.

The following sections detail the proposed methodology.

10.2.1 Objectives

The studies presented in Section 10.1 tend to show the growing interest to implement agile methods in an e-government context. However, few research has yet investigated their adaptability and the challenges that may arise when implementing them without much discernment.

With this regard, our research aims to:

1. find out, discuss and validate the most critical context-specific challenges that e-government stakeholders encounter when implementing agile software development, and

2. identify the causal factors that underlie those challenges and how they may impact agility goals.

More specifically, the second objective intends to generate comprehensible guidelines (theoretical coding) that can be reused by other practitioners and therefore pave to way for the customization of agile methods in the context of e-government.

These specific objectives relate to RQ2 and RQ3, the second and third research questions of the thesis (see Section 3.5).

10.2.2 Data Collection

In order to gather the research data, we chose to organize focus groups with the purpose of capturing and understanding the challenges as perceived by public sector representatives. We decided to stop grounding theory after three focus groups as each challenge had reached saturation and no more original findings were determined.

The focus groups were conducted according to the guidelines and best practices reported in [Krueger, 2014; Morgan, 1996].

Following these guidelines, we made the choice to involve from 5 to 10 participants (selected based on convenience sampling) at each focus group. We also chose to follow a single category design, i.e., to focus on a set of practitioners familiar with agile methods so that the validation of the constructs is empirically grounded.

However, to stimulate discussion and contrasting opinions, we ensured diversity in terms of organizations as participants came from different governmental levels (local and regional) and from different governmental sectors (employment, administrative simplification, innovation, etc). Furthermore,

the focus groups were diverse in terms of agile expertise (which range among novice, intermediate and expert) and in terms of roles (developers, analysts, operational, team leaders (middle level management) and strategic leaders (top level management)).

Table 10.1.: Focus Groups Participants

ID	E-gov Service	Role	Agile Exp. Level	Method
P1	Communal services	Project Leader	Intermediate	Custom
P2	Communal services	Dev.	Novice	Custom
P3	Communal services	Analyst, Dev., Ops.	Novice	User Driven Design
P4	Communal services	Dev., Ops.	Novice	Custom
P5	Communal services	Dev., Ops.	Novice	Custom
P6	Communal services	Dev., Ops.	Expert	Devops
P7	Communal services	Dev., Ops.	Novice	Custom
P8	Communal services	Dev., Ops.	Novice	Custom
P9	Admin. Simplification	Project Leader	Intermediate	Custom
P10	Employment	IT Advisor	Novice	Custom
P11	Employment	IT Director	Expert	Custom Scrum
P12	Enterprise & Innovation	IT Advisor	Novice	Custom
P13	Digital Technology	IT Manager	Intermediate	Scrum/Kanban
P14	Environment	General Director	Expert	Custom Scrum
P15	Environment	Product Owner	Intermediate	Custom Scrum
P16	Transversal IT	General Inspector	Intermediate	Custom
P17	Transversal IT	IT Project Leader	Intermediate	Custom
P18	Routes and highway	General Director	Novice	Custom
P19	Research & Innovation	General Inspector	Novice	Custom Scrum
P20	Research & Innovation	IT Project Manager	Novice	Custom Scrum

Table 10.1 details the profiles of the different focus groups participants. It describes the e-government service that they contribute to, their current role, their agile expertise and the agile methods they experienced so far. P1 to P8 participated to the first focus group, P9 to P13 to the second and P14 to P20 to the third. It is worth to note that some participants relied on

their previous agile experience in other public organizations to identify the challenges.

Following the guidelines in [Krueger, 2014; Morgan, 1996] and applying the *Affinity Diagram* method [Pyzdek and Keller, 2014], we designed the following questioning route:

Step 1. The participants were asked to present themselves, their organization as well as their knowledge and experience with agile ;

Step 2. Each practitioner was asked to write on post-its the challenges that he/she faced so far when trying to implement agile methods in a public organization. We recommended to write down a unique challenge per post-it ;

Step 3. Each practitioner discussed the challenges he/she identified and placed them on a board.

Step 4. Practitioners were asked to collaborate on arranging the challenges into separate categories to design the Affinity Diagram. We only helped them naming the categories. Indeed, at this step, our assistance was reduced to the minimum to avoid bias from the researchers as we only intervene to facilitate consensus among participants about the assignment of the post-its ;

Step 5. The final grouping are reviewed and discussed by all the practitioners, with the researchers playing a mediation role. As suggested by [Morgan, 1996], we used the theoretical background to generate discussion for this step.

Step 6. In order to identify the most important category of challenges (as perceived by practitioners), we used the dot voting technique[2] which allows to visually show what are the most important ideas among a reasonably small set. Each participant was given 3 dots votes (dv) (each vote will be represented by a dot on the post it) that they can place freely. More that one dot could be placed on a single category.

Step 7. We finally drove the discussion to insist on the most populated categories and to bring practitioners to provide the causal interpretation of the challenges it contains. This is known in grounded theory as *in-vivo coding.*

---

[2]http://www.funretrospectives.com/dot-voting/

10.2.3 Data Analysis

As mentioned in [Rabiee, 2004], relying exclusively on the focus groups transcriptions may be overwhelming since they generate a large amount and complex qualitative data. In this research, the interpretation of the generated data was largely facilitated by the use of the Affinity Diagram method. Indeed, using this simple technique, the generated data were preprocessed (classified) and we were able to visualize the most critical categories of challenges and the most frequently appearing challenges.

We analyzed the generated data following the recommendations of [Krueger, 2014; Rabiee, 2004] and based on the generated affinity diagrams and on the focus groups records. After each focus group, we examined, categorized and arranged the generated data in tables following the same template (see Table 10.2). The resulting tables contain the identified challenges, their explanations by the practitioners, their categorization and the causal interpretation that practitioners pointed out to explain them (see Tables 10.3, 10.4, 10.5). In terms of grounded theory, these tables report what is known as substantive coding, i.e., the process of conceptualizing the categories and properties of the theory which emerge from the substantive area being researched [Abdel-Fattah, 2015].

Table 10.2.: Focus groups transcription template

ID	Category	Votes /15	Challenges	Causes
..	..	..	..	..

To recombine the evidence, we integrated the generated tables after the three focus groups were completed. The categories were reformulated into generic challenges and evidence for the contextual interpretation (obtained by in-vivo coding) was added from the literature.

Finally, using this synthetic table, we proceeded to the coding of the interpretations, i.e., the impact of the e-gov context factors, in the form of a matrix diagram [Pyzdek and Keller, 2014] which is useful to analyze the correlations between two groups of ideas and to systematically analyze correlations. They provide a suitable form for encoding practical recommendations to practitioners, which is the initial goal of the study. We structured the matrix diagram as follows: the columns represent the contextual factors and the rows represent the desirable agility goals (which are formulated based on the agile principles reported in Table 2.1).

10.3 **Results**

In this section, we first present the challenges identified in the focus groups (see Tables 10.3, 10.4, 10.5). Then, based on the in-vivo coding and on the transcriptions of the focus groups, we discuss the impact of those challenges and generate the theoretical coding, i.e., the matrix diagram.

The resulting diagram is shown in Table 10.6. It summarizes the impact of context factors on agility principles as described by the agile Manifesto (see Table 2.1). The matrix columns contain the context factors and its rows consist of the impacted agility principles. The (−) symbol designates a hypothetical negative impact identified in focus groups. The diagram should be considered as a way to focus the reflection and guide the customization when implementing agile methods for e-government service development.

In the following sections, we detail the set of challenges that we were able to identify and discuss the derivation of the hypothetical correlations that construct the matrix diagram.

10.3.1 Internal Competences

The most important category of challenges reported by the practitioners is the lack of internal competences (15 dot votes). Mainly, this challenge designates the unavailability of specific IT profiles in e-government teams. For example, some participants pointed out that team members are over-specialized and that they often lack time and resources to continuously improve their technical competences outside of their initial expertise. The lack of soft-skills and knowledge of agile methods was also pointed out. Indeed, practitioners reported that when the agile knowledge isn't disseminated to the whole organization, it is hard to find a common lexica and understanding with other public agents to discuss the advancement of projects. Another crucial challenge resides in the need for influential drivers (or sponsors) able to impulse the implementation of agility.

Impact

This challenge is directly related to the ***Technical Excellence*** principle (AP09). Most developers in e-government projects are specialized in clear-cut tasks, which can lead to a decrease in technical excellence (AP9, Technical excellence)

Table 10.3.: 1st focus group transcription (synthesized)

ID	Category	dv/24	Challenge	Causes
FG1-1	Management support	2	No motivation to change	Low innovation and risk taking
			Agile for "marketing" with no investment	
			Lack of sponsoring	
FG1-2	Culture	0	No self-organization	Siloed organization
			Over-processing of every demand	Bureaucracy
FG1-3	Competences	9	Lack of competences	Limited budgets
			Developers working on multiple projects	
FG1-4	Regulation and Politics	4	Communes autonomy	High domain complexity
			Continuous switch of priorities	High regulatory compliance
			Public marketplace pressure	
			Changing regulations	
FG1-6	User Involvement	4	One customer representative is insufficient	Diversity of user profiles
FG1-7	Business Availability	5	Inaccessibility of representatives	Formal governance
			No dedicated representatives	

Causes

The lack of competences is most likely not purely specific to the e-government domain but can be a result of the ***low attractiveness*** of the public sector as reported by some practitioners. Indeed, participants from the 2nd and 3rd focus groups reported that governments have difficulties to attract specific profiles to facilitate the digital transition in general. Similarly, attracting specific agile profiles such as facilitators, coaches and DevOps experts is challenging.

Moreover, it can also be the result of the ***lack of investment on innovation and learning*** by leaders and top level management of e-government organizations. Indeed, practitioners from the 3rd focus group particularly pointed that the implementation of methods may be perceived as superfluous with no immediate pay-off.

Table 10.4.: 2nd focus group transcription (synthesized)

ID	Category	dv/15	Challenges	Causes
FG2-1	Management support	4	No management buy-in	Low innovation and risk taking
			Stakeholders resistance	
FG2-2	Culture	4	Over-specialized individuals	Hierarchal structure
			Low team empowerment	
			No culture of change acceptance	
FG2-3	Competences	0	Lack of agile drivers	Sector attractiveness
			Agile understandability / knowledge	Short-term orientation
FG2-4	Innovation	4	Lack of visibility of products	Low innovation
			Outdated and non-competitive products	Short-term orientation
FG2-5	Business Availability	3	Multiple strategic levels	Hierarchal structure
			Strategic alignment between projects	Formal Governance
FG2-6	User Involvement	0	User-orientation products	Formal Governance
			Collaboration/participation of users	Users number and diversity

Hypotheses

H1: Impact of Low Attractiveness of Public Sector (CF1) on Technical Excellence (AP09)

H2: Impact of Low focus on innovation and learning (CF4) on Technical Excellence (AP09)

10.3.2 Business Availability

The relationship between e-government business experts and the IT stakeholders was reported by focus groups participants as the second most critical challenge (11 dot votes).

This challenge is not not purely specific to the e-government domain. Actually, 31% of the respondents to the 12th Annual State of Agile survey [VersionOne, 2018] reported that they encounter the same challenge.

Table 10.5.: 3rd focus group transcription (synthesized)

ID	Category	dv/21	Challenges	Causes
FG3-1	Management support	2	No management buy-in	Short-term vision
FG3-2	Culture	0	Multiple delegations for each decision Centralized structure / decisions	Formal Governance
FG3-3	Domain complexity	2	Intensive initial planning Business model: Fixed time/budget Complex requirements and documentation (iterative dev.?) Software of high impact (data security is critical)	High regulatory compliance
FG3-4	Regulation and Politics	7	Frequent changes in regulations Late and complex modifications Audits Public marketplace Investment cycles Business continuity	High regulatory compliance Formal governance
FG3-5	Competences	6	Competence for driving agility Soft skills Lack of technical experts Outdated systems hard to maintain	Low sector attractiveness Limited budgets
FG3-6	Business Availability	3	Multiple focus of business representatives	Formal governance
FG3-7	User Involvement	1	Difficult to have sufficient citizen representatives	Users size and diversity

However, this challenge seem to be particularly emphasized by the constraints of e-government. Indeed, governments constitute a diverse ecosystem with multiple stakeholders working at different strategic levels and having different objectives. These business stakeholders don't always communicate with each other, leading to a certain level of silo structure. This complex ecosystem makes the close collaboration between business and IT difficult.

Moreover, participants reported that business experts usually manage a portfolio of several e-government projects which consequently limits their availability for face-to-face conversations.

Impact

This challenge impacts two essential agile principles: ***Business/IT collaboration*** (AP04) and ***Face-to-face communication*** (AP06) .

Causes

When it comes to identify the context factors that contributes to the customer unavailability, practitioners pointed out the culture of ***formal governance*** (i.e., structures, policies and procedures). Indeed, multiple strategic levels and several business teams are usually involved in e-government projects. However, there don't always communicate with each other, leading to a difficult alignment internally. The problem makes it particularly difficult to implement agile methods in the governments at scale.

Hypotheses

H3: Impact of Formal Governance (CF3) on Business/IT collaboration (AP04)

H4: Impact of Formal Governance (CF3) on Face-to-face communication (AP06)

10.3.3 Regulatory Compliance

Governments have to take into account regulations in their processes and in their development projects. Participants from the focus groups reported the significant impact of regulatory compliance and political agendas on their development practices (11 dot votes).

Unsurprisingly, this challenge is taught by several practitioners and researchers. Indeed, 14% of the respondents to 12th Annual State of Agile survey [VersionOne, 2018] reported that they are challenged by some kind of regulatory compliance. According to [English and Hammond, 2017], new regulatory requirements has tripled globally in five years. Studies such as [Mehrfard and Hamou-Lhadj, 2011], report that heavy regulations introduce changes to business processes (at the governance and strategic level) and to software engineering practices by which software systems are built, tested, and maintained (at the team level).

Impact

Focus groups participants pointed out that regulations may be in tension with their willingness to implement agile methods. Precisely, the third focus group reported that regulations for protecting and securing sensitive information and for ensuring a higher business continuity may slow down the team's ability to ***deliver software frequently*** (AP03). Having short iterations in such a context seem to be more difficult to implement.

Participants to the first focus group discussed the fact that the regulatory compliance increases the complexity of the development process and therefore impedes the 10th agile principle, i.e., the ***work simplicity*** (AP10). Moreover, they reported that new regulations may be released late in the project life-time. Indeed, the project team members reported not being able to work at a ***constant pace*** (AP08), i.e., they can be stacked waiting for fundamental regulations at sometime and rushing on modifying features at some other time.

When we discussed the problem in more depth, it appears that the implementation of agile is inadequate. More customization is required to remove time wastes. Leads for solutions include the establishment of a core team for the project and a separate acceleration team to provide timely-support[3], a better prioritization and grooming of the backlog, a more efficient collaboration with the regulator, etc.

Moreover, the second focus group reported that the specific regulation regarding government procurement require lots of non-functional documents which distracts teams from their initial goal, i.e., the working software (AP07).

Government procurement also necessitates the planning of the development projects to be fixed upfront which makes it difficult to change scope of the project afterwards. In contrast, they observe that regulatory requirements are unclear at the beginning, open to interpretation, and evolve over time. Even thought this seem to be an argument for implementing an agile model, participants seem to perceive it as extremely problematic. Indeed, the regulation changes are not necessarily budgeted but have to be integrated in the development anyway. Moreover, the changes are often very complex to implement and cause lots of pressure. Regarding the aforementioned discussion, the high regulatory compliance in the e-government domain is in tension with one more agile principle: the acceptance or change, i.e., ***Welcome Change*** (AP02).

---

[3]`https://www.bcg.com/publications/2017/financial-institutions-technology-digital-when-agile-meets-regulatory-compliance.aspx`

Causes

This challenge is already expressed as a context factor, i.e., high regulatory compliance.

Hypotheses

H5: Impact of High Regulatory Compliance (CF5) on Welcome Change (AP02)

H6: Impact of High Regulatory Compliance (CF5) on Frequent Delivery of Software (AP03)

H7: Impact of High Regulatory Compliance (CF5) on Focus on Working Software (AP07)

H8: Impact of High Regulatory Compliance (CF5) on Constant Pace (AP08)

H9: Impact of High Regulatory Compliance (CF5) on Focus on simplicity (AP10)

10.3.4 Management and Political Support

Inadequate management support and sponsorship is another critical challenge reported by the focus group participants (8 dot votes).

Particularly, P14 and P15 reported the importance of having the mid-level management involved in the process of transition. They argue that when the willingness to change the development practices emerges from operational development teams, with no support from management, the resistance to change is likely to be an important drawback. Other practitioners report that the decision to transition to agile is not sufficient when it is not complemented by concrete actions (hiring of agile specialists, support of pilot projects, etc.).

Impact

This challenge impacts the overall ability of an organization to transition to agile.

Causes

When asked to explain this challenge regarding the e-government domain, participants stated that a ***short-term orientation*** often drive the IT strategy in governments. More generally, the lack of support from management also raised the question of the ***innovation*** in the public sector : who has the capacity and the responsibility to drive innovation in the products and services of the governments ? (see Section 10.3.7).

10.3.5 User Involvement

In the case of governments, users can be the citizens, administrations or other public servants. Participants reported that the number and diversity of the user profiles makes it difficult to involve users in the development process and/or to identify a fitting user participation strategy (5 dot votes).

The use of representatives was discussed in the focus groups but several questions remain unanswered: Can the representatives fully understand the needs of the whole user population? How to ensure their availability? The specific case of public servants being users has also been discussed. However, ensuring their available and reactivity is crucial for communicating their requirements and frequent feed-back.

Impact

This challenge is directly related with the first agile principle which prescribes to ***Focus on users and customers*** (AP01) by involving them in the development process.

Causes

As earlier specified, this challenge is caused by the ***number and diversity of e-government users***.

Hypotheses

H10: Impact of Users Diversity (CF2) on Focus on User / Customer (AP01)

10.3.6 Hierarchal Structure

Practitioners from the three focus groups pointed out that governmental organizations have a culture at odds with the agile values (4 dot votes). This challenge is not specific to the e-government domain since it is reported as the first most challenging issue by the Annual State of Agile survey [VersionOne, 2018] (53% of respondents).

Practitioners to the focus groups pointed out that governments tends to function hierarchically. This top-down way of working is present within governments as all major decision validations regarding the project or resource requests have to pass through several official decision-making bodies (Steering Committee, Working Group, etc.).

Impact

The pyramidal and siloed structure slows the development process and therefore prevents the team from maintaining a ***constant pace*** (AP08). This structure also leads to a ***lack of communication*** between business units and developers (AP4 and AP6).

Furthermore, participants reported that leaders in governments are reluctant towards scope flexibility, i.e., to the principle of ***welcoming change*** (AP02) which is perceived as a loss of control on projects.

Furthermore, the top-down culture is also a consequence of the influence of political representatives on the functioning of governments. Developments teams see their work heavily influenced as politicians require the projects to be modified to fit their own needs and agenda, often linked to the agenda of the elections. This reduces the ***self-organization*** margin of teams (AP11) and makes the ***regular improvement*** of the overall development process difficult (AP12).

Causes

As earlier discussed, the hierarchal and siloed structure of organizations is a consequence of its *formal governance*.

Hypotheses

Regarding the previous discussion, the following hypothetical correlations were formulated:

 H11: Impact of Formal Governance (CF3) on Welcome Change (AP02).

H12: Impact of Formal Governance (CF3) on Constant Pace (AP08).

H13: Impact of Formal Governance (CF3) on Self-organization (AP11).

H14: Impact of Formal Governance (CF3) on Regular Improvement (AP12).

10.3.7 Innovation Management

IT project leaders from the second focus group reported the lack of innovation as a very critical issue that e-government practitioners encounter when trying to adopt agile methods (4 dot votes).

[Nerur et al., 2005] states that *"organizational cultures conducive to innovation may embrace agile methods more easily than those built around bureaucracy and formalization"*. The lack of innovation in the public sector was pointed out by several studies as a drawback to their digital transformation [Mergel, 2016; Holgersson et al., 2017].

Impact

Practitioners mainly pointed out the impact of low innovation on the ability of e-government projects to deliver value to citizens. It is therefore correlated with the ***valuable delivery of products*** principle (AP03). Moreover, the challenge was also reported as negative for the ***team motivation*** (AP05).

Causes

This challenge is already formulated as a context factor, i.e., ***low focus on innovation and learning***.

Hypotheses

H15: Impact of Low innovation (CF4) on Valuable Delivery (AP03).

H16: Impact of Low innovation (CF4) on Team Motivation (AP05).

Table 10.6.: Identified correlations between E-Gov Context Factors and Agility Goals

		E-Gov Context Factors					
		CF1	CF2	CF3	CF4	CF5	CF6
		Domain Attractiveness	Users Diversity	Governance	Innovation and Learning	Regulatory Compliance	Domain Complexity
		LOW	*HIGH*	*FORMAL*	*LOW*	*HIGH*	*HIGH*
AP01	Focus on User / Customer		(−) **H10**				
AP02	Welcome Change			(−) **H11**		(−) **H5**	
AP03	Frequent and valuable Delivery				(−) **H15**	(−) **H6**	(−) **H17**
AP04	Business/IT collaboration			(−) **H3**			
AP05	Team motivation and empowerment				(−) **H16**		
AP06	Face-to-face communication			(−) **H4**			
AP07	Focus on Working Software				(−) **H7**		(−) **H18**
AP08	Constant Pace			(−) **H12**		(−) **H8**	
AP09	Technical Excellence	(−) **H1**			(−) **H2**		
AP10	Focus on simplicity					(−) **H9**	
AP11	Self-organization			(−) **H13**			
AP12	Regular Improvement			(−) **H14**			

Agility Goals (principles)

10.3.8 Domain Complexity

The last e-government challenge that was reported by practitioners is a generic one. It relates to the complexity of the e-government domain. This challenges includes several sub-challenges among which the complex requirements, the security and quality concerns, the interoperability between systems, size and duration of the projects, etc (2 dot votes).

Impact

Some participants reported that this complexity is in tension with the notion of short time-boxed iterations and therefore with the ***Frequent Delivery*** principle (AP03). Indeed, regarding the domain complexity, the most important requirements need time to be integrated in the software. Moreover, they report that some requirements can't be delivered as a working piece of software which is in tension with the ***Focus on working software*** principle (AP07).

Causes

This challenge is already formulated as a context factor, i.e., ***High domain complexity***.

Hypotheses

H17: Impact of High Domain Complexity on Business/IT Collaboration (AP03)

H18: Impact of High Domain Complexity on Focus on Working Software (AP07)

10.4 Discussion

Table 10.6 shows that Formal Governance (CF3) is the most critical context factor since it seems to have the highest impact on agile principles. The lack of focus on innovation and learning and the high regulatory compliance seem also to have consequent impacts on agile methods implementation.

However, these results need to be considered carefully and further validated since the discussion mostly rely on the judgment and past (negative) experience. Further research should rely on a bigger set of data to empirically validate the hypotheses.

Moreover, our findings reflect the situation in Belgium and should be validated in other countries with different state structures, cultures and maturities in e-government.

Finally, the composition of the focus group may have influenced the results. In this study, the aspects of federal level and hierarchy positions were controlled but other factors could influence the results: individuals agile knowledge, digital literacy, size of the organization, etc.

10.5 Lessons Learned

In this part of the thesis, we investigated the influence of the e-government domain on agile methods implementation. We also formulated a reusable matrix diagram to help practitioners identify the discontinuing and challenging context factors and their impact on agile principles.

The representation of the learnings in the form of a matrix diagram constitute an interesting lead for structuring the knowledge base (see Chapter 12). In this study we chose to visualize the impact of the context factors on agile principles as stated in the Agile manifesto. A better guidance for practitioners would be to rather give an indication of which practices are usable, which would require customization or special consideration and which are not applicable. Therefore, a matrix representing the impact the context factors on operational agile practices would be a suitable representation of the knowledge base. Using such a matrix, more precise recommendations could be provided to practitioners to guide them implement custom methods.

At this stage, we have already reflected on leads for solutions, i.e., customizations that could be recommended in case of agile implementation in e-government context:

- **User Involvement**: leads for solutions could be found in the information system research base. [Simonofski et al., 2017] have established an inventory of participatory methods to stimulate user participation. Among these methods, one particular method may fit to stimulate user participation: Crowd-centric Requirements Engineering (CCRE) that applies the crowdsourcing paradigm to the requirements engineering process. With such a method, the large user group of e-government services could be targeted easier.

- **Regulatory Compliance**: this challenge may be handled by keeping a waterfall process at the beginning of the project or around the release time while implementing an agile process throughout the other

phases. For example, e-government projects are usually required to prepare plans for security emergencies on critical infrastructures. The preparation of such documents may require the intervention of several specialists and is often a precondition for the approval of the project. In such case, a waterfall process could precede the iterative development phase. Similarly, when several operations, including the verification of regulations (e.g., citizens privacy rights) are required to take software to production, a waterfall process can be implemented afterwards.

- **Formal Governance**: Challenges such as the lack of alignment between e-government stakeholders can be addressed through the implementation of a change management initiative at strategic levels of organization. Various change management models could be considered, e.g., the Satir process model and the Kotter's eight steps model [Cameron and Green, 2015].

In summary, by examining all the identified challenges in-depth, we should be able to provide a concrete agile methodology that fits the specificities of e-government.

In the next chapter, a similar study is conducted to investigate the impact of culture on agile methods implementation.

Chapter 11

Culture Case Study

Agile Cultural Challenges in Europe and Asia

In the previous two chapters, we investigated the implementation of agile methods in two distinct contexts and identified a set of influencing context factors. This chapter focuses on analyzing and understanding the relationships between agile methods customization and the cultural background of people.

Precisely, this study investigates how the European and Asian cultural backgrounds may impact agile methods implementation. We focused on three countries: Belgium, Malaysia and Singapore. Data about practices, challenges and impediments encountered by software development teams were gathered from interviews of 19 practitioners and from two agile events (focus groups). The results of the analysis are prioritized and discussed using the Hofstede Model for national cultures comparison.

The resulting outcome is a functional set of hypotheses representing the potential relationships between cultural traits and agile success factors. Although these preliminary findings call for more validation, they motivate further research and offer an important venue for practitioners and coaches regarding the fine tuning of agile methods with regards to culture. This knowledge is particularly relevant for professionals working in globally distributed projects.

The remainder of this chapter is organized as follows. Section 11.1 provides an overview on selected related works. Sections 11.2 and 11.3 present our research methodology and results. Section 11.4 explores the limitations of this preliminary study and provides closing comments. Finally, Section 11.5 synthesizes the lessons learned from this case study and explains how it contributes to the thesis.

11.1 **Background**

Agile methods initially spread in North America before the expected benefits of agile software development began attracting interest and adoption grew in other parts of the world. More than a decade after the release of the first methodologies, practitioners all over the world have accumulated field experience working with numerous practices and techniques and trying to achieve higher maturity.

Recently, this search for maturity has been focusing more and more on the context awareness of agile methods. Among the context-related factors, the cultural background (i.e., mind-set, values and behaviors that shape culture over time) of the project-team is one of the most crucial and yet neglected aspects addressed by research.

According to [Laroche, 2012], culture consists in *"patterned ways of thinking, feeling, and reacting, acquired and transmitted mainly by symbols, constituting the distinctive achievement of human groups, including their embodiments in artifacts; the essence of culture consists of traditional (i.e., historically derived and selected) ideas and especially their attracted values"*.

As outlined by [MacGregor et al., 2005], cultural factors are recognized as critical issues that influence the teams way of working and therefore should be taken into account when discussing the ability of software teams to work with agile methods effectively and successfully.

Several studies have been conducted in software engineering research to investigate the factors influencing agile methods implementation [Chow and Cao, 2008; Stankovic et al., 2013; Kruchten, 2013]. However, most of them do not consider the inter-cultural differences and human factors. The few existing studies to do so are related to Global Software Development (GSD) research.

[Lee and Yong, 2010] presents a set of challenges encountered by Yahoo in a globally distributed project across Asia Pacific, Europe and the US. These are categorized into 3 areas: communication, control and trust. Cultural differences seem to make the challenges even more difficult. The study reports that communication lines were not always as open as expected, since Asian teams were sometimes reluctant to discuss negative issues. It also explains how local business conditions and sensitive issues may create disjointed and conflicting priorities within the product backlog.

[Fowler, 2006] reports lessons learned in ThoughtWorks concerning an off-shore development experience in Bangalore India to support software development projects in North America and Europe. It argues that Asian cultures

may reinforce deference to superiors which contradicts the value of team autonomy. This makes the communication harder: Asian team members may be discouraged from exposing problems, warning about non-feasible deadlines, or proposing alternatives to perceived directives from superiors. The authors end the discussion about cultural differences by explaining the sensitiveness of the problem and that it is obviously not specific to Asia since we may find the same problems or even worse in some western companies.

[Holmström et al., 2006] explores how agile practices can reduce sociocultural distance and details the risks and opportunities in the context of global software development. The results also show that risks are related to communication, inconsistency in work practices and different perceptions of authority/hierarchy.

As we may see, the cultural influence on agile development is already reported in the body of knowledge. However, it often refers to the organizational culture and not the cultural background of people. Moreover, the few studies that refers the issue are generally focused on GSD research.

11.2 Methodology

As an exploratory methodology to find out and validate the cultural challenges and to analyze them, we decided as in the previous chapter to follow a grounded theory approach (see Section 4.1). Following this methodology, data was iteratively coded into theory at each step of the process.

The following sections detail the proposed methodology.

11.2.1 Objectives

Studies such as [Borchers, 2003] tend to show that cultural factors should not be disregarded when evaluating the efficiency and/or relevance of software engineering practices. As pointed out in Section 11.1, some studies tackle the challenges related to cultural differences within agile development teams. However, these efforts regarding cultural aspects in agile software development tend to focus on the conflict arising between team members coming from different cultural backgrounds (in the context of globally distributed projects).

Our research aims at extending this focus by questioning the impact of culture in homogeneous cultural environments. Due to agile methods originating

from North America, it stands to reason that many agile practices may in fact rely on observations or assumptions based on views and behaviors shared by American practitioners. Successfully applying such practices would therefore suppose the ability to conform to said views and behaviors. In multi-cultural development teams, the mitigation of culturally induced mental blocks through conflict may lead to success. However, in culturally homogeneous development teams outside of North America, the inability to conform to specific views may lead to the disregard, inefficiency or failure of specific agile practices.

In order to challenge this hypothesis, we've been investigating the practices of 9 agile teams spread across 3 culturally diverse contexts outside of North America, i.e., Belgium (BE), Malaysia (MY) and Singapore (SG). Through this preliminary study, our goal is to address the following research questions within these 3 contexts:

1: Is there an observable relationship between adopted practices and cultural background?

    a) Can we rely on a commonly accepted typology of cultural factors?

    b) Is there a noticeable difference in the investigated contexts?

2: Can we formulate plausible hypotheses about potential relationships between cultural factors and agile practices adequacy?

These specific questions relate to RQ2 and RQ3, the second and third research questions of the thesis (see Section 3.5).

11.2.2 Data Collection

The data of our study has been collected using semi-structured interviews (see Appendix A) and focus groups. We decided to stop collecting data after having interviewed 19 practitioners as each challenge has reached saturation and no more original findings were determined.

The interviewees (see Table 11.1) are distributed as follows:

- in BE: 13 practitioners (3 teams) interviewed during 2013 (2h each)

- in MY: 4 practitioners (one per team) interviewed between April 2016 and June 2016 (1h30 each)

- in SG: 2 practitioners (one per team) interviewed between April 2016 and June 2016 (1h30 each)

We asked the interviewees about their teams' context: size, business domain, potential geographical distribution, years of agile experience, documentation strategy, project duration, iteration length, etc. We also asked about the organizational culture, i.e., compliances, management support to agile, leadership style and innovativeness. We used a formal interview guide with the list of question and topics to be covered. To drive the discussion, we asked them to rank some context aspects such as the domain complexity, the scope variability, the customer involvement and the team self-organization. We also included open-ended questions which helped identifying new ways of seeing and understanding the topic.

Table 11.1.: Interviewees profiles

ID	Team	Country	Role	Agile Exp. Level
P1	T1	BE	Analyst	Novice
P2	T1	BE	Developer	Intermediate
P3	T1	BE	Developer	Novice
P4	T1	BE	Product Owner	Intermediate
P5	T2	BE	Analyst	Intermediate
P6	T2	BE	QA Manager	Novice
P7	T2	BE	Architect	Expert
P8	T2	BE	Developer	Novice
P9	T3	BE	Product Owner	Novice
P10	T3	BE	Architect	Intermediate
P11	T3	BE	Developer	Novice
P12	T3	BE	Project Manager	Expert
P13	T3	BE	Product Owner	Intermediate
P14	T3	BE	Architect	Novice
P15	T3	BE	Portfolio Manager	Novice
P16	T4	MY	Dev. Manager	Expert
P17	T5	MY	Product Owner/Coach	Intermediate
P18	T6	MY	Product Owner	Expert
P19	T7	MY	Dev. Manager	Expert
P20	T8	SN	Scrum Master	Novice
P21	T9	SN	Scrum Master/Quality Manager	Expert

These informations are synthesized in Table 11.2. We also discussed in details the challenges and impediments that the teams encounter and asked (when needed) to explain these challenges according to the team members cultural background.

As we can see in Table 11.1, the selection of interviewees is unbalanced (4 in MY, 2 in SG and 13 in BE). This is because we relied on convenience

Table 11.2.: Teams Overview

ID		Size	Domain	Ag. Exp.	Proj. Length	Itera. Length	Req. Change	Method
T1	BE	10	E-Gov.	1 y	2 y	2 w	Low	Scrum
T2	BE	6	E-Gov.	1 y	1 y	4 w	Low	Scrum
T3	BE	15	E-Gov.	2 y	1 y	2 w	Low	Scrum
T4	MY	20	B2B	8 y	3 m	na	Medium	Kanban/Lean
T5	MY	7	Real estate	3 y	3 m	2 w	High	Scrum/Kanban
T6	MY	8	Oil&Gas, E-Gov	10 y	6 m	2 w	Low	Scrum/Kanban
T7	MY	15	E-Gov.	5 y	2 y	4 w	High	Custom
T8	SN	5	E-Gaming	2 y	6 m	2 w	Low	Scrum/Kanban
T9	SN	7	B2B, Banking	4 y	1 y	6 w	Medium	Scrum/AUP

sampling which is a non-probability sampling technique where subjects are selected because of their convenient accessibility and proximity to the researchers.

We also collected data from the following focus groups:

- in BE: two project retrospectives that we organized in 2013 with the teams T1 (10 participants) and T3 (10 participants) (see Table 11.2)

- in MY: two agile software development meetings in April 2016 and June 2016. In every meeting, 3 sub-groups were created of ~5 participants each.

The focus groups were conducted according to the guidelines and best practices reported in [Krueger, 2014; Morgan, 1996]. Following these guidelines, each focus group involved from 5 to 10 participants (selected based on convenience sampling).

The two focus groups conducted in BE consisted of project retrospectives in a transitioning context. The team-level challenges in implementing Scrum to their specific context were discussed and categorized using an affinity diagram. We also asked them to evaluate the severity of the challenges (low, medium or high). Then we questioned them about the potential influence of the cultural factors.

In MY, the two meetings of ~15 participant each were animated by the researcher. Participants were of different profiles: coaches and practitioners from mature teams and newly transitioning teams. In each meeting, we

asked the participants to form 3 sub-groups for a free discussion of the issues and drawbacks they encounter when implementing agile methods. A reporter from each sub-group was asked to synthesize the most critical challenges and to evaluate their (low, medium or high) and an affinity diagram is constructed collaboratively with the researcher playing the role of the facilitator. The question of the influence of the Asian culture on agile methods implementation emerged naturally.

The advantage of collecting data from focus groups is that people tend to discuss openly their challenges.

11.2.3 Data Analysis

In order to be able to analyze and compare the cultural backgrounds, we first need to determine the discriminating dimensions or variables to describe culture-related positions.

Several models have been designed by researchers to conceptualize cultural differences. These can be categorized into three types: single dimension models, multi-dimensional models and historical-social models [Morden, 1999]. The first two types of models assume that people have a distinctive, identifiable and influential national culture that we can dimensionalize, potentially measure and operationalize.

The historical-social models, however, question the distinctiveness of national cultures and provide broader perspectives which cross geographical boarders. Such models identify key historical-social variables to analyze cultural backgrounds. These variables are usually context-specific and thus may not be useful for cross-cultural understanding. For example, [Chen, 2004] and [Cragg, 1995] propose south east Asian management models and study the influence of specific variables such as Confucianism or Taoism which are not representative of other cultures. Thus, this type of models was excluded.

Table 11.3 synthesizes the models of national culture that we explored. As we can see, culture has been defined in different ways. Each model has its own set of discriminating factors characterizing the concept of national culture. All these models are based on the assumption that cultures can be distinguished based on differences in what they value. That is, *"some cultures place a high value on equality among individuals, while others place a high value on hierarchies. Likewise, some cultures place a high value on certainty in everyday life and have difficulty coping with unanticipated events, while others have a greater tolerance for ambiguity"* [Bhagat and Steers,

2009]. A comparison and an alignment of the multi-dimensional models can be found in [Bhagat and Steers, 2009].

Table 11.3.: Models of National Culture

	Model	Cultural dimension(s)
Single dimension	[Hall et al., 1959]	High/Low Context
	[Lewis, 2011]	Monochronic/Polychronic
	[Triandis, 1995]	Ideocentric/Allocentric
	[Fukuyama, 1995]	High/Low Trust
Multiple dimensions	[Kluckhohn and Strodtbeck, 1961]	Relationship with nature
		Relationship with people
		Human activities
		Relation with time
		Human nature
	[Hofstede, 2011]	Power Distance
		Uncertainty Avoidance
		Individualism/Collectivism
		Masculinity/Femininity
		Indulgence
	[Hampden and Trompenaars, 1993]	Universalism/Particularism
		Individualism/Collectivism
		Specific/Diffuse roles
		Neutral/Affective emotions
		Achieved/Ascribed social status
		Time perspective
		Inner/outer directed
	[House et al., 2004]	Power Distance
		Uncertainty Avoidance
		Human Orientation
		Institutional Collectivism
		In-Group Collectivism
		Assertiveness
		Gender Egalitarianism
		Future Orientation
		Performance Orientation

Among all these definitions, the Hofstede's model [Hofstede, 2011] is mostly cited in Information System (IS) research. Much of the literature concerned with cultural and cross-cultural issues in the IS field has relied on Hofstede's work, typically the study of global software development teams [Borchers, 2003][MacGregor et al., 2005]. This is maybe not surprising, given that the Hofstede typology of culture has been derived from empirically strong study of employee values at a major multinational IT corporation (IBM). The original project included 60,000 employees and over 40 countries. In total, Hofstede carried out his research over a period of 15 years and analyzed some 116 000 questionnaires from 67 countries in a single multinational corporation.

The validity of the Hofstede model has been discussed in several research. Criticisms can be found in [Schmitz and Weber, 2014] and [McSweeney, 2002] and concern the research methodology (e.g., the unsuitability of surveys, the uniqueness of the data source: IBM and the measure of national culture as a statistical average of individuals' views), the dimensions validity (e.g., [Schmitz and Weber, 2014] argue that the dimension of "Uncertainty Avoidance" have lost relevance over the years) and the model's crucial assumptions (e.g., the assumption of a shared and stable national culture). Arguments for the model include its strong empirical basis (a large data set), its applicability to different contexts and the sufficiency of its variables to study the differences between national cultures (the conceptual and statistical independence of the variables). In [Hofstede, 2002], Hofstede provide a comprehensive response to the model criticisms and argues that nations may not be the best unit of culture analysis but they are a legitimate one regarding the accessibility and availability of information.

Based on the aforementioned merits, we decided to analyze the challenges of implementing agile practices according to the Hofstede Model.

The model proposes 6 factors to characterize the national culture:

- **Power Distance (PDI)**: indicates the extent to which the less powerful members of organizations accept and expect that power is distributed unequally. People in societies with high PDI score accept the hierarchical order easily. People in societies with low PDI, consider hierarchy as established only for convenience and try to equalize the distribution of power.

- **Individualism vs. Collectivism (IDV)**: refers to the degree to which people in a society are integrated into groups. In individualist societies (high IDV), the ties between individuals are loose: individuals are expected to take care of only themselves and the immediate groups

to which they belong. In collectivist societies (low IDV), people belong to "in groups" that take care of them.

- **Masculinity vs. Femininity (MAS)**: refers to the distribution of values such as assertiveness, achievement, power and control between the genders. Masculine society (high MAS) indicates maximum emotional and social role differentiation between the genders and Feminine society (low MAS) indicates minimum emotional and social role differentiation between the genders.

- **Uncertainty Avoidance (UAI)**: evaluates the degree to which a society is reluctant to ambiguity and unstructured situations. Societies with a high UAI score feel threatened by ambiguous and unknown situations. In societies with a low UAI, uncertainty is accepted as an inherent in life and "each day is taken as it comes".

- **Long-term vs. Short-term Time Orientation (LTO)**: is related to the choice of focus of people's efforts; the future or the present and past. In a long-term oriented society (high LTO), people attach more importance to future. They prescribes to long term commitments in a pragmatic way: they encourage efforts in the present to prepare the future. In short-term oriented society (low LTO), people attach more importance to present, i.e., they prefer to maintain past/time-honored traditions and view societal change with suspicion.

- **Indulgence vs. Restraint (IDG)**: measures the extent to which people express their desires and impulses. An indulgent society (high IDG) allows relatively free gratification and natural human desires related to enjoying life. Restraint (low IDG) stands for a society that controls gratification of needs and regulates it by means of strict social norms.

Figure 11.1 compares the 6 scores for BE, MY and SG. The data is provided by [Hofstede, 2010]. The scale runs from 0 - 100 with 50 as an average score. The rule of thumb of the Hofstede model is that if a score is under 50, the culture scores relatively low on that scale and if any score is over 50, the culture scores relatively high on that scale. This means that the country scores on the dimensions are relative. In other words, culture can be only used meaningfully by comparison.

At the PDI dimension, MY is the country which scores the higher. This explains the culture of deference to superiors: the management in the country tends to follow a command-and-control style. It may also indicates that superiors may have privileges and may be inaccessible. SG and BE have also relatively high PDI scores. The IDV score in BE is the higher, which
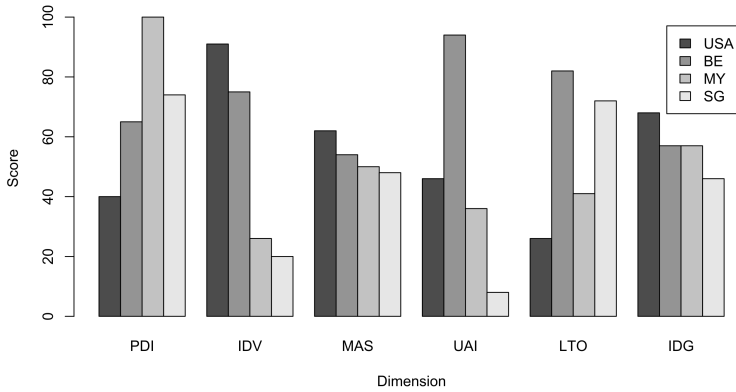
Figure 11.1.: US, BE, MY and SG Cultural Dimensions according to [Hofstede, 2011]

indicates that the BE society is more individualist than MY and SG. The MAS dimension ranks average in BE, MY and SG (respectively 54, 50 and 48) which indicates that it will not be effective to explain the potential observations and derive hypotheses. We therefore excluded it.

BE has the highest score in the UAI dimension. This expresses a difficulty of coping with uncertainty and unanticipated situations. At the LTO dimension, BE and SG rank high, while MY ranks lower. This indicates that BE and SG are long-term oriented while MY is short-term oriented.

The IDG score of BE, MY are of 57 which makes them more indulgent societies than SG which score is a bit lower (46).

11.3 Results

Table 11.4 summarizes the interviews and focus groups findings. These are categorized and prioritized according to the challenges severity in BE, MY and SG. Table 11.6 summarizes the hypotheses identified in the following sections.

Table 11.4.: Comparison of challenges

	Priorities	
	BE	MY/SN
Team commitment to practices	Medium	Low
Team Empowerment	Medium	High
Team transparency and cohesion	Low	High
Team's external communication	High	Low
Team multidisciplinarity	Medium	-
Team motivation	Low	Medium
Customer involvement	Medium	Medium
Management involvement	Low	Low
Process improvement	Medium	High

11.3.1 Team commitment to Practices

To develop a detailed understanding of the deployed agile practices, we asked the 21 practitioners to rank from none to high the level of adoption of 40 agile practices[1] (see an overview in Table 11.5).

Most of the practitioners in MY/SG assign high or medium ranking for most of the practices. During the focus groups in MY, the majority of participants reported that team members show high level of commitment to the agile practices: they respect the method guidelines as much as possible. P18 (see Table 11.1) reported that team commitment to agile practices such as retrospectives decreases when the process facilitator (Scrum master) is not enough leading and motivating.

In BE, during a project retrospective, T2 (see Table 11.2) reported some violations to process rules such as the daily stand-ups duration (same meeting for two sub-projects which makes the meeting longer than it should be). However, these violations to process rules are perceived as negative (referred to as "to drop" practices during retrospectives).

The relatively high level of commitment to practices can be explained by the PDI factor: cultures with high PDI usually accept established rules easier. With a score of 100, MY has the highest score of PDI. The PDI scores for SG and BE are also relatively high (respectively 74 and 65).

Another critical issue reported by BE practitioners is the reluctance to the newly introduced agile practices. We especially observed this in T2 which

---

[1]Agile software development practices as defined by the Agile alliance [Alliance, 2008]

was not enough prepared to the transition from traditional skill-centric siloed organization (structured in specialized units architecture, business analysis, QA, etc.).

According to the Hofstede model, the acceptance of change is mainly correlated to the UAI and LTO dimensions. Since BE have a high LTO score, the commitment to accept new practices when the change is well prepared should also be high. When the change is not enough prepared (as it was in T2), cultures which tend to avoid uncertainty (high UAI) will probably reject it. This may explain the reluctance towards the introduction of new practices in T2. The problem was not raised by MY and SG practitioners. It seems reasonable to assume that this is may be due to the fact that the transition to agile methods in all the Asian companies that we interviewed follows a top-down model, i.e., the change comes from the management. Since MY and SG have higher PDI scores, teams should be more prone to commit to introduced practices.

Hypotheses

H1: Positive impact of high PDI scores on commitment to (new) practices

H2: Negative impact of high UAI scores on commitment to (new) practices (and vice versa)

H3: Positive impact of high LTO scores on commitment to (new) practices (if change is well prepared)

11.3.2 Team Empowerment

Traditional teamwork has the project manager controlling everything. The manager is given responsibility and authority over all decisions and plans. In Agile, team empowerment is an important feature that removes bottlenecked decision making and therefore allows the team to be more efficient.

Several practitioners from MY and SG reported the lack of team empowerment as a critical issue to their Agile implementations: P16, P17, P19, P20 (see Table 11.1). Participants to the focus groups reported the same issue. They all relate this to the command-and-control mindset. For instance, P20 (see Table 11.1) reports that the management gives the team the freedom to decide about their way of working but the team members *"don't dare to think out of the box"* because of their lack of experience.

In BE teams, we observed a misunderstanding of team empowerment: T3 (see Table 11.2) reported that the team members were considered accountable

Table 11.5.: Overview of practices adoption level

	Practitioners ranking of commitment to practices [1]							
	BE				MY/SG			
	High	Medium	Low	None	High	Medium	Low	None
Collective Ownership	4	5	3	2	5	1	0	0
Stakeholders Participation	11	4	0	0	4	2	0	0
Customer Representative	7	7	1	0	6	0	0	0
Time-boxed Iterations	7	6	2	0	3	2	0	1
Task board	3	7	4	0	5	0	0	1
Frequent Releases	11	4	0	0	4	0	0	2
Agile Modeling	3	3	6	2	4	0	0	2
Test Driven Development	1	3	8	3	1	1	1	3
Acceptance Testing	10	2	2	1	5	1	0	0
Unit Testing	6	5	3	1	5	1	0	0
Pair-programming	0	4	8	3	4	0	1	1
Code Refactoring	3	3	6	0	2	2	1	1
Continuous Integration	7	7	0	1	3	2	1	0
Daily Stand-ups	6	8	0	1	5	0	0	1
...								
Percentage	37.62%	32.38%	20.47%	6.66%	66.66%	14.28%	4.76%	14.28%

for some business decisions and priorities definition. They also reported that they were sometimes confronted directly to the customer demands. The proximity of customer in itself is positive but this should not interfere on the team work during the iteration. The relatively high PDI score in BE can also explain the customer interference. In countries showing high PDI, hierarchy is well established and superiors, i.e., the customers in this case, may consider that they have special privileges such as asking the team for changes directly and anytime they want to.

Hypothesis

H4: Negative impact of high PDI scores on team empowerment (and vice versa)

11.3.3 Team Transparency and Cohesion

Transparency refers to open communication, including the communication on negative points and represents an important agile software development value. From intensive collaboration and great team cohesion emerges transparency. The two concepts of transparency and cohesion are therefore interrelated.

During focus groups and interviews, lack of transparency is reported by several MY and SG practitioners as very critical. P16, P18, P20 and P21 (which were interviewed separately)(see Table 11.1) all reported that team members have a tendency to not expose problems such as non-feasible deadlines or technical difficulties. They refer to the command-and-control mindset as a possible explanation. In such a context, i.e., a high PDI score, [Hofstede, 2011] argues that *"it is advised for the manager to establish a second level of communication, having a personal contact with everybody in the structure, allowing to give the impression that everybody is important in the organization, although unequal"*.

We hypothesize that the lack of transparency can be explained by both the PDI and IDG dimension, depending on the real reason that pushes individuals to not communicate about problems. If this is related to uneasiness with superiors, then the cause of the problem would be the high PDI score in MY and SG. If the cause is the uneasiness with other team members (which are at the same hierarchal level), then the communication is threatened by the IDG dimension. IDG score is 57 in MY and 46 in SG.

BE practitioners report a relatively good communication inside the team. During the project retrospective of T3, a critical issue concerning each other's responsibilities and workload was reported. The issue was never been

mentioned before which might suggest a lack of internal communication. The team also reported having communication problems with the external environment (see Section 11.3.4).

Hypotheses

H5: Negative impact of high PDI scores on team transparency (and vice versa)

H6: Positive impact of high IDG scores on team transparency (and vice versa)

11.3.4 Team's External Communication

In BE, T1 and T2 (see Table 11.2) report challenges related to the communication with external teams. The IDV dimension determines the degree to which individuals are socialized and therefore may be correlated to this issue. The IDV score is relatively high in BE, which enhances the degree of interdependence inside the team (cohesion) and in contrast decreases external relationships. This could mean that the Belgians favor focusing on their immediate entourage rather than belonging to larger groups.

Another type of external communication is the team's integration within the software engineering community. Communicating with other agile teams represents an excellent opportunity to leverage knowledge. In MY, agile software development meetings are organized each month in a different host company. The initiative demonstrates a willingness to mature practices. A low IDV score indicates a collectivist culture were strong relationships exist. This is what was observed in MY and SG agile software development events.

Hypothesis

H7: Negative impact of high IDV scores on team's external communication (and vice versa)

11.3.5 Team Multidisciplinarity

Multidisciplinary or cross-functional teams are believed to be essential in developing innovative solutions to many types of business problems. It refers to expertise diversity and shared knowledge emerging from intensive

collaboration inside the team. During the semi-structured interviews, we asked the participants to comment the way they are managing this Agile value.

The interviewed teams in BE reported a lack of multidisciplinarity and ranked it as a mid-level challenge (see Table 11.2). This may be explained by the fact that BE teams come originally from a environment formally structured in skill-centric silos. Individuals are already aware of the issue and making efforts to overcome it by collaborating closer and communicating as much as possible about each other workload.

We hypothesize that the organizational culture (conjugated with a high UAI) favors this issue and that the LTO orientation moderate it. In fact, high UAI (94 in BE) relates to a culture where members feel threatened by unknown situations (e.g., learning new skills) and high LTO (82 in BE) refers to a pragmatic culture where individuals invest time in the present to prepare for future change.

In MY, P17, P19 reported unsatisfactory levels of multidisciplinarity which may confirm the correlation between this value and the UAI dimension (MY has a UAI score of 42). SG practitioners, which have a very low UAI score of 8, have not exposed any issue related with multidisciplinarity.

Hypotheses

H8: Negative impact of high UAI scores on team multidisciplinarity (and vice versa)

H9: Positive impact of high LTO scores on team multidisciplinarity

11.3.6 Team Motivation

Successful organizations understand that teams enthusiasm and motivation is an essential factor for achieving high productivity [Asproni, 2004]. Motivating the team is therefore considered as a critical success factor of Agile deployment for coaches and managers.

Practitioners from BE showed a certain satisfaction about their motivation (referred to as "to keep" during the projects retrospectives) while some MY and SG practitioners showed concerns about it. Practitioners P16, P20 and P21 (see Table 11.1) from MY and SG reported team fatigue as a serious concern, mainly related to very critical time schedules. Team members therefore loose motivation to some practices which they become to consider

as unnecessarily time-consuming such as retrospectives (but they commit to them as explained in Section 11.3.1).

One of the cultural dimensions that might be related to motivation is IDG. With a score of 57, BE and MY societies are considered as relatively indulgent. In high indulgence societies, individuals tend to show a positive attitude which helps to maintain team motivation. With a score of 47, the IDG dimension is below the average in SG. This may be explain why the two interviewed practitioners from SG report team fatigue as an essential challenge.

Hypothesis

 H10: Positive impact of high IDG scores on team motivation (and vice versa)

11.3.7 Customer Involvement

Active user involvement is a key principle of Agile to enable clear understanding and appropriate prioritizing of requirements. It refers to daily basis collaboration between the team and the customer representatives. We asked the participants to explain their relationship with the customers (or customers' representatives). If the participants don't have a clear opinion, we ask them to rank this from 0 to 5 to facilitate the discussion.

Some of the MY and SG seem to be unsatisfied about the customer commitment level. P19 (see Table 11.2) reports concerns about the customer relationship and explains it regarding the e-Gov domain: *"How to convince customers such as representatives of ministries and government agencies?"*.

BE participants have also expressed concerns about it. T3 (see Table 11.2), the same team that reports being sometimes confronted directly to customer demands during the iteration (see Section 11.3.2), raises the problem of customer representatives' absence in some iteration planning sessions, which impacts on the team velocity and constrains them to make choices about business and priorities.

We hypothesize that the customer's availability issue is related to the organizational culture primarily but may also be impacted by the national culture, in particular the PDI cultural dimension. A high PDI may be correlated with privileged superiors (i.e., the customers in this case), not always following the rules.

Hypothesis

H11: Negative impact of high PDI scores on customer involvement (and vice versa)

11.3.8 Continuous Improvement

Assessing and adapting its way of working continuously is a core Agile value. Implementing this value was found challenging in BE, MY and SG, probably because it requires ongoing attention. T1, T2, T3, P16, P20 and P21 (see Table 11.1) particularly insisted on the lack of commitment to process improvement. When we ask them to detail the reasons for this, they report: (1) the insufficient level of team empowerment (improvement initiatives usually come from top management levels) and (2) the lack of time to think about what should be improved.

As explained in Section 11.3.2, the lack of empowerment might be caused by the PDI dimension (H4) and we hypothesize that the lack of time might actually be a consequence of the LTO dimension. In fact, BE and SG are long-term oriented which means that the individuals need time to effectively adjust their way of working. This might explain the feeling of frustration due to the lack of time.

However, all the interviewed practitioners seem to be aware about the issue: P16, P17, P18 and P20 (see Table 11.1) reported growing interest to mature practices using maturity models such as the Kotters 8 steps to change [Kotter et al., 1995] and the Satir Interaction Model [Satir and Banmen, 1991].

In BE, process improvement appears to be both a team's and management's issue. The 3 teams we interviewed reported considering retrospectives very seriously. P15 (see Table 11.1), a portfolio manager, showed great interest in formal guidance and documentation of the process. This initiative of process modeling may seem opposite to agile values (since work improvement should be the essential measure, not process documentation) but this actually should be considered according to the cultural background and the organization context.

BE has a high LTO score, which suggests that individuals will be more pragmatic in change management: they encourage efforts as a way to prepare for the future. Of course, we should mitigate this according to the business domain. In e-government, heavy formalization is a requirement in itself (see Section 10.3.3) and this is something we observed in BE and MY: T1, T2, T3, T6 and T7 (see Table 11.2).

Hypotheses

H12: Negative impact of high PDI scores on continuous process improvement (and vice versa)

H13: Positive impact of high LTO scores on continuous process improvement (and vice versa)

11.4 Discussion

In order to understand the impact of the cultural background on agile software development, we conducted a study in culturally diverse contexts (BE, MY, SG). We collected data about practices, challenges and impediments encountered by teams in each context. As earlier discussed, we have been able to observe potential relationships (Question 1) between agile methods adoption and 5 cultural factors (out of the 6 defined in the Hofstede model): PDI, IDV, UAI, LTO and IDG. We derived 16 hypotheses (Question 2) (see Table 11.3) which basically represent positive or negative correlations between cultural factors and agile success factors. The results of the study tend to show that the Hofstede model helps us find relevant hypotheses regarding the impact of cultural background on agile methods implementation.

In order to validate the plausibility of the hypotheses (Question 2), we tried to find evidence in the literature. Studies corroborating the hypotheses were found, e.g., [Asnawi et al., 2011] and [Asnawi et al., 2012] which discuss the management role in agile projects in MY and seem to consolidate **H1**. These studies tend to show that the hypotheses we propose, although not always expressed explicitly, are based on some concrete reality.

We also found studies that assert the impacts of some cultural factors (coherent with the hypotheses we derived) and detail their solutions to overcome the adoption challenges.

For example, [Yasuoka and Sakurai, 2012] investigates the applicability of the participatory design practice (coming originally from the Scandinavian and North American context) in the Japanese context. It refers to **H5** and explains how team transparency and cohesion can be achieved by building confidence (creating a fun environment) and mitigating the LTO of Japanese by installing certain conditions such as a feel of urgency. In the end, our study seems to show that cultural background has a tangible impact on how agile practices are perceived and applied.

Table 11.6.: Hypothetical correlation between Agile challenges and cultural factors

	Cultural factors									
	PDI		IDV		UAI		LTO		IDG	
Agility Goals (Success Factors)	High	Low	High	Low	High	Low	High	Low	High	Low
Commitment to practices	(+)	(?) **H1**			(−)	(+) **H2**	(+)	(?) **H3**		
Team Empowerment	(−)	(+) **H4**								
Transparency and cohesion	(−)	(+) **H5**							(+)	(−) **H6**
External Communication			(−)	(+) **H7**						
Team multidisciplinarity					(−)	(+) **H8**	(+)	(?) **H9**		
Team motivation									(+)	(−) **H10**
Customer involvement	(−)	(+) **H11**								
Process improvement	(−)	(+) **H12**					(+)	(−) **H13**		

However, the study still has several limitations that should be addressed. Firstly, as [Borchers, 2003] reminds us, any study addressing culture-related aspects should be cautious not to stereotype individuals through cultural traits. To an extent, our study is not void of such stereotyping. Indeed, by focusing on cultural background instead of individual mental models, our research simplifies complex human relationships but provides a good entry point for practitioners. The study should be regarded as pointing out *potential* links between cultural background and ease of adoption, *individual preferences notwithstanding*.

Furthermore, the definition of culture in terms of nations is problematic and somewhat simplistic. In fact, studying socio-cultural differences is sensitive. Some models explore a more dynamic view of culture, one that sees culture as contested, temporal and emergent. To this regards, [Hofstede, 2002] responds that national culture may not be the best unit of cultural differences analysis but its use is legitimized by the accessibility and availability of information.

The results need also to be considered carefully since more empirical data are required to confirm the practitioners' opinion. Moreover, the results might be affected by the non-systematic analysis method and by the imbalance of data.

Another significant threat to validity is the possible influence of other factors on the challenges listed in Table 11.4. In fact, the observations may be caused by other variables (independent from the national culture) since the participant groups share more than just cultural differences. Other factors, namely, the organizational culture and the projects' constraints (size, experience, business domain, technology, etc.) may also have an impact on the reported challenges. In order to validate the actual impact of cultural background, other varying factors should be controlled.

The decision to consider only the national culture for this analysis was made for simplification reasons (we decided to study one factor at a time). It is motivated by the fact that during the first focus group, several practitioners reported culture as a critical challenge towards implementing agile methods. Another argument for this decision, may be found in [Child, 1979] which studies the impact of organizational culture on management practices. He argues that macro-level variables characterizing the cultures of organizations, may not be the most determinant. The micro-level variables, i.e., the behavior of people, continue to retain their cultural identity and have significant impact on the work practices. Therefore, the cultural background of people ought to be studied.

Regarding data collection, the study is based on small data sets that cannot claim statistical relevance in any way. The study should be regarded as preliminary, aiming at pointing out relevant hypotheses to be validated in the future through more sophisticated surveys (i.e., larger scale and more controlled experiments) and using correlation and regression analysis.

Finally, due to the theoretical coding of the relationships between cultural dimensions and data, our analysis of the data may be undermined by subjective opinions. The input of practitioners during the redaction of this paper is one mechanism designed to overcome this risk but is still not sufficient. A more systematic approach should be followed in future research. Increasing the number of analysts (and therefore of inputs) should guarantee a more objective outcome.

Regarding the aforementioned discussion, the study should be considered as preliminary, aiming at pointing out potentially relevant hypotheses that have to be further validated in the future. Future perspectives include the validation of the hypotheses through larger surveys and more controlled experiments (see Chapter 14). Future studies should also switch from convenience sampling of participants to a more controlled selection (which is only possible if the number of participants is high enough). They should also go further by investigating the same research question in other contexts than those we studied (BE, MY and SG).

11.5 Lessons Learned

As previously discussed, the learnings from this chapter can be valuable for practitioners implementing agile software development in distributed environments as well as practitioners working in culturally homogeneous teams, that is, teams where no possibility of conflicts arise from cultural differences.

The direct implication for the AMQuICk framework is that the cultural background is another context dimension to be considered in the customization process (see Chapter 12).

As in the previous study, we also used the form of a matrix diagram to represent the learnings which initiates the idea of structuring the knowledge base using customization decision matrices (see Chapter 12). Using such matrices, more precise recommendations could be provided to practitioners in a particular cultural configuration: which practices are recommended,

which would require customization or special consideration and which are not applicable.

Another learning of this study is that the cultural dimension can be measured using the 6 factors of the Hofstede model. A cultural self-assessment survey can be conducted internally in the organization to precisely measure the team members individual culture. A self-assessment questionnaire or the original Hofstede questionnaire[1] may be reused. The results of the assessment would help to decide about the customization guidelines using culture-related decision matrices documented by experts and/or other practitioners (see the example in Table 13.3).

Finally, the fact that context factors can be measured using specific models helped us introduce a new concept in AMQuICk Essence, the *Assessment Model* (see Chapter 12) which aims at providing precise context indicators.

---

[1] `https://www.surveymonkey.com/r/QVY9YXV`

Part V.

Customization and Capitalization Perspective

This part of the dissertation generalizes the learnings of the exploratory studies and proposes an extension to the AMQuICk framework accordingly.

More precisely, Chapter 12 describes the theoretical process to be followed by practitioners to instantiate a suitable agile method and presents the AMQuICk customization matrix, a core AMQuICk artifact used to represent the practitioners expertise. It also presents some facilitation tools proposed as a guidance to help practitioners easily capitalize their customization knwoledge at an organizational level.

Chapter 13 discusses the usability of the proposed AMQuICk artifacts based on illustrative examples and on a complete case study.

Chapter 12

AMQuICk Customization and Capitalization

We explored in the previous chapters a number of challenging contexts where agile methods have been deployed and identified a set of context factors influencing the success of such deployment. We also captured and structured the identified customization guidelines so they can be easily reused by other practitioners working on similar contexts. The aim of this chapter is to generalize the learnings of the exploratory studies and to propose an extension to the AMQuICk framework accordingly.

In Section 12.1, we first synthesize the case studies learnings and propose a theoretical process to be followed by practitioners to instantiate a suitable agile method. Then, in Sections 12.2 and 12.3, we discuss the extension of the AMQuICk Essence metamodel to include context and customization related concepts. In Section 12.4, we present the AMQuICk customization matrix, a visual artifact used to represent the practitioners expertise. Finally, in Section 12.5, we present some facilitation tools that may be used to easily capitalize the customization knwoledge from practitioners at an organizational level. The usability of the proposed artifacts is discussed in Chapter 13.

12.1 Proposal

The exploratory studies presented in chapters 9, 10 and 11 allowed us to observe that the customization of agile methods is a dual-level process involving both the organizational and the project-team level. More precisely, the studies highlight the importance of capitalizing (disseminating) the project-team process knowledge at the organizational level. Moreover, the studies show that both the organizational and the project context influence the choice of suitable practices to select. For instance, if the team decides to implement Scrum without considering the availability of a customer representative, the implementation of Sprint Reviews is likely to be ineffective. The practice should be configured accordingly.

The studies also allowed us to figure out the multi-dimensional and measurable nature of a situational context. In other terms, a specific context where an agile method is to be implemented can be characterized by a set of factors, each of which can be estimated or measured using assessment models. For example, the organizational culture factors can be characterized by the 6 factors of the [Hofstede, 2011] model (see Chapter 11) which can be measured using some specific surveys[1]. Another learning consists of the fact that a context factor may be favorable or unfavorable for an agility goal. This aspect should also be taken into account in the customization process. Finally, the exploratory studies allowed us to initiate the idea to use matrix diagrams to structure the customization knowledge base.

Regarding the aforementioned learnings, we argue that the customization of an agile method should be the result of a consensus between the organizational learnings (stored in a knowledge base), the project-team preferences (a set of practices that the team wants to apply) and a context model.

Figure 12.1 depicts the customization process of AMQuICk. At the organizational level, the process repository documents a set of agile practices (as well as interesting references, techniques, tools and measures). Moreover, a knowledge base is available to store the past experiences and teams' feedback regarding the implementation of practices in their specific contexts.

At the project level, the method facilitator, in collaboration with the team members, defines the preferred way of working in terms of quality goals and agile practices (this provides more empowerment to the development team since practices are not imposed but suggested). Moreover, it instantiates the context model using an adequate assessment model. The consistency of the selected practices is checked based on the documented practice associations (see Section 7.3.6). The suitability of the selected practices is verified with regards to the available customization knowledge.

Based on the previously stored knowledge, a set of recommendations can be generated. These would approve, disapprove or propose configurations for the desirable practices. An additional set of beneficial practices may also be suggested as solutions to deal with challenging factors. The set of dependent or complementary practices (see Section 7.3.6) may also be proposed for selection.

The customization knowledge is stored using decisional matrices and is continuously refined and expanded at the organizational level. The matrices combine a set of recommendations and therefore provide a better guidance to practitioners in a particular situational context: which practices are helpful,

---

[1]For example: `https://www.surveymonkey.com/r/QVY9YXV`
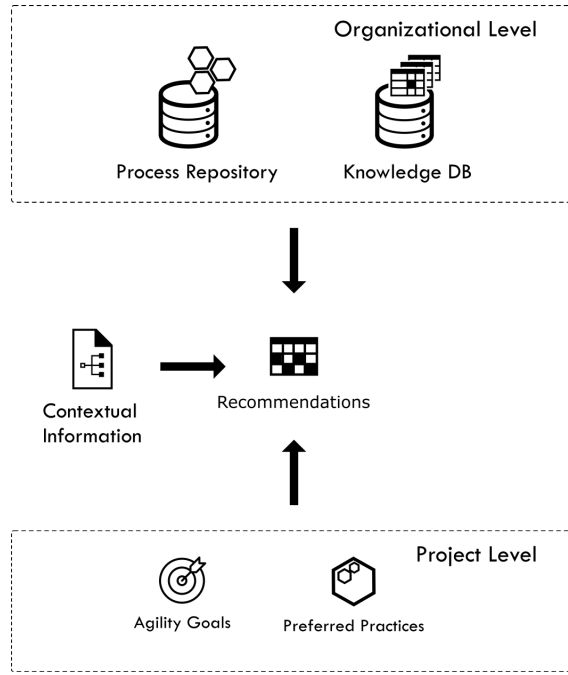
Figure 12.1.: Customization process levels

which would require customization or special consideration and which are not applicable or are harmful (see Section 12.3.1).

As they are deployed, the set of selected practices is continuously improved and new configurations may emerge. These are documented by the method facilitator and are later capitalized at an organizational level. The team also capitalizes its experience regarding the suitability of the selected practices in their specific context using a the common format of AMQuICk decisional matrices.

To summarize, the AMQuICk customization process is composed of the following steps:

1. **Context Study**

    a) assessing the situational context,

    b) identifying challenging context factors and

    c) prioritizing the agility goals (process quality goals).

2. **Customization** (see Figure 12.1)

    a) selecting preferred practices,

    b) gathering knowledge from similar contexts (retrieving adequate matrices), and

    c) recommending configurations.

The AMQuICk implementation and capitalization processes are composed of the following steps:

1. **Implementation** (see Figure 12.8)

    a) Plan

    b) Do

    c) Check

    d) Adapt

2. **Capitalization** (see Section 12.5.2)

    a) Store Experience

    b) Update Customization Knowledge

In the next section, we discuss the necessary conceptual elements that need to be added to AMQuICk Essence in order to support such a context-oriented customization.

12.2 Context Modeling

When an organization is going to develop a project in an agile way, a context profile has first to be instantiated. This instantiation is compliant with the AMQuICk context characterization described below.

12.2.1 Context Defined

The software development context influencing agile methods implementation can be defined as follows:

Definition 12.1 (Situational Context)**.** *The influential circumstances and variables that make the project situation unique and comprehensible and that affect the way of working of stakeholders involved in the project.*

The situational context factors for example consist of the *market uncertainty, budget constraints, application domain, project criticality, project duration, team size, familiarity with the involved technology, etc.*.

In an agile context, we have been able to identify a set of influential factors but more can be defined by an organization that is willing to use AMQuICk. Indeed, the set of relevant context elements to support the agile methods adjustments may be potentially different from an organization to another. Therefore, using AMQuICk, a custom context model can be designed to structure organizational or inter-organizational knowledge.

As earlier mentioned, some contextual models have been proposed by researchers and practitioners to guide the adoption and adaptation of agile software development practices.

[Cockburn, 2004b] in the crystal family of processes define different processes based on *Product Size*, *Criticality*, and *Skills*.

[Boehm and Turner, 2003] define a home ground of agile vs. plan-driven methods as associated to five critical factors namely, *Product Size*, *Criticality*, *Dynamism* (i.e. requirements change rate), *Personnel* (i.e., level of method understanding Cockburn [2000]) and *Culture* (of the team: thriving on chaos or on order).

[Kruchten, 2013] defines 2 sets of factors that make up the context: factors that apply at the level of the whole organization, and factors that apply at the level of the project. The organization-level factors do influence heavily the project-level factors which should drive the process to adopt. The organization level factors are defined as: *Business domain, Number of instances, Maturity of the Organization, Level of Innovation* and *Culture*. Project-level context factors are: *Size*, *Stable Architecture*, *Business Model* (contracting, money flow, etc.), *Team Distribution*, *Rate of Change*, *Age of System*, *Criticality* and *Governance* (management style).

[Ambler, 2009] in the Agile Scaling Models (ASM) framework defines a range of 8 scaling factors for effective adoption and tailoring of Agile strategies: *Team size, Geographical distribution, Regulatory compliance, Domain complexity, Organizational distribution, Technical complexity, Organizational complexity and Enterprise discipline.*

Even though the context models reported above have been defined for different purposes (i.e., crystal family of methods configuration, defining Agile vs. Plan-driven home grounds, practices adoption guidance and scaling agility to larger scopes), they seem to be more or less similar with only minor variations. They are all composed of context *"dimensions"* at the

higher levels refined into a set of *"factors"* or *"attributes"* at the lower levels. The main issue with such models is that they do not provide any explanation concerning the measurement of the context factors and their interpretation.

Based on this observation, we aim in the following section to abstract the context modeling in a common paradigm, so that agile facilitators (or any other relevant role) can characterize and assess the development context objectively.

12.2.2 AMQuICk Essence Extension - Context Package

This AMQuICk Essence context package aims to define the concepts and relationships to be used for the specification of a context model.
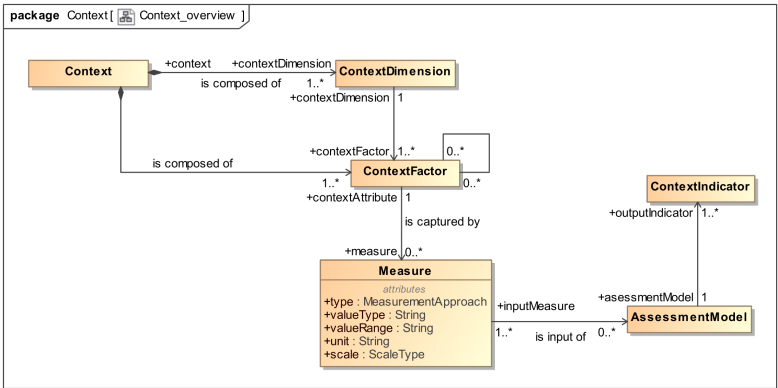
Figure 12.2.: AMQuICk Essence - Context Package

Figure 12.2 depicts its core elements. These are detailed below.

Context

The *context* construct type allows to instantiate custom context models, i.e., custom descriptions of situational contexts. Possible instantiations of this concept are the [Boehm and Turner, 2003] and [Kruchten, 2013] models. A context model is composed of a set of *dimensions* and measurable *factors*.

Context Dimension

A *context dimension* is a construct type that allows to define the high-level key-concepts used to categorize a situational context. Possible instantiations are "project", "organization", "team", "solution" or "customer". Each dimension is characterized by a set of *context factors.*

Context Factor

A *context factor* is used to describe a set of variables underlying a context dimension. For example, the "project dimension" may be characterized by the "scope variability" factor. A context factor may be divided into a set of sub-factors. For example, the "organizational culture" factor may be further refined into the "PDI, IDV, MAS, UAI, LTO and IDG" sub-factors of the Hofstede model [Hofstede, 2011] (see Chapter 11).

Moreover, a *context factor* is a measurable entity, i.e., it can be determined quantitatively using a set of measures. For example, a possible measure for the "scope variability" factor would be the "percentage of requirements change per month" and "a team appreciation of the criticality of requirements change".

As earlier discussed, in order to be conceptually correct and allow for the right operation and comparison, the measure has to be identified by a series of variables: its *value type*, *unit*, *scale* and a m*easurement/estimation* method. These indicate how the sheer value of a measure must be understood, compared to other measurement values and interpreted in fine (see Section 7.4.6).

Context Indicator

An *indicator* corresponds to the interpretation of context factors. It indicates how the sheer value of measures must be understood, compared to other measurement values and interpreted. Indicators inform about the suitability of the context providing agile methods implementation. In other terms, they interpret the values of context factors and provide the information needs for the customization process. They are the result of assessment models.

Indicators are key to the approach since they allow bridging the gap between objectified (through measurement) context elements and derived practices or practice configurations. As explained in [Vanderose et al., 2012], indicators are only as useful as their interpretation rules. In the context of product quality assessment, the interpretation associated to a given indicator determines the action to be undertaken in the next steps of the development.

Building upon this notion, the *AMQuICk* approach proposes to link the interpretation to practice customization guidance so that the interpretation of the indicator impacts directly the way the method is to be implemented (see Section 12.3).

Assessment Model

An *assessment model* is used for interpreting the set of context measures. It defines the acceptable values for the metrics according to a set of thresholds and generates a collection of context indicators. An assessment model will be for example the Hofstede model [Hofstede, 2011] which describes how to interpret the measures of the culture-related context factors (PDI, IDV, MAS, UAI, LTO and IDG) and which generates indicators for each of the factors (High vs Low).

Assessing the context factors highly depends on the tacit knowledge and inner experience of agile experts and may therefore be subjective. For instance, [Qumer, 2010] is an assessment model that generates an indicator for the *degree of agility*. The assessment model relies on the estimates of 5 attributes: flexibility, speed, leanness, learning and responsiveness. Each attribute is estimated to 0 or 1. The degree of agility is determined by the aggregation of the number of practices (or phases) which have been estimated to fulfill the attribute divided by the total number of applied practices. In a similar assessment model [Gandomani and Nafchi, 2014], this indicator is measured regarding to the degree of adoption and the weight (importance) of the practice to the company.

Several assessment models can be found in the literature for guiding agile methods implementation and for different purposes (see Chapter 3).

12.2.3 Example

As an illustrative example of context modeling, we provide a purposely simple context model in Figure 12.3.

The model describes one context dimension (customer) characterized by 3 context factors (availability, agility culture and business model). The customer availability context factor may be measured in different ways:

(CF3.M1) *Commitment time*: the effective time of collaboration between the customer and the development team per sprint. This measure is of type *ratio* and is expressed in *hours per sprint*,
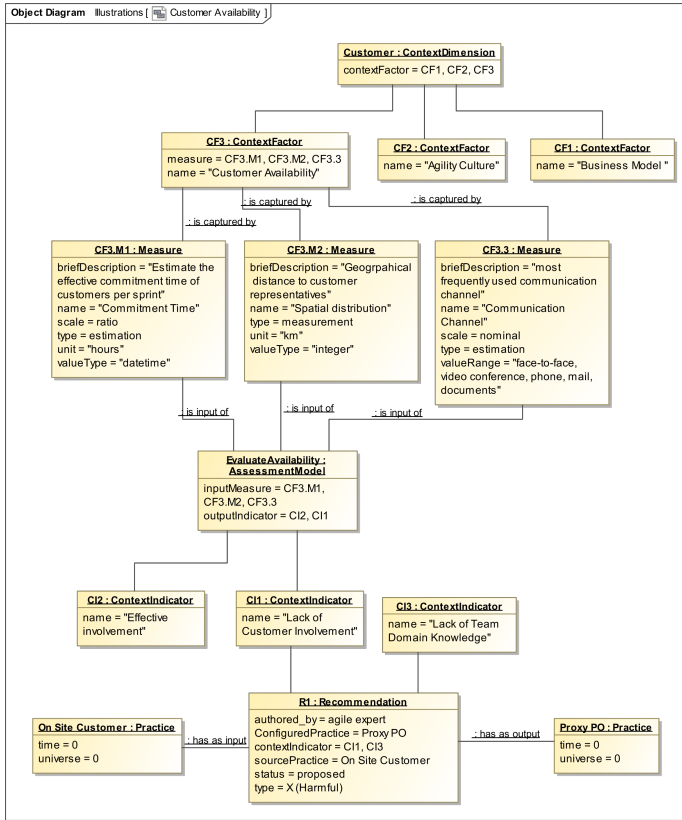
Figure 12.3.: Context Modeling Example

(CF3.M2) *Spatial distribution*: refers to the geographical distance and is expressed in *km*,

(CF3.M3) *Communication channel*: refers to the more frequent channel used to communicate with customers. The measure is of type *nominal* with a range of possible values, i.e., face-to-face, video, phone, mail or documents.

The interpretation of these measures in an assessment model results in 2 possible indicators that rank the level of customer availability (Low, High). In the case of lack of customer involvement, a possible customization to be undertaken is to recommend the *"Proxy PO"* practice as a configuration of the *"on-site customer"* practice.

12.3 **Customization Modeling**

A standalone context model has no meaning if not linked to adequate customization abilities. When an organization is going to develop a project in an agile way, a context profile has first to be instantiated using the context elements described in Section 12.2. Then, this profile and eventually a prioritized list of quality goals will be used to generate customization guidance.

We describe in the next sections how such guidance can be structured and formalized in AMQuICk Essence.

12.3.1 Customization Defined

As earlier mentioned, the customization of agile methods is often done in an ad-hoc way relying on the tacit knowledge of experts. We argue that it could be beneficial to structure such a knowledge so it can be reused to guide the customization decision-making. Accordingly we define the customization of agile methods as follows:

Definition 12.2 (Agile Methods Customization)**.** *The process in which agile teams define a suitable development method for their specific situation through a set of guided and knowledge-based adaptation decisions.*

Using AMQuICk, the customization body of knowledge would mainly document the contexts where practices have been deployed so far (inside and outside the organization), a set of experienced and recommended customization decisions and references to suitable practice configurations. The core AMQuICk Essence elements used to structure such knwoledge are presented in the following section.

12.3.2 AMQuICk Essence Extension - Customization Package

In order to properly structure the customization guidance and to support an easier decision-making, we define the constructs shown in Figure 12.4.

The figure describes the AMQuICk elements used to recommend customizations. These are detailed in the following paragraphs.
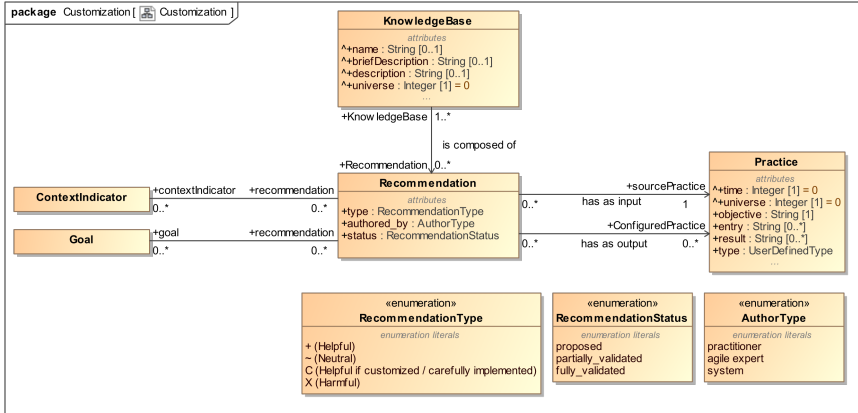
Figure 12.4.: AMQuICk Essence - Customization Package

Recommendation

The *Recommendation* construct type is the main element used to capture the customization knowledge and recommend practice configurations in a particular combination of context indicators. A recommendation has as an input a *source practice* and a set of *context indicators* and provides as an output information regarding the suitability or not of the selected practice and possibly a number of *configured practices*. It is suggested to associate recommendations with explanations using the *Resource* construct type. Doing so, a rational for the customization may be provided to end users. Moreover, experience reports of organizations where the configuration have succeeded may be provided as an additional guidance for practitioners.

Recommendations are to be visualized using the AMQuICk customization matrices (see Section 12.4). Each cell of the customization matrices actually represents an instantiation of the *Recommendation* construct type for a specific *Practice* and a unique *Context Indicator* (and/or one (or many) quality *Goal(s)*).

Knowledge Base

The *Knwoledge Base* construct type is used to store the collection of practitioners' and experts' recommendations which are available to assist the customization decision making. A knowledge Base belongs to a specific organization. This information is represented using the *universe* attribute (see Section 7.3.3). The default configuration of a knwoledge base (i.e., which has

a BASE_UNIVERSE = 0) may be reused / inherited by other organizations. A knwoledge base is more easily visualized using the AMQuICk customization matrices (see Section 12.4).

Goal

A practice may be recommended with regards to one or many *Quality Goals*. For example, the *"pair programming"* practice is reported in several empirical studies to contribute to *"design quality"* and to the *"reduction of defects in code"*. Therefore, the practice may be recommended to practitioners that seek to improve these goals. More importantly, goals also provide the rational for the customization decisions. For example, one may recommend to customize or pay a special attention to the *"daily meeting"* practice in the context of *"a low indulgence (IDG) level within the team"* (see Chapter 11) in order to achieve a *"better transparency"*.

Goals are expressed with regards to the SMART criteria (see Section **??**) and in terms of agile principles such as *"frequent delivery of the software"* or *"knowledge sharing"*. A goal may also be generic (e.g., "agility") so that recommendations can be formulated at a higher level. A prioritized list of agility goals may be used as an input of the AMQuICk customization process. Practices that contribute to the fulfillment of the selected goals may be recommended.

Recommendation Type

The *Recommendation Type* construct is used to express the nature of the relationship between a *Practice*, a *Context Indicator* and an agility *Goal*. A practice is to be recommended to satisfy a particular goal and/or provided a specific context indicator. Indeed, as earlier mentioned, a recommendation may be of different types to determine whether the practice can be implemented or not provided the specified context indicators. AMQuICk relies by default on 4 recommendation types : `~` Neutral practice, `+` Helpful practice, `c` Helpful if customized or carefully implemented and `–` Harmful practice (see Section 12.4.2). If different recommendation types are to be defined, one may use the *UserDefinedType* construct type.

Recommendation Status

The *Recommendation Status* can be understood as the confidence level that an AMQuICk user may have in a specific recommendation. Indeed, recommendations may represent the expertise of a specific agile team, an

agile expert or be generated automatically based on the aggregation of a set of previously documented customizations (for instance using a recommendation system). When a recommendation is documented by a unique team, there is a high risk that this one would be subjective. However, when it has been validated by an expert or generated based on several experience reports, then the risk is reduced.

Typically, we propose the following statuses: *proposed*, *partially validated* or *fully validated*. When a recommendation is documented by a practitioner, then it has the status of *proposed* in the knowledge base. If it is documented by an expert, then it has the status of *partially validated*. In order to be *fully validated*, more evidence is required based on several experience reports. Basically, a recommendation has this status if it is automatically generated. If different recommendation statuses are to be defined, one may use the *UserDefinedType* construct type.

12.3.3 Example

Team distribution is one of the most frequently reported challenges to the implementation of agile methods [Vallon et al., 2018]. Indeed, agile software development as it is initially defined in the manifesto [Beck et al., 2001] promotes direct and informal communication with the team and with customer representatives. Several practices such as the daily stand-up, sprint reviews, on-site customer and so on recommend co-location.

In contrast, global software engineering is becoming very common. In such environments, the customization of the common agile methods is necessary. Several adaptations are possible. Return of experiences can be found in research and practice.

Figure 12.5 shows an example of a customization that can be recommended to a distributed team planning to implement Scrum. The example is inspired by an experience report[2] from the Agile alliance.

The example shows that in case of a high spatial and temporal distribution (big difference in time zones), an agile team should adapt its way of working. Typically, it describes how the daily stand-up practice (identified as P7.0.0) could be configured:

(P7.0.1.) *Separate and Document Daily Stand-ups*: A possible configuration to the practice would be to hold one meeting per site and to

---

[2]`https://www.scrumalliance.org/community/articles/2013/july/managing-distributed-teams`
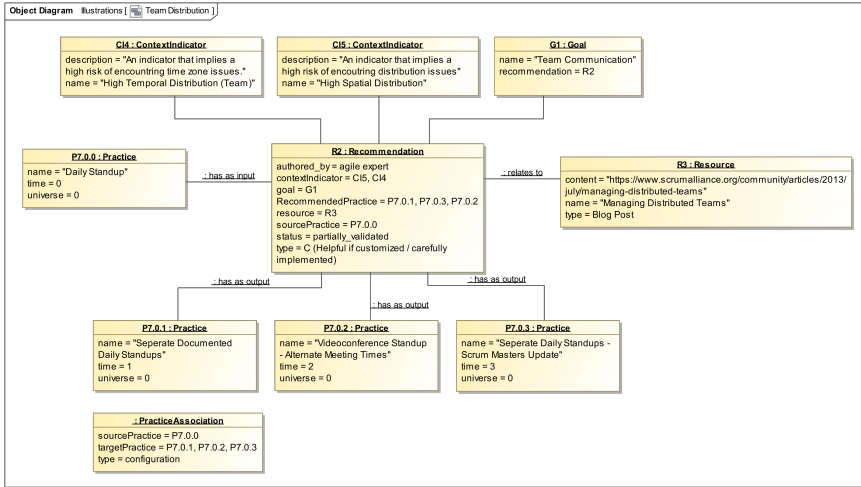
Figure 12.5.: Customization Modeling Example

document what has been discussed. This configuration may be a source of controversy since agile methods advocate the reduction of unnecessary documentation

(P7.0.2.) *Hold a unique video-conference stand-up and alternate times*: Usually, it is recommended to hold the daily meeting in the morning as it helps set the context for the coming day's work. For team members working in different time zones, it is still possible to hold a common stand-up meeting using video-conferencing (or a phone call) but with an alternation of the meeting time.

(P7.0.3.) *Hold separate stand-ups and update the other team*: Another possibility is to hold one daily stand-up per site and inform the other team about the important points that were discussed in the meeting. The update may be ensured by the Scrum masters.

The example also shows that the recommendation is activated by the *physical and temporal distribution* context indicators and with regards to the *team communication* agility goal. As earlier mentioned, the example also shows how a recommendation can be associated to a *resource*. In this case, we associated the recommendation to the experience report[3]2 from which the example was retrieved.

12.4 **Customization Matrices**

As shown in Figure 12.4, the recommendation of customization decisions is a many-to-many relationship between a list of practices and a set of context indicators. Matrix diagrams provide a convenient and compact way to represent many-to-many relationships between two or many list of items. This simple tool helps to analyze relatively complex situations and causal relationships in a straightforward way [Burge, 2011].

Taking the aforementioned consideration into account and inspired by the exploratory studies in Chapter 10 and Chapter 11 and by [Kruchten, 2013], we considered the opportunity for presenting the customization knowledge using matrix diagrams. The format, interpretation and population of such matrices is discussed in sections 12.4.1, 12.4.2 and 12.4.3, respectively.

12.4.1 **Format**

AMQuICk customization matrices act as a reference model for the expertise of agile practitioners. Depending on how a matrix is to be used, it may represent a unique context (see Table 13.2) or aggregate several possible contexts (see Table 13.3). They aim to help teams visually identify the presence of relationships between the agile practices they wish to implement (rows) and the relevant indicators regarding their specific context (columns). Table 12.1 shows how the matrices are formatted.

At each intersection (cell), a relationship between a specific practice and a context indicator is either present (+ , ~ , *C* , −) or absent (?). The different colors indicate the specific nature of the relationship, i.e., the nature of the specific knwoledge available regarding the implementation of practices in the specified context indicators.

As discussed in Chapter 5, the matrices should not be considered as prescriptions that tell what the team should exactly do. Rather, they are a mean to drive reflection on the value of practices and in fine to drive customization.

Another point worth discussing is the fact that customization matrices may be subject to many discussions and controversies. Indeed, they store the subjective knowledge of practitioners which may contain contradictory evaluations of practices or may be biased by the success or failure experiences. Consequently, it is important to consider the level of expertise of those who report the knowledge and to allow experts to validate their recommendations,

Table 12.1.: Customization Matrix Format

	Context Factors (Indicators)														
	CF1			CF2			CF3			..			CFy		
	V1	V2	...	V1	V2	...	V1	V2	...	V1	V2	...	V1	V2	...
Goal 1															
P1	~	~	?	~	+	?	~	C	?	C	~	~	~	–	?
P2	~	C	~	~	–	?	~	~	–	~	+	~	–	~	?
Goal 2															
P3	+	?	?	~	?	?	?	~	?	+	?	?	?	?	?
..	~	+	+	?	~	–	~	~	?	?	~	?	~	~	?
Goal n															
Px	+	~	?	~	~	–	~	~	–	~	C	~	?	~	–

Practices (row group label)

+	Helpful practice
~	Neutral practice
C	Harmful practice if not adequately customized / Helpful if adequately customized
–	Harmful practice
?	Not enough evidence to conclude

i.e., allow them to tell their confidence level about the reported knwoledge using the *Recommendation Status* construct type.

Moreover, it is important to consider the aggregation of several customization matrices (with the same structure) to guide the final customization decisions. In other terms, the final customization decision should depend on the interpretation of a collection of cells from one or many matrices (see Section 12.4.2 and Figure 12.6).

As shown in Table 12.1, the proposed customization matrices correspond to L-type matrix diagrams, i.e., basic matrix diagrams that illustrate the relationship between two lists [Burge, 2011]. Additionally, they include a third dimension consisting of the goals fulfilled by practices in specific contexts. This allows a goal-oriented decision making and provides the rationale for the customization (see Section 12.3.2).

A possible option to represent this third dimension is to use a C-type matrix diagram (a cubic arrangement), i.e., one that allows for three dimensional relationships [Burge, 2011]. However, since this can be difficult to visualize, the two dimensional representation of customization matrices seem to be more convenient assuming that the "inside" of cells contains additional information: the goal achieved from the customization and more (inspiring resources on how to implement the customization, pointers to suitable practice configurations, etc.). When necessary and for convenience, we depict the relationship between practice recommendations and goals in the matrix diagram as row headers (see Table 12.1).

12.4.2 Interpretation

A recommendation of a practice with regards to a specific context indicator can be of the following types:

- **Helpful practice: helps to satisfy a particular agility goal in the specified context indicator (** + **)**

  A practice that helps to reduce the negative impact of a non-sweet spot context indicator on an agility goal. For example, *pair programming* is a helpful practice to satisfy *technical excellence* in the context of *a team that is composed of junior developers profiles* [Dybå et al., 2007].

  Helpful practices typically include the facilitation practices that may be implemented by an agile coach to help a team resolve the context related issues. For example, given the indicator *lack of organizational*

283

innovativeness, method facilitators may implement the practice of *a monthly agile innovation meeting* (where the team members brain storm on a set of specific features) with the purpose of satisfying the goal of *better competitiveness.*

- **Neutral practice: may or may not satisfy a particular agility goal in the specified context indicator (** ~ **)**

  If previous experiences show no evidence that the practice is impacted (positively or negatively) by the specific context indicator, then the practice is indicated as neutral. A neutral practice means that the success of the practice or its failure is not conditioned by the indicator. For example, the *Sprint Review* practice is neutral with regards to the context indicator *diversity of users profiles.*

- **Harmful practice if not adequately customized (or helpful if adequately customized): needs to be customized and/or carefully implemented in order to satisfy a particular agility goal in the specified context indicator (** *c* **)**

  In non-sweet spot context, a practice may need to be carefully implemented or customized. A practice is either customized in order to become helpful or to not be harmful to satisfy a specific goal. For example, if the *team is distributed*, the *Sprint review* practice should be carefully implemented or configured in order to still satisfy *continuous feedback.*

- **Harmful practice: presents a high risk to satisfy a particular agility goal in the specified context indicator (** − **)**

  A possible recommendation is to avoid the implementation of the practice because it would be infeasible or unfavorable. For example, a *monthly release to users* may be infeasible and even harmful if the project is of a very *high criticality.*

Each cell of the matrix must be read as follows:

Practice [*specific practice*] **is** [*helpful, neutral, harmful if not customized (helpful if customized) or harmful*]
to satisfy [*quality goal*].
in the context of [*specific indicator*]

For example, *pair programming* may be recommended or not in different contexts and with regards to different goals. The following recommendations are formulated based on experience reports from the literature[Dybå et al.,

2007; Arisholm et al., 2007; Padberg and Muller, 2003; Sfetsos et al., 2009; Choi et al., 2008; Pikkarainen et al., 2008]:

- *Pair Programming* **is helpful** (+)
  to satisfy *a better code correctness*
  in the context of *complex system* and *junior profiles*
  Resource [Arisholm et al., 2007]

- *Pair programming* **is harmful** (–)
  to satisfy *a better time to market*
  in the context of *large project*, *small team size* and *high market pressure*
  Resource [Padberg and Muller, 2003]

- *Pair Programming* **is helpful** (+)
  to satisfy *productivity*
  in the context of *heterogeneous personality profiles*
  Resource [Sfetsos et al., 2009; Choi et al., 2008]

- *pair programming* **is harmful (or counterproductive)** (–)
  to satisfy *productivity*
  in the context of *homogeneous personality profiles* and *similar expertise*
  Resource [Choi et al., 2008; Pikkarainen et al., 2008]

- *pair programming* **is harmful** (–)
  to satisfy *productivity*
  in the context of *low system complexity* and *senior profiles*
  Resource [Dybå et al., 2007]

Another worth discussing question is the horizontal and vertical interpretations of matrices. [Burge, 2011] recommends to adopt a disciplined and systematic approach to examine the potential relationships of a matrix diagram. Whether to proceed row-wise or column-wise depends largely on the situation. The interpretation of AMQuICk matrices is mainly row-wise.

For example, let's consider the matrix depicted in Figure 12.6 which is instantiated for a specific context profile (one value per indicator). A quick horizontal scanning of the matrix shows that P1 is almost not affected and may therefore be implemented easily (~ cells are the greater part of the row) while P2 seem to be particularly challenging (the row contains only c and – cells). The recommendation regarding the P4 practice may

be more arguable. The customization or not of such a practice should be debated until the team reaches a consensus. The vertical lecture of the matrix also may provide some valuable information. For example, in Figure 12.6 we may see that the high regulatory compliance is one of the most challenging factors.
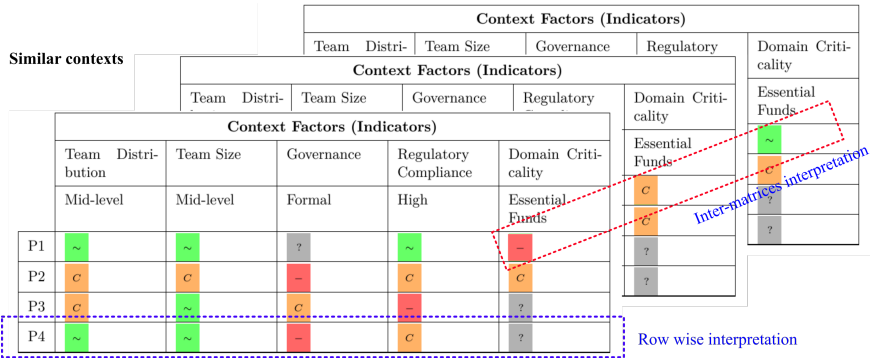
Figure 12.6.: Customization Matrices Interpretation

The aggregation of several customization matrices documented by different teams with the same structure may generate valuable information. For instance, we may possibly observe that in a *distributed context* 80% of the customization matrices documented by practitioners recommend the configuration of the *Monthly Release* practice. Consequently, the configuration of the practice could be recommended for future distributed projects and possible configurations of the practice would be proposed for selection.

12.4.3 Population

AMQuICk Essence provides a mean to structure and classify the customization knowledge and to link context-related elements to relevant practices (see Section 12.3.2). However, it do not instantiate a default knowledge base filled with experience reports.

Such a knowledge can be documented by practitioners (based on their previous experiences) and validated by a number of agile experts (see Section 12.5). They may also be automatically generated based on the aggregation of a set of previously documented customizations.

If a recommendation is to be automatically generated, one of the greatest challenges will be to collect a relevant learning set. A possibility that could

be considered is to collect such data based on a survey. Participants would be asked to fill-in information regarding their situational context and to rank the adaptability of a set of agile practices (e.g., adapted, adapted but needs few configuration, adapted but needs consequent configuration, not adapted). The results of some few surveys[4][5] conducted by agile practitioners are available but yet are not directly exploitable. In order to be able to extract relevant customization knowledge, the collected data can be processed using a multivariate factor analysis or a recommendation system (see Chapter 14). However, to apply such a methodology, it is crucial to have an important set of observations in order to have an empirical relevance.

Another possibility is to collect as much data as possible using a systematic literature review of scientific publications or practitioners' return of experiences. Indeed, several Scrum masters, agile coaches and experts frequently report their knowledge in the form of blog posts or articles in an agile consortium (e.g., Agile alliance[6], Scrum alliance[7], scrum.org[8]). This knowledge is valuable but considerable effort is needed to structure it.

In the next section, we describe how the customization knowledge can be directly collected during the implementation of the method and the capitalization at the organizational level.

12.5 Facilitation Tools

Asking teams to fill the customization matrices after the project is completed may not be an intuitive task. Moreover, team members may not remember all the different customizations that they experienced so far. We therefore propose to support the AMQuICk approach with some facilitation tools to be used during the method implementation and at the closure of the project.

The first facilitation tool consists of a visual board to be used during iteration retrospectives. It aims to facilitate the conversion of the reasoning behind process improvements into such an explicit format that it can be reused for eliciting the learning at the organizational level.

The second facilitation tool consists of a postmortem activity to perform during the capitalization step. It aims to summarize and structure all the

---

[4]`http://stateofagile.versionone.com/`
[5]`http://www.ambysoft.com/surveys/`
[6]`https://www.agilealliance.org/`
[7]`https://www.scrumalliance.org/`
[8]`https://www.scrum.org/`

team learnings in a customization matrix at the organizational level. Team members are the primary users of both tools. They are assisted by the method facilitator.

These two facilitation tools should not be perceived as the unique way to collect the customization knowledge from the team. They actually describe one possibility from many others that may be proposed by agile facilitators. During the course of this research, these tools were iteratively built and experienced in the context of a master's students project (see Section 13.4). However, their effectiveness comparing to other experienced retrospective tools was not validated empirically because of the unavailability of a longitudinal study to allow for a controlled experiment and because many factors actually may influence the effectiveness of retrospectives. The next sections discuss in further details the proposed tools.

12.5.1 Improvement Backlog

We discussed in Section 2.2.5 that while traditional SPI approaches usually adopt a top-down approach for improving the development process, agile methods seek to move the process control from the organizational level to the team. More precisely, the agile manifesto [Beck et al., 2001] advocates to empower development teams so that they adjust and improve their daily working practices continuously and in a face-to-face manner (see Table 2.1).

Complying with this principle many agile methods propose practices for iterative improvement. For example, Scrum proposes to perform retrospectives at the end of each sprint and Crystal [Cockburn, 2004b] describes a reflection workshop technique. Lean approaches rely on just-in-time improvement, i.e., improvement is done if and when necessary (see Section 2.2.3). Several process improvement practices and techniques are documented and shared in the agile community [Derby et al., 2006][9].

However, while these practices most likely accomplish their objective of empowering the team process control, they do not or hardly allow for a systematic knowledge transfer from one team to another. Indeed, they *"do not provide guidance on how the tacit or even explicit knowledge and learning of the project teams can be converted into an explicit form that can be utilized in organizational learning as well"* [Salo, 2005].

In order to be able to disseminate the team learnings at an organizational level, we argue that it is beneficial to systematically collect the information

---

[9]`http://www.funretrospectives.com/category/retrospective/`

available regarding the operated improvements and their effectiveness. The continuously collected information can be used for the encoding of the final team learnings in a customization matrix.

With the purpose of collecting the knowledge from practitioners as soon as possible and to not overload the development process, we argue that it would be beneficial to do it jointly with retrospectives.

In this section, an extension to the commonly used retrospective techniques is proposed using practice cards and an improvement backlog to support the systematic collection of improvement actions.

Format

Basically, retrospective techniques focus on identifying the things that went well, the things that doesn't and the things that the team can improve in the future. Different visualization techniques are suggested by method facilitators so they can still maintaining the motivation of the team. Usually, the team observations are written on post-its and openly discussed. Improvement actions are decided and eventually prioritized. However, the improvement actions are not expressed in terms of practices, nor are they goal-driven. Moreover, their effectiveness is not validated and their progress is not tracked.

In order to add more structure to the generated observations using the elements of the AMQuICk metamodel, we use visualization techniques familiar to development teams. Precisely, practice cards are displayed on the team room, post-its are used to represent practices that we want to improve or implement, goals, recommendation types and improvement actions. The latter are sorted by practices and goals and enacted in the improvement backlog (see Figure 12.7). The visualization of the practice cards in the team working space is called *practices wall*. This wall aims at animating the discussion during the retrospective sessions and to visualize a history of the practices evolution.

Based on the history of improvement actions and or their enactment, practice configurations may be proposed and are formalized in practice cards.

Retrospectives represent the iterative learning of the project. The last retrospective of the project is considered as a postmortem capitalization workshop. It captures the learnings at the organizational level in the form of a matrix (see Section 12.5.2).
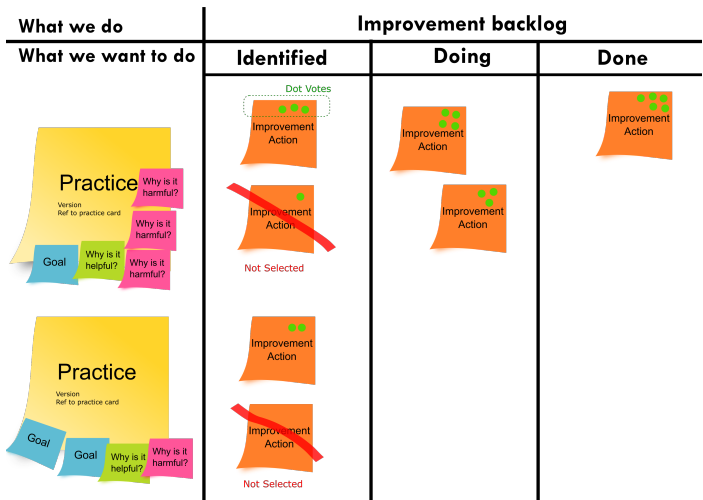
Figure 12.7.: AMQuICk Improvement Tool

Usage

The continuous improvement and customization of agile practices is aligned with the development iterations. Improvement actions are planned, implemented and checked continuously. A set of improvement actions may lead to the configuration of one practice. Taken the aforementioned into account, we modified the Deming's PDCA cycle [Deming and Edwards, 1982] by replacing the Act step by Adapt. Figure 12.8 depicts this cycle.

Retrospective sessions are the place where improvement actions are planned and checked and where practice adaptation decisions are discussed. The PDCA steps are detailed below:

1. **Plan**: This step aims at formulating feasible and collectively agreed practice improvement actions to be tried out in the next iteration. These would eventually lead to the definition of new practice configurations. Based on the perception of practices implemented in the previous iteration (green and red post-its), the team decides about the most urgent practices to improve. For each selected practice to improve, improvement actions (and alternatives) are discussed and written on post-its. The facilitator should confirm that these are within the possible organizational limitations for process improvement. Dot votes[10] are then used to prioritize all the improvement actions:

---

[10]http://www.funretrospectives.com/dot-voting/
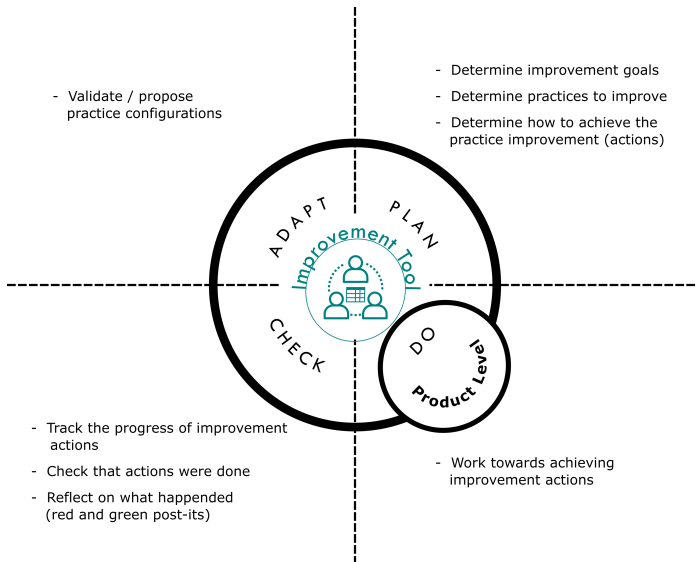
Figure 12.8.: Implementation Cycle - Plan Do Check Adapt (adapted from [Deming and Edwards, 1982])

every team member is given 3 dot votes for example and should place them on whatever interesting action he/she thinks would be beneficial. Many dot votes may be placed on the same post-it. Only a number of improvement actions with the highest dot votes are selected to integrate in the iteration. The team then decides about qualitative or quantitative means (observation, estimation or measurement) to validate the improvements. For example, if the team decides to improve the *unit testing* implementation, they have to decide about a minimal threshold for the *test coverage* to achieve.

2. **Do**: Team members self assign for the improvement actions. The assigned person is not necessarily the unique performer of the improvement actions. He/she rather plays the role of a responsible for ensuring that the action will be considered during the next sprint. An improvement action should be enough specific to be completed during one iteration.

3. **Check**: This step is carried out at the beginning of the next improvement iteration. First, the team has to check out if the actions have taken place as expected or if they need to be postponed to the next iteration. The effectiveness of the implemented actions is also verified.

Then, the team reflects on all the practices implemented during the Sprint and discuss how helpful/or harmful they were. To do so, each team member is given a number of green and red post-it and is asked to individually write notes on the positive and negative aspects of the practices. The practices with many harmful notes are discussed and are eventually selected to improve.

4. **Adapt**: Considering the previously introduced improvements, and the discussion on the effectiveness of practices, the team may identify more suitable ways of doing things. It decides about the new practice configurations (configuration of tools, metrics, practice activities, resources, etc.), the practices to discard and the new practices to introduce. This step should be carefully monitored by the method facilitator. If a new configuration is agreed, he/she documents it in a practice card and enriches the practices wall.

12.5.2 Capitalization Workshop

The capitalization workshop is a postmortem meeting that aims at summarizing and transforming the tacit knowledge of practitioners (highly personal, hard to formalize and, therefore, difficult to communicate) to explicit knwoledge (formal and systematic). Postmortem meetings are not a new concept. They have been used both in disciplined and agile environments as one of the most powerful tools *"that facilitates the transformation of personal knowledge into organizational knowledge"* [Takeuchi, 1995].

The output of this step is the customization matrix that captures the final experience of the project-team. To capitalize the team knowledge successfully, the role of the method facilitator is essential. The latter has to challenge the team for what they take for granted, ensure that they express both their negative and positive tacit knowledge and remind them about the history of improvement actions.

The meeting is animated around a *capitalization board* or knwoledge elicitation board (see Figure 12.9) which facilitates the generation of knowledge. At the end of the workshop, the facilitator has the responsibility to encode the generated knowledge using in a matrix diagram. The workshop may be animated following the steps below:

- **Preparation**: this step aims at stimulating the reflection. The facilitator reminds the team about the most important steps of the project and draw them in a time-line and overviews the history of improvement actions and practice configurations. The team members are asked to
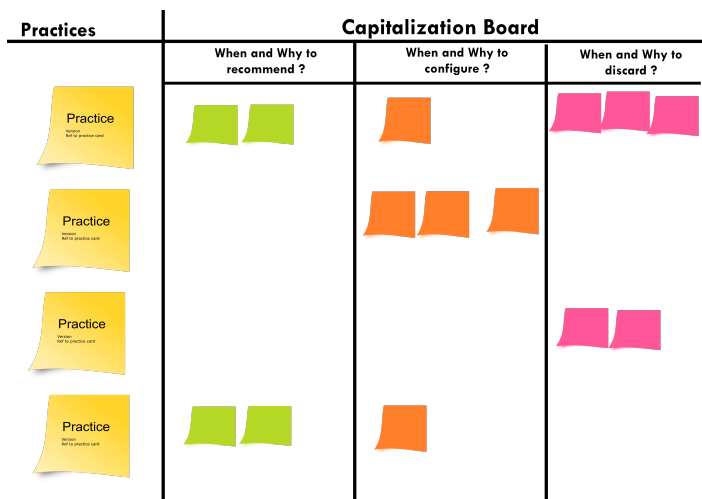
Figure 12.9.: Capitalization Workshop - Knowledge Elicitation Board

reflect on how the positive and negative events were distributed across the project life-cycle and on how practices contributed to leverage the encountered impediments. They may also discuss the possible practices that could have helped but that were not experienced.

- **Generation of explicit knowledge**: the implemented and the likely helpful practices (that could have helped) are displayed on the capitalization board (see Figure 12.9). Team members are given green, red and orange post-its that respectively represent the reason why they think that the practice may be helpful, harmful or helpful if configured. The facilitator reminds the usage of practices one by one as well as the experienced configurations. For each practice, team members are asked to record their feedback using as many post-its as they want. Then, a row-wise examination of the board is performed and the team finds a consensus about the final recommendations. The facilitator is the responsible for encoding the generated knowledge in a customization matrix to be stored in the organizational knowledge base.

As previously mentioned, this facilitation workshop should not be perceived as the unique way to collect the customization knowledge from the team. It is actually one possibility among others. Specifically, the preparation step may be helpful for getting a read on how team members felt about where the team was at particular points in time via a visual representation but clearly is not mandatory.

12.6 **Summary**

We focused in this chapter on the formalization of the customization knwoledge and its capitalization at the organizational level. Precisely, we proposed an extension to AMQuICk Essence, a new formalism to document the customization guidance using decisional matrices and finally some facilitation techniques.

The extension of AMQuICk Essence provides means to characterize in a fine-grained way the customization knwoledge so it can be easily reused by other practitioners. Basically, this characterization relies on the breakdown of the context into finer measurable factors. Practices and practice configurations are recommended or not depending on the value of these factors, on their interpretation and on the agility goals that a team is looking for.

Using AMQuICk customization matrices, the team expertise is captured more easier and in a more compact and systematic way. The proposed facilitation techniques are aimed to assist practitioners when capturing and capitalizing their expertise.

In the next chapter, we demonstrate the usability of the AMQuICk customization matrices through a number of illustrations. We also discuss the usability of the facilitation techniques which we experienced in the context of a masters' projects (see section 13.4) to elicit the customization knwoledge that we acquired regarding the implementation of Scrum in the context of an academic project.

Chapter 13

Illustrations

We argued in the previous chapter how important it is to structure the customization knowledge for enabling organizational learning. To this regards, we presented the newly introduced artifacts consisting of the extension of AMQuICk Essence, the knowledge base that is composed of a set of customization matrices and the facilitation tools for the collection of the customization experiences and their capitalization.

This chapter discusses the usability of these artifacts based on some illustrations. More precisely, Section 13.1 extends the Intel Shannon case study (see Chapter 7) by structuring the extracted knowledge in an AMQuICk customization matrix. Similarly, Section 13.2 structures the knowledge reported by the SPW practitioners (see Chapter 9). Section 13.3 provides an example of some recommendations that can be formulated with regards to the cultural context (see Chapter 11). Finally, Section 13.4 provides a more complete illustration that formalizes our key learnings regarding the implementation of a custom Scrum in the context of a masters' capstone course.

13.1 Intel Shannon Customization

We studied in Section 7.6 the implementation and customization of Scrum and XP in the specific context of Intel Shannon (a case study selected from literature). In this section, we demonstrate how the customization decisions taken by practitioners can be documented using an AMQuICk customization matrix (see Table 13.1).

The matrix lists only the XP and Scrum practices that were configured or discarded. For convenience, the agility goals fulfilled by the practices are not presented in the matrix but are detailed in the following list which provides a row-wise explanation of customization decisions:

Table 13.1.: Intel Shannon Customization Matrix

		Context factors												
		Project / Requirements						Team						
		Complexity		Variability		Compliance		Expertise		Size		Turnover		Distribution	
		High	Low	High	Low	High	Low	High	Low	Large	Small	High	Low	High	Low
Agile practices	Pair programming [**C15**]	+	−	−	~	?	?	+	−	?	?	?	?	?	?
	Simple Design [**C16**]	C	~	?	?	?	?	?	?	?	?	?	?	?	?
	Coding Standards [**C17**]	?	?	?	?	?	?	?	?	?	?	+	~	?	?
	Collective Ownership [**C18**]	?	?	?	?	?	?	−	~	+	~	?	?	?	?
	Continuous Integration [**C19**]	C	?	?	?	?	?	?	?	?	?	?	?	?	?
	On-site Customer [**C20**]	?	?	?	?	?	?	?	?	?	?	?	?	C	~
	Small Release [**C21**]	−	~	?	?	?	?	?	?	?	?	?	?	?	?
	Daily Meeting [**C22**]	?	?	?	?	?	?	?	?	+	C	?	?	C	+
	Sprint Planning [**C23**]	C	?	?	?	C	?	?	?	?	?	?	?	?	?
	Sprint Closure [**C24**]	?	?	?	?	C	?	?	?	?	?	?	?	?	?

+	Helpful practice
~	Neutral practice
C	Harmful practice if not adequately customized / Helpful if adequately customized
−	Harmful practice
?	Not enough evidence to conclude

[C15] **Pair Programming**: The customization of pair programming is discussed in the study with regards to the complexity of the project, the variability of scope and the expertise of the team. More precisely, it was found that pair programming helps the team members to understand the complex requirements. Moreover, the practice seemingly contributed to the reduction of defect density when junior profiles integrate the team. However, pair programming was found unsuitable for easy features and maintenance tasks (low variability of requirements) and when pairs are formed of expert profiles since it reduces the team productivity and concentration.

Provided the aforementioned, the following recommendations can be formulated:

- Pair Programming is *helpful* to satisfy *requirements understandability* in the context of *high requirements complexity*

- Pair Programming is **harmful** to satisfy *Productivity and Teamwork quality (concentration)* in the context of *low requirements complexity*

- Pair Programming is *harmful* to satisfy *Productivity and Teamwork quality (concentration)* in the context of *low requirements variability* (and *neutral* in the context of *high variability*)

- Pair Programming is *harmful* to satisfy *Productivity* in the context of *high team expertise*.

- Pair Programming is *helpful* to satisfy *Code Reliability (reduction of defect density)* and *Technical Excellence* in the context of *low team expertise*

[C16] **Simple Design (Quick Design Sessions)**: The simple design practice is implemented through quick design sessions where practitioners brainstorm on white-boards before each block of code is written. However, they report that this practice needs to be configured to satisfy the high expectations of maintainability and traceability. More precisely, the practice was adapted so that design documents were systematically updated after each quick design session.

Provided the aforementioned, the following recommendations can be formulated:

- Simple Design, *if not customized, is harmful* to satisfy *maintainability* and *traceability* in the context of *High project*

297

> ***complexity*** (and ***neutral*** in the context of ***low project complexity***)
>
> - Simple Design, ***if not customized, is harmful*** to satisfy ***maintainability*** and ***traceability*** in the context of ***high process compliance*** (and ***neutral*** in the context of ***low process compliance***)

[C17] **Coding Standards**: The Intel Shannon practitioners applied the coding standards practice by defining their own C coding standard rather then relying on an accepted community standard[1]. They found the practice helpful for ensuring a better maintainability in the context where teams membership is volatile (a high turnover rate).

Provided the aforementioned, the following recommendation can be formulated:

> - Coding Standards (the practice) is ***helpful*** to satisfy ***maintainability*** in the context of ***high turnover*** (and ***neutral*** in the context of ***low turnover***)

[C18] **Collective Ownership**: In the default configuration of the practice (as it is defined in XP), everyone shares responsibility for code quality and everyone can make necessary changes and fix bugs anywhere. However, in the context of Intel Shannon, the practice was only considered appropriate on a single-team basis, i.e., code was not shared across multiple teams even when those work on the same project. Practitioners also felt that the practice could not scale to a wide team. Moreover, the practice was found helpful to ensure maintainability in the context of a high turnover rate. The study reports : *"Collective ownership allowed management more flexibility as it resulted in teams being able to maintain the code base."*.

Provided the aforementioned, the following recommendations can be formulated:

> - Collective Ownership, ***if not carefully implemented/customized, is harmful*** in the context of ***large team size*** (unspecified goal) (and ***neutral*** in the context of ***small team size***)
>
> - Collective Ownership is ***Helpful*** to satisfy ***maintainability*** in the context of ***high team turnover*** (and ***neutral*** in the context of ***low team turnover***)

---

[1] `http://agileinaflash.blogspot.com/2009/02/coding-standards.html`

[C19] **Continuous Integration**: Because of the complexity of the projects and regarding the need for external test equipments, the code was not fully integrated at the end of each sprint. Rather, the system is integrated only two weeks before a major release.

Provided the aforementioned, the following recommendation can be formulated:

- Continuous integration, ***if not carefully implemented/customized, is harmful*** to satisfy ***software testability*** in the context of *high project complexity*

[C20] **On-site Customer**: In the context of Intel Shannon, customers are not available during the early development stages and customer representatives are distributed. To ensure a value-driven delivery of features, the co-located marketing team played the role of a customer proxy, *"prioritizing features based on potential revenue"*.

Provided the aforementioned, the following recommendation can be formulated:

- On-site Customer, ***if not carefully implemented/customized, is harmful*** to satisfy ***value-driven delivery*** in the context of ***high distribution (between the team and customer representatives)***

[C21] **Small Release**: With no configuration, the practice was found unfeasible because of the complexity of the project. Indeed, the software development and deployment on processors depend on silicon availability, require test equipment and intensive integration. As a balance between continuous delivery and valuable software delivery, the practice was transformed to a two-level release strategy where minor versions are released every 4 to 6 weeks and major versions every two quarters.

Provided the aforementioned, the following recommendation can be formulated:

- Small Release, ***if not adequately customized, is harmful*** to satisfy ***valuable delivery*** in the context of ***high project complexity***

[C22] **Daily Meeting**: The practice was found helpful in the context of small teams to help achieve transparency (shared visualization of tasks and of the project progress). Daily meetings are held around a post-it board where team members record their daily tasks and discuss them.

In the context of large teams, the practice was configured to also record the tasks in the project repository. Moreover, the practice was configured for distributed teams using an online tool.

Provided the aforementioned, the following recommendation can be formulated:

- Daily Meeting is ***helpful*** to satisfy ***transparency*** in the context of ***small team size*** and ***small distribution (of the team)***

- Daily Meeting, to satisfy ***transparency*** in the context of ***large team size***

- Daily Meeting, ***if adequately customized, is helpful*** to satisfy ***transparency*** in the context of ***high distribution (of the team)***

[C23] **Sprint Planning**: Because of the complexity of projects at intel shannon, a two-level planning is performed: one at the start of the project and one at the start of each sprint. Another reason for this configuration resides in the high process compliance (audited) which requires an upfront planning of the project including a predefinition of the project scope (at a high level).

Provided the aforementioned, the following recommendation can be formulated:

- Sprint planning, ***if adequately implemented or customized, is helpful*** to satisfy ***on-time delivery*** in the context of ***high project complexity*** and of ***high compliance (audited)***

[C24] **Sprint Closure (lightweight)**: Scrum recommends to close the sprint by performing a review and a retrospective and delivering/deploying a valuable product version. In Intel Shannon, the practice was customized to include a wrap-up session aiming at finalizing the documentation of the end-of-sprint deliverables (wrap up report listing the completed/extra tasks, lessons learned, measurement of the actual effort, test scripts, tests scenarios, demo deliverables, etc.) This is explained by the importance of compliance and the fact that it is audited. In such a context, one may say that the only delivery of code is not sufficient. Documentation is important and should be considered as a valuable feature that needs to be delivered (i.e., to treat documentation like requirements[2]).

---

[2]`http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm`

Provided the aforementioned, the following recommendation can be formulated:

- Lightweight Sprint Closure, ***if not adequately customized, is harmful*** to satisfy ***valuable delivery*** in the context of ***high and audited process compliance***

13.2 SPW Customization

In Chapter 9, we presented a set of customization decisions collected during the project retrospective in collaboration with the team and the Scrum master. Table 13.2 restructures the collected knowledge using an AMQuICk customization matrix. For each practice, we provide a row-wise explanation of the documented customizations (from C1 to C14). The are presented in Section 9.3.2.

The table do not depict neutral relationships, i.e., ~ cells since in the SPW case study we only focused on identifying the practices that may help the team and those that were found particularly difficult or impossible to implement. When the practice applicability with regards to a specific context indicator was not discussed with the team members, we consider that not enough evidence exist and therefore mark the cell with ? .

For convenience, the agility goals fulfilled by the practices are not presented in the matrix. These appear in Table 9.4.

13.3 Culture-based Customization

To show how the customization knwoledge can be documented using decisional matrices, we provide in this section an example related to the case study discussed in Chapter 11. Table 13.3 illustrates a customization matrix that can be provided to AMQuICk users to guide them through the implementation of methods taking into account their specific cultural context. The recommendations it presents should be seen as illustrative since they are suggested by the researchers and not designed by agile practitioners.

A row-wise explanation of the set of illustrative recommendations is provided in the following:

Table 13.2.: SPW Customization Matrix

Agile practices		Organizational and Project's Context Indicators							
		Team Communication	Org. Structure	Agile Skills	Team Size	Team Distribution	Domain Criticality	Customer Availability	Contract Model
		Average	Siloed	Low	Average	Mid-level	Ess. Funds	Low	Fixed scope
	User Stories Estimation[C1]	C	?	C	?	?	?	C	?
	Sprint Planning[C2]	?	?	?	C	?	C	?	?
	Grooming Session per Release[C3]	?	?	?	?	?	?	+	?
	Domain Walk-through[C4]	?	?	?	?	?	+	+	?
	Model in Small Increments[C5]	?	−	?	?	?	−	?	−
	US Writing Workshops[C6]	?	?	+	?	?	?	?	?
	User Story (US)[C7]	?	C	?	?	?	?	?	C
	Role Coaching[C8]	?	+	+	?	?	?	?	?
	Sprint-0 (Technical Envisioning)[C9]	?	?	+	?	?	?	?	C
	Pair-programming Sessions[C10]	?	?	+	?	?	?	?	?
	Parallel Independent Testing[C11]	?	+	?	?	?	+	?	?
	Scrum Team[C12]	?	?	?	C	?	?	?	?
	Retrospective facilitation[C13]	+	?	?	C	?	?	?	?
	Physical Taskboard[C14]	?	?	?	−	−	?	?	?

Table 13.3.: An example of cultural customization matrix

Agile practices	Cultural factors									
	PDI		IDV		UAI		LTO		IDG	
	High	Low	High	Low	High	Low	High	Low	High	Low
Team Commitment to Practices										
Satir Change Model (Research Model) [C25]	?	?	?	?	+	−	+	−	?	?
Kotter 8-Step (Industrial Model) [C26]	?	?	?	?	+	−	+	−	?	?
Team Empowerment										
Self-organizing teams (Scrum) [C27]	C	~	?	?	?	?	?	?	?	?
Sign-up or self-assign (XP) [C28]	C	~	?	?	?	?	?	?	?	?
Dickinson et al. teamwork model (Research Model) [C29]	+	?	?	?	?	?	?	?	?	?
Transparency and Cohesion										
Task Board (Scrum) [C30]	C	~	?	?	?	?	?	?	~	C
Daily Meeting (Scrum) [C31]	C	~	?	?	?	?	?	?	~	C
Code Ownership (XP) [C32]	C	~	C	~	?	?	?	?	~	C
Gemba Walk (LEAN) / Go See (LeSS) ³ [C33])	+	?	?	?	?	?	?	?	?	C
etc...										

[C25] **Satir Change Model:** Implementing a change model such as the Satir Model [Satir and Banmen, 1991] is usually recommended by agile experts to improve the commitment to agile practices where more resistance to change is likely to happen. In environments where changes are welcome, there is no need to implement such heavyweight models to guide transformation.

Regarding the LTO factor, H3 (see Chapter 11) argues that when change is enough planned and prepared in a long-term oriented context, the team is more likely to commit to practices. Therefore the use of change models is assumed to be useful in contexts where organizations are long-term oriented. References where this model is used for these purposes can be found in literature [Babuscio, 2009; Liebmann, 2003; Kelly, 2008].

It seem also reasonable to assume that in a short team oriented context, the implementation of strategic change models would bring unnecessary complexity to the process.

Provided the aforementioned, the following recommendations can be formulated:

- Satir Change Model is *helpful* to satisfy *the team commitment to practices* in the context of *high UAI*

- Satir Change Model is *helpful* to satisfy *the team commitment to practices* in the context of *high LTO*

- Satir Change Model is *harmful* to satisfy *Process Simplicity* in the context of *low UAI* or *low LTO*

[C26] **Kotter 8 step model**: Similarly to the previous recommendation, the Kotter 8 step model[4] [Kotter et al., 1995] can be recommended for guiding agility transition and improving team commitment to practices. References where this model is used for these purposes can be found in literature [Kelly, 2008; Hayes and Richardson, 2008; Hui, 2013].

Provided the aforementioned, the following recommendations can be formulated:

- Kotter 8 step model is *helpful* to satisfy *the team commitment to practices* in the context of *high UAI*

- Kotter 8 step model is *helpful* to satisfy *the team commitment to practices* in the context of *high LTO*

---

[4]`https://www.kotterinc.com/8-steps-process-for-leading-change/`

- Kotter 8 step model is **harmful** to satisfy **Process Simplicity** in the context of **low UAI** or **low LTO**

[C27] **Self-organizing teams**: In culture environments with an established hierarchy (high PDI), the implementation of *self-organizing teams* should be configured and/or carefully implemented. To this regards, [Hoda et al., 2012] states that *"teams facing a management that was still dictating terms are unlikely to self-organize"*. The study identifies a set of practices to configure the *self-organizing team* practice: Balancing freedom & responsibility, Balancing cross-functionality & specialization and Balancing continuous learning & iteration pressure.

In a low PDI context, its more easier to have self-organized teams.

Provided the aforementioned, the following recommendations can be formulated:

- Self-organizing teams **if adequately customized, is helpful** to satisfy **team empowerment** in the context of **high PDI** (and **neutral** in the context of **low PDI**)

[C28] **Sign-up**: Sign-up or Self-assign is another practice that is impacted by a high PDI index and therefore the practice should be configured or carefully implemented. [Hoda et al., 2012] discusses the need for configuring this practice.

Provided the aforementioned, the following recommendations can be formulated:

- Sign-up **if adequately customized, is helpful** to satisfy **team empowerment** in the context of **high PDI** (and **neutral** in the context of **low PDI**)

[C29] **Teamwork improvement models**: Specific models such as [McIntyre and Dickinson, 1997] which provide a better understanding of the nature of self-organization may be recommended for empowering team members to share the tasks and responsibilities of leadership.

Provided the aforementioned, the following recommendation can be formulated:

- The [McIntyre and Dickinson, 1997] teamwork model **is helpful** to satisfy **team empowerment** in the context of **high PDI**

[C30] **Task Board**: Task boards are one of the most important information radiators used by agile teams to track their progress. However, when a culture of transparency isn't available, the practice should be carefully

305

verified by the agile facilitator. The issue is discussed in some return of experiences from practitioners [Perry, 2008].

Provided the aforementioned, the following recommendations can be formulated:

- Scrum Task Board, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of *high PDI* (and *neutral* in the context of *low PDI*)

- Scrum Task Board, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of *low IDG* (and *neutral* in the context of *high IDG*)

[C31] **Daily Meeting**: Daily meetings are recommended by several agile practices. In an environment where transparency lacks, their implementation may be problematic. To this regards, it is crucial that the agile facilitator configures and/or cares about the implementation of this practice by building trust. Studies such as [Pries-Heje and Pries-Heje, 2011] discuss this question.

Provided the aforementioned, the following recommendations can be formulated:

- Daily Meeting, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of *high PDI* (and *neutral* in the context of *low PDI*)

- Daily Meeting, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of *low IDG* (and *neutral* in the context of *high IDG*)

[C32] **Code Ownership**: The practice of shared code ownership is related to team cohesion (i.e., to the team IDV context factor). [Bacchelli and Bird, 2013] explains how managers promote collective code ownership and code review processes to improve team cohesion and transparency. In an environment of low IDG and high PDI, the implementation of such a practice is likely to fail. The implementation of the practice may also be influenced by the IDV dimension. Developers with an individualistic mindset usually feel that they individually own the code and that they are the only ones understanding it in details[5]. Therefore, it is recommended that agile facilitators particularly take care of the implementation of this practice in this combination of these context factors.

---

[5]`https://www.coderhood.com/12-reasons-avoid-individual-code-ownership/`

In contrast, in a low PDI, low IDV and high IDG context, it is arguable that the practice may be implemented easily.

Provided the aforementioned, the following recommendations can be formulated:

- Code Ownership, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of **high PDI** (and **neutral** in the context of **low PDI**)

- Code Ownership, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of **high IDV** (and **neutral** in the context of **low IDV**)

- Code Ownership, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of **low IDG** (and **neutral** in the context of **high IDG**)

[C33] **Gemba**: Gemba is a practice from Lean meaning the real place. It encourages both managers and developers to go for "walks" to observe the team or users' working places. The practice is recommended[6] for providing more transparency and cohesion between team members and external stakeholders in a context characterized by a command-and-control mindset. However, one should ensure, particularly in a low IDG context, that the practice is not perceived as invasive by the team members or users [Womack, 2013].

Provided the aforementioned, the following recommendations can be formulated:

- Gemba Walk, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of **high PDI** (and **neutral** in the context of **low PDI**)

- Gemba Walk, *if carefully implemented/customized, is helpful* to satisfy *Transparency* in the context of **low IDG** (and **neutral** in the context of **high IDG**)

13.4 Agile Customization in a Master's Capstone Course

A capstone course have the objectives to help the students integrate the material previously learned in the curriculum, improve their understanding

---

[6]`https://www.scrumalliance.org/community/articles/2013/june/transparency-in-agile-product-development`

of that material, extend their knowledge to other areas and help them acquire a realistic simulation of a professional experience [Mahnic, 2012; Adcock et al., 2009].

Several studies such as [Cleland and Mann, 2003; Mahnic, 2012; Schroeder et al., 2012] tend to show the benefits of implementing agility in capstone courses to improve the technical and soft skills of students. Besides, they argue the need for special consideration and/or adaptation of numerous agile practices and to this regards, some studies such as [Williams et al., 2008; Reed, 2008] formulate guidelines to help other educators be as successful as possible when introducing agility to their courses.

In this section, we report our experience on the implementation of agile software development in three masters' students projects during the academic years 2015-2016, 2016-2017 and 2017-2018. Relying on the feedback collected from the students, on observational data, and on project management reporting, we formalize our key learnings in an AMQuICk customization matrix and discuss the general applicability of the learnings.

13.4.1 Background

The laboratory of software engineering[7], most commonly called MDL (for Méthodologies de Développement de Logiciel in french, i.e., methodologies of software development), is a master's capstone course in the faculty of computer science of the University of Namur. The course is of 14 weeks duration and is actually of 9 ECTS which according to [European Union, 2015] corresponds to 16/20 hours workload per week (1 ECTS = 25 to 30 hours workload) including an average of 4 hours per week in class. In each academic year 4 to 5 groups of 5 to 6 students each were created.

The three projects P1, P2 and P3 were of an average complexity with some few features of a relatively *high complexity* (for non experimented developers). The most challenging features of the projects were purposely chosen to be similar. In P1 and P2 (2015-2016, 2016-2017), students worked on the same business case with some few variations. More precisely, they were asked to develop a parcels delivery system, starting from the customer's order placement to the final delivery of the parcel. Several features were required to achieve the *Minimum Viable Product (MVP)*[8]. These mainly include the order placement, the parcels routing through the sorting centers, the management of the fleet of vehicles and parcels trolley, the delivery of parcels

---

[7]`https://directory.unamur.be/teaching/courses/INFOM111/2016`
[8]`https://www.agilealliance.org/glossary/mvp/`

to the final destination (that may be outside the country) and the potential return of parcels. In this business case, the major complexity consisted in the development of a routing algorithm which corresponds to a rich Traveling Salesman Problem (TSP) [Hoffman et al., 2013].

In P3 (2017-2018), students worked on a different business case. More precisely, they were asked to develop a dynamic public bike-sharing system [Contardo et al., 2012], starting from the citizens subscription to the dynamic bikes booking. Several features were required to achieve the MVP. These include the subscription, the dynamic booking of bikes, the tracking of the bikes ride, the closure of a ride and the dynamic balancing of the fleet of bikes. As in the other business case, the most challenging feature consisted in the dynamic balancing of the bikes which again requires to use a heuristic to solve the TSP [Hoffman et al., 2013].

Regarding the *scope variability*, students working on P1 and P2 were not enough empowered to adapt the scope except for some few features. In P3, they were much more free to innovate in the implementation of some features such as the booking of bikes and the balancing algorithm.

For the three projects, the technologies used are the Java Enterprise Edition (JEE)[9], the Java API for RESTful Web Services (JAX-RS)[10] and Android[11]. For the development of the routing algorithm, the students were free to choose any heuristic API or to develop their own algorithms. Most of them chose to use JSprit, an open source Java based library for solving rich TSPs[12]. The average student is not familiar to the proposed technologies.

The course requires students to work as Scrum teams, responsible for the implementation of a set of EPICs[13] and user stories.

Based on an opinion survey, project management reporting (using *Jira Atlassian*[14]) and on observational data, we discuss the customization decisions that have been decided during the course of the projects and summarize our lessons learned using an AMQuICk matrix. These lessons reflect the issues to be considered when using agile software development in a teaching context.

---

[9]http://www.oracle.com/technetwork/java/javaee/overview/index.html
[10]https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html
[11]https://developer.android.com/
[12]https://github.com/graphhopper/jsprit
[13]Large features that can be broken down into a number of smaller user stories
[14]https://www.atlassian.com/software/jira/agile

13.4.2 Objectives and Data Collection

The main focus of this study is to investigate the implementation of agile methods in a teaching context. More precisely, it intends to identify and structure the customization decisions that emerged during the implementation of a custom Scrum method.

The applied methodology to collect the data is mainly observational. Our intervention was limited to the facilitation of Scrum, the introduction of facilitation techniques to collect the customization knwoledge (see Section 12.5.2) and the suggestion of additional practices or practice configurations when necessary. We also assisted the team getting started with Jira and configured the tool to their needs when required.

Data was collected from the following:

1. **Survey at the closure of P1**: After the last release of P1, students were asked to answer a questionnaire[15] to collect data regarding their general perception of the course, the technologies that were used, their appreciation of the relevance of different practices that were applied (see Table 13.6) and the eventual practice configurations they think would be helpful. We decided to not use the survey in the other years. Indeed, in order to collect the groups learnings in a more systematic way, we applied the AMQuICk capitalization workshop. Since in this study, we only focus on reporting the customization decisions, the survey results regarding the course agenda, the achievement of the course objectives and the appreciation of tools and used technologies are not reported.

2. **Observation of Scrum meetings**: We (the teaching assistants) participated to all sprints planning, review meetings and to some retrospectives. The latter were held by the student groups right after their sprint review and generally without our intervention. We participated to only few retrospectives with the purpose of facilitating the customization and introducing improvement and capitalization techniques.

3. **Reporting**: For the 3 projects, we collected data on project management using Jira with the purpose of analyzing the amount of work completed, the compliance with sprint plans, the team velocity and ability in effort estimation, etc. Teams were frequently provided with comments regarding their usage of Jira.

---

[15] https://goo.gl/forms/QPscpm5VVlYmJRJJ3

13.4.3 Implemented Method

General Settings

In order to effectively apply Scrum, a dedicated space furnished with white boards was allocated to each group of students for the duration of the project. Moreover, we used Jira which was deployed in an internal server under the academic license. The choice of Jira was made because of its reporting abilities and because of the fact that it is one of the most used tools in professional settings. Groups were also encouraged to maintain a *physical tasks board* and to use different visualization techniques in their working place. The average student was not familiar to agile software development and Jira. We also provided each group with a repository for code versioning and a web application server.

Regarding the Scrum roles, in P1, two students from each group played the roles of a Scrum Master and a proxy Product Owner respectively. Only the role of a Scrum Master was maintained in P2 and P3. The teaching assistants and eventually external stakeholders (colleagues) played the roles of customer representatives, domain experts (product owners), agile facilitators and technical consultants.

Sprint-0 and Design Sprint

During the first month, students were not fully committed to the project since other courses are taught in parallel. Therefore, we payed attention during this period to not implement time-consuming practices.

More precisely, we exploited the first 2 weeks to introduce the business case and animate workshops to familiarize the students with the project's technologies, the agile practices and the business domain. We refer to this initial preparation period as *Sprint-0*.

Regarding the application complexity (distributed and multi-plateform), the remaining 2 weeks were exploited to conduct a *Design Sprint* consisting of a high-level architecture envisioning, prototyping, preparing the skeleton of the data access layer, etc.

Development Sprints

The rest of the project was decomposed into 4 development Sprints of 2 weeks (10 open days) (see Table 13.4). Each sprint started with a *Sprint Planning* of ∼2h and ended with a *Sprint Review* of ∼1h and a *Retrospective* of ∼1h.

At the start of the second sprint, and with the purpose of helping the groups to continuously verify the quality of their code, we conducted a training on *Continuous Code Inspection* (see Section 13.4.4.4). Moreover, groups were encouraged to perform *Pair programming* sessions, to *Continuously Document* their code and to perform *Unit Testing* for the critical features. The initially produced design documents were continuously maintained throughout the development sprints. We refer to this practice as *Continuous Documentation*[16]. Groups were encouraged to perform *Model Storming*[17] sessions whenever necessary.

Final Release

The final release is a special sprint where the student groups finalize the development, deploy their web applications on the application server and define the test case scenarios for user acceptance testing. Two review types are performed: a functional review (more focus on technical aspects) and a customer oriented review (more focused on soft skills).

13.4.4 Key Learnings and Recommendations

Conducting the course as an observational study made it possible to study the behavior of student groups using Scrum for the first time. Generally speaking, and according to our observations and to the survey conducted at the closure of P1 (N=23), the students opinions' regarding the implementation of Scrum in such courses were overwhelmingly positive: 65% reported that they found it as useful (and 13% as very useful) and 86% reported that they would recommend agile methods for future projects. Moreover, most of the practices were reported as being relevant to their context, even though we were able to observe that they were not systematically applied.

The survey also allowed us to collect some information regarding the customization needs. Mainly, it shows that the questioned groups (T1 to T4) didn't found relevant pair-programming, user stories estimation (relative and time-based) and burndown charts (i.e., velocity tracking).

Even thought very few configurations were tried out by the teams during P1, when answering the 4th question of the survey (see Table 13.6), the students provided the rational why some practices were found irrelevant which was helpful to formulate the final recommendations (see Section 13.4.4).

---

[16]`http://agilemodeling.com/essays/documentContinuously.htm`
[17]`http://agilemodeling.com/essays/modelStorming.htm`

Table 13.4.: MDL Course Schedule for P1, P2 and P3

Sprint-0	Weeks 1-3	Practical agility workshop
		Technical workshop (P1, P2)
		Customer representatives interview (P1, P2) or World Café (P3)
Design Sprint	Weeks 4-5	Domain walkthrough (P1, P2, P3)
		Design Sprint planning
		Design Sprint review (with technical consultants)
Sprint-1	Weeks 6-7	Testing workshop (in anther course) (P2, P3)
		Sprint planning
		Sprint review
Sprint-2	Weeks 8-9	Product quality workshop (SonarQube) (P2, P3)
		Process quality workshop (retrospectives)
		Sprint planning
		Sprint review
Sprint-3	Weeks 10-11	Sprint Planning
		Sprint Review
Sprint-4	Weeks 11-12	Sprint Planning
		Test cases for user acceptance testing (P2)
		Sprint Review
Final Release	Weeks 12-14	Deployment (P1, P2)
		Functional Review
		Final Customer Review (P1, P2)

The observation of retrospectives and the examination of the identified improvement actions also revealed valuable information regarding the practices suitability and the needs for customization even though some improvement actions were vague (especially in P1). This tends to prove the need for a better support of such a practice.

The examination of Jira, namely the product backlog, sprints task boards, sprints reports, velocity charts and burndown charts also allowed us to assess the groups ability to define well-formed user stories, to estimate them and to track their own progress.

Finally, the results of the capitalization workshops conducted at the closure of P2 and P3, allowed us to encode some customization decisions and to identify a set of potentially interesting configurations.

314

13. Illustrations

Table 13.5.: MDL customization matrix

Agile practices	Domain Expertise (Low)	Process Expertise (Low)	Technical Expertise (Low)	Sense of Accountability (Low)	Time Pressure (High)	Scope Variability (High)	Distribution (High)	Project Dedication (Halftime)	Scope Complexity (High)	Power Distance (High)
Req. Brainstorming[C34]	+	?	?	?	?	?	?	?	+	?
Backlog Grooming[C35]	C	?	?	?	?	+	?	?	?	?
Proxy PO (Student)[C36]	?	?	?	−	?	?	?	?	?	?
Relative Estimation[C37]	?	C	?	?	?	?	?	?	?	?
Time Estimation (Tasks)[C38]	?	+	?	C	?	?	?	?	?	?
Taskboard[C39]	?	C	?	?	C	?	?	?	?	?
Sprint Planning[C40]	?	?	C	?	?	?	?	?	C	?
Architecture Envisioning[C41]	?	?	C	?	?	?	?	?	+	?
Domain Walkthrough[C42]	+	?	?	?	?	?	?	?	?	?
Iterative Modeling[C43]	?	?	?	?	C	?	?	?	?	?
Test First Design[C44]	?	?	C	?	C	?	?	?	?	?
Code Inspection[C45]	?	?	?	?	C	?	?	?	?	?
Pair Programming[C46]	?	?	?	?	C	?	?	?	+	?
Story Mapping[C47]	?	?	?	?	?	+	?	?	+	?
Daily Meeting[C48]	?	?	?	?	?	?	C	C	?	?
Retrospective[C49]	?	?	?	?	C	?	?	?	?	C

The remaining of this section provides an overview of data collected and analyzes the major customization decisions that were made by the different teams during the 3 projects.

13.4.4.1 Requirements Gathering

There exists several practices that product owners usually use to gather the requirements and to help share a common understanding of the project with the development teams. One of the objectives of this course was to get students more involved in the process of requirements gathering and thus more familiar with *Backlog Grooming* (the main responsibility of product owner consisting of writing user stories and continuously refining them).

The definition of EPICs and User Stories appeared to be particularly challenging. Indeed, when examining the product backlogs defined by the student groups in Jira, it usually appears that user stories are so coarse grained or vague that they wouldn't allow a complete comprehension of the feature or that they they cannot be implemented in just one sprint.

Moreover, we often observed that the groups discover the real complexity of user stories during the development sprint which reveals that the planning was not sufficiently granular.

To address these issues, we used the following facilitation techniques:

- **User Stories Exercises**: During the agility workshop, we insisted on the concept of the *Definition of Ready* which specifies that a ready user story (a user story that is ready to be pulled to the sprint) is necessarily *clear, feasible, estimated and testable*. We also proposed a set of practical exercises for writing user stories. Moreover, we proposed to the teams to first identify EPICs, i.e., capture coarse-grained features without committing to the details. Then break epics into more detailed user stories.

- **User Stories Gathering Meetings**: At the start of the Sprint-0 and in order to help the groups understand the requirements and instantiate the Product Backlog, we implemented some facilitation techniques for requirements gathering.

  In P1 and P2, we organized *Customer Interview* sessions. Student groups interviewed the customer representatives twice during a 1h meeting (one per group) to discuss the unclear and yet ambiguous parts of the project.

315

In P3, we conducted a *Requirements Brainstorming* session using the *World Café* technique (a structured brainstorming process for knowledge sharing in which groups of people discuss a topic at several tables and switch periodically except of one table host)[18]. The practice was found as helpful regarding to the requirements flexibility (or volatility).

The face to face communication with the customer representatives was found useful to get the students better understand the requirements. However, students were sometimes expecting customers to be more prescriptive. The fact that they do not systematically make decisions about the way features should be developed was perceived negatively.

- **Student as Proxy Product Owner (PO)**: In order to assist the team writing user stories, we experienced in P1 to attribute the role of a Proxy PO to one student. The proxy PO had the responsibility to contact the customer representative and the product owners (assistant teachers) to ask for more precisions whenever needed, to help the team make sound decisions on the business process and to maintain the backlog on Jira. As shown in Table 13.6 and in the answers of students to open-ended questions from the same survey, the practice was found to not have a real added value. This may be explained by the avoidance of individual accountability. Such a role represents more responsibilities. The student groups in P1 report that they would better prefer to share this accountability.

- **Story Mapping**: As earlier mentioned, in P3, we allowed more flexibility on the scope and the prioritization of the features. To help the team instantiate and organize their product backlog in such a context, we decided to implement the story mapping practice which consists of a collaborative top-down approach of requirement gathering. The practice aims at ordering user stories along two independent dimensions. The generated "map" arranges stories along the horizontal axis in rough order of business value and along the vertical axis, in order of development priority or sophistication. The practice was particularly found helpful and all the 4 teams of P3 continued using it for their backlog grooming. When asked about its relevance during the capitalization workshop, they reported that the practice helped them to decide about the scope refinement since they could focus on more valuable user stories.

Regarding the aforementioned, the following recommendations can be formulated:

---

[18]http://www.theworldcafe.com/

[C34] Backlog Grooming (involving the team), ***if carefully implemented/customized, is helpful*** to satisfy ***detailed and valuable requirements definition*** in the context of ***low Domain Expertise***

[C35] Requirements Brainstorming ***is helpful*** to satisfy ***detailed and valuable requirements definition*** in the context of ***low Domain Expertise***

Requirements Brainstorming ***is helpful*** to satisfy ***detailed and valuable requirements definition*** in the context of ***high Requirements Volatility***

[C36] Student as Proxy PO ***is harmful (counterproductive)*** to satisfy ***detailed and valuable requirements definition*** in the context of ***low Sense of Individual Accountability***

[C37] Story Mapping ***is helpful*** to satisfy ***detailed and valuable requirements definition*** in the context of ***high Scope Variability***

13.4.4.2 User Stories Estimation

Despite our focus on the estimation of user stories during the agility workshop, this practice was found to be particularly challenging. More precisely, we have been able to observe the following issues:

- **No estimation**: the examination of the sprints planning in Jira show that some groups simply skipped user stories estimation in the first sprint (we purposely didn't remind them to estimate their stories) (see Table 13.7 and Figure 13.1).

- **Estimation changes**: another common issue that we have been able to observe is the change of the initial user story estimation during the sprint. A typical situation is that the sprint is launched without encoding the initial estimation (which would be done after, typically in the first day of the sprint).

- **No consensus-based estimation**: we also have been able to observe that the estimation of user stories was sometimes exclusively done by the Scrum Master without involving the team members, i.e., the estimation was not consensus-based as it is recommended by agile methods (no use of *Planning Poker* or an alike technique). The issue was particularly discussed during the capitalization workshop of project P2.

Table 13.6.: Practices Appreciation (2015-2016), N= 23

1. Rate your overall experience with agile methods			
Poor	4.34%		
Fair	17.39%		
Good	65%		
Excellent	13%		

2. Would you recommend to apply agile methods in similar projects?			
Yes	86.95%		
No	13.04%		

3. Rate the relevance of the following practices (to your context)
Not at all Helpful, not Helpful, Neutral, Helpful, Very Helpful

	Helpful	**Neutral**	**Not Helpful**
2 weeks sprint	91.30%	4.34%	4.34%
Team Room	86.95%	4.34	8.70%
Standup Meetings	82.60%	8.70%	8.70%
Pair Programming	34.78%	39.13%	26.08%
Agile Modeling	73.91%	4.34%	21.73%
Sprint Review	86.95%	4.34%	8.70%
Sprint Retrospective	65.21%	13.04%	21.73%
User Stories	82.60%	4.34%	13.04%
Planning Poker	30.43%	43.47%	26.08%
Relative Estimation	39.13%	4.34%	56.52%
Time Estimation	60.86%	4.34%	34.78%
Student as Proxy Product Owner	43.47%	8.70%	47.82%
Student as Scrum Master	56.52%	8.70%	34.78%
Physical Task Board	52.17%	26.08%	21.73%
Virtual Task Board	73.91%	4.34%	21.73%
Burndown Chart	43.47%	8.70%	47.82%
Unit Testing	86.95%	4.34%	8.70%
Acceptance Testing	82.60%	13.04%	4.34%
Continuous Code Inspection	82.60%	8.70%	8.70%
Continuous Documentation	43.47%	8.70%	47.82%

4. How the previously cited practices were eventually improved?
...

5. Rate the relevance of the following facilitation workshops
Not at all Relevant, not Relevant, Neutral, Relevant, Very Relevant

Customer interviewing	73.91%	13.04%	13.04%
Domain walkthrough	73.91%	21.73%	4.34%
Agility workshop	56.52%	21.73%	21.73%
Agile testing workshop	73.91%	21.73%	4.34%
Retrospectives workshop	65.21%	13.04%	21.73%

During the retrospectives workshop (Sprint 2) (see Table 13.4), we particularly discussed these issues and reminded the importance of estimation. We also configured Jira so that the groups that want it can use time to estimate the subtasks of user stories.

The survey shows that students found time estimation even more helpful than user stories. Some students reported in the Capitalization workshop that they found it easier to discuss in terms of time. The time tracking encouraged them to use Jira more frequently. However, regarding their avoidance of individual accountability, some groups discarded the use of time estimation and preferred to track velocity based on the number of completed tasks (issues in Jira).

Table 13.7.: Reported Story Points via Jira (committed vs. completed per sprint)

	Sprint 1		Sprint 2		Sprint 3		Sprint 4		Relative Error (%)
T1	0	6	42.5	37.5	33	0	21	36	50%
T2	No estimation		0	6	38	38	12	21	55.55%
T3	No estimation		47	6	89	8	97	124	57.30%
T4	17	17	0	71	71	51	40	31	50.25%
T5	21	21	78	66	102	70	68	28	31.22%
T6	25	25	81	81	107	53	75	50	27.43%
T7	20	20	35	31	62	57	21	21	3.62%
T8	20	20	6.5	12.5	66.5	66.5	45	43	5.55%
T9	84	76	82	44	119	74	74	35	36.21%
T10	62	25	51	19	190	37	161	111	43.75%
T11	0	21	153	62	309.5	219.5	281	282	26.51%
T12	No estimation		32	5.5	51	7	157	120	44.79%
T13	0	12	52	39	83	44	101	58	43.1%

Regarding the aforementioned, the following recommendations can be formulated:

[C38] Relative Estimation ***if not carefully implemented/customized, is harmful*** to satisfy ***an efficient velocity tracking*** in the context of ***low Process Expertise***

[C39] Time Estimation ***is helpful*** to satisfy ***an efficient velocity tracking*** in the context of ***low Process Expertise***

Time Estimation *is harmful* to satisfy *an efficient velocity track-*
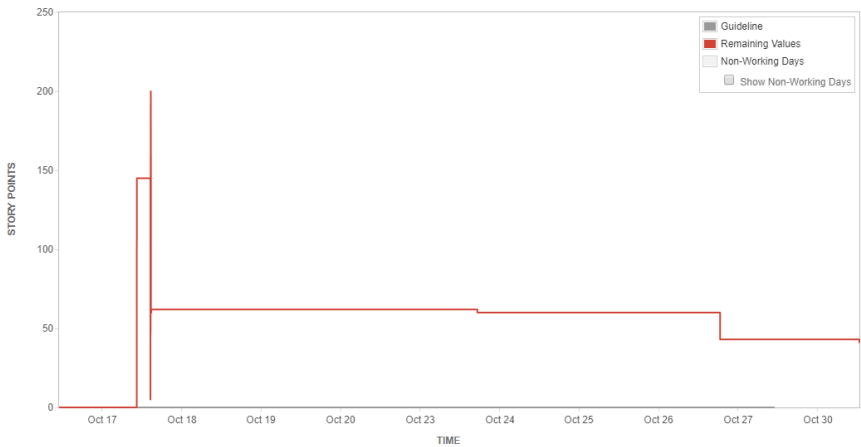
Figure 13.1.: Burndown chart example - Scope change during the sprint (T11)
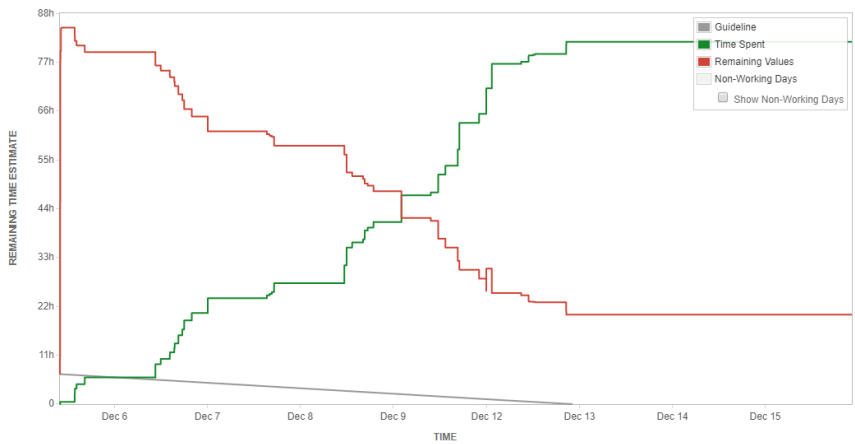
Figure 13.2.: Burndown chart example - Time tracking of tasks (T9)

13.4.4.3 Velocity Tracking (Continuous Update of Task Boards)

The main idea behind velocity tracking is to provide a lightweight method to measure the pace at which teams consistently deliver business value for future planning purposes. Velocity is measured based on the point estimates of the completed user stories and/or the time estimates of the user story tasks. Since teams encountered several estimation issues, the tracking of velocity was not always effective.

Precisely, we observed two main issues regarding velocity tracking: no frequent update of task boards and the addition or removal of tasks during the sprint (see Figures 13.1 and 13.3). For example, Figure 13.3 shows an example of a sprint closed later than expected and without an update of the user stories status.
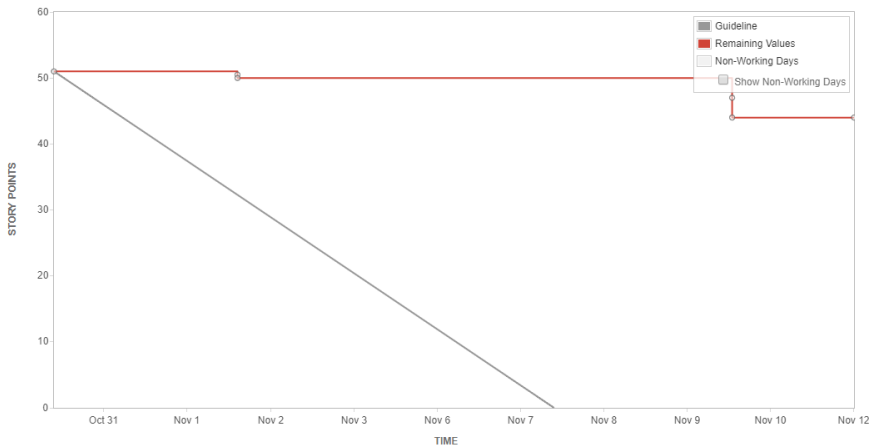
Figure 13.3.: Burndown chart example - Scope change during the sprint (T8)

These issues suggest that more attention needs to be payed to raise the students awareness of the importance of the frequent update of task boards. When asked about the reasons for the misuse of practice, students pointed out the time pressure. Indeed, during the last sprints they are usually so focused on the functional completeness that they considered the update of Jira task boards as a waste of time.

Regarding the decomposition of user stories into tasks during the sprint, they reported that it is difficult to anticipate and plan all the tasks during the *Sprint Planning* because of the complexity of some features and/or the

difficulty to estimate the development effort from the technical point of view.

Regarding the aforementioned, the following recommendations can be formulated:

[C40]   Continuous Update of Taskboard ***if not carefully implemented/-customized, is harmful*** to satisfy ***an efficient velocity tracking*** in the context of ***low Process Expertise***

Continuous Update of Taskboard ***if not carefully implemented/-customized, is harmful*** to satisfy ***an efficient velocity tracking*** in the context of ***high Time Pressure***

[C41]   Sprint Planning ***if not carefully implemented/customized, is harmful*** to satisfy ***an efficient velocity tracking*** in the context of ***high Requirements Complexity***

Sprint Planning ***if not carefully implemented/customized, is harmful*** to satisfy ***an efficient velocity tracking*** in the context of ***low Technical Expertise***

13.4.4.4 Agile Design

As discussed in Section 2.2.2, agile design is emergent. There is a range of agile design practices from the high-level architectural practices to the low-level development practices that intend to help teams iteratively improve their design. In the context of our capstone course, we introduced a set of practices from the *Agile Modeling* method (see Section 2.2.3.4).

Precisely, *Architectural Envisioning*[19] is a practice that consists of undertaking a high-level architectural modeling at the very beginning of a project and then performing further detailed modeling during the development sprints. The practice was applied to our context during the *Design Sprint*. Regarding the projects complexity, the practice helped the student teams identify and think about the critical architecture-level issues (system distribution, multi-platform communication (android and java web), database replications, data access layer, etc.) and therefore reduce several technical risks early in the life-cycle.

During the design sprint, the student groups were also asked to work on some high-level design models to get a better understanding of the entire application domain (not just the features that are captured in the final solution). With this purpose, we conducted a *Domain Walkthrough* session

---

[19]`http://agilemodeling.com/essays/agileDesign.htm`

where we further discussed the details of the business domain and assisted the groups brainstorm on the creation of a *Domain Model*. At the end of the design sprint, we reviewed the produced artifacts and provided feedback on the architectural choices. The students reported that the practice was helpful and that having more alike sessions would be even more helpful (see Table 13.6).

Other agile design practices were helpful but more challenging. These consist of *Iteration Modeling*, *Continuous Documentation*[20], *Test First Design* and *Continuous Code Inspection*.

Iterative Modeling[19] consists of performing lightweight modeling sessions for a few minutes at the beginning of each sprint to help the team identify its design strategy for the sprint. The practice is complementary with *Model Storming*. The student groups reported that such sessions where helpful regarding to the complexity of features but that they we not systematically performed at the start of the sprint. They would rather not plan the design needs for the sprint a nd brainstorm when a specific design model needs to be refined. Students, especially in P1 and P2, also reported to lack of time to *Iteratively Document* their design changes. This explained by the fact that the scope of the two projects was larger and more complex then in P3. We therefore scheduled an additional week to allow students to finish their documentation.

Test First Design[19] which consists of writing a single test before writing enough production code to fulfill that test was almost not applied. The main reason for this is that the practice is very time consuming and that the students lack of sufficient expertise in unit testing.

Continuous Code Inspection or *Refactoring* consists of making small changes to the developed solution without changing the code semantics and thus, whenever needed. To help students continuously refactor their code, we conducted a training on *SonarQube*[21] and *SonarLint*[22], a tool and its IDE extension for the statical and dynamic inspection of the code quality. The groups were encouraged to use the tool to continuously inspect the quality of their code. Students overwhelmingly agreed on the relevance of the practice, however, regarding the lack of time few of them frequently used the SonarQube server to analyze the whole code. They preferred to user the lightweight SonarLint for IDEs which helps to get instantaneous feedback as the code is typed but do not provide means to analyze the whole project to find bugs.

---

[20]http://agilemodeling.com/essays/documentContinuously.htm
[21]https://www.sonarqube.org/
[22]https://www.sonarlint.org/

Regarding the aforementioned, the following recommendations can be formulated:

[C42] Architecture Envisioning **is helpful** to satisfy **reduced technical risk** in the context of **high requirements complexity**

[C43] Domain Walkthrough **is helpful** to satisfy **requirements understandability** in the context of **low domain expertise**

[C44] Iterative Modeling **if carefully implemented/customized, is helpful** to satisfy **continuous design improvement** in the context of **high time pressure**

Iterative Modeling **is helpful** to satisfy **continuous design improvement** in the context of **high requirements complexity**

[C45] Test Driven Design **if carefully implemented/customized, is helpful** to satisfy **continuous design improvement** in the context of **high time pressure**

[C46] Continuous Code Inspection **if carefully implemented/customized, is helpful** to satisfy **continuous code improvement** in the context of **high time pressure**

13.4.4.5 Pair Programming

Pair programming is another practice intended to continuously improve the quality of the produced code, i.e., to continuously reduce the technical debt. Very few groups have experienced pair programming or pair design because it was found very time consuming. Regarding to our observation, the practice was seemingly helpful when implemented for the development of complex features such as the routing algorithm and the data access web service. Particularly, we observed that when students worked in pairs on the routing algorithm, they were able to produce high quality solutions (the observation is worth to be validated through a controlled experiment).

Student groups reported different configurations to pair programming among which: (a) pair programming/testing (a variation of pair programming where a student writes a test and the others write code to make it pass) and (b) pair programming with one pivot role (designate each sprint a pivotal role that can be solicited for pairing when necessary).

Regarding the aforementioned, the following recommendations can be formulated:

[C47]   Pair programming **if carefully implemented/customized, is helpful** to satisfy **code quality** in the context of **high time pressure**

Pair programming **is helpful** to satisfy **code quality** in the context of **high requirements complexity**

13.4.4.6 Daily Meeting

The implementation of daily meetings was found helpful but needed to be configured regarding because of distribution and because of the fact that students were not fully dedicated to the project (i.e., they had other courses in parallel). Maintaining the same schedule and daily to preform the meeting was simply not feasible.

Students configured the practice in different ways: daily feedback on slack[23] (a professional messaging system), variable schedules for daily meetings (a planning done at the start of the sprint), a daily meeting on Skype, etc.

Regarding the aforementioned, the following recommendations can be formulated:

[C48]   Daily Meeting **if carefully implemented/customized, is helpful** to satisfy **a better team collaboration** in the context of **high distribution**

Daily Meeting **if carefully implemented/customized, is helpful** to satisfy **a better team collaboration** in the context of **no full dedication to the project**

13.4.4.7 Retrospectives

In P1, a set of retrospective techniques were presented to the groups at the start of Sprint 2. We asked each group to choose some specific techniques to experiment. A list of improvement actions was maintained by each group (on their physical board and documented at the end of the sprint). An average of 1,6 completed improvement actions per sprint were reported.

In P2 and P3, the students only used the AMQuICk improvement and capitalization boards presented in Section 12.5.2. Once the improvement actions elicited, they were encoded in Jira as a *"Work Improvement"*. A specific extension to Jira (see Figure 13.4) was developed to enable teams encode their practice-based improvements. An average of 3,6 improvement actions per sprint were reported as completed by the groups of P2 and P3.

---

[23]https://slack.com/

These were later used as an input for the capitalization workshop. Some examples of concrete improvement actions are the following:

- **Practice**: Versioning
  Improvement Goal: Maintainability
  Improvement Action: Convention naming for commit messages
  Priority: Medium
  Reporter: T6 (Project 2)

- **Practice**: Functional Testing
  Improvement Goal: Reliability
  Improvement Action: Pair functional testing of developed features
  Reporter: T5 (Project 2)

- **Practice**: Functional Testing
  Improvement Goal: Reliability
  Improvement Action: Maintain a list of bugs and technical impediments
  Reporter: T12 (Project 3)

- **Practice**: Code Integration
  Improvement Goal: Correctness
  Improvement Action: Stop coding in the last 2 days of Sprint in order to have enough time to integrate, design test scenarios and test.
  Reporter: T7 (Project 2)

We have been able to observe several issues regarding retrospectives. First, in some cases retrospectives were not really effective since no concrete actions were decided at the end of the meeting. It also happened that a retrospective is skipped because of the time pressure, the non availability of team members (distribution, or other courses in parallel) or demotivation at the end of sprints.

Another common issue consists of the fact that improvement actions are formulated in such a way that they cannot be implemented in one sprint or cannot be verifiable. Example of vague improvement actions that were formulated are the following: **improve functional testing** or **provide scrum master with more feedback**.

Finally, our participation as observers was also reported as harmful since it impacts the team's openness and ability to discuss the issues freely.

Regarding the aforementioned, the following recommendations can be formulated:
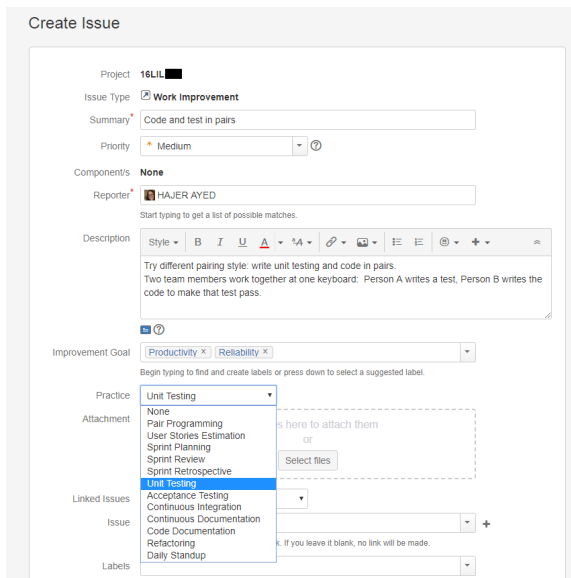
Figure 13.4.: Jira Extension to encode identified practice improvements

[C49]   Retrospective *if carefully implemented/customized, is helpful* to satisfy *continuous process improvement* in the context of *high time pressure*

Retrospective *if carefully implemented/customized, is helpful* to satisfy *continuous process improvement* in the context of *power distance* (between teachers and students)

13.5 Summary

In this chapter, we did managed to provide different illustrations, with different contexts and different representations of the customization knwoledge. The results tend to give confidence in the feasibility and pertinence of the AMQuICk approach. However, we cannot ignore the threats to validity that surround this evaluation work.

First, as mentioned in Chapter 12, the validation of the AMQuICk matrices is intrinsically complex, and would ideally require the customization knwoledge to be confronted to other experience reports from different practitioners. Indeed, the study of similar contexts overtime with different and multiple

participants could in all likelihood reveal different behavior and other representations of the customization knowledge. Moreover, we only collaborated with few end-users (SPW and MDL case studies) and we played the role of the facilitator that encode the generated knwoledge, which inevitably reduced the potential divergent uses of the matrices. Finally, the feedback of practitioners when implementing the approach in real-life projects is still lacking. To this regards, in the next chapter, we will focus on collecting feedback from practitioners regarding the relevance of the approach and discuss its possible improvements.

Part VI.

Closing Comments

In this part of the dissertation, we discuss the AMQuICk approach and envision possible future works. In particular, Chapter 14 summarizes the specificities and merits of the approach and discusses its limitations and improvements that could be considered. Chapter 15 finally concludes this dissertation.

Chapter 14

Discussion

Contribution, Practitioners' Feedback and Perspectives

In the previous parts of the thesis, we iteratively constructed a theoretical framework for guiding agile methods customization. This chapter summarizes the research outcomes and discusses the framework limitations and future research perspectives.

First of all, Section 14.1 reviews the main contributions and specificities of the AMQuICk framework. Secondly, Section 14.2 gives insights on how this framework could be perceived by the industry. Then, Section 14.3 discusses the current limitations and improvement points. Finally, Section 14.4 provides an overview of the research perspectives and future works.

14.1 Contribution

During the course of this research work, we developed a theoretical framework to agile methods customization called AMQuICk in an attempt to provide a better support for practitioners going through agile methods deployment in challenging or non-sweet spot contexts.

First of all, based on a review of the research context and gaps (see Chapters 2 and 3), we iteratively identified the following core perspectives to explore for the construction of our framework:

1. **Situational Method Engineering (SME) perspective** (see Part III): SME is a disciplined approach for building situational methods by selecting, tailoring and integrating a set of method components. The latter are designed using a meta-method modeling process and are stored in a repository of reusable components. In this perspective, our objective was to investigate the opportunity to use a SME approach to customize agile methods without impeding the agility values.

331

2. **Context Study perspective** (see Part IV): A thorough understanding and characterization of the context is needed for enabling the elicitation of accurate, precise and reusable customization knowledge. In this perspective, our objective was to study in practice the implementation of agile methods in different non-sweet spot contexts with the purpose of supporting our customization approach with an extensible and accurate representation of development contexts.

3. **Customization and capitalization perspective** (see Part V): The experience of practitioners, if conveniently structured and interpreted, provides a valuable customization knowledge that can be reused to formulate recommendations intended to guide other practitioners. This is what we refer to as experience-based learning. In this perspective, our objective was to find out a practical way to structure the customization knowledge and to interpret it.

The exploration of these perspectives, following the design science research framework and relying on a set of founding principles (inherited from the field of software process improvement and from agile best practices), allowed us to successfully design this approach to agile methods customization in a practical framework.

This Agile Methods Quality Integrated Customization framework (AMQuICk) defines an *iterative incremental* and *multi-level cycle* adapted from the Quality Improvement Paradigm (QIP) [Basili and Caldiera, 1995] where the *team process learning* is structured into a reusable *organizational knowledge*. Its core activities consist of: (1) the context assessment, (2) the recommendation of suitable practices, (3) the continuous improvement of the implemented method, (4) the capitalization of the project-team learnings and the generation and reuse of the customization knowledge. These activities, associated with a set of designed components, have the potential to provide agile organizations and teams with the necessary support for guiding context-aware implementation of agile methods.

The framework allows the description of agile methods through a dedicated method engineering metamodel called *AMQuICk Essence*. The metamodel helps to support the *agile facilitators* structure and organize their knowledge regarding agile practices. It also provides means for capturing the context and for characterizing the customization knowledge in a fine-grained way so it can be easily reused by other practitioners.

AMQuICk relies extensively on the involvement of the *development team* in the process of agile methods customization and therefore the metamodel is supplemented with a set of tools that improve its usability. Precisely,

it involves a *Repository* of agile practices that can be browsed or used to document the newly introduced practices. Additionally, it provides a *Practice Cards Modeler* to design informative user-friendly practice cards that the teams visualize in their working space to ensure that all stakeholders share the same vision on the process and its evolution throughout the project.

Regarding the usability of the customization knwoledge collection, reuse and capitalization, AMQuICk is enhanced with decisional matrices and with some facilitation techniques. AMQuICk customization matrices aim to capture the team expertise more easily and in a more compact and systematic way while the facilitation techniques aim to assist practitioners to capture and capture their expertise during the enactment of their method and in the final project retrospective.

The proposal of this framework and its constituent components allows a continuous back-and-forth exchange of learnings between the organizational level and the project-teams level. The approach has the potential to be applied at a broader scale if the learnings are exchanged at a community level, i.e., at an inter-organizational level.

Since the framework was iteratively grounded, its usability has been continuously discussed through a set of examples, illustrations and case studies. However, its overall effectiveness and relevance to agile practitioners still needs to be validated through an implementation in the industry. This validation would require the active participation of several practitioners and an extensive timeframe for observing, collecting, analyzing and comparing the assessment data from different industrial settings. Therefore, as we come to the end of this research, we rather propose in the next section to explore insights from experienced agile practitioners on how this framework could be perceived by the industry.

14.2 Practitioners' Feedback

A common approach for validating the relevance of new research frameworks to industry is to conduct a longitudinal study, a research method consisting of gathering data from the same subjects repeatedly over a period of time. In this kind of studies, one or more organizations use the newly designed framework and researchers collect data following a specific protocol to evaluate the relevance of the framework with regards to different quality criteria. Regarding the aforementioned, a longitudinal study would be a suitable approach to evaluate the relevance of the AMQuICk Framework to practitioners [Yin, 1994; Vitalari, 1985].

However, the main challenges of this kind of evaluative studies is that they call for consequent amount of time and require practitioners to buy into the use of the proposed framework before any empirical evidence can be gathered. The latter requirement is particularly challenging. Indeed, even though several agile facilitators are highly interested in the question of agile methods contextualization, they usually are reluctant to use a new framework with no evidence of its validity.

A good alternative to longitudinal case studies is to gather feedback from agile experts as a preliminary assessment of the relevance of the framework to industry. This feedback may be qualitative or quantitative and should highlight the expected benefits as well as the potential problems to the framework implementation. This kind of straightforward evaluation would increase the credibility of the framework and prepare for further empirical validation. Moreover, it would increase the likelihood to get practitioners experience the framework in the future.

The next section describes the procedure used to collect feedback from agile experts.

14.2.1 Feedback Collection

Our initial idea was to collect data using a survey that would be sent to the agile practitioners of different expertise levels through agile community groups[1,2]. However, since the potential practitioners are not informed about the framework, it would have been necessary to supplement the survey with material on the framework life-cycle, metamodel, context modeling and assessment and customization knwoledge capitalization in AMQuICk matrices. The problem with this approach is that practitioners with their tight schedules, will presumably not take the time to read such material and fill out a survey.

Another possibility was to collect feedback from available practitioners face-to-face which is more in adequacy with the collaborative nature of agility. In the context of our previous case study on culture-based customization (see Chapter 11), we were able to have the commitment of 6 agile experts which we visited for a 1h30 semi-structured interview session. These meetings appeared to be a good opportunity to present the framework face-to-face to experienced practitioners familiar with the agile adoption and improvement issues. Since then, we interviewed two more practitioners and therefore

---

[1] https://www.linkedin.com/groups/49087
[2] https://www.meetup.com/fr-FR/Agile-Belgium/

Table 14.1.: Interview Steps and Questions

Discuss the way agile practices are currently customized	
1.a. How your agile practices are currently customized?	
1.b. What outstanding customizations have been proposed so far and who initiated them?	
Present and discuss the framework	
2. Would you suggest any improvements or adjustments to the approach?	
3. Rate the following liker-scale questions	
Comprehensiveness	3.a. To what extent do you agree that the overall objective of the approach is comprehensive?
Relevance	3.b. To what extent do you agree that the approach is appropriate/relevant to agile practitioners?
Practicality	3.c. To what extent do you agree that the approach is practical and is applicable in industry?
Necessity	3.d. To what extent do you think that the capitalization of the customization knowledge is necessary to agile teams?
4. Do you have any additional comments?	

a total of 8 participants provided us with their feedback regarding the AMQuICk framework.

In order to assure a response at the time of our visit, the interview questions had to be short and concise, especially that we also interviewed them about their context, the practices they implemented so far and their cultural issues (see Chapter 11 and Appendix A).

First, we asked them about their current agile practices improvement and customization process. Second, we gave a presentation on the AMQuICk framework (AMQuICk Essence, repository, context modeling, customization matrices and capitalization tools) and, based on their specific context, provide them with examples on how the framework might be used. Then, based on a 5-point Likert scale, from 1 "Strongly disagree" to 5 "Strongly agree", we asked them to rank the framework according to the following criteria: *comprehensiveness*, *practicality*, *necessity* and *relevance*. During the presentation of AMQuICk and when ranking the approach, participants expressed their concerns through informal discussions. Finally, there were 2 open-ended questions about the possible changes and enhancement of

the framework. The answer to these questions coupled with the informal discussion, helped to elicit a better feedback from participants about the possible improvements and considerations for future work. Table 14.1 shows the questions that were asked.

The 8 participants are all from different companies, working on different contexts and have different agile expertise (see Table 14.2). They all are either managers or agile facilitators. If a larger number of practitioners was available, it would have been interesting to differentiate the feedback analysis based on their expertise level but with only 8 participants this would have been meaningless.

The next section presents an analysis of the feedback gathered from the participants.

Table 14.2.: Interviewees profiles

ID	Role	Agile Expertise	Method	Domain
P1	Dev. Manager	Expert	Kanban / Lean	B2B
P2	Product Owner/Coach	Intermediate	Scrum/Kanban	Real Estate
P3	Product Owner	Expert	Scrum/Kanban	Oil&Gas, E-Gov
P4	Dev. Manager	Expert	Custom	E-Gov
P5	Scrum Master (SM)	Novice	Scrum/Kanban	E-Gaming
P6	SM/QA Manager	Expert	Scrum/AUP	B2B, Banking
P7	Portfolio Manager	Novice	Custom Scrum	E-Gov
P8	Analyst/SM	Intermediate	Scrum	Web/Mobile Dev.

14.2.2 Feedback Discussion

Due to the nature of the feedback and to convenient sampling from the agile community, the number of participants was relatively small and thus, no complex statistical analysis was possible. Therefore, we mainly discussed the feedback of practitioners qualitatively. Only few quantitative discussion is made based on simple statistical descriptions and comparison of percentages.

The main feedback and improvement suggestions (which are identified from F1 to F13) are classified according to the framework components and summarized in Table 14.3. These are identified detailed in the following paragraphs.
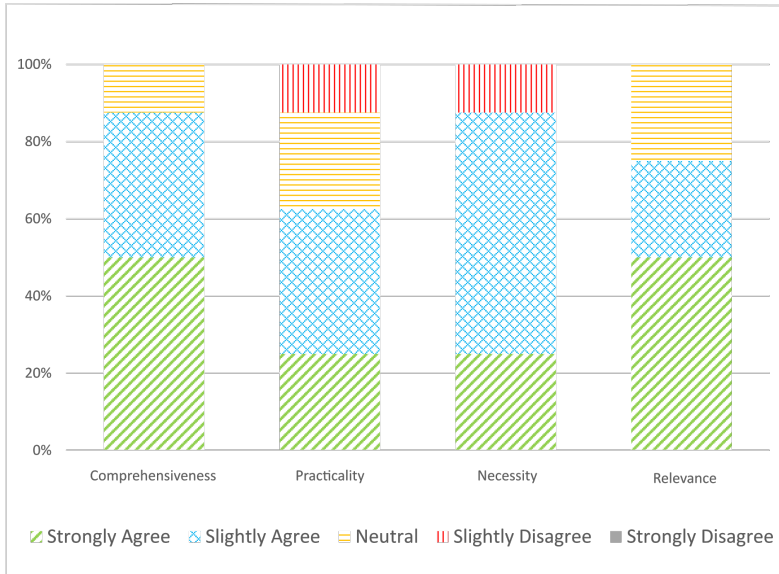
Figure 14.1.: Feedback of Agile Experts (8 participants)

Comprehensiveness

As shown in Figure 14.1, almost all participants except one (who ranked comprehensiveness as neutral) reported to understand the approach, with 50% strongly agreeing and 38% agreeing to its comprehensiveness. The overall mean is 4,25 (between slightly and highly agree).

In the informal discussions, practitioners mainly pointed out that some of the chosen terminologies were found to be difficult to understand or inappropriate (F6). In particular, one practitioner (P6) suggested to use more familiar terms to industry. He would suggest to borrow some terms from the CMMI: *Process Asset Library* rather than *Repository* of practices and *Process Asset* or *Pattern* rather than *Agile Component* or *Building Block*. Another participant (P5) also commented that it would be more appropriate to use the term *Suggestion* rather than *Recommendation* to be less prescriptive and formal.

Moreover, two practitioners (P6 and P8) reported that the customization knowledge need to be systematically supported with more detailed guidelines to ensure the comprehensiveness of the recommendations (F12). This comment confirms what we discussed in Section 12.3.2 regarding the possibility to supplement the AMQuICk recommendations with further information to

improve their usability. This is by all means a possible use of the *Recommendation* construct type of the metamodel. However, further investigation is needed to decide on whether or not we should constrain practitioners that report their customization knowledge to document detailed guidelines. Especially since another practitioner (P5) reported in contrast that recommendations should not be very specific.

Practicality

While none of the participants highly disagreed about the practicality of the framework, the strong consensus reported for comprehensiveness decreased, with 25% of the participants strongly agreeing, 28% slightly agreeing, 25% remaining neutral and 13% slightly disagreeing. The overall mean is 3,62 (a little below agree).

In the informal discussions and when answering the open ended questions, practitioners mainly pointed out that simplicity should be kept when the approach is introduced (F1). Specifically, they reported that process modeling is a complex feature that must be kept as simple as possible to ensure its comprehensiveness by the team and to avoid too much discipline. This is a valid and important concern which we have discussed in Chapter 6 and which we actually addressed by the fact that AMQuICk Essence isn't intended to be directly used by the development teams. Indeed, the latter will rather use the repository of practices and the practice cards.

The overall feedback regarding the AMQuICk matrices was positive. Their utility could have been confirmed using examples from the practitioners' own contexts. For instance, we discussed with P1 the reasons why they switched from Scrum to Kanban/Lean and were able to formulate the following recommendation:

> *Continuous Software Development (no iteration)* **is helpful** (+)
> **to satisfy** *short time schedules*
> **in the context of** *maintenance-like projects (small and fluid tasks)* and *high scope variability*

Nonetheless, we were able to discuss two main concerns regarding the practicality of the customization knowledge capitalization.

Firstly, one concern that we were aware of and that we discussed with P7, consists in the need to provide AMQuICk users with a default taxonomy of context factors. Indeed, as we discussed in Chapter 12, with no default characterization of context factors, each facilitator would have to instantiate a custom context model which would prevent inter-organizational learning.

It seems also reasonable to assume that a default configuration of the quality attributes would enhance the framework practicality.

Secondly, P4 and P6 reported that the ranking of recommendations is an important improvement to consider (F13). Indeed, only the statistical learning may not be sufficient to confirm the well-formedness of a customization decision and recommend it. More precisely, some customization decisions may be very relevant but may face more resistance. We partially address this concern using the *Recommendation Status* construct type but the question may be further investigated, may be by defining a recommendation's confidence level.

Necessity

When it comes to discuss the necessity of the approach, we first had to ask about the current way agile methods are customized in the participants' organizations with the purpose of determining whether they would be interested in a more systematic experience-based customization approach.

Almost all the participants reported that, in their contexts, the major customization decisions are made by managers at an organizational level and that only minor adaptations are decided at the team level. They also report that retrospectives are not always effective since process adjustment or improvement is considered to be *"a matter of the management"* (P5) (which, as discussed in Section 2.2.2, fundamentally contradicts the agility values). This tends to demonstrate the necessity to consider another way to support agile methods improvement and customization other than retrospectives so that the team-level improvements are more valued.

Returning back to Figure 14.1, almost all participants (except P5) agreed with the necessity of having an approach such as AMQuICk to better support the customization decisions at the team level and to capitalize on the teams' experience. 25% strongly agreed and 63% slightly agreed and 13% (one participant) disagreed, with a mean of 3,37 (little above agree).

The discussion of necessity with practitioners showed that they expect the approach to be beneficial for early agile adopters (P3, P7 and P8) (F2). The more experienced agile practitioners may not require to use the approach as a customization guidance but rather as a reference to learn about new agile practices or to document their own practices and practice configurations.

Table 14.3.: Improvements and Suggestions for the AMQuICk framework

	Id	**Feedback: Improvements and Suggestions**	**Reporter(s)**
General	F1	Keep the framework as simple and open as possible	1, 2, 5
		Make sure it is not complex for teams to use	
	F2	Target mainly early adopters to use it	3, 7, 8
		Investigate how it can be used efficiently by experienced agile practitioners (may be use it as reference?)	
Life-cycle	F3	Support the organizational-level (top down) improvement	6
	F4	Study the relationship of the team-level practices and organizational maturity levels	6
Metamodel	F5	Support Organizational Process Modeling	6
	F6	Use more familiar terminologies to practitioners	2, 6, 7
	F7	Consider the dynamic aspects of process modeling	6
Repository	F8	Support organizational-level practices	6
Context Modeling	F9	Consider the team maturity as a context factor	2, 8
	F10	Default taxonomy of context factors	7
Capitalization	F11	Involve customer representatives, conventional project managers, transformation leaders, CEOs, coaches	6, 2, 1
	F12	Investigate when to provide detailed vs. less specific guidance?	5, 6, 8
	F13	Rank the recommendations	6, 4

Relevance

As shown in Figure 14.1, none of the participants disagreed about the relevance of the approach. 50% strongly agreed, 25% agreed and 25% neither

agreed or disagreed. The overall mean is of 4,25 (between strongly agree and slightly agree).

Nonetheless, participants expressed some concerns which relate to the need to contextualize the approach. First of all, as earlier mentioned, the approach was found to be more relevant to the context of new agility adopters (F2) and thus, if the framework is to be adopted in industry, yet immature teams should be the primary targeted users.

Secondly, one participant (P6) suggested that the approach needs to scale up to specific contexts such as large scale projects where the customization has mainly to be done in a top-down way (F3). He argues that relying exclusively on the experiences collected from the team level to recommend customizations is an elusive target. This is an interesting point that we briefly discussed in Section 2.3 and Chapter 3 and that we certainly need to further investigate in the future. Arguments for the fact that the two strategies to agile methods customizations (top-down and bottom-up) and for a hybrid approach[3] can be found in the literature [Rolland et al., 2016; Ambler and Lines, 2016]. In connection with this suggestion (F3), participants reported the need to consider the organizational-level practices (i.e., practices of organizational agility) (F8) and to study the relationship between the team-level practices and the organizational maturity levels (F4) which we argued against in Chapter 3 because the approach would be too much prescriptive and disciplined.

Moreover, a question about the roles to involve in the customization process was raised. Indeed, some participants (P1, P2 and P6) argued the need to involve conventional managers, transformation leaders and coaches in the process of capitalization and particularly during the customization workshops. P2 particularly argued that since the final goal in agile methods is to satisfy the customer, it would be probably interesting to consider the customer representatives in the capitalization of the customization knowledge.

Finally, another improvement suggestion regards the relevance of AMQuICk Essence. P6 suggested that customizations may not only concern static parts of an agile method (i.e., practices, resources and artifacts) (F7). Indeed, he states that the behavior aspects of process models (e.g., the sequence of activities, the flow of workproducts construction, etc.) may also be customized. Despite the pertinence of the suggestion, the choice to not consider the dynamic behavior was made deliberately to not impede the simplicity advocated by agility as argued in Chapter 6. Nonetheless, it remains interesting to investigate the question in future work.

---

[3]`http://www.disciplinedagiledelivery.com/tag/process-tailoring/`

14.3 **Limitations**

In spite of the positive outcome from the case studies and of the overall positive feedback from practitioners, the AMQuICk framework, its underlying theoretical foundations and its artifacts still have to be further investigated. Indeed, the illustrations and the conducted case studies gave confidence in the relevance and applicability of the framework, however more evidence is required so that it gains more maturity and can be perceived as a viable approach that may be applied at an industrial scale. Throughout the design process and the review of the practitioners feedback, we identified and discussed a set of potential limitations that still need to be addressed. These are reported and detailed in the remainder of this section.

First of all, since the research was conducted following build-and-evaluate loops, an evaluation of the whole approach in an actual industrial case study from scratch would be necessary to validate the efficiency of the approach (i.e., whether or not the approach helps practitioners to improve the quality of their development process). Indeed, although we provided several demonstrations on the expressiveness of AMQuICk (precisely, the expressiveness of its metamodel and customization matrices) and on its feasibility in different contexts, its relevance and efficiency to practitioners cannot be confirmed without an extensive empirical testing.

Only through repeated empirical studies in various contexts will the approach collect enough customization knowledge to exploit and therefore show enough evidence of its advantages or reveal more drawbacks that need to be addressed. This empirical validation would require the commitment of practitioners with whom a close collaboration would be necessary to configure the framework, (i.e., to develop individual instances of the context model, context assessment model, repository of practices and customization knowledge base) and to observe its usage overtime.

Additionally, one of the limitations of this research might appear to be the lack of generalizability of the exploratory studies results (see Chapters 9, 10 and 11), as the data collected were specific to particular contexts. The results of these case studies might also be biased by the non-systematic analysis method and by the researcher subjective interpretation of learnings and therefore they should be validated through larger surveys or through more controlled experiments. The researcher role as a facilitator can also be considered as a factor of bias when capitalizing the customization knwoledge in the AMQuICk matrices.

Another limitation of the approach is the fact that we only focused on the expressiveness of the customization knwoledge using AMQuICk Essence

and the customization matrices. The tooling to generate the right recommendations starting from a set of documented experiences is still lacking. Future work on the refinement of the framework should undeniably focus on automating the recommendation process (see Section 14.4).

The scalability of the approach is another aspect that needs to be further investigated in order to ensure its viability in the industry. Indeed, when discussing the approach with agile experts, the question of whether AMQuICk is appropriate in all the contexts was raised. More specifically, the viability of the framework in already mature agile teams or in large-scale organizations should be verified.

Finally, another noteworthy threat to validity regards the fundamental agile value of process simplicity. Indeed, the introduction of a systematic and more disciplined approach to guide the customization of agile methods may be perceived by the development teams as a costly process. Even thought we discussed in Chapter 2 that the implementation of agile methods also require some kind of discipline, we should ensure that the use of AMQuICk is not heavyweight and do not fundamentally impede the simplicity required by the agile manifesto [Beck et al., 2001]. This is an issue that was reported to us during the collection of the feedback from practitioners and when communicating about the framework in conferences. Therefore, the proposal of practical tools to operationalize the framework was an essential part of our design process. The AMQuICk repository of practices, its practice cards modeler, the facilitation tools and the capitalization matrices are all tools that are intended to enable a more lightweight and team-focused customization process. However, their ability to preserve a lightweight way of working should be further assessed in the future. Eventually, their design may be further refined with the purpose of improving their usability to agile teams.

14.4 Perspectives

The research described in this dissertation attempts to answer a complex question on *how* to guide the customization of agile methods. It provides an original view that changes the way customization is currently performed by agile teams. As such, the proposed framework opens multiple perspectives for improvement through further research. The remainder of this section discusses some of these perspectives.

Conducting further evaluation in industry

As explained in the previous section, the AMQuICk framework still needs to be used in professional environments in order to validate its relevance and usability and to gain in maturity. The more the framework will be applied in concrete contexts, the more its credibility and the confidence on its relevance will increase. Moreover, conducting different evaluations will allow a continuous growth of the customization knwoledge base.

During the course of this research, we did have the opportunity to collaborate with practitioners interested in the question of agile methods adaptation. These collaborations helped us to confirm the relevance of the research problematic to agile facilitators, to deeper our knowledge of it and to iteratively design the framework.

In the future work, finding more opportunities to apply the whole framework in specific organizations is still required. In order to convince more industry partners to collaborate in the maturation of the framework, it is crucial to understand which aspects would slow its adoption rate.

Based on our experience with the SPW (see Chapter 9), we observed that the main reluctance regarding the adoption of a new process improvement framework consists of its experimental status. Therefore, prior to the solicitation of professionals, the AMQuICk tool-support should be improved and fully integrated and the feedback from practitioners should be integrated.

Another aspect that should be considered is that when the development process is already well established, it is hard to get people involved in a new approach. Therefore, when seeking for industrial collaboration, we should put the accent on the fact that AMQuICk do not require additional effort from developers since it builds on the existing retrospectives and uses facilitation techniques familiar to what the agile teams usually use. The commitment of the facilitator is also not time consuming especially if he is assisted by the researcher. Another argument that should convince the agile facilitators to collaborate on the evaluation of AMQuICk, is the feature of inter-team and inter-organizational knwoledge sharing. Indeed, a knwoledge sharing culture is essential among the agile community and facilitators frequently share their learnings through community groups or blog posts.

Proposing a default context model and a default knwoledge Base

Proposing a default context model for AMQuICk (i.e., a selected set of relevant context factors that are generic enough to apply to various situa-

tions prior to further customization) and a default knowledge base would enhance the usability of the framework. This can be performed through a Systematic Literature Review (SLR) of empirical studies that report the situational implementation of agile methods in different contexts. The systematic sampling from experience reports would increase the likelihood to constitute a complete context model, i.e., which covers the most critical context factors reported by practitioners and allow the early adopter of the framework to have a reusable default knowledge base as a starting point. Indeed, a growing number of empirical agile implementation studies are published every year in the agile community. For instance, a simple search in IEEE Explore[4] for papers published in the last decade and which title containing *"Pair programming"* returns more then 150 results. These studies form an important body of knowledge from which we can extract valuable information regarding agile practices implementation in specific contexts. The [Kitchenham and Charters, 2007] guidelines can be followed to perform this review.

[Kitchenham and Charters, 2007] states that one of the most important pre-review steps is to clearly define the objectives and research question(s). In software engineering, systematic literature reviews are often performed for 4 main reasons: (1) to summarize the existing evidence concerning a research field, (2) to identify any gaps worth to be investigated, (3) to provide background to position new research work and (4) to provide empirical evidence to support a theory or to generate new hypotheses.

Our motivation is actually closer to the 4th reason. The purpose of this future work would not be to proof a theory or a hypothesis, but to develop a practical context model for AMQuICk and validate it taking into account the published practitioners' experiences. The research questions of the SLR could be formulated as follows: What are the most critical context factors that should be considered for customizing agile methods? What implications do have these context factors?

Better tool support

The conceptual level of the AMQuICk framework sets the foundation for the tool-support that has to be provided. The current version of AMQuICk is supported by a repository of practices and a practice cards modeler. Additional development efforts has to be provided in order to improve these tools, to integrate them and to provide additional support to the framework users.

---

[4]`https://ieeexplore.ieee.org/`

First of all, the Agilia repository of practices, which is designed to help share a common vision on the development practices and to emphasis the inter-team knowledge sharing, needs to be extended to integrate the customization knowledge base and to enable the inter-organizational learning. Its current version is limited to the essential features for managing and consulting the content of a database of practices and related material (artifacts, metrics, tools, templates, literature references, etc.). However, the tool is able to support the extension towards a full community-based platform (which is technically possible in the current version). The main challenge of this community-wide repository is to ensure the validity of the documented knowledge, i.e., to ensure that there is no co-existence of more or less similar practices, that the associations between practices are correctly informed, that the description of practices is relevant and more importantly that related recommendations are relevant. As earlier discussed, this requires the validation of the proposed content by an agile expert. The repository of practices also needs to be fully integrated with the practice cards modeler so that the cards can be automatically generated from the description of a practice and vis versa.

Additionally, the framework may be improved by supporting automation of the context assessment process (prior to the customization) and to assist the evaluation of the context indicators. This may be feasible through the development of a context assessment form builder or thanks to an automatic chatbot for knowledge acquisition [Wu et al., 2008].

Moreover, the tool-support for AMQuICk may involve the automation of the facilitation tools. The extension to Jira that we developed in the context of the MDL case study (see Chapter 13) to capture the practice-based improvement actions is an interesting lead for solution but we should provide a way to bridge the gap between Jira and the integrated AMQuICk tool support.

Finally, in the current description of the AMQuICk framework, the process of relating the context elements with the right practices remains mainly a matter of expert knowledge and intuition. The automation of this process would be feasible using a multivariate analysis approach to select the context factors that are correlated with the successful implementation of practices or using a collaborative recommendation system that would recommend practices based on the previous ratings of users in similar contexts.

Chapter 15

Conclusion

As this dissertation comes to an end, let us take the time to recall the different stages we went through during the elaboration of this research work.

First of all, the research started by our willingness to challenge the false assumption or belief that agile methods are a *magic bullet silver* that works well in all possible situations and that delivers quality anyway. Indeed, a cursory glance at the multiple success stories shared by agile "evangelists" can induce many false beliefs[1] among which:

- Agile Software Development lead obviously to success

- Agile Software Development is anti-documentation and anti-planning

- Agile Software Development is undisciplined

- Agile Software Development is a set of simple practices to apply at the team level

Regarding to all these misconceptions, seventeen years after the release of the agile manifesto [Beck et al., 2001], some of the early agile thought leaders (including authors of the manifesto) claim that *"Agile is in decline"*[2] or even that *"Agile is dead!"*[3,4]. They argue that many teams are claiming to be *Agile* where they are just picking up some of the simplest practices such as *sprints* and *stand-up meetings*, without adopting any of the essential technical practices such as *pair programming*, *test driven development*, *continuous integration*, and *test automation* which are necessary to produce high-quality software. They also argue that the popularity of Scrum doesn't make it a sufficient method to deliver quality and explain that Scrum is popular *"because it's easy"*. They even criticize the misuse of the word *Agile* as a noun rather than an adjective to define the way to develop software[3].

---

[1] http://www.agilenutshell.com/agile_myths
[2] http://www.jamesshore.com/Blog/The-Decline-and-Fall-of-Agile.html
[3] https://www.youtube.com/watch?v=a-BOSpxYJ9M
[4] https://pragdave.me/blog/2014/03/04/time-to-kill-agile.html

From this initial existential questioning, we started by examining the context of the agile software development research. Criticisms and questioning regarding the efficiency of agile methods could also be found in the software engineering research community. Indeed, several research studies have questioned the real impact of agile methods on quality [Stamelos and Sfetsos, 2007] or investigated the conditions to succeed using agility [Chow and Cao, 2008; Ahimbisibwe et al., 2015].

As such, it appeared that an increasing number of practitioners and researchers support a contextual approach to agile software development, where agile methods are customized to suit the specific context of their organizations and teams. [Conboy and Fitzgerald, 2007] notes that *"the very name agile suggests that the method should be easily adjusted to suit its environment."* and to this regards, [Kruchten, 2013] defines agility as *"the ability of an organization to react to changes in its environment faster than the rate of these changes.".* These studies demonstrate the importance of context-awareness when implementing agile methods and provide us with the assumption that motivated our research work: *"A situational or contextual implementation of agile methods is more likely to succeed and therefore to produce high quality software".*

Going back to the literature, we found that the systematic guidance approaches for supporting a situational deployment of agile methods are still lacking. Practitioners are seeking for methods to help determine whether they are ready for the change to agility and to help them identify the right agile practices to adopt. In addition, they are interested in learning more about the potential difficulties in their journey towards agility [Qumer, 2010; Sidky, 2007]. Without sufficient support, they strive to mature their methods in many non-sweet spot contexts. Moreover, it's arguable to assume that relying simply on *retrospectives* is not sufficient to help them fully and efficiently customize their methods.

The creation of the Agile Methods Quality Integrated Customization framework (AMQuICk), presented in this dissertation, is motivated by the absence of a structured approach capable of providing agile teams and organizations with practical customization guidance.

The framework builds on existing software process improvement methods such as the Quality Improvement Paradigm (QIP) [Basili, 1985], on Situational Method Engineering [Henderson-Sellers and Ralyté, 2010] and on the paradigm of reuse-based organizational learning [Basili et al., 1994b]. By doing so, it brings more discipline to the overall customization process but we argue that this doesn't necessarily lead to less agility. Indeed, one of our major concerns throughout the development of the framework was to

provide means for keeping the customization process as simple as possible and for extensively including teams in the overall process to not impede agility values.

The framework, that has been iteratively grounded thanks to the design science research methodology, relies on an iterative, incremental and multi-levels life-cycle that allows a continuous back-and-forth exchange of learnings between the organizational level, the process-implementation level and the product-development level.

Its core artifact consists of a metamodel for authoring agile building blocks called *AMQuICk Essence.* This metamodel incorporates the necessary elements for structuring an *agile repository of practices* (a kind of an experience factory), a *context model* and a *customization knowledge base* (see Figure 5.1). Additional operational tools are the *AMQuICk Backlog* and the *Capitalization Workshop.*

Since the framework was iteratively grounded, its validity was discussed at different design steps using several exploratory and evaluative case studies. However, further work is still needed in order to ensure its viability in industry.

Finally, by following an iterative and incremental process to design an iterative and incremental approach to customize agile methods which are iterative and incremental, we can say that we experienced a (meta)agility at all levels !

In the end, we can conclude that the AMQuICk framework is an attempt to find the middle-ground between disciplined process improvement (relying on heavyweight prescriptive approaches) and agile process improvement (relying exclusively on the lightweight retrospectives), demonstrating that once again, agility and discipline, these apparently opposite approaches are, in fact, complementary.

Appendices

Appendix A

Semi-structured Interview Guide

The goal of this interview session is first to outline **the issues you have experienced so far and specifically the culture-related ones** when adopting and practicing agile and second to **understand how do you usually assess the effectiveness** of the applied agile practices? How do you **tune and adjust them** and why?

This interview is semi-structured. The rest of the document represents the interview guide, which is the list of questions and topics that needs to be addressed.

1. <u>**Understand your context**</u>

    1.1. Your profile:

Years of agile experience
Role

    1.2. Company

Attribute	Value and Comments
1.2.1. Size: *number of personnel*	
1.2.2. Domain: *Organization Business Domain*	
1.2.3. Structure: *departments boundaries* *communication procedures between them*	
1.2.4. Agile experience: *years of agile methods practice*	

1.2.5. Culture:
  a. Compliance and governance: ITIL, Cobit, ISO,
     CMMi ...
  b. Additional Regulatory compliance (financial,
     privacy...)	

c. *Level of agile management support: 1-5*	
d. *Innovativeness: 1-5*	

1.3. Typical Project and Team

Attribute	Value and Comments
1.3.1. Size: *Typical team size*	
1.3.2. Average agile experience: *Team average experience (in years)*	
1.3.3. Geographical distribution: *Co-located – Distributed (same division) - Distributed (different divisions) – Global (1-4)*	
1.3.4. Multidisciplinary Team *True – False*	
1.3.5. Schedule: *Typical Project time*	
1.3.6. Domain Complexity: *Straightforward -- Very Complex (1-5)* Note: More domain complexity means greater modeling and planning and sophisticated testing strategies	
1.3.7. Technical Complexity: *Straightforward - - Very Complex (1-5)* Note: Very complex technical environment may be due to refactoring of existence systems, usage of several technology platforms … Very complex means more architecture and design effort	
1.3.8. Criticality: *Comfort – discretionary funds – essential funds - single life – many lives (1-5)*	
1.3.9. Scope variability: *Percentage of change per month (%)* Note: This attribute refers to how stable is the business environment	
1.3.10. Volatility: *Likelihood of hard to control change (1-5)*	
1.3.11. Iteration duration: *Iteration duration for a typical project*	
1.3.12. Applied Method: *What method is most frequently applied?*	
1.3.13. Customer Involvement: *Commitment and capability of the customer and his representative (1-5)*	

Average commitment time per week?

1.3.14. Documentation Strategy:
Lightweight – Heavy (1-5)

1.3.15. Team self-organization and dependencies:
Self-organized -- Dependent (1-5)

2. Discuss the applied practices, work products, tools and Metrics?

Please rank your commitment to the following agile practices and eventually comment how these were customized

Practice, work product or metric *Please comment when filling*	Not adopted	Planning adoption	Adoption in progress	Partially adopted	Completely adopted
Unit testing	1	2	3	4	5
Acceptance testing	1	2	3	4	5
Integration testing	1	2	3	4	5
Regression testing	1	2	3	4	5
Continuous integration	1	2	3	4	5
Test driven design	1	2	3	4	5
Test driven development (or test first)	1	2	3	4	5
Active stakeholder participation	1	2	3	4	5
Collaborative requirements workshops	1	2	3	4	5
Dedicated customer representative	1	2	3	4	5
On-site customer	1	2	3	4	5
40-hours per week	1	2	3	4	5
Architecture envisioning	1	2	3	4	5
Burn-down charts	1	2	3	4	5
Coding standards	1	2	3	4	5
Collaborative teams	1	2	3	4	5
Collective ownership	1	2	3	4	5
Common room (war-room)	1	2	3	4	5
Collaborative workspace	1	2	3	4	5
Daily standup	1	2	3	4	5
Definition of done	1	2	3	4	5
Definition of ready	1	2	3	4	5

	1	2	3	4	5
Document continuously	1	2	3	4	5
Document later	1	2	3	4	5
Domain modeling	1	2	3	4	5
Empowered team	1	2	3	4	5
Iteration Review	1	2	3	4	5
Metaphor	1	2	3	4	5
Models brainstorming	1	2	3	4	5
Pair work (design, programming …)	1	2	3	4	5
Personas	1	2	3	4	5
Prioritized Feature List	1	2	3	4	5
Prioritized requirements	1	2	3	4	5
Prototyping	1	2	3	4	5
Code refactoring	1	2	3	4	5
Retrospective	1	2	3	4	5
Short release	1	2	3	4	5
Simple code design	1	2	3	4	5
Story post-it board	1	2	3	4	5
Time-boxed Iteration Planning	1	2	3	4	5
User stories	1	2	3	4	5
Velocity	1	2	3	4	5
Waste elimination	1	2	3	4	5

3. **Discuss the common encountered issues and impediments encountered when adopting agile practices:**
Please report 5 of the most critical and challenging ones

Issue	Details

4. Discuss the culture-related issues and impediments encountered when adopting agile practices.

Discuss their impact on agile methods implementation

Issue	Details

5. Discuss the way agile practices are currently customized:

One of the manifesto principles is "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly"

How your agile practices are currently customized? What outstanding customizations have been proposed so far and who initiated them?

6. Explain the targeted issues of AMQuICk, Present its lifecycle and different components

Would you suggest any improvements or adjustments to the approach?

Rate the approach comprehensiveness, practicality, necessity, relevance
Comment while ranking

	strongly agree	agree	neutral	disagree	strongly disagree
Comprehensiveness	1	2	3	4	5
Practicality	1	2	3	4	5
Necessity	1	2	3	4	5
Relevance	1	2	3	4	5

Do you have any additional comments?

Thank you!

Appendix B

SPW case study - D443 IT department details

This annex presents some more details regarding the contex of the SPW case study (see Chapter 9). A more complete overview of the study can be found in [AYED, 2013].

B.1 Structure

The D443 service is structured hierarchically (see Figure B.1). It is composed of 5 services : Development (DEV), Development Support (DEV Support), Security, Exploitation and IT Help-desk. In Chapter 9, we report the implementation of agile practices in DEV and DEV Support.
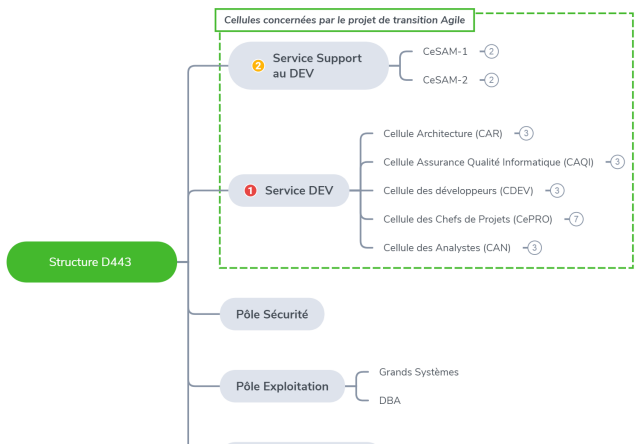

Figure B.1.: D443 Structure

The DEV Support service is composed of two units for the maintenance, support and production operations (MA). 2 practitioners from the service

were interviewed. The DEV service is composed of the following business units:

AR   Architecture unit. 3 practitioners from the unit were interviewed.

QA   Quality Assurance unit. 3 practitioners from the unit were interviewed.

DEV   Development unit. 3 practitioners from the unit were interviewed and 8 participated to the project retrospective.

PM   Project Management unit. 7 practitioners from the unit were interviewed.

FA   Functional Analysis unit. 3 practitioners and 1 external consultant were interviewed.

B.2 Development Life-cycle

The development process (at the time the study started) of the D443 is mainly composed of a classic waterfall process (in preliminary and closure phases). The development process is incremental which makes the refinement. It rarely happens that an increment is developed iteratively.

B.3 Example Workflow

An example of the heavy-weight development process is provided in Figure. The figure shows some details regarding the development process that the architecture unit follows.
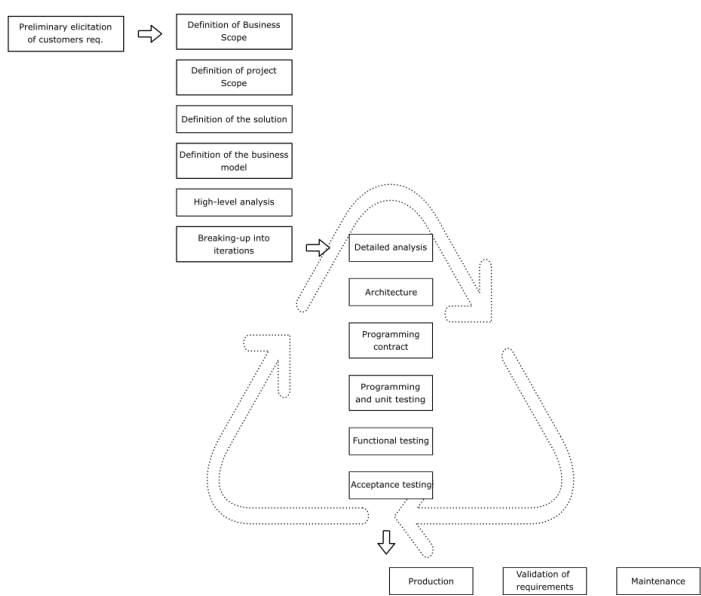
Figure B.2.: Overview of the D443 Development Life-cycle before the intro-
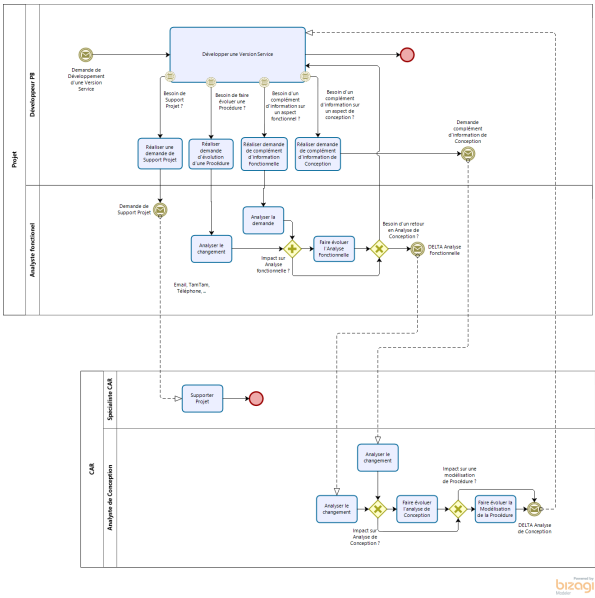duction of agile methods
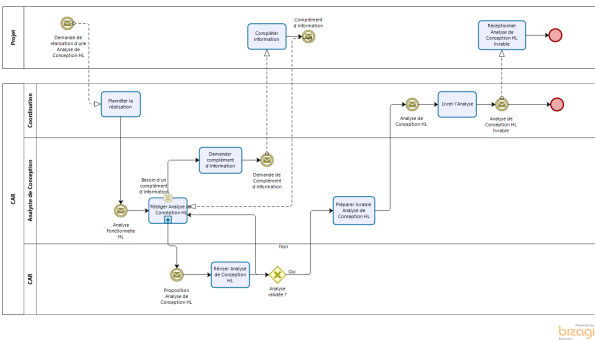
Figure B.3.: Overall view of the architecture unit process

Figure B.4.: High-level analysis workflow (architecture unit)

Bibliography

Abbas, N., Gravell, A. M., and Wills, G. B. (2010). Using factor analysis to generate clusters of agile practices (a guide for agile process improvement). In *AGILE Conference, 2010*, pages 11–20. IEEE.

Abdel-Fattah, M. A. (2015). Grounded theory and action research as pillars for interpretive information systems research: A comparative study. *Egyptian Informatics Journal*, 16(3):309–327.

Abrahamsson, P., Conboy, K., and Wang, X. (2009). "lots done, more to do": the current state of agile systems development research.

Adcock, R., Alef, E., Amato, B., Ardis, M., Bernstein, L., Boehm, B., Bourque, P., Brackett, J., Cantor, M., Cassel, L., et al. (2009). Curriculum guidelines for graduate degree programs in software engineering.

Agarwal, R., Nayak, P., Malarvizhi, M., Suresh, P., and Modi, N. (2007). Virtual quality assurance facilitation model. In *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 51–59. IEEE.

Agile Alliance (2012). Agile alliance guide to agile practices. *Online at: http://www.guide.agilealliance.org/*.

Ahimbisibwe, A., Cavana, R. Y., and Daellenbach, U. (2015). A contingency fit model of critical success factors for software development projects: A comparison of agile and traditional plan-based methodologies. *Journal of Enterprise Information Management*, 28(1):7–33.

Alliance, A. (2008). Agile alliance home. *Online at: http://www.agilealliance.org*.

Alqudah, M. and Razali, R. (2016). A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, 6(6):828–837.

Alqudah, M. and Razali, R. (2017). Key factors for selecting an agile method: A systematic literature review. *International Journal on Advanced Science, Engineering and Information Technology*, 7(2):526–537.

363

Ambler, S. (2005). Quality in an agile world. *Software Quality Professional*, 7(4):34.

Ambler, S. and Agile, I. (2010). Context counts: Position paper for semat. In *Proceedings of the Semat, Zurich Workshop.*

Ambler, S. W. (2006). The Agile Unified Processe (AUP). *Ambysoft - Best Practices for Software Development.*

Ambler, S. W. (2009). The agile scaling model (asm) : Adapting agile methods for complex environments. Technical report, IBM.

Ambler, S. W. and Lines, M. (2012). Disciplined agile delivery.

Ambler, S. W. and Lines, M. (2016). Scaling agile software development tactically: Disciplined agile delivery at scale.

Amescua, A., Bermón, L., García, J., and Sanchez-Segura, M.-I. (2010). Knowledge repository to improve agile development processes learning. *IET software*, 4(6):434–444.

Anderson, D. J. (2005). Stretching agile to fit CMMI level 3-the story of creating MSF for CMMI/spl reg/process improvement at Microsoft corporation. In *Agile Conference, 2005. Proceedings*, pages 193–201. IEEE.

Anthopoulos, L., Reddick, C. G., Giannakidou, I., and Mavridis, N. (2016). Why e-government projects fail? an analysis of the healthcare. gov website. *Government Information Quarterly*, 33(1):161–173.

Anthopoulos, L. G., Siozos, P., and Tsoukalas, I. A. (2007). Applying participatory design and collaboration in digital public services for discovering and re-designing e-government services. *Government Information Quarterly*, 24(2):353–376.

April, A. and Coallier, F. (1995). Trillium: a model for the assessment of telecom software system development and maintenance capability. In *Software Engineering Standards Symposium, 1995.(ISESS'95)'Experience and Practice', Proceedings., Second IEEE International*, pages 175–183. IEEE.

A.Qumer and Henderson-Sellers, B. (2007). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology*, 50:280–295.

Arisholm, E., Gallis, H., Dyba, T., and Sjoberg, D. I. (2007). Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Transactions on Software Engineering*, (2):65–86.

Asnawi, A. L., Gravell, A. M., and Wills, G. B. (2011). Empirical investigation on agile methods usage: issues identified from early adopters in malaysia. In *International Conference on Agile Software Development*, pages 192–207. Springer.

Asnawi, A. L., Gravell, A. M., and Wills, G. B. (2012). Emergence of agile methods: perceptions from software practitioners in malaysia. In *2012 Agile India*, pages 30–39. IEEE.

Asproni, G. (2004). Motivation, teamwork, and agile development. *Agile Times*, 4(1):8–15.

Axelsson, K., Melin, U., and Lindgren, I. (2010). Exploring the importance of citizen participation and involvement in e-government projects: practice, incentives, and organization. *Transforming Government: People, Process and Policy*, 4(4):299–321.

AYED, H. (2013). Spw transition project - context analysis report.

Ayed, H., Habra, N., and Vanderose, B. (2013). Agile processes evolution challenges.

Ayed, H., Vanderose, B., and Habra, N. (2012). A metamodel-based approach for customizing and assessing agile methods. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*, pages 66–74. IEEE.

Ayed, H., Vanderose, B., and Habra, N. (2014). Supported approach for agile methods adaptation: an adoption study. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pages 36–41. ACM.

Ayed, H., Vanderose, B., and Habra, N. (2017). Agile cultural challenges in europe and asia: insights from practitioners. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*, pages 153–162. IEEE Press.

Babuscio, J. (2009). How the fbi learned to catch bad guys one iteration at a time. In *Agile Conference, 2009. AGILE'09.*, pages 96–100. IEEE.

Bacchelli, A. and Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering*, pages 712–721. IEEE Press.

Bajec, M., Vavpotič, D., Furlan, Š., and Krisper, M. (2007). Software process improvement based on the method engineering principles. In

Situational Method Engineering: Fundamentals and Experiences, pages 283–297. Springer.

Basili, V., Caldiera, G., and Rombach, D. H. (1994a). The Goal Question Metric approach.

Basili, V. R. (1985). Quantitative evaluation of software methodology. Technical report, DTIC Document.

Basili, V. R. and Caldiera, G. (1995). Improve software quality by reusing knowledge and experience. *Sloan Management Review*, 37.

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994b). Experience factory. *Encyclopedia of software engineering.*

Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10):70–77.

Beck, K. and Andres, C. (2004). *Extreme programming explained: embrace change.* Addison-Wesley Professional.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development. *Online at : http://www.agilemanifesto.org*, 13.

Becker, J., Janiesch, C., and Pfeiffer, D. (2007). Reuse mechanisms in situational method engineering. In *Situational method engineering: Fundamentals and experiences*, pages 79–93. Springer.

Benbasat, I. (1984). An analysis of research methodologies. *The information systems research challenge*, 47:85.

Berki, E., Siakas, K., and Georgiadou, E. (2007). Agile quality or depth of reasoning? applicability vs. suitability with respect to stakeholders' needs. In *Agile software development quality assurance*, pages 23–55. IGI Global.

Bhagat, R. S. and Steers, R. M. (2009). *Cambridge handbook of culture, organizations, and work.* Cambridge University Press.

Boehm, B. (2006). A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 12–29. ACM.

Boehm, B. and Hansen, W. (2001). The spiral model as a tool for evolutionary acquisition. *CrossTalk*, 14(5):4–11.

Boehm, B. and Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed.* Addison-Wesley Professional.

Boehm, B. and Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE software*, 22(5):30–39.

Boehm, B. W. (1981). Software engineering economics. *Prentice-Hall Advances in Computing Science and Technology Series, Englewood Cliffs: Prentice-Hall, 1981*, 1.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5):61–72.

Borchers, G. (2003). The software engineering impacts of cultural factors on multi-cultural software development teams. In *Proceedings of the 25th international conference on Software engineering*, pages 540–545. IEEE Computer Society.

Bourque, P., Fairley, R. E., et al. (2014). Software engineering process. In *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*, chapter 8. IEEE Computer Society Press.

Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information and Software Technology*, 38(4):275–280.

Brinkkemper, S., Saeki, M., and Harmsen, F. (1999). Meta-modelling based assembly techniques for situational method engineering. *Information Systems*, 24(3):209–228.

Buglione, L. (2011). Light maturity models (lmm): an agile application. In *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, pages 57–61. ACM.

Burge, S. (2011). The systems engineering tool box – matrix diagram.

Cameron, E. and Green, M. (2015). *Making sense of change management: A complete guide to the models, tools and techniques of organizational change.* Kogan Page Publishers.

Campanelli, A. S. (2016). A tailoring criteria model for agile practices adoption.

Cannon-Bowers, JA, S. E. and Converse, S. (1993). Shared mental models in expert team decision making. *Individual and group decision making: Current issues*, 221.

Bibliography

Cao, L., Mohan, K., Xu, P., and Ramesh, B. (2004). How extreme does extreme programming have to be? adapting xp practices to large-scale projects. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10–pp. IEEE.

Chau, T. and Maurer, F. (2004). Knowledge sharing in agile software teams. In *Logic versus approximation*, pages 173–183. Springer.

Chen, M. (2004). *Asian management systems: Chinese, Japanese and Korean styles of business*. Cengage Learning EMEA.

Child, J. (1979). *Culture, contingency and capitalism in the cross-national study of organizations*. University of Aston Management Centre.

Choi, K. S., Deek, F. P., and Im, I. (2008). Exploring the underlying aspects of pair programming: The impact of personality. *Information and Software Technology*, 50(11):1114–1126.

Chow, T. and Cao, D.-B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, 81(6):961–971.

Cleland, S. and Mann, S. (2003). Agility in the classroom: Using agile development methods to foster team work and adaptability amongst undergraduate programmers. *16th Annual NACCQ*.

CMM (1993). Capability Maturity Model, version 1.1. *IEEE software*, 10(4):18–27.

CMMI (2006). Capability Maturity Model Integration for Development, version 1.2.

CMMI-DEV (2010). CMMI for Development, Version 1.3.

Cockburn, A. (2000). Selecting a project's methodology. *IEEE software*, 17(4):64–71.

Cockburn, A. (2004a). *Agile Software Development*. Addison-Wesley.

Cockburn, A. (2004b). *Crystal clear: a human-powered methodology for small teams*. Pearson Education.

Conboy, K. and Fitzgerald, B. (2007). The views of experts on the current state of agile method tailoring. In *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, pages 217–234. Springer.

Conboy, K. and Fitzgerald, B. (2010). Method and developer characteristics for effective agile method tailoring: A study of xp expert opinion. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(1):2.

Contardo, C., Morency, C., and Rousseau, L.-M. (2012). *Balancing a dynamic public bike-sharing system*, volume 4. Cirrelt Montreal.

Cragg, C. (1995). The new taipans a vital source book on the people and business of the pacific rim.

Cunningham, J. B. (1997). Case study principles for different types of cases. *Quality and quantity*, 31(4):401–423.

Cunningham, W. (1993). The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30.

Dalkir, K. (2013). *Knowledge management in theory and practice*. Routledge.

Davis, N. and Mullaney, J. L. (2003). The team software process (tsp) in practice: A summary of recent results.

Deming, W. E. and Edwards, D. W. (1982). *Quality, productivity, and competitive position*, volume 183. Massachusetts Institute of Technology, Center for Advanced Engineering Study Cambridge, MA.

Denzin, N. K. (2017). *Sociological methods: A sourcebook*. Routledge.

Derby, E., Larsen, D., and Schwaber, K. (2006). *Agile retrospectives: Making good teams great*. Pragmatic Bookshelf.

DOD-STD-2167A (1988). A Defense System Software Development, Version A.

Dresch, A., Lacerda, D. P., and Miguel, P. A. C. (2015). A distinctive analysis of case study, action research and design science research. *Revista brasileira de gestão de negócios*, 17(56):1116–1133.

Dromey, R. G. (1995). A model for software product quality. *Software Engineering, IEEE Transactions on*, 21(2):146–162.

DSDM Consortium (2008). Dsdm atern: the handbook.

Dybå, T., Arisholm, E., Sjøberg, D. I., Hannay, J. E., and Shull, F. (2007). Are two heads better than one? on the effectiveness of pair programming. *IEEE software*, (6):12–15.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9):833–859.

El-Said, S. M., Hana, M., and Eldin, A. S. (2009). Agile tailoring tool (att): A project specific agile method. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 1659–1663. IEEE.

Bibliography

Englebert, V. and Heymans, P. (2007). Towards more extensible metacase tools. In *International Conference on Advanced Information Systems Engineering*, pages 454–468. Springer.

English, S. and Hammond, S. (2017). Cost of compliance 2017. *Online at:* `https://risk.thomsonreuters.com/content/dam/openweb/documents/pdf/risk/report/cost-of-compliance-2017.pdf`.

Esfahani, H. C. and Yu, E. (2010). A repository of agile method fragments. In *Proceedings of the 2010 international conference on New modeling concepts for today's software processes*, pages 163–174. Springer-Verlag.

Essence (2014). Kernel and language for software engineering methods, version 1.0. Online at : `http://www.omg.org/spec/Essence/1.0`.

Essence (2015). Kernel and language for software engineering methods, version 1.1. Online at : `http://www.omg.org/spec/Essence/1.1`.

European Union (2015). Ects users' guide.

Feigenbaum, A. V. (1999). The new quality for the twenty-first century. *The TQM magazine*, 11(6):376–383.

Fitzgerald, B. (2000). Systems development methodologies: the problem of tenses. *Information Technology & People*, 13(3):174–185.

Fitzgerald, B., Hartnett, G., and Conboy, K. (2006). Customising agile methods to software practices at intel shannon. *European Journal of Information Systems*, 15(2):200–213.

Fitzgerald, B., Russo, N., and O'Kane, T. (2003). Software development method tailoring at motorola. *Communications of the ACM*, 46(4):64–70.

Forsberg, K. and Mooz, H. (1991). The relationship of system engineering to the project cycle. In *INCOSE International Symposium*, volume 1, pages 57–65. Wiley Online Library.

Fowler, M. (2006). Using an agile software process with offshore development. *Online at:* `http://martinfowler.com/articles/agileOffshore.html`.

Francois, F., Nain, G., Morin, B., Daubert, E., Barais, O., Plouzeau, N., and Jézéquel, J.-M. (2014). Kevoree modeling framework (kmf): Efficient modeling techniques for runtime use. *arXiv preprint arXiv:1405.6817*.

Fritzsche, M. and Keil, P. (2007). Agile Methods and CMMI: Compatibility or Conflict?

Fukuyama, F. (1995). *Trust: The social virtues and the creation of prosperity.* Free press New York.

Gandomani, T. J. and Nafchi, M. Z. (2014). Agility assessment model to measure agility degree of agile software companies. *Indian Journal of Science and Technology*, 7(7):955–959.

Glazer, H., Dalton, J., Anderson, D., Konrad, M., and Shrum, S. (2008). CMMI or Agile: Why Not Embrace Both! *Software Engineering Institute (SEI).*

Grady, R. B. and Caswell, D. L. (1987). Software metrics: Establishing a company-wide program.

Gregory, J. and Crispin, L. (2014). *More Agile Testing: Learning Journeys for the Whole Team.* Addison-Wesley Professional.

Gregory, P., Barroca, L., Taylor, K., Salah, D., and Sharp, H. (2015). Agile challenges in practice: a thematic analysis. In *International Conference on Agile Software Development*, pages 64–80. Springer.

Griffiths, M. (2007). Agile suitability filters. *Online at:* `http: // leadinganswers. typepad. com/ leading_ answers/ 2007/ 06/ agile_ suitabili. html`.

Hall, E. T. et al. (1959). *The silent language*, volume 3. Doubleday New York.

Hampden, C. and Trompenaars, F. (1993). *The seven cultures of capitalism.* Doubleday New York.

Harmsen, F., Brinkkemper, S., and Oei, H. (1994). Situational method engineering for information system project approaches. In *Proceedings of the IFIP*, volume 8, pages 169–194.

Harry, M. J. (1998). Six sigma: a breakthrough strategy for profitability. *Quality progress*, 31(5):60.

Hayes, S. and Richardson, I. (2008). Scrum implementation using kotter's change model. In *International Conference on Agile Processes and Extreme Programming in Software Engineering*, pages 161–171. Springer.

Henderson-Sellers, B. and Gonzalez-Perez, C. (2005a). A comparison of four process metamodels and the creation of a new generic standard. *Information and software technology*, 47(1):49–65.

Bibliography

Henderson-Sellers, B. and Gonzalez-Perez, C. (2005b). The rationale of powertype-based metamodelling to underpin software development methodologies. In *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling-Volume 43*, pages 7–16. Australian Computer Society, Inc.

Henderson-Sellers, B., Gonzalez-Perez, C., and Ralyté, J. (2008). Comparison of method chunks and method fragments for situational method engineering. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 479–488. IEEE.

Henderson-Sellers, B. and Ralyté, J. (2010). Situational method engineering: state-of-the-art review. *Journal of Universal Computer Science*, 16(3):424–478.

Henninger, S. (2003). Tool support for experience-based software development methodologies. *Advances in Computers*, 59:29–82.

Hevner, A. and Chatterjee, S. (2010). Design science research in information systems. In *Design research in information systems*, pages 9–22. Springer.

Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4.

Highsmith, J. (2009). *Agile project management: creating innovative products*. Pearson Education.

Hoda, R., Kruchten, P., Noble, J., and Marshall, S. (2010). Agility in context. In *ACM Sigplan Notices*, volume 45, pages 74–88. ACM.

Hoda, R., Noble, J., and Marshall, S. (2012). Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Software Engineering*, 17(6):609–639.

Hoffman, K. L., Padberg, M., and Rinaldi, G. (2013). Traveling salesman problem. In *Encyclopedia of operations research and management science*, pages 1573–1578. Springer.

Hofstede, G. (2002). Dimensions do not exist: A reply to brendan mcsweeney. *Human relations*, 55(11):1355–1361.

Hofstede, G. (2010). Hofstede cultural dimensions. *Online at : `https://geert-hofstede.com/`*.

Hofstede, G. (2011). Dimensionalizing cultures: The hofstede model in context. *Online readings in psychology and culture*, 2(1):8.

Holgersson, J., Lindgren, I., Melin, U., and Axelsson, K. (2017). Not another new wine in the same old bottles–motivators and innovation in local government e-service development. In *25th European Conference on Information Systems (ECIS), Guimarães, Portugal, June 5-10, 2017*, pages 691–702. Association for Information Systems.

Holmström, H., Fitzgerald, B., Ågerfalk, P. J., and Conchúir, E. Ó. (2006). Agile practices reduce distance in global software development. *Information Systems Management*, 23(3):7–18.

Holmström, J., Ketokivi, M., and Hameri, A.-P. (2009). Bridging practice and theory: A design science approach. *Decision Sciences*, 40(1):65–87.

House, R. J., Hanges, P. J., Javidan, M., Dorfman, P. W., and Gupta, V. (2004). *Culture, leadership, and organizations: The GLOBE study of 62 societies*. Sage publications.

Httermann, M. (2012). *DevOps for developers*. Apress.

Hui, A. (2013). Lean change: Enabling agile transformation through lean startup, kotter and kanban: An experience report. In *Agile Conference (AGILE), 2013*, pages 169–174. IEEE.

Humble, J. and Russell, R. (2009). The agile maturity model applied to building and releasing software. *ThoughtWorks White Paper*.

Humphrey, W. S. (1993). Introduction to software process improvement.

Humphrey, W. S. (2000). The personal software process (psp).

Humphrey, W. S. (2006). *TSP (SM) Coaching Development Teams*. Pearson Education.

IEEE 1061 (1998). standard for a software quality metrics methodology. *IEEE Computer Society, Tech. Rep.*

Iivari, J. and Iivari, N. (2011). The relationship between organizational culture and the deployment of agile methods. *Information and Software Technology*, 53(5):509–520.

Ishikawa, K. (1976). Total quality management. *The Japonese way. Prentice.*

ISO/IEC 12207 (1995). Information technology - Software life cycle processes. Standard, International Organization for Standardisation (ISO).

ISO/IEC 15504 (2004). Information technology – process assessment.

ISO/IEC 15504-2 (2004). Information technology – process assessment – part 2: Performing an assessment.

Bibliography

ISO/IEC 19502 (2005). Information technology - meta object facility (mof).

ISO/IEC 24744 (2007). Metamodel for development methodologies.

ISO/IEC 33001 (2015). Information technology – process assessment – concepts and terminology.

ISO/IEC 330xx (2015). Information technology – process assessment series of standards.

ISO/IEC 9000 (2005). Quality management systems - fundamentals and vocabulary.

ISO/IEC 9126 (2001). Software engineering product quality.

Jackson, S. E., Joshi, A., and Erhardt, N. L. (2003). Recent research on team and organizational diversity: Swot analysis and implications. *Journal of management*, 29(6):801–830.

Juran, J. (1998). *Juran's quality handbook*. McGraw Hill (New York).

Kalus, G. and Kuhrmann, M. (2013a). Criteria for software process tailoring: a systematic review. In *Proceedings of the 2013 International Conference on Software and System Process*, pages 171–180. ACM.

Kalus, G. and Kuhrmann, M. (2013b). Criteria for software process tailoring: a systematic review. In *Proceedings of the 2013 International Conference on Software and System Process*, pages 171–180. ACM.

Kanji, G. K. (1990). Total quality management: the second industrial revolution. *Total Quality Management*, 1(1):3–12.

Kawakita, J. (1975). The kj method–a scientific approach to problem solving. Technical report, Technical report, Kawakita Research Institute, Tokyo.

Kelly, A. (2008). *Changing software development: Learning to become agile*. John Wiley & Sons.

Kelly, S. (2004). Comparison of eclipse emf/gef and metaedit+ for dsm. In *19th annual ACM conference on object-oriented programming, systems, languages, and applications, workshop on best practices for model driven software development*.

Kelly, S., Lyytinen, K., and Rossi, M. (1996). Metaedit+ a fully configurable multi-user and multi-tool CASE and CAME environment. In *International Conference on Advanced Information Systems Engineering*, pages 1–21. Springer.

Kent, S. (2002). Model driven engineering. In *International Conference on Integrated Formal Methods*, pages 286–298. Springer.

Khan, P. and Beg, M. S. (2013). Extended decision support matrix for selection of sdlc-models on traditional and agile software development projects. In *Advanced Computing and Communication Technologies (ACCT), 2013 Third International Conference on*, pages 8–15. IEEE.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report.

Kluckhohn, F. R. and Strodtbeck, F. L. (1961). Variations in value orientations.

Kniberg, H. and Ivarsson, A. (2012). Scaling agile @spotify. *online], UCVOF, ucvox. files. wordpress. com/2012/11/113617905-scaling-Agile-spotify-11. pdf.*

Kotter, J. P. et al. (1995). Leading change: Why transformation efforts fail.

Krasteva, I., Ilieva, S., and Dimov, A. (2010). Experience-based approach for adoption of agile practices in software development projects. In *International Conference on Advanced Information Systems Engineering*, pages 266–280. Springer.

Kroll, P. and MacIsaac, B. (2006). *Agility and Discipline Made Easy: Practices from OpenUP and RUP*. Addison-Wesley Professional.

Kruchten, P. (2011). We do not need richer software process models. *Online at: https://philippe.kruchten.com/2011/03/11/we-do-not-need-richer-software-process-models/.*

Kruchten, P. (2013). Contextualizing agile software development. *Journal of Software: Evolution and Process*, 25(4):351–361.

Krueger, R. A. (2014). *Focus groups: A practical guide for applied research.* Sage publications.

Kuhrmann, M., Fernández, D. M., and Steenweg, R. (2013). Systematic software process development: Where do we stand today? In *Proceedings of the 2013 International Conference on Software and System Process*, pages 166–170. ACM.

Kuvaja, P. (1995). Bootstrap: A software process assessment and improvement methodology. *Objective software quality*, pages 31–48.

Larman, C. (2008). *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum.* Pearson Education India.

Bibliography

Larman, C. and Vodde, B. (2016). *Large-scale scrum: More with LeSS.* Addison-Wesley Professional.

Laroche, L. (2012). *Managing cultural diversity in technical professions.* Routledge.

Layman, L., Williams, L., and Cunningham, L. (2004). Exploring extreme programming in context: An industrial case study. In *Agile Development Conference, 2004*, pages 32–41. IEEE.

Layman, L., Williams, L., and Cunningham, L. (2006). Motivations and measurements in an agile case study. *Journal of Systems Architecture*, 52(11):654–667.

Lee, G. and Kwak, Y. H. (2012). An open government maturity model for social media-based public engagement. *Government Information Quarterly*, 29(4):492–503.

Lee, S. and Yong, H.-S. (2010). Distributed agile: project management in a global environment. *Empirical Software Engineering*, 15(2):204–217.

Leffingwell, D. (2013). Scaled agile framework. *Online at: `http://scaledagileframework.com`.*

Leffingwell, D. (2016). *SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering.* Addison-Wesley Professional.

Leppänen, M. (2013). A comparative analysis of agile maturity models. In *Information Systems Development*, pages 329–343. Springer.

Lewis, R. (2011). *Finland, cultural lone wolf.* Nicholas Brealey Publishing.

Liebmann, L. W. (2003). Layout impact of resolution enhancement techniques: impediment or opportunity? In *Proceedings of the 2003 international symposium on Physical design*, pages 110–117. ACM.

Lindgren, I. (2014). Stakeholder involvement in public e-service development–broadening the scope of user involvement. *Electronic Government and Electronic Participation: Joint Proceedings of Ongoing Research and Projects of IFIP WG 8.5 EGOV and ePart 2014*, 21:84–92.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., and Zelkowitz, M. (2002). Empirical findings in agile methods. *Extreme Programming and Agile Methods, XP/Agile Universe 2002*, pages 81–92.

376

Linstone, H. and Turoff, M. (1975). The delphi method: Techniques and applications.

Llamas, V., Coudert, T., Geneste, L., Bejarano, J. R., and De Valroger, A. (2016). Experience reuse to improve agility in knowledge-driven industrial processes. In *Industrial Engineering and Engineering Management (IEEM), 2016 IEEE International Conference on*, pages 651–655. IEEE.

MacGregor, E., Hsieh, Y., and Kruchten, P. (2005). The impact of intercultural factors on global software development. In *Canadian Conference on Electrical and Computer Engineering, 2005.*, pages 920–926. IEEE.

Mackinnon, T. (2004). Xp: Have you got the discipline? *TickIt International magazine.*

Mahnic, V. (2012). A capstone course on agile software development using scrum. *IEEE Transactions on Education*, 55(1):99–106.

Mann, C. and Maurer, F. (2005). A case study on the impact of scrum on overtime and customer satisfaction. In *Agile Conference, 2005. Proceedings*, pages 70–79. IEEE.

Marchenko, A. and Abrahamsson, P. (2008). Scrum in a multiproject environment: An ethnographically-inspired case study on the adoption challenges. In *Agile, 2008. AGILE'08. Conference*, pages 15–26. IEEE.

McCall, J. A., Richards, P. K., and Walters, G. F. (1977). *Factors in software quality.* General Electric, National Technical Information Service.

McFeeley, B. (1996). Ideal: A user's guide for software process improvement. Technical report, DTIC Document.

McIntyre, R. M. and Dickinson, T. L. (1997). A conceptual framework for teamwork measurement. In *Team performance assessment and measurement*, pages 31–56. Psychology Press.

McSweeney, B. (2002). Hofstede model of national cultural differences and their consequences: A triumph of faith-a failure of analysis. *Human relations*, 55(1):89–118.

Mehrfard, H. and Hamou-Lhadj, A. (2011). The impact of regulatory compliance on agile software processes with a focus on the fda guidelines for medical device software. *International Journal of Information System Modeling and Design (IJISMD)*, 2(2):67–81.

Mens, T. and Van Gorp, P. (2005). A taxonomy of model transformation. In *International Workshop on Graph and Model Transformation (GraMoT): a satellite event of the Fourth International Conference on Generative Programming and Component Engineering (GPCE), Tallinn, Estonia, September 28, 2005.*

Mergel, I. (2016). Agile innovation management in government: A research agenda. *Government Information Quarterly*, 33(3):516–523.

Microsoft: DEV212x (2017). Introduction to devops (edx online course).

Mikulėnas, G. and Butleris, R. (2010). An approach for constructing evaluation model of suitability assessment of agile methods using analytic hierarchy process. *Elektronika ir Elektrotechnika*, 106(10):99–104.

Mikulėnas, G., Butleris, R., and Nemuraitė, L. (2011). An appraoch for the metamodel of the framework for a partial agile method adaptation. *Information Technology And Control*, 40(1):71–82.

MIL-STD-1521A (1976). Technical Reviews and Audits for Systems, Equipment, and Computer Programs, Version A.

MOF (2011). Meta Object Facility (MOF) Core Specification, Version 2.4.1. Online at : `http://www.omg.org/spec/MOF/2.4.1/`.

Morden, T. (1999). Models of national culture: a management review. *Cross Cultural Management: An International Journal*, 6(1):19–44.

Morgan, D. L. (1996). Focus groups. *Annual review of sociology*, 22(1):129–152.

Naur, P. and Randall, B. (1968). Software engineering: A report on a nato conference. *Garmisch, Germany, October*, pages 7–10.

Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5):72–78.

Odell, J. (1996). A primer to method engineering. In *Method Engineering*, pages 1–7. Springer.

OPF (2009). Open process framework. Online at : `http://www.opfro.org/`.

Osterweil, L. (1987). Software processes are software too. In *Proceedings of the 9th international conference on Software Engineering*, pages 2–13. IEEE Computer Society Press.

Overby, E., Bharadwaj, A., and Sambamurthy, V. (2006). Enterprise agility and the enabling role of information technology. *European Journal of Information Systems*, 15(2):120–131.

Packlick, J. (2007). The agile maturity map a goal oriented approach to agile improvement. In *Agile Conference (AGILE), 2007*, pages 266–271. IEEE.

Padberg, F. and Muller, M. M. (2003). Analyzing the cost and benefit of pair programming. In *Software Metrics Symposium, 2003. Proceedings. Ninth International*, pages 166–177. IEEE.

Palmer, S. R. and Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.

Patel, C., De Cesare, S., Iacovelli, N., Merico, A., et al. (2004). A framework for method tailoring: a case study. In *2nd OOPSLA Workshop on Method Engineering for Object-Oriented and Component-Based Development*, pages 1–14. Citeseer.

Patel, C. and Ramachandran, M. (2009). Agile maturity model (AMM): A Software Process Improvement framework for agile software development practices. *International Journal of Software Engineering, IJSE*, 2(1):3–28.

Perry, T. (2008). Drifting toward invisibility: The transition to the electronic task board. In *Agile 2008 Conference*.

Petersen, K. (2010). *Implementing Lean and Agile software development in industry*. PhD thesis.

Pfleeger, S. L. and Rombach, H. D. (1994). Measurement based process improvement. *IEEE Software*, 11(4):8–11.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337.

Pikkarainen, M., Salo, O., and Still, J. (2005). Deploying agile practices in organizations: a case study. *Software Process Improvement*, pages 16–27.

Poppendieck, M. (2007). Lean software development. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 165–166. IEEE Computer Society.

Poppendieck, M. and Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley Professional.

Powner, D. (2012). Software development: effective practices and federal challenges in applying agile methods.

Bibliography

Pries-Heje, L. and Pries-Heje, J. (2011). Why scrum works: A case study from an agile distributed project in denmark and india. In *Agile Conference (AGILE), 2011*, pages 20–28. IEEE.

Pyzdek, T. and Keller, P. A. (2014). *The six sigma handbook*, volume 4. McGraw-Hill Education New York.

Qumer, A. (2010). *A framework to assist in the assessment and tailoring of agile software development methods.* PhD thesis.

Qumer, A. and Henderson-Sellers, B. (2006). Comparative evaluation of xp and scrum using the 4d analytical tool (4-dat). In *Proceedings of the European and Mediterranean Conference on Information Systems.*

Qumer, A. and Henderson-Sellers, B. (2008). A framework to support the evaluation, adoption and improvement of agile methods in practice. *Journal of Systems and Software*, 81(11):1899–1919.

Qumer, A., Henderson-sellers, B., and Mcbride, T. (2007). Agile adoption and improvement model.

Rabiee, F. (2004). Focus-group interview and data analysis. *Proceedings of the nutrition society*, 63(4):655–660.

Reed, P. (2008). An agile classroom experience. In *Agile, 2008. AGILE'08. Conference*, pages 478–483. IEEE.

Reifer, D. J., Maurer, F., and Erdogmus, H. (2003). Scaling agile methods. *IEEE software*, 20(4):12–14.

Ricci, F., Rokach, L., and Shapira, B. (2015). Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer.

Ringstad, M. A., Dingsøyr, T., and Brede Moe, N. (2011). Agile process improvement: Diagnosis and planning to improve teamwork. *Systems, Software and Service Process Improvement*, pages 167–178.

Robinson, H. and Sharp, H. (2005). Organisational culture and xp: three case studies. In *Agile Conference, 2005. Proceedings*, pages 49–58. IEEE.

Robson, C. and McCartan, K. (2016). *Real world research.* John Wiley & Sons.

Rolland, K. H., Mikkelsen, V., and Næss, A. (2016). Tailoring agile in the large: Experience and reflections from a large-scale agile software development project. In *International Conference on Agile Software Development*, pages 244–251. Springer.

Royce, W. W. (1970). Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26, pages 328–338. Los Angeles.

Saaty, T. L. (1988). What is the analytic hierarchy process? In *Mathematical models for decision support*, pages 109–121. Springer.

Saleh, M. H. (2013). *Methodology for selection of agile practices*. PhD thesis.

Salo, O. (2005). Systematical validation of learning in agile software development environment. In *Biennial Conference on Professional Knowledge Management/Wissensmanagement*, pages 106–110. Springer.

Salo, O. (2006). Enabling software process improvement in agile software development teams and organisations.

Salo, O. and Abrahamsson, P. (2005). Integrating agile software development and software process improvement: a longitudinal case study. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10–pp. IEEE.

Salo, O. and Abrahamsson, P. (2007). An iterative improvement process for agile software development. *Software Process: Improvement and Practice*, 12(1):81–100.

Satir, V. and Banmen, J. (1991). *The Satir model: Family therapy and beyond*. Science and Behavior Books.

Schlager, K. J. (1956). Systems engineering - key to modern development. *IRE Transactions on Engineering Management*, 3(EM-3):64–66.

Schmitz, L. and Weber, W. (2014). Are hofstede's dimensions valid? a test for measurement invariance of uncertainty avoidance. *interculture journal: Online-Zeitschrift für interkulturelle Studien*, 13(22):11–26.

Schroeder, A., Klarl, A., Mayer, P., and Kroiß, C. (2012). Teaching agile software development through lab courses. In *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pages 1–10. IEEE.

Schwaber, K. (1995). Scrum development process. In *OOPSLA Business Object Design and Implementation Workshop*, volume 27, pages 10–19. Austin, TX.

Sfetsos, P., Stamelos, I., Angelis, L., and Deligiannis, I. (2009). An experimental investigation of personality types impact on pair effectiveness in pair programming. *Empirical Software Engineering*, 14(2):187.

Bibliography

Sharp, H., Robinson, H., and Petre, M. (2008). The role of physical artefacts in agile software development: Two complementary perspectives. *Interacting with computers*, 21(1-2):108–116.

Sherehiy, B., Karwowski, W., and Layer, J. K. (2007). A review of enterprise agility: Concepts, frameworks, and attributes. *International Journal of industrial ergonomics*, 37(5):445–460.

Siakas, K. V. and Siakas, E. (2007). The agile professional culture: A source of agile quality. *Software Process: Improvement and Practice*, 12(6):597–610.

Sidky, A. S. (2007). *A structured approach to adopting agile practices: The agile adoption framework*. PhD thesis, Virginia Tech.

Simon, H. A. (1996). *The sciences of the artificial*. MIT press.

Simonofski, A., Ayed, H., Vanderose, B., and Snoeck, M. (2018). From traditional to agile e-government service development: Starting from practitioners' challenges. In *Americas Conference on Information Systems, Boston*. Association for Information Systems (AIS).

Simonofski, A., Snoeck, M., Vanderose, B., Crompvoets, J., and Habra, N. (2017). Reexamining e-participation: Systematic literature review on citizen participation in e-government service delivery.

Sjoberg, D. I., Dyba, T., and Jorgensen, M. (2007). The future of empirical methods in software engineering research. In *Future of Software Engineering, 2007. FOSE'07*, pages 358–378. IEEE.

SPEM (2008a). Software & systems process engineering metamodel specification, version 2. Online at : `http://www.omg.org/spec/SPEM/2.0/`.

SPEM (2008b). SPEM 2.0 UML 2 Profile. Online at : `http://www.omg.org/spec/SPEM/2.0/`.

Stamelos, I. and Sfetsos, P. (2007). *Agile software development quality assurance*. Igi Global.

Stankovic, D., Nikolic, V., Djordjevic, M., and Cao, D.-B. (2013). A survey study of critical success factors in agile software projects in former yugoslavia it companies. *Journal of Systems and Software*, 86(6):1663–1678.

Strode, D. E., Huff, S. L., and Tretiakov, A. (2009). The impact of organizational culture on agile method use. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–9. IEEE.

Susman, G. I. and Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative science quarterly*, pages 582–603.

SW-CMMI (2002). CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Continuous Representation.

Takeuchi, N. (1995). The knowledge creating company: How japa-nese company create the dynamics. *Oxford: Oxford Un-iversity Press.*

Thiollent, M. (2011). Action research and participatory research: An overview. *International Journal of Action Research*, 7(2):160–174.

Tolfo, C. and Wazlawick, R. S. (2008). The influence of organizational culture on the adoption of extreme programming. *Journal of systems and software*, 81(11):1955–1967.

Tomasini, A. and Kearns, M. (2012). Agile transition-what you need to know before starting. *InfoQueue Enterprise Software Development Series.*

Tran, Q., Henderson-Sellers, B., and Hawryszkiewycz, I. (2009). Some method fragments for agile software development. In *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*, pages 223–242. IGI Global.

Triandis, H. C. (1995). *Individualism & collectivism.* Westview press.

Tudor, D. and Walter, G. A. (2006). Using an agile approach in a large, traditional organization. In *Agile Conference, 2006*, pages 7–pp. IEEE.

Turner, R. and Jain, A. (2002). Agile meets CMMI: Culture clash or common cause? In *Conference on Extreme Programming and Agile Methods*, pages 153–165. Springer.

Vallon, R., da Silva Estácio, B. J., Prikladnicki, R., and Grechenig, T. (2018). Systematic literature review on agile practices in global software development. *Information and Software Technology*, 96:161–180.

Van Solingen, R. and Berghout, E. (1999). *The Goal/Question/Metric Method: a practical guide for quality improvement of software development.* McGraw-Hill.

van Velsen, L., van der Geest, T., ter Hedde, M., and Derks, W. (2009). Requirements engineering for e-government services: A citizen-centric approach and case study. *Government Information Quarterly*, 26(3):477–486.

Vanderose, B., Ayed, H., and Habra, N. (2014). Software quality in an increasingly agile world. *ERCIM News*, 2014(99).

Vanderose, B. et al. (2012). *Supporting a model-driven and iterative quality assessment methodology: The MoCQA framework*. PhD thesis, PhD thesis, University of Namur.

Vanderose, B. and Habra, N. (2011). Tool-support for a model-centric quality assessment: Quatalog. In *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*, pages 263–268. IEEE.

VersionOne (2017). 11th annual state of agile report. *Online at : `http://www.versionone.com/`*.

VersionOne (2018). 12th annual state of agile survey. In *Technical Report*. Version One.

Vitalari, N. P. (1985). *The need for longitudinal designs in the study of computing environments*. Public Policy Research Organization, University of California.

Wieringa, R. (2010). Design science methodology: principles and practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pages 493–494. ACM.

Williams, L. and Cockburn, A. (2003). Guest editors' introduction: Agile software development: It's about feedback and change. *Computer*, 36(6):39–43.

Williams, L., McCrickard, D. S., Layman, L., and Hussein, K. (2008). Eleven guidelines for implementing pair programming in the classroom. In *Agile, 2008. AGILE'08. Conference*, pages 445–452. IEEE.

Womack, J. (2013). *Gemba Walks: Expanded 2nd Edition*. Lean Enterprise Institute.

Wu, Y., Wang, G., Li, W., and Li, Z. (2008). Automatic chatbot knowledge acquisition from online forum via rough set and ensemble learning. In *Network and Parallel Computing, 2008. NPC 2008. IFIP International Conference on*, pages 242–246. IEEE.

Wynekoop, J. L. and Walz, D. B. (2000). Investigating traits of top performing software developers. *Information Technology & People*, 13(3):186–195.

Yasuoka, M. and Sakurai, R. (2012). Out of scandinavia to asia: adaptability of participatory design in culturally distant society. In *Proceedings of the 12th Participatory Design Conference: Exploratory Papers, Workshop Descriptions, Industry Cases-Volume 2*, pages 21–24. ACM.

Yin, A., Figueiredo, S., and da Silva, M. M. (2011). Scrum maturity model. *Proceedings of the ICSEA*, pages 20–29.

Yin, R. K. (1994). Case study research: Design and methods (applied social research methods, vol. 5). *Sage Publications, Beverly Hills, CA. Rick Rantz Leading urban institutions of higher education in the new millennium Leadership & Organization Development Journal*, 23(8):2002.

Yin, R. K. (2017). *Case study research and applications: Design and methods*. Sage publications.

Yu, X. and Petter, S. (2014). Understanding agile software development practices using shared mental models theory. *Information and Software Technology*, 56(8):911–921.

Zahran, S. (1998). *Software process improvement*. Addison-wesley.

Zeithaml, V. A., Rajan Varadarajan, P., and Zeithaml, C. P. (1988). The contingency approach: its foundations and relevance to theory building and research in marketing. *European Journal of Marketing*, 22(7):37–64.