Architectural Bad Smells for Self-Adaptive Systems

Santos, Edilton Lima dos; Schobbens, Pierre-Yves; Machado, Ivan; Perrouin, Gilles

Published in:
Proceedings of the 17th International Working Conference on Variability Modelling of Software-Intensive Systems, VaMoS 2023, Odense, Denmark, January 25-27, 2023

DOI:
10.1145/3571788.3571802

Publication date:
2023

Document Version
Peer reviewed version

Architectural Bad Smells for Self-Adaptive Systems: Go Runtime!

Edilton Lima dos Santos
edilton.limados@unamur.be
PReCISE, NaDI,
Faculty of Computer Science, University of Namur
Namur, Belgium

Pierre-Yves Schobbens
pierre-yves.schobbens@unamur.be
PReCISE, NaDI,
Faculty of Computer Science, University of Namur
Namur, Belgium

Ivan Machado
ivan.machado@ufba.br
Institute of Computing,
Federal University of Bahia
Salvador, Brazil

Gilles Perrouin
gilles.perrouin@unamur.be
PReCISE, NaDI,
Faculty of Computer Science, University of Namur
Namur, Belgium

ABSTRACT

Self-adaptive systems (SAS) change their behavior and structure at runtime depending on environmental changes or user requests. For this purpose, the SASs combine architectural fragments or solutions in their adaptation process. However, this process may negatively impact the system's architectural qualities, exhibiting architectural bad smells (ABS). Current studies perform ABS detection for SAS at design time, ignoring their *intrinsic runtime variability*. We demonstrate that this ignorance leads to inaccurate smell detections and possibly wrong maintenance decisions. We delineate the challenges runtime variability raise on ABS detection and argue that we should analyze SAS architectures at runtime.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; • **Computer systems organization → Self-organizing autonomic computing**.

KEYWORDS

Self-adaptive Systems, Software architecture, Architectural Smells, Architectural Quality, Runtime Validation.

ACM Reference Format:
Edilton Lima dos Santos, Pierre-Yves Schobbens, Ivan Machado, and Gilles Perrouin. 2023. Architectural Bad Smells for Self-Adaptive Systems: Go Runtime!. In *17th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2023), January 25–27, 2023, Odense, Denmark*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3571788.3571802

1 INTRODUCTION

Architectural Bad Smells (ABS) are architectural decisions that negatively impact internal software quality. The presence of ABS might imply reduced system testability, maintainability, extensibility, and reusability [3, 9, 12, 13, 16]. There are many ABS reported in the literature [4, 9–13, 17, 18, 21]. In this paper, we focus on structural smells and exclude their impact on the system's behavior. Examples include the **Cyclic Dependency (CD)** [1] and **Hub-Like dependency (HL)** [1, 18]. The former occurs when two or more components depend on each other directly or indirectly. The latter arises when a component has (outgoing and ingoing) dependencies with many other abstractions (*e.g.*, other components) [6].

The literature on self-adaptive systems (SAS) encompasses approaches to support ABS identification at design time through static analysis [18]. Such approaches enable the program source code analysis statically without executing it. However, it does not consider the system's (re)configuration process at runtime [18] and the variability space. In particular, we argue that we cannot infer the whole variability space for two reasons. First, since most SAS do not document configuration options, it is difficult to analyze them automatically. Second, SAS are realizing open variability [23] at runtime thanks to variability mechanisms such as polymorphism and via the possibility to download new features on the fly (*e.g.*, the code for a plug-and-play sensor [20]). A particular characteristic of a SAS is to reconfigure dynamically at runtime. A SAS might change its behavior due to unexpected environmental changes, reconfiguration plans, and goals [5]. The adaptations at runtime may affect architectural qualities and properties, given that the (re)configuration process may combine architectural fragments or apply architectural abstractions at the wrong granularity level through the newly loaded features [15].

Based on such observations and our experience [15], we have devised the following seemingly controversial idea: *achieving an effective ABS identification in SAS will only be possible at runtime*, once variability is bound. Accordingly, we *strongly encourage carrying out dynamic analysis in addition to/rather than solely relying on static analysis*. It contradicts the common practice of identifying architectural smells only at design time. The following section motivates why this current practice is doomed to fail.

2 CHALLENGES

We identified the following challenges that SAS raises for ABS detection.

VaMoS 2023, January 25–27, 2023, Odense, Denmark

ACM ISBN 979-8-4007-0001-9/23/01…$15.00
https://doi.org/10.1145/3571788.3571802

Runtime variability's impact on SAS architectures. Even if not implemented as such (see below), one can see SAS re-configuration as activating and deactivating features at runtime. Not taking this aspect into account leads to inaccurate reports on the existence and importance of ABS runtime. For instance, in a recent study, we compared ABS detected at design time and runtime [7]. We observed significant differences between smells' occurrences at such different binding times for the Adasim project [25]. In addition, some smells appearing at runtime could not be found at design time for the mRUBiS project [24].

Lack of variability documentation in SAS. Variability management is crucial for SAS [7], and this lead research community on variability management to coin the concept of *dynamic software product lines* [2] (DSPLs). DSPLs realize SAS by carefully modeling SAS adaptations using variability models and tracing variability down to implementation artifacts, *e.g.*, [20]. This would allow the variability-aware analysis of SAS and possible extension of variability-aware code smells [8, 22] to ABS. However, most SAS are not implemented as DSPLs. For example, none of the Java-based exemplars provided by the SEAMS community[1] had any variability documentation (feature model, feature annotations). Dos Santos *et al.* introduced a manual process to identify source code features based on information available in the system's repository [15]. However, adding a mechanism in the systems' source code for ABS identification requires expertise and time because the mandatory and variable features are not documented.

Capturing adaptations. For identifying ABS at runtime, it is necessary to run the system and identify the exact moment each adaptation starts and ends [19]. It is also necessary to capture all features and dependencies loaded in the adaptation loop at runtime. This task is challenging because it is necessary to identify the method responsible for executing the adaptation loop and the invoked methods inside it. Algorithm 1 illustrates such a scenario using a simplified MAPE-K loop [14] implementation. The `adaptationMechanism()` method is responsible for executing the system's adaptation mechanism. It uses a loop to execute the adaptation process encompassing `dataLoad()`, `dataAnalysis()`, and `runAdaptationStep()` methods. The first method reads the data from the environment, *e.g.*, sensor data, and sends them to the `dataAnalysis()` method. The `dataAnalysis()` defines the features we should activate to support the adaptation required at runtime. Then, the `runAdaptationStep()` method performs the adaptation. We adopt a runtime monitoring approach to this challenge by observing the evolution of methods and objects, progressively identifying the code responsible for the adaptation, and tracing methods entries and exits [15].

Handling polymorphism at runtime. Some SAS architectures are implemented based on polymorphism through abstract classes or interfaces. Polymorphism is a strategy to support variability at runtime [19]. Such a strategy could hide the absolute number of features involved in CD and HL, particularly when the analysis (of ABS) only considers the design time [7]. This is due to the analysis taking only concrete classes into account. Figure 1 shows a simplified architecture model of a Traffic Routing system. The model shows that the `Vehicle` class uses the Core and

[1] https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/

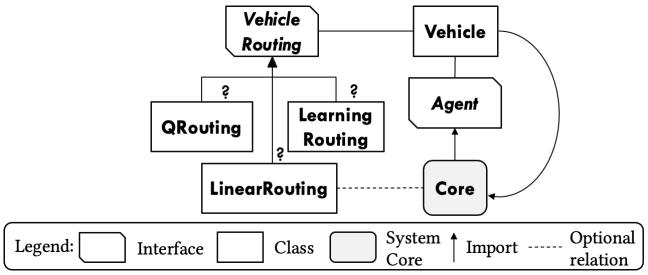

Figure 1: Traffic Routing system simplified architecture (simplified).

`VehicleRouting` interface to bind a specific routing (*e.g.*, `QRouting`, `LearningRouting`, and `LinearRouting`) mechanism at runtime for each `Vehicle` instantiated. Also, the `Vehicle` class implements the `Agent` interface used to connect the system core, and each agent type is instantiated at runtime. The system core can use `VehicleRouting` (*e.g.*, `LinearRouting`) to manage vehicles with a specific routing type at runtime. In this scenario, the cyclic dependency between `Vehicle` and `Core` will happen only at runtime. Thus, the static analysis does not identify that type of ABS at design time because there is no direct relationship among all classes involved in CD. Also, the same situation may happen with classes involved in HL.

Algorithm 1 Interception loop design.

```
1:  procedure adaptationMechanism
2:      while !isFinished() do
3:          dataLoad();
4:          dataAnalysis();
5:          runAdaptationStep();
6:      end while
7:  end procedure
8:  procedure runAdaptationStep
9:      featureIdentification();
10:     bindingFeatures();
11: end procedure
```

3 CONCLUDING REMARKS

We made the case to switch from the classic design time and static detection of architectural bad smells to a more dynamic, runtime perspective when considering intrinsically variability-aware self-adaptive systems. We have been involved in developing this uncommon perspective, providing methods and tools to identify smells at runtime and overcoming the previous challenges [15]. However, analyzing (re)configurations of SAS in a product-based fashion is controversial: How can we cover the most relevant adaptations in an unknown variability space? What is the point of looking for smells in ephemeral architectures? Would smell detection be a driver for architecture-based adaptation?

We do not yet have answers to these questions, but we have a framework to study them. As immediate future work, we would like to characterize the impact of smells on qualities such as resiliency

and security. More finely characterizing this impact would help to decide whether one should modify the system's implementation or its adaptation loop to avoid smelly configurations affecting the system performance or security. In addition, we would like to extend our analysis to more smells and SAS.

ACKNOWLEDGMENTS

Edilton Lima dos Santos is funded by a CERUNA grant from the University of Namur. Gilles Perrouin is an FNRS Research Associate. This work was partly funded by the ERDF IDEES Co-innovation project, and partly funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

[1] Umberto Azadi, Francesca Arcelli Fontana, and Davide Taibi. 2019. Architectural smells detected by tools: a catalogue proposal. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 88–97.

[2] Nelly Bencomo, Peter Sawyer, Gordon S Blair, and Paul Grace. 2008. Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems.. In *SPLC (2)*. 23–32.

[3] Hugo Sica de Andrade, Eduardo Almeida, and Ivica Crnkovic. 2014. Architectural bad smells in software product lines: An exploratory study. In *Proceedings of the WICSA 2014 Companion Volume*. 1–6.

[4] Jorge Andrés Díaz-Pace, Antonela Tommasel, and Daniela Godoy. 2018. Towards anticipation of architectural smells using link prediction techniques. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 62–71.

[5] Edilton Lima dos Santos. 2021. STARS: Software Technology for Adaptable and Reusable Systems. In *Proceedings of the 25th International Systems and Software Product Line Conference (SPLC)*. ACM.

[6] Edilton Lima dos Santos, Sophie Fortz, Gilles Perrouin, and Pierre-Yves Schobbens. 2021. A Vision to identify Architectural Smells in Self-Adaptive Systems using Behavioral Maps. In *15th European Conference on Software Architecture (ECSA 2021)*. CEUR Workshop Proceedings, 1.

[7] Edilton Lima dos Santos, Pierre-Yves Schobbens, and Gilles Perrouin. 2022. Featured Scents: Towards Assessing Architectural Smells for Self-Adaptive Systems at Runtime. In *19th International Conference on Software Architecture*. IEEE, 71–74.

[8] Wolfram Fenske and Sandro Schulze. 2015. Code smells revisited: A variability perspective. In *Proceedings of the Ninth International Workshop on Variability Modelling of Software-intensive Systems*. 3–10.

[9] Francesca Arcelli Fontana, Paris Avgeriou, Ilaria Pigazzini, and Riccardo Roveda. 2019. A Study on Architectural Smells Prediction. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 333–337.

[10] Francesca Arcelli Fontana, Valentina Lenarduzzi, Riccardo Roveda, and Davide Taibi. 2019. Are architectural smells independent from code smells? An empirical study. *Journal of Systems and Software* 154 (2019), 139–156.

[11] Francesca Arcelli Fontana, Ilaria Pigazzini, Riccardo Roveda, and Marco Zanoni. 2016. Automatic detection of instability architectural smells. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 433–437.

[12] Joshua Garcia, Daniel Popescu, George Edwards, and Nenad Medvidovic. 2009. Identifying architectural bad smells. In *13th European Conference on Software Maintenance and Reengineering*. IEEE, 255–258.

[13] Joshua Garcia, Daniel Popescu, George Edwards, and Nenad Medvidovic. 2009. Toward a catalogue of architectural bad smells. In *International conference on the quality of software architectures*. Springer, 146–162.

[14] IBM. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* 31 (2006), 1–6.

[15] Edilton Lima dos Santos, Sophie Fortz, Pierre-Yves Schobbens, and Gilles Perrouin. 2022. Behavioral Maps: Identifying Architectural Smells in Self-adaptive Systems at Runtime. In *European Conference on Software Architecture*. Springer, 159–180.

[16] Isela Macia, Roberta Arcoverde, Elder Cirilo, Alessandro Garcia, and Arndt von Staa. 2012. Supporting the identification of architecturally-relevant code anomalies. *ICSM12* (2012), 662–665.

[17] Haris Mumtaz, Paramvir Singh, and Kelly Blincoe. 2020. A systematic mapping study on architectural smells detection. *Journal of Systems and Software* (2020).

[18] Claudia Raibulet, Francesca Arcelli Fontana, and Simone Carettoni. 2020. A preliminary analysis of self-adaptive systems according to different issues. *Software Quality Journal* (2020), 1–31.

[19] Andres J Ramirez and Betty HC Cheng. 2010. Design patterns for developing dynamically adaptive systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 49–58.

[20] Edilton Santos and Ivan Machado. 2018. Towards an Architecture Model for Dynamic Software Product Lines Engineering. In *IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, 31–38.

[21] Marcel A Serikawa, André de S Landi, Bento R Siqueira, Renato S Costa, Fabiano C Ferrari, Ricardo Menotti, and Valter V De Camargo. 2016. Towards the characterization of monitor smells in adaptive systems. In *X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*. IEEE, 51–60.

[22] Iuri Santos Souza, Ivan Machado, Carolyn Seaman, Gecynalda Gomes, Christina Chavez, Eduardo Santana de Almeida, and Paulo Masiero. 2019. Investigating Variability-aware Smells in SPLs: An Exploratory Study. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*. 367–376.

[23] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. 2001. On the notion of variability in software product lines. In *Proceedings Working IEEE/IFIP Conference on Software Architecture*. IEEE, 45–54.

[24] Thomas Vogel. 2018. mRUBiS: An exemplar for model-based architectural self-healing and self-optimization. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*. 101–107.

[25] Jochen Wuttke, Yuriy Brun, Alessandra Gorla, and Jonathan Ramaswamy. 2012. Traffic routing for evaluating self-adaptation. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 27–32.