MASTER EN SCIENCES INFORMATIQUES

Domain Driven Design comme facteur d'évolution des modèles mentaux partagés dans le développement logiciel

ZENOBI, Jérôme

Award date:
2023

Awarding institution:
Universite de Namur

Annexe VI - Entretien PDE

Moi : Sur quoi travaillez-vous actuellement ?

Pascal : Alors actuellement je travaille comme consultant sur le projet Gerico. Il s'agit d'une application de gestion pour huissiers de justice qui est sensé les aider à automatiser leurs procédures. J'occupe le poste de techlead java et je chapeaute les 3 développeurs juniors de l'équipe.

Moi : Quel est l'état d'avancement du projet ?

Pascal : On arrive doucement mais surement à la fin de la première version commercialisable. Normalement d'ici 2 à 3 mois les tests seront terminés et nous pourrons attaquer l'analyse de la nouvelle version. On a pris un peu de retard par rapport aux estimations de base mais sachant qu'on est parti from scratch et que l'on a encore des retours réguliers des huissiers ce n'est pas très étonnant.

Moi : Comment définiriez-vous la complexité du domaine métier du projet ?

Pascal : Plutôt très complexe ! Je n'ai pas de compétences particulières en matière juridique donc c'est assez nouveau pour moi. Comme tout le monde je connaissais le rôle de base d'huissiers mais je ne m'attendais pas à une telle complexité en fin de compte. Rien que le module des bilans mériterait une formation à lui seul ! Heureusement que nous avons des analystes compétents pour dégrossir un peu le tout avant que nous devions intervenir.

Moi :Avez-vous déjà travaillé dans des projets d'une complexité égale ou supérieur ?

Pascal : A ce point là jamais. J'ai déjà travaillé sur des développements pour des assurances, c'était déjà pas mal mais là c'est encore un autre niveau.

Moi :Comment décririez-vous votre niveau de compréhension de celui-ci ?

Pascal : Plus que ce que j'aurai imaginé au départ ! De nouveau nos analystes prémâchent le travail donc ça devient plus facile pour nous. De plus ils essayent d'utiliser des termes métiers afin de nous mettre directement dans le bain. Ils y intègrent un maximum de justifications afin de conserver la pertinence du terme utilisé par rapport à sa signification métier. En tant normal je dirais que ce n'est pas nécessaire ou obligatoire, mais plus de 2 ans après le début du projet je reconnais la plus-value qu'il y a derrière.

Moi :Comment était-il sur vos précédents projets ?

Pascal : En tant que développeur on nous demandait rarement d'avoir une compréhension métier de bout en bout. On devait comprendre grosso modo les tenants et aboutissants de la partie dont on était responsable, et encore. Tant qu'au final le code fonctionnait, on ne nous demandait pas spécialement de comprendre les enjeux métiers qui se cachaient derrière.

Moi :Que pensez-vous du niveau de compréhension de vos collègues ?

Pascal : Il est le même que le miens je dirais. En tout cas les développeurs. Les analystes ont une vision et une compréhension bien plus élevé que là notre mais bon c'est un peu normal…

C’est eux qui sont en relation constante avec les experts métiers. Mais comme on essaie d’utiliser les termes métiers directement dans notre code, à force de les utiliser, notre compréhension s’améliore je pense.

Moi :Pensez-vous avoir une compréhension commune des termes métiers ?

Pascal : Oui, en tout cas pour les termes les plus généraux. Il nous manque surement certaines clés de compréhension pour des notions très particulières du métier d’huissier mais je pense que dans l’ensemble on ne doit pas trop mal se débrouiller.

Moi :Était-ce le cas dans les projets antérieurs ?

Pascal : Pas vraiment. Pour être franc je ne me suis jamais vraiment tracassé de la compréhension métier de mes collègues. Tant qu’ils respectaient les bonnes pratiques inhérentes à notre métier de développeur, le reste me semblait assez accessoire. Et s’il y avait un souci de compréhension on allait le voir l’analyste.

Moi :Comment découvrez-vous les changements dans le projet ?

Pascal : Nous avons des dailys journalières et la cheffe de projet nous fait part des éventuels changements. Ou alors nous l’apprenons grâce aux tickets qui nous sont assignés.

Moi :Comment cela a-t-il été fait dans les projets antérieurs ?

Pascal : De la même manière, cela fait maintenant quelques années que je travaille en Agile et le fonctionnement est assez similaire d’un projet à l’autre.

Moi :Comment faites-vous face aux changements dans le projet ?

Pascal : ça dépend de l’ampleur des changements mais la plupart du temps nous recevons de la part des analystes les modifications qu’ils ont apporté dans la documentation. Nous en discutons avec eux si nécessaire puis nous attendons qu’un ticket nous soit assigné. Ensuite c’est le cycle de vie classique d’un ticket dans Jira avec son workflow défini.

Moi :Comment cela a-t-il été fait dans des projets antérieurs ?

Pascal : Assez similaire aussi même s’il y avait moins d’interaction avec les analystes.

Moi :L’utilisation du Tactical Pattern vous aide-t-il dans la compréhension du métier ?

Pascal : Oui un peu car le tactical pattern nous demande de réaliser des bonnes pratiques en matière de code, par exemple le respect de principes SOLID, mais en essayant d’y appliquer des concepts métiers. Par exemple en ce qui concerne les entités et les value objects, les premières sont des objets mutables et pas les deuxièmes. Pour comprendre comment définir correctement notre objet nous devons comprendre son implication pour le client. Même chose pour la définition des aggregats roots qui doivent vraiment être considérés comme le point d’entrée, le chapeau, des différentes entités. Si on arrive a bien cerné les concepts métiers les plus importants, il devient facile d’utiliser le tactical pattern a bonne escient.

Moi :Comment cela a été fait dans les projets antérieurs ?

Pascal : Nous utilisions déjà des concepts similaires au tactical pattern mais sans y mettre ce nom. De plus nous n'avons jamais essayé de l'appliquer sur des concepts métiers. On restait focalisé sur notre code et on ne s'intéressait pas trop à ce qui se faisait ailleurs.

Moi :L'utilisation de Context Map et des Bounded Context vous aide-il dans la compréhension du métier ?

Pascal : Oui aussi. Dans un métier aussi large et complexe que celui de huissiers, avoir défini à l'avance les grandes frontières aide à rentrer dans le projet. Ça permet aussi de s'investir dans un module sans avoir une compréhension exhaustive du reste du projet. Il y a bien sûr des liens entre les différents contextes via une gestion d'évènements asynchrones mais c'est justement pour respecter au mieux la découpe de l'application. Nous avons des modules Tiers, Dossiers, Communication, etc. Chaque module représente une partie distincte de la logique des huissiers et on essaie de calquer cette logique sur le code. Au moins ça permet de parler le même langage que les autres membres de l'équipe qui n'ont pas le même poste ou background que nous. Si un analyste me parle de la gestion des décomptes débiteur d'un dossier, je sais que ça se trouve dans l'aggregate bilan et lui aussi. On en parle directement et et l'on sait que l'on parle de la même chose.

Moi :Comment cela a été fait dans les projets antérieurs ?

Pascal : il s'agit d'une structure plus classique, pas mauvaise pour autant mais elle était bien moins orientée business. On découpait le tout de façon plus logique, par rapport au code sans pour autant coller au maximum à ce que les experts métiers pouvaient nous expliquer. Ça restait assez clair quand on en discutait entre devs mais ça devenait plus compliqué si on devait intégrer des analystes ou le chef de projet.

Moi :L'équipe a-t-elle un objectif commun pour le projet ?

Pascal : Oui oui bien sûr, on sait tous où l'on doit aller et comment le faire. C'est vrai qu'il y a parfois des ajustements, c'est entre autre pour ça que l'on travaille en Agile, mais on sait depuis le début l'objectif final et je pense qu'on était tous aligné.

Moi :Les projets précédents avaient-ils un objectif commun ?

Pascal : Il y en avait un mais on se focalisait plus sur nos objectifs à courts termes.

Moi :Comment est la performance de l'équipe ?

Pascal : Je pense qu'elle est assez bonne. J'ai la chance de travailler avec deux jeunes ingénieurs sortis il y a moins d'un an et malgré leur statut de Junior ça avance plutôt bien. C'est l'un des aspects positifs du DDD, même s'il est difficile à appréhender au début, son application facilite la suite du projet. Encore une fois on sait que nous serons alignés plus facilement entre nous, mais aussi avec les analystes ou les testeurs car cette approche fonctionne pour tout le monde. Nous avons assez peu de rework à effectuer contrairement à d'autres projets, et ceux-ci se font en générales de manières assez rapides car nous savons exactement où trouver ce qu'il nous intéresse dans le projet.

Moi :Comment cela a-t-il été fait dans les projets précédents ?

Pascal : On travaillait en Agile, sans autres approches en particulier. Donc ça commençait vite ! Dès le premier jour nous commencions à coder, puis nous avions des itérations en sprint pour les développements futures, les modifications, etc. Même si c'est assez satisfaisant car on a l'impressions de toujours travailler et de ne jamais s'arrêter, avec du recul ce n'est peut-être pas l'idéal. Il manque une longue période de réflexion et de mise en place. On travaille énormément en flux tendu ce qu'il fait qu'on a souvent des bugs ou des modifications à effectuer. Je pense que sur le long terme cela a un impact sur le projet dans son ensemble, peut-être un peu plus décousu. Et fatalement on prend pas mal de retard.

Moi :Qu'est-ce qui fonctionne ?

Pascal : Ce qui est très satisfaisant c'est que tout est clair pour tout le monde. Il n'y a pas d'ambiguïté possible car nous avons pris le temps.

Moi :Qu'est-ce qui ne fonctionne pas ?

Pascal : L'approche DDD en tant que telle n'est pas évidente à comprendre au début. Ça nécessite un certain temps d'adaptation et souvent le client à l'impressions qu'on avance pas du tout. Il faut trouver un moyen de lui faire comprendre que c'est une plus-value pour lui et nous de notre coté trouver un moyen de rendre l'apprentissage de ces concepts plus accessibles pour les nouveaux venus. D'autant plus que si tu ne comprends qu'une partie du DDD et que tu l'appliques à moitié, c'est bien pire que de ne rien faire du tout !

Moi :Comment les informations métiers vous sont-elles transmises ?

Pascal : Il y a plusieurs moyens. En général on nous donne les premières infos lors des dailys quotidiennes, afin de dégrossir en 2 minutes le travaille de la journée. Puis nous avons un ticket qui nous est assigné et il y a un lien vers la documentation. Il s'agit d'une documentation sur Confluence qui décrit à la fois l'aspect business, fonctionnel et technique. Même si en tant que devs nous ne sommes intéressés que par l'aspect technique et un peu fonctionnel, le fait d'avoir sur la même page le concept business du métier nous aide dans notre compréhension. Etant donné que le DDD se base en partie sur la notion d'Ubiquitous Language, nous retrouvons les termes de l'analyse business dans les attributs des entités, vo, etc définis par les analystes. Comprendre les notions métiers de ce que l'on code est je trouve intéressant tant personnellement que pour l'équipe et les futurs développements.

Moi :Comment cela a été fait dans les projets antérieurs ?

Pascal : On n'avait très peu d'informations business. Les analystes prémâchaient déjà tout le travail et nous n'avions pas vraiment besoin de connaitre les tenants et aboutissants de l'application que nous développions. J'avoue que je n'en voyais pas vraiment l'intérêt avant de travailler avec la méthode DDD.

Moi :Jugez-vous la découpe en VO, Entities, etc pertinente pour le développement ?

Pascal : Oui tout à fait ! Cette découpe est vraiment pensée sur les principes fondamentaux de l'orienté objet tout en y intégrant une grande part de business. Par exemple les entités sont des objets mutables donc possédant un identifiant en DB. On les utilise pour les objets métiers importants qui pourront évoluer dans le temps, comme par exemple un dossier. A

l’inverse une vo est un objet immutable et ne possède pas d’identifiant. L’exemple le plus parlant que j’explique à chaque fois est celui de l’adresse : si nous habitons rue de l’église numéro 1 et que nous décidons de changer le numéro en 2, ce ne sera plus la même adresse, plus le même objet ! Donc quand on doit changer une vo, on crée une nouvelle vo à référencer sur l’entité. Ça permet de coller beaucoup plus au métier. Et je pense que c’est assez intéressant pour les jeunes développeurs de travailler de cette façon. Ça les aide à mieux structurer les pensées quand ils sortent de l’école. Enfin je dirai que c’est intéressant car cette notion est utilisée également par les autres membres du projet, comme les analystes. C’est d’ailleurs eux qui définissent ces objets au départ, car ceux-ci doivent refléter le métier au maximum.

Moi :Comment cela a été fait dans les projets antérieurs ?

Pascal : C’était une construction plus classique, on ne faisait pas toujours la distinction entre les objets mutables et immutables même si l’on sait que cela fait partie des bonnes pratiques de notre métier.

Moi :Comment jugez-vous la lisibilité du code au travers des découpes précitées ?

Pascal : Très clair ! Les context map et bounded context donnent les grandes lignes du projet, les grandes découpes métiers tandis que le tactical pattern nous donne une meilleure granularité dans le code même.

Moi :Pouvez-vous facilement lire, comprendre et/ou reprendre le code d’un de vos collègues ?

Pascal : Oui ça devient très facile ! On ne travaille pas tous sur les mêmes epic et parfois il nous arrive de devoir effectuer une tâche sur une partie du code que l’on découvre pour la première fois. voir effectuer une tâche sur une partie du code que l’on découvre pour la première fois. Bien sûr il n'y a pas que le Tactical Pattern qui rentre en compte, mais ça n'en reste pas moins un facilitateur pour celui qui découvre le projet et le code produit par un développeur. C’est tout de suite beaucoup plus intuitif la découpe étant la même dans l’ensemble du projet.

Moi :Comment cela a été fait dans les projets antérieurs ?

Pascal : Et bien chaque développeur avait un peu un style qui lui était propre, même si l’on avait des guidlances de la part des responsables. Ce n’était pas toujours évident, surtout si on ne bossait pas en binôme ou que c’était plus rare.

transcript

Annexe II - Entretien CCO

Moi (Jérôme): Voir la première interview dans le cadre du mémoire On est le lundi sept et je interview Cédric Cox, qui est business analyste et producteur au Nord pour la société Pulse. Cédric qui est ce que tu peux me dire en quelques mots sur quoi tu travailles actuellement ?

Lui (Cédric): Alors actuellement, je travaille sur une application de gestion du recouvrement qui inclut dans ses méthodes d'analyse du DDD. Et plus précisément je travaille sur la. Tout ce qui est bilan comptable et financier au sein de l'application.

Moi (Jérôme): Actuellement tu estime que le projet a quel état d'avancement ?

Lui (Cédric): Pour sa première version commercialisable, on se situe aux environs de 80 % en termes d'analyse et 60 % en terme de développement.

Moi (Jérôme): OK. Hum. Comment est-ce que tu définirais la complexité du domaine métier sur lequel tu travailles ?

Lui (Cédric): Ça dépend des modules, des bounded context qui sont analysées. Il y a des modules qui sont relativement simples comme quand il s'agit de signalétique et où il n'y a pas de processus dynamique d'évolution des données. En tout cas elles sont très limitées, ils sont très limités. Donc là ça va. Par contre, quand la modification des données devient plus dynamique, ça devient complexe. C'est notamment le cas des bilans, parce qu'il y a beaucoup, beaucoup de processus qui viennent modifier les données en permanence. Et donc, puisque le DDD engendre aussi du traitement asynchrone, il y a des ping pong assez souvent entre le bounded contexte sur lequel je travaille et les autres. Et donc là, à ce moment-là, c'est vraiment complexe. Mais le DDD permet justement de casser une partie de cette complexité.

Moi (Jérôme): Est-ce que tu as déjà travaillé avec des projets d'une telle complexité égale ou supérieure ?

Lui (Cédric): Oui, ça m'est déjà arriver mais les méthodes d'analyse n'étaient pas les mêmes, et l'organisation dans l'équipe n'était pas la même non plus.

Moi (Jérôme): Donc, concernant ce projet ci, comment est-ce que tu définirais ton niveau de compréhension de ce métier ? Est-ce que tu estimes déjà avoir une compréhension générale excellente ou y-a-t-il encore pas mal de points d'amélioration ?

transcript

Lui (Cédric): Je pense qu'elle est bonne. Excellente d'ici une demi-année à un an à travailler sur le projet. Il y a des détails et des subtilités que je sais que ne pas encore reconnaître et qui vont arriver. Parce que le métier est plein de détails, de petits points d'attention et d'exceptions. Et je ne les connais pas tous mais, j'en ai une compréhension largement suffisante que pour pouvoir sortir une première version.

Moi (Jérôme): D'accord. Justement, si on compare par rapport à tes projets précédents. Est-ce que tu considères que ta compréhension du métier des précédents projets était tout aussi bonne ?

Lui (Cédric): C'était obligatoire d'avoir une bonne connaissance métier. Elle était meilleure que celle que j'ai du produit que je suis en train de développer aujourd'hui. Mais c'est aussi parce que j'ai travaillé pendant neuf ans dessus. Donc les périodes de travail ne sont pas du tout comparables. Et si je dois me projeter, on va dire un an environ après de travail dans mon poste dans mon précédent boulot. Je pense que s'était relativement similaire en fait, mais j'ai été vraiment pris et plongé dans le dans le cœur du truc. Pas là. J'avais vraiment dû apprendre rapidement et aussi pour ne pas avoir l'air bête devant les personnes que j'allais rencontrer. Donc j'ai pu bénéficier de pas mal de formations en interne par mes pairs au niveau du métier lui-même, parce que les connaissances métier avaient déjà été assimilées par d'autres collègues à moi. Et donc faire voyager la formation était relativement facile à ce moment-là. Ici, dans le cadre actuel, c'est différent puisque personne chez nous n'a cette connaissance, ce métier de A à Z dans tous les détails et on l'acquiert en rencontrant les externes en fait. Et donc on doit vraiment avoir tout un processus de digestion des connaissances pour récolter le besoin et arriver à rendre une analyse.

Moi (Jérôme):. Comment est-ce que tu estimes le niveau de compréhension de tes collègues de ce même métier et du projet en général par rapport à toi ?

Lui (Cédric): Elle est fragmentaire, mais ça, cela vient du fait que justement, on a cassé la complexité de l'ensemble de l'application en la divisant en différents domaines et qu'on n'a pas nécessairement demandé à tout le monde de connaître tous les domaines. Et on a préféré que les analystes se concentrent sur les domaines qu'on leur demandait d'analyser, ce que le DDD permettait, tout en ayant quand même une connaissance suffisamment poussée des domaines de base. Les domaines de base ici, c'est le domaine de tiers et le domaine dossiers. Mais après, mise à part ces deux-là, on ne demande pas à tout le monde de connaître l'ensemble des domaines et les domaines plus périphériques. Bien que les modules périphériques communiquent avec dossier et avec tiers, ils ne communiquent pas nécessairement entre eux. Parfois oui, mais pas toujours. Et donc ça permet de pouvoir s'investir dans un domaine sans nécessairement avoir la connaissance de l'entièreté des domaines de l'application.

transcript

Moi (Jérôme): OK, parfait. De manière générale, dans la gestion même du projet au quotidien, comment est-ce que vous découvrez ce qui est mis en place pour faire face au changement potentiel du projet ? Ça peut être aussi bien un changement d'analyse qu'un changement pour donner suite à une discussion métier.

Lui (Cédric): D'accord, ok, tu sais, c'est un processus relativement similaire aux autres. Va ça, Il n'y a pas grand-chose qui change à ce niveau-là. La première chose qu'on va se demander, c'est quel, quelles sont les parties d'analyse que ça va changer. Alors ici en l'occurrence, et dans quel domaine est-ce que ça va impacter ? Et puis au sein d'un domaine, quelle story d'analyse ça va impacter. Maintenant, c'est vrai que la façon dont on a découpé les choses permet d'aller relativement vite dans le ciblage. C'est plus dans la façon dont on a rédigé ces analyses avec la philosophie du DDD. Ça permet, quand on lit une analyse, qu'on sait qu'il y a des changements qui vont avoir lieu. Ça permet de se rendre compte aussi relativement rapidement de ce que ça va engendrer comme modifications par exemple, ce qu'on doit va devoir générer de nouveaux Event ? Est ce qu'on va en supprimer ? Est ce qu'on va devoir ajouter des nouvelles commandes ou pas ? Alors que si on avait une application davantage monolithique, moins découpée alors on utiliserait d'autres méthodes pour se rendre compte de ce que ça va modifier. Et je pense que plus on a d'expérience pour une application, plus on va venir mettre le doigt sur ce qui ne va pas, ce qui doit être modifié. Mais je pense que si les deux systèmes pour deux applications qui sont développées from scratch donc à partir de rien, une avec le DDD un sans le DD. Je pense que le DDD va aider plus rapidement à mettre le doigt sur ce qui va ou ce qui doit être modifié qu'une application n'est qu'une méthode d'analyse qui qui ne demande pas un découpage aussi strict.

Moi (Jérôme): OK justement. Alors pour revenir à l'approche du DDD en tant que telle et les différents patterns qu'elle met en place, est ce que tu considères que le tactical patterns, donc le fait de diviser les différentes classes du domaine en agrégat, services et autres ? Est-ce que concrètement ça a aidé dans ta compréhension du domaine métier ?

Lui (Cédric): Oui, bien que. Mais ce qu'il y a en fait, c'est que les analystes arrivaient à un certain degré d'expérience. Je pense qu'on va, comment dire ça ? Ils ont des concepts d'analyse en tête qui, sans nécessairement le savoir, se rapprochent à mon avis des concepts qu'ils ont expliqué dans le DDD. C'est juste qu'ils ne le savent pas nécessairement. Et quand ils lisent le DDD, ça permet de clarifier des choses qu'ils avaient déjà en tête. Moi, c'est vraiment ce qui m'est arrivé en fait avec le DDD. Finalement, j'ai mis des mots et des noms de concepts sur des choses que je faisais déjà avant. Pas nécessairement fourretout, mais en tout cas sur une partie de ces choses-là. Et ça, ça a permis de clarifier les choses dans mon esprit. Ça ne m'a pas spécialement aidé à faire de meilleures analyses. On va dire des analyses plus rapides parce que je ne pense pas que le DDD permet d'aller plus vite. Je pense qu'il permet de mieux faire les choses.

transcript

Moi (Jérôme): OK, mais tout à fait. Justement, je l'ai également dit parce que tu disais que tu pensais déjà faire. En tout cas appliquer une partie de ce que le DDD préconise dans tes précédentes expériences. Ben justement, comment est-ce que vous faisiez dans les précédentes expériences ? C'était implicitement déjà du DDD sans le savoir.

Lui (Cédric): Ou alors en fait ce qui se passait c'est quand on avait une application qui était une application monolithique, qui était devenue un véritable spaghetti parce que ça faisait dix ans qu'elle existait et que personne n'était vraiment au fait de l'architecture fonctionnelle de l'application et on a commencé à avoir des problèmes de performances. Des choses assez sournoises qui se passaient dans l'application sans qu'on se rende compte nécessairement pourquoi. Puis, à un moment donné, on s'est dit qu'on allait devoir prendre un peu de hauteur. On avait l'impression d'avoir accumulé une dette technologique relativement forte pour plonger l'application en maintenance relativement lourde et revoir son architecture fonctionnelle. Donc, ça veut dire finalement refaire plein de copier-coller de code d'un endroit à l'autre et de remettre les choses au bon endroit. Et à ce moment-là, on a réfléchi à comment est-ce qu'on allait procéder pour remettre les choses dans les bonnes cases ? En fait, parce qu'on avait des jeunes développeurs qui étaient venus, qui ne s'étaient vraiment jamais posé la question de comment et quelle est la bonne façon d'aller chercher des informations. Et comme l'application n'était pas bien découpée, c'était fait un peu à la hussarde parfois. Donc on se retrouvait avec, par exemple, la même récupération de données qui était fait de cinq fois différents, de cinq façons différentes à cinq endroits. Donc voilà, il a fallu remettre un peu de ménage dans tout ça. Donc on s'est posé la question comment est-ce qu'on va procéder ? On ne connaissait pas le DDD à ce moment-là. Par contre, on connaissait tout ce qui était micro-service qui est une façon d'implémenter le DDD en fait. Et on s'est dit ce qu'on va faire, ce sont des faux micro-service parce que les micro-services te disent que tu peux scinder et que chaque micro-service c'est une application avec sa propre base de données. On avait la même base de données de données derrière. Donc la seule chose qu'on a décidé de faire, c'est dire que ça reste, c'est du java pour chaque micro-service, ça aurait été la même base de données pour chaque micro service. C'étaient juste des projets Java différents en fait. Et on s'est dit on va essayer de tracer des frontières entre les différents modules de l'application. Il y avait déjà des noms de modules qui étaient plus ou moins clairs, même si c'était fort mélangé. Et entre ces différents, ces différents modules, on a affaire à ça. Vraiment, tout était synchrone. On va garder du synchrones pour tout un ensemble de choses, mais on va essayer aussi de faire de l'asynchrones quand c'est possible. Quand on a lu toute la théorie sur la micro-services à ce moment-là, on s'est rendu compte que enfin on a pris connaissance de ce qu'était là l'éventual consistancy et on s'est dit bah oui ! Finalement, c'est tout à fait vrai cette histoire. Tout n'a pas nécessairement besoin d'être synchrone à la seconde près. Avoir un petit peu de tests de désynchronisation au niveau des données, ce n'est pas quelque chose de grave. Donc on a réfléchi au re factoring de l'application en fait, dans ce sens-là, c'est ce qui a été très difficile et je pense que ce qui est difficile aussi dans le DDD à l'heure actuelle, c'est d'arriver à positionner correctement les frontières entre les processus. C'est vraiment la partie la plus la plus difficile, c'est de se dire OK, ces données-là, je les mets dans quel

contexte, dans quels domaines ? Est-ce qu'en les mettant là, je ne vais pas devoir dupliquer de l'information ailleurs ? Si oui, est ce que c'est grave ? Et ces questions-là, je me le suis posé aussi dans mon précédent job, sans savoir que je faisais quelque part un peu du DDD à ce moment-là. Et ça, c'était relativement compliqué. Et le fait de de poser des frontières entre des ensembles de données et des ensembles de processus distincts, alors d'une part ça, ça fait perdre du temps clairement aux analystes, ça fait perdre du temps aux développeurs. J'avais même des développeurs qui ne comprenaient pas, ils n'étaient pas du tout au courant de ce genre de philosophie, de découpage et d'architecture fonctionnelle. Donc on ne comprenait pas pourquoi on faisait ça. Les développeurs on a toujours fait comme ça. Donc on va continuer à faire comme ça et on ne comprenait pas pourquoi ils ne pouvaient pas faire un commit sur un objet avec treize niveaux de profondeur ou ils avaient trucmuche pour tout bidule. Et finalement vous oui, tirez la moitié de la DB chaque fois que vous avez une requête sur un objet et disait Bah oui, c'est bien pratique. Oui c'est bien pratique, mais c'est ce qu'il faut ton application en l'air après dix ans en fait. Et. Et donc on s'est demandé qu'on va. Quelle est être le ratio, la perte en termes de vélocité et de développement. Et on était entre virgule cinq et deux, deux fois plus lent avec cette méthode-là. Ce qu'il faut savoir. Et ça, tout le monde était plus ou moins d'accord, c'est qu'on gagnerait énormément en évolutivité et en stabilité de l'application.

Moi (Jérôme): Tu as déjà à moitié répondu à la question suivante qui concernait justement cette découpe en contexte map ou bounded context. Et justement à l'aide que cela va apporter à la compréhension générale du métier. Tu disais toi même que c'est parfois compliqué de savoir où caser tel et tel mené ou tel processus, et cetera Mais une fois que c'est fait de manière correcte, c'est une plus-value. Est-ce bien ça ?

Lui (Cédric): Exactement, C'est l'investissement de départ, il est difficile. Et puis une fois que c’est clarifié. Après c'est juste un moment chouette. C'est que dès tu as une nouvelle demande métier ou un changement à effectuer, si tu sais directement dans quel cas ça va tomber et tu vas cibler très rapidement ce qui n'est là où ça doit être modifié.

Moi (Jérôme): D'accord, il y aura de nouveau la question par rapport aux anciens projets. Je suppose que déjà vous tendit vers ce système-là, sans peut être spécialement en avoir totalement conscience ou en tout cas de mettre un nom dessus.

Lui (Cédric): Alors oui, exactement. Je suis resté neuf ans dans mon précédent boulot, les sept premières années et en fait on ne le faisait pas du tout ça. Donc on dès qu'il y avait une modification à faire. En plus, on travaillait en flux tendu par rapport à des demandes de support relativement urgentes, ce qui fout encore plus de bordel dans le l'application. On ne se souciait pas à ce moment-là de savoir est ce que c'est le bon endroit pour venir implémenter ça? On l'a implémenté plutôt en méthodes agiles. On voyait les clients, ils protestaient, me disaient OK, c'est passé, oui, c'est bon, oui, ce n'est pas bon. Une fois que c'était fait, c'était fait, on s'en souciait plus. On ne se souciait pas non plus de la façon dont

transcript

ça avait été fait. Mais c'est des tech lead en fait, qui normalement devaient se soucier de la bonne tenue du code et de la bonne tenue de l'architecture. Mais le problème, c'est qu'ils avaient tellement de pression sur les épaules venant des clients que le principal c'était que ça soit en prod et que ça fonctionne et pas nécessairement que ça a été fait dans les règles de l'art les deux dernières années. Comme je l'ai dit, là, on s'est dit qu'on allait rentrer en phase de maintenance. On a commencé à migrer les différentes parties de l'application vers cette nouvelle architecture fonctionnelle et là, ça nous a permis de nous rendre compte. Quand on a continué à recevoir des nouvelles demandes, c'était plus facile de dire c'était là que ça allait influer celle-là. C'était là que ça allait influencer au niveau du code, au niveau de l'analyse. Mais on avait toujours le vieux code à devoir maintenir et on était toujours aussi dans cette phase de réflexion. Il faut dire que c'était une application qui faisait plus d'un million de lignes de code et plus de 800 tables. Donc ça, c'était un énorme mastodonte et on l'avait découpé en 35 contextes différents, ce qui est énorme aussi. Et chaque bounded contexte faisait environ. Je crois que c'était une pfffffff au moins 15 à 20 tables. Quelque chose comme ça ?

Moi (Jérôme): Moi ça manquait vraiment. En plus d'une compréhension en amont pour justement une meilleure découpe, une simplification etc.

Lui (Cédric): Et comme on était une équipe de dix personnes, comme on essaie vraiment d'insuffler cette méthode de découpage en disant chaque chose à sa place, et quand on veut faire quelque chose, ça existe peut-être déjà, il faut essayer de le réutiliser. Ce n'est pas toujours évident, notamment chez des développeurs qui ont toujours fonctionné d'une certaine façon. Et tout le monde n'a pas l'esprit ouvert à des nouvelles données, à des nouveaux patterns. En fait, que ce soit en termes d'analyse ou en termes de développement, parce que le DDD a des impacts en termes d'analyse mais a aussi des impacts en termes de développement.

Moi (Jérôme): Oui, tout à fait. De manière générale, toi et l'équipe, vous estimez que vous avez tous en tête le même objectif et le même résultat attendu de ce que l'application doit être finalement ?

Lui (Cédric): C'est une question assez vaste et subjective. J'ai envie de dire.

Moi (Jérôme): Est-ce que tu as une idée de ce à quoi doit ressembler le produit final? Est-ce que tu penses que tes collègues, développeurs, analystes, et cetera ont exactement la même idée du rendu final?

Lui (Cédric): Tu n'auras pas nécessairement la même réponse en interrogeant le front end, en interrogeant le back end et en interrogeant l'analyste, l'analyste qui peut te dire ça doit être hyper tip top carrée à la virgule près pour que ça puisse sortir le front end. Donc les

transcript

grandes boites on va dire sont là, c'est globalement fonctionnel, ça va et le back end bien. Je ne sais pas vraiment des ça au backend en fait.

Moi (Jérôme): Oui.

Lui (Cédric): Je ne pense pas que tout le monde à en ait la même perception. Mais ce n'est pas la faute de DDD.

Moi (Jérôme): Oui, pas de soucis, il n'y a pas de bonnes ou mauvaises réponses de nouveau. Ça c'est pareil. Ton point de vue. Comment est-ce que tu estimes la performance de l'équipe? Tu disais toi même que justement l'implémentation du DD vous estimiez que c'était à certains moment avec une simple voire deux fois pour le potentiellement. Le temps de l'analyse plutôt. Mais de manière générale, développement aussi.

Lui (Cédric): Bien la mise en place, oui. Maintenant, une fois que tu auras une première version de l'application qui est développée. Après, on est plus dans ce ratio. Fois avec le cycle deux, on est dans un ratio. Normalement, on devrait même être en dessous d'un. Ça nous devrait nous permettre de gagner du temps. Pourquoi? Parce qu'une fois que l'analyse a été faite, ce qui peut être modifié est clairement ciblé. Le développeur sait clairement où il doit aller faire ses modifications, ne pas se poser des quinze questions sur comment est-ce que je dois faire ça. Et là, on va gagner du temps. Mais la mise en place, la création d'une première version prend plus de temps en DDD maintenant, ce qui influence quand même aussi négativement je pense. Le temps de développement clairement c'est la gestion asynchrone des choses parce qu'il faut faire beaucoup plus attention à ce qu'on fait en asynchrone qu'en synchrone. Il y a de ces histoires de driven qui arrivent en même temps, qui peuvent bloquer les données, qui peuvent faire la valeur. Et il y a plein de petit évènement qui sont engendrés par la gestion asynchrone des données en fait et même si c'est ça, le DDD nous permet de savoir rapidement où on va devoir placer de modification. Le fait de devoir gérer certaines modifications de façon asynchrone ralentit quand même l'écriture de l'analyse parce que c'est plus complexe et peut ralentir aussi l'écriture du développement. Maintenant, je pense quand même qu'il faut faire l'effort, surtout pour sortir une première version. Et après? Je pense qu'au récupère, on arrive à un rythme qui doit être à mon avis le même qu'une équipe de développement qui n'utilise pas le DDD. Sauf qu'on aura une évolutivité qui sera beaucoup plus grande.

Moi (Jérôme): OK maintenant justement avec un peu de recul, depuis que tu appliques le DDD et que toute l'équipe le fait, qu'est-ce qui fonctionne et ne fonctionne pas selon toi ?

Lui (Cédric): Alors ce qui ne fonctionne pas, ça va être difficile de dire puisqu'on n'a pas en cours de version en production. Et quand j'ai quitté mon précédent job pour commencer à

transcript

migrer les premiers modules, donc je n'ai pas pu vraiment tâter de deux choses qui ne fonctionnerait pas vraiment.

Moi (Jérôme): Ça peut très bien aussi être des interactions entre collègues de par justement une mauvaise compréhension de certains. Il n'est pas nécessaire d'attendre la fin du projet pour voir ce qui va et ne va pas.

Lui (Cédric): De mon point de vue, à partir du moment où l'ensemble des personnes ont compris comment fonctionnait DDD, la notion de limite de transaction par exemple, on comprit où il fallait placer des données par rapport à ce qu'on voulait en faire, je pense que ça aide, mais je n'ai pas vraiment vu de contraintes de conséquences négatives du DDD au niveau de la compréhension.

Moi (Jérôme): Par contre, a contrario, je pense qu'une mauvaise compréhension du DDD a un impact.

Lui (Cédric): Exactement. Voilà, c'est ça, c'est unique. Par exemple, ça n'aide pas à comprendre ce qu'est-ce que la modification asynchrone. Ou le fait par exemple, qu'on doit se limiter aux frontières d'un agrégat pour une transaction ou ce genre de choses là, Si tu ne le comprends pas, si tu ne comprends pas quelle est la différence entre une value object et une entité par exemple au niveau de leur cycle d'évolution, ça va se transformer en catastrophe et la personne va vraiment avoir la pression de subir l'analyse. Ça va mal se passer. Je pense qu'à partir du moment où tous les concepts du DDD sont clairs dans la tête des analystes, il n'y a pas de réel problème. Je pense que ça favorise même la communication. Par exemple, j'ai dit j'ai demandé il y a quelques semaines à Marion Il faut que tu ailles modifier un event, telle entité sous telle condition, tu me pop and event qui s'appelle comme ça. Je n'ai même pas dû lui expliquer pourquoi mais elle est allée dans la bonne colonne. Donc elle savait où il fallait aller pour le faire. Et elle n'avait pas nécessairement besoin d'avoir la connaissance de ce qui se passait après que l'event ait été généré. Alors je pouvais quand même dire pour lui expliquer parce que je pense que c'est quand même intéressant de qu'elle sache à quoi sert cette event. Mais elle n'avait pas besoin de savoir à quoi il servait en termes de conséquences pour vous, pour savoir où il fallait aller le mettre dans l'analyse.

Moi (Jérôme): OK, parfait. De manière générale, dans les projets réalisés aujourd'hui, surtout dans les méthodes agiles, on travaille énormément en flux tendu et on a peu de temps pour les analystes de prendre un peu d'avance. On fait régulièrement ce qu'on appelle un sprint zéro où on laisse parfois deux ou trois semaines d'avance pour faire les premières analyses et pour le temps, les développeurs fassent les premières configurations, et cetera Mais c'est à peu près tout. Le DDD, lui, préconise justement une plus longue analyse initiale, et cetera. Est ce qu'on vous a laissé le temps justement de réaliser une analyse avant que réellement le développement même ne commence ?

transcript

Lui (Cédric): Oui, clairement, oui, mais pas encore suffisamment, Je.

Moi (Jérôme): Quoi comme une attente.

Lui (Cédric): Je pense qu'on a eu trois mois d'avance, plus ou moins, parce qu'il y a eu toute la mise en place technique qui allait permettre de soutenir le DDD justement avec l'écriture d'architecture logique mais aussi des décisions d'implémentation technique. On va faire ça comme ça. Les énumérations, on va les gérer comme ça. Les transactions, on va le gérer ainsi. Tout ça a des conséquences en termes techniques et en termes de choix de librairie ou le choix de Framework. Et il a fallu du temps pour l'équipe technique pour rêver. Implémenter tout ça aussi. Faire les premiers tests, faire des POC. Donc ça nous a laissé justement ces trois mois d'analyse. Et je pense que pour la mise en place d'un produit justement, faire correspondre cette phase de mise en place technique avec le temps supplémentaire octroyé aux analystes par rapport à l'équipe technique pour vraiment démarrer l'implémentation métier. Je trouve que c'est une chouette chose. En fait, je pense qu'il nous aurait fallu à mon avis aller au moins un mois et demi de plus d'avance. Parce qu'un an après, on s'est heurté au fait que le back end nous à rattraper les analystes en termes d'implémentation. Et il y a eu collision d'équipes. Donc je pense qu'on a eu un peu plus avant, ça aurait été bien. Maintenant, on est une petite boîte ici. Peut-être que dans des plus grosses boîtes, ce qu'on aurait dit à ce moment-là et au back end, on va vous prendre ou vous mettre sur un autre projet, une d'autres ressources.

Moi (Jérôme): Mais je suppose quand même qu'on est un peu plus à l'aise ici que sur des projets précédents ou vous voudront. Et ils sont beaucoup plus.

Lui (Cédric): Oui, ce qui se passe de ce que j'ai connu, je l'ai dit d'ailleurs il y a quelques minutes, c'est qu'on travaillait en flux tendu sur base de demandes de clients pour des choses qui devaient être faites hier. Il y a deux choses qui en pâtissait les analystes et les tests. Voilà ce qui faisait en sorte qu'on ne faisait pas des choses qui étaient évolutives, qu'on faisait des choses qui étaient parfois bogué. Qui expliquait finalement dans les grandes lignes ce que le l'application faisait ou devait faire, mais n'allait jamais dans les dans les détails.

Moi (Jérôme): Dans lequel je demande pendant ces phases de pré analyses ce que vous aviez souvent des contacts avec les experts métier?

Lui (Cédric): Oui.

Moi (Jérôme): À quelle fréquence? Plus ou moins.

transcript

Lui (Cédric): Au minimum une fois par semaine.

Moi (Jérôme): Justement, le fait de rencontrer le client, entre autres, c'est pour faire des brainstorming. Nous avons l'eventstorming, qui est un peu plus proche de ce qu'on fait dans le DDD. Est-ce que tu estimes que ces réunions avec les experts métiers ont eu un fort impact sur la construction de l'ubiquité du langage? Donc il y a le langage omniprésent qui est vraiment l'un des piliers du DDD. C'est à dire que peu importe la couche dans laquelle tu travailles, nous avons tous le même langage métier. Et la même compréhension du langage.

Lui (Cédric): Moi, je sais que le DDD est censé permettre ce genre de choses là. Mais ça, je l'ai clairement moins ressenti. Tout le langage n'a pas pour moi été suffisamment respectée dans l'application a l'heure actuelle, je pense que les phases de recueil des besoins n'ont pas été gérées avec des évent storming des choses de ce genre-là ont été gérées comme des phases de recueil de besoins tout à fait normal, où on pose des questions au client ou à l'expert métier, on écoute ce qui nous raconte et on creuse et on creuse. Donc là, au niveau du recueil des besoins, on s'est rapproché beaucoup plus de ce qu'on l'habitude de voir ailleurs. Et ce n'est qu'une fois qu'on s'en retrouve en débriefing ou face à son écran que, à ce moment-là, on a essayé d'appliquer des concepts, des idées. Mais vis à vis de l'expert métier ou du client, ça n'a pas été appliqué. On a essayé de le faire une fois, il y a eu un Event storming qui a eu lieu. Je pense que le problème lors de cette phase a été que les frontières de ce dont on voulait parler étaient beaucoup trop vastes. Donc c'était l'ensemble de la phase amiable et judiciaire qui nous a occupé, dont la phase amiable nous a pris un an et demi à être développée. Et on a terminé avec, je pense, 20 mètres de murs remplis de post-it. Et ces post-it n'allaient pas toujours suffisamment dans les détails au niveau des Events, au niveau des actions, au niveau de tout ce qu'on peut retrouver dans le métier. Donc c'était à la fois, ça nous a permis d'avoir une vue très high level de ce à quoi on allait être confronté. Cette réflexion de découpage de domaines n'a pas eu lieu sur base de event storming, mais a eu lieu sur base de réflexions en interne.

Moi (Jérôme): Oui, bien sûr, mais qui prend, je suppose quand même un point d'entrée qui est les premières réflexions?

Lui (Cédric): Oui, bien sûr, parce qu'on s'est quand même rendu compte de ce qu'on allait devoir faire. Mais je pense qu'une phase de recueil de besoins est tout à fait normale. On s'assied devant un huissier de justice et en lui disant OK, racontez-nous votre métier, Je vais prendre des notes. Je pense que pour être efficace, les events storming auraient dû se concentrer sur des processus beaucoup plus ciblés. Par exemple, demander aux huissiers parce qu'on sait qu'on va devoir implémenter la gestion des tournées, ok. Vous, quand vous partez sur la route, qu'est-ce que vous prenez avec vous ? Ça consiste en quoi ? Qu'est-ce que vous faites ? Et racontez-nous votre journée. C'est ciblé ou prendre un gestionnaire de dossier et lui dire OK. Comment est-ce que vous faites quand vous recevez un mail de

transcript

demande d'ouverture d'une récupération de créance ? Qu'est-ce que vous encoder comme données par exemple ? Et quand est-ce que vous considérez que le dossier est ouvert et peut commencer à vivre sa vie? C'est beaucoup plus ciblé aussi. Ou dire quelles sont les différentes actions que vous faites, ce que vous avez fait à distance pour récupérer de l'argent, on va dire Bah, on envoie une mise en demeure ou on passe un coup de téléphone ou on demande à l'huissier de passer si on est en phase. On va peut-être faire un event storming uniquement sur la gestion des tournées de l'huissier ou mais vraiment je pense que ça c'est vraiment important de faire des storming qui sont beaucoup plus ciblés et pas essayer de se dire je fais un évent storming pour essayer de dégager le scoop complet de mon application, parce que ça, ça va, ça va, ça va partir trop dans tous les sens et on saura, je pense, pas retirer grand-chose ou en tout cas pas plus d'informations concernant les différents silos du DDD. Pas plus que si on avait fait des séances de recueil du besoin. Le Standard.

Moi (Jérôme): Ok. Mais par contre, pour revenir à l'ubiquité sur le langage qui normalement est censé découler de l'event storming. Ici, c'est un peu différent. Mais bon, l'ubiquité, ce langage est censé être un langage valable aussi bien pour les experts métier que pour nous. Analyse que pour les développeurs, et cetera. Est-ce que justement, tu estimes que ce langage est présent ? Implémenté de manière correcte chez nous. Est-ce que dans ce cas-ci, un huissier qui jettera un œil à notre analyse ou même regarderait une classe Java à implémenter, et cetera comprendraient les termes utilisés ? Est-ce que nous avons respecté cette culture évolutive ?

Lui (Cédric): Non, non, non, on n'a pas voulu le faire. Je pense qu'il y a une réelle volonté de ne pas avoir voulu le faire, en fait, parce que on est face à un métier qui est très complexe. On s'est peut-être à peu à peu présomptueux de dire ça, mais j'en suis relativement convaincu qu'il fallait prendre un peu de hauteur par rapport au métier lui-même. Prendre un peu de hauteur par rapport par rapport aux concepts métier comme on les a énoncés et se dire finalement OK, je vais faire une application qui gère du recouvrement. Quels sont les différents concepts dont je vais avoir besoin ? Oui, ils m'ont parlé de plein de choses, mais. Est-ce que c'est vraiment ce terme là que je dois réutiliser ? Est-ce que c'est vraiment cette façon-là, que je dois implémenter les choses ? Est-ce que c'est vraiment les processus qu'eux m'ont décrit que je dois implémenter tels quels et je ferai peut-être évoluer par après ? Ou est-ce que je prends de la hauteur ? J'ai de réfléchir à faire de l'abstraction, à réfléchir en termes un peu plus générique finalement que les termes qu'eux énonçait. Et est-ce que c'est n'est pas mieux d'aller dans ce sens-là ? Et je pense que la réponse est oui. Alors après on verra quand ça démarrera, si ça leur parle. Mais prenons un exemple, prenons les dossiers. On a des notions dans les bilans, la notion d'activités, de contexte et d'activités impactée. Ce sont des notions extrêmement abstraites et que les huissiers ne comprendraient absolument pas.

Quand on a un paiement qui s'appuie sur une créance, là l'activité de contexte, c'est le paiement. L'activité impactée, c'est la créance parce que le paiement vient de payer la créance et apurer une partie de la créance. Mais cette réflexion d'activités, de contexte

transcript

d'activité impactée ou d'activité révisée est générique finalement au sein de l'application. Et ça nous a permis de développer les choses de façon plus abstraite que si on avait appliqué directement les directives des huissiers. Je pense que ça, ça a de l'intérêt d'appliquer le langage métier directement dans nos classes, dans les concepts de l'analyse.

Moi (Jérôme): OK, parfait, je te rassure, on va avoir terminé. En tant qu'analyste, c'est à toi de déterminer les différents agrégats. C'est parfois un peu plus technique que ce qu'on peut demander à un business. Analyse de manière classique. Est-ce que tu estimes que c'est pertinent pour l'analyste de le faire ? En quoi est-ce que ça l'est ? Et s'il y a une réelle plus-value en tant qu'analyste.

Lui (Cédric): Clairement parce que ça oblige un analyste à réfléchir. Il n'y a pas juste survolé les analyses, ça permet de quand on va définir en case one qu'on définisse tous les cas et le else à la fin aussi que ce que si on a if on pense aussi au else. J'ai fait ma partie création de données. Ah ben oui, mais maintenant je suis obligé de faire toute la partie suppression de données. Qu'est ce qui se passe quand je supprime les données ? Est-ce que ça a des impacts ailleurs ? Ben oui, ben ça va sûrement avoir des impacts inverses que les processus de création et ce genre de choses-là. Et je pense que ça donne aussi une certaine ouverture d'esprit aux analystes pour appréhender des analyses d'applicative de plus en plus de qui peuvent être parfois complexes et de casser la complexité, de développer un problème complexe pour un problème. Je pense que le DDD, justement offre la possibilité d'aller plus loin. J'aime particulièrement aller au bout des choses et ça m'est déjà arrivé d'être relativement frustré au niveau des analyses parce que je n'avais pas le temps de faire ce que je devais faire. Le DDD t'oblige à aller au bout des choses quelque part. Ou alors tu fais du demi DDD et finalement ça ne sert pas à grand-chose.

Moi (Jérôme): Voilà. OK, et alors ? Enfin, avec tous ceux dont on a déjà parlé et la manière dont on réalise les analyses via le DDD, et cetera, est ce que ça arrive régulièrement de devoir effectuer des corrections sur des choses qui ont déjà été analysées ? Si oui, est ce que c'est quelque chose qui arrive de manière régulière ou est ce qu'on estime que le temps pris pour analyser certains modules évite justement les Re facto par la suite ?

Lui (Cédric): On est dans une phase de réflexion d'une première version. Je pense qu'il y a eu un peu trop de re factoring dans le cadre de cette première version. Maintenant, je pense que c'est inévitable parce que créer un programme de rien du tout, on est vraiment dans un processus itératif où on va écrire une première version, puis on va se rendre compte que l'on peut mieux faire. Parce que, en analysant le module qui suit, par exemple, on se rend compte de certaines choses qui peuvent avoir des impacts potentiels sur le module précédent qu'on a qu'on a écrit. Le DDD nous aide à déterminer rapidement les endroits nécessitant du re factoring, mais pas à traiter ce re factoring en tant que tel.

Moi (Jérôme): Carrément eu une erreur de commise, plutôt mauvaise compréhension.

transcript

Lui (Cédric): Ou non. Non, s'il y a eu une ou deux fois sur l'année ou on avait mal compris le métier et où on a écrit une ou deux story finalement, qu'on a dû jeter à la poubelle ce qu'on a. On avait mal compris les choses, mais j'estime que sur l'ensemble des heures story qui ont été écrites, ça reste quand même extrêmement limité.

Moi (Jérôme): D'accord, ok, c'est fini. Je te remercie beaucoup pour ton temps.

transcript

Annexe III - Entretien RMA

Speaker 1: Alors. Interview du 21 novembre de Romain Mentonnais, développeur backend chez Pulse. Romain est ce que tu sais me dire sur quoi tu travailles actuellement ?

Speaker 2: Donc je travaille sur plusieurs choses. Un projet de développement d'une application pour le milieu du recouvrement pour des huissiers de justice qui se fait en utilisant l'approche DDD et un projet pour un partenaire qui ressemble un peu à un Google Drive en fait. Spécialisé pour des images lourdes de rayons X, etc. Qui se fait de manière classique.

Speaker 1: En ce qui concerne le projet pour le recouvrement amiable, selon toi, quel est l'état d'avancement à ce jour du projet?

Speaker 2: Je dirais que pour la première version du logiciel, qui sera la première version commercialisée, on doit être aux alentours de 80 % de finition. A priori nous devrions sortir cette version d'ici 3 à 4 mois.

Speaker 1: Comment est-ce que tu définirais la complexité du domaine métier de ce projet?

Speaker 2: La complexité est assez haute parce qu'on est assujetti à beaucoup de règles qui sont des règles légales. Il y a beaucoup d'éléments qui sont des choses définies par la loi étant donné que nous travaillons dans l'intérêt des huissiers. Les huissiers ne peuvent pas faire n'importe quoi quand ils réclament de l'argent, quand ils envoient une demande qui compte des frais, etc. Donc, c'est quelque chose qui est très, très bien structuré d'un point de vue légal. Et donc ça, ça a énormément d'impact sur les analyses mais également sur le code puisqu'il faut qu'on respecte précisément toutes ces règles là et cette façon de faire.

Speaker 1: Tu as déjà travaillé sur des projets qui ont une complexité égale ou supérieure.

Speaker 2: Non pas du point de vue métier, en tout cas. La plupart des projets sur lesquels j'ai déjà travaillé ne nécéssitaient qu'un simple CRUD et une écriture basique en base de données. Avec l'application de recouvrement nous sommes quand même à un niveau supérieur, et ce dans toutes les couches du projet.

Speaker 1: Très bien. Comment est-ce que tu décrirais ton niveau de compréhension du monde des huissiers?

Speaker 2: Je dirais que j'ai les bases. Après voilà, comme on travaille avec des analyses qui

sont en général assez complètes et assez bien faites, on comprend en général au fur et à mesure qu'on code une fonctionnalité ou un élément, on le comprend au fur et à mesure sur en lisant l'analyse. Donc on apprend au fur et à mesure de l'analyse qu'on doit mettre en place. Il n'est pas nécessaire d'avoir une vue globale de ce qui se trouve dans l'application, tout du moins pas dans un premier temps. La découpe faite sur les différentes fonctionnalités de l'application pousse à une architecture micro-service et à une gestion très indépendante des différents modules. Mais, sans être obligatoire, une vision high level reste quand même un atout dans ce genre de projet. Heureusement nous avons l'architecte ou encore le product owner du projet qui possèdent cette vision.

Speaker 1: Et vis à vis de tes collègues, vous avez l'impression que votre niveau de compréhension est le même pour tous ou est-ce que tu as l'impression de mieux comprendre, moins bien comprendre?

Speaker 2: Alors moi j'ai eu une période sur lesquelles je n'étais pas sur le projet, on avait d'autres projets à côté et qu'il fallait quelqu'un pour s'en occuper. Donc il y a certaines choses où justement j'ai peut-être eu plus de lacunes sur la connaissance métier et même sur la connaissance du code du projet. Après, comme souvent les fonctionnalités, ben on essaie chacun. Quand quelqu'un développe une finalité, on essaye de prendre les x tickets qui concernent cette fonctionnalité. On se retrouve par exemple je me suis occupé de la gestion des moyens de contact à ce moment-là, j'ai beaucoup plus de connaissances. Je pense que quelqu'un d'autre sur les moyens de contact aura peut-être moins de connaissance que moi, au début en tout cas. Mais voilà, Stéphane s'est occupé d'une partie de bilan financier mais a beaucoup plus de connaissances que moi. Je veux dire selon sur quels éléments on travaille, on a plus ou moins de connaissances puisqu'on n'a pas forcément le temps de prendre de lire toutes les analyses et de tout comprendre puisque voilà, les deadlines font qu'on doit avancer au quotidien. Mais oui je pense qu'on a tous une certaine maitrise du domaine, selon notre poste ou notre implication dans le projet. Mais on ne peut pas dire qu'on ne connait rien, sûrement pas ! Je pense d'ailleurs que certains pourraient même se revendiquer huissiers eux-mêmes.

Speaker 1: Ok. De manière générale, comment est-ce que vous découvrez et appréhender les changements qui se font sur le projet? Par changement, j'entends par exemple du re factoring à faire sur certaines fonctionnalités. Comment est-ce que vous vous découvrez ça? Comment est-ce que ça vous est transmis ? Donc soit les analyses dans le cadre d'un review ou se rendre compte qu'il y a quelque chose à adapter.

Speaker 2 : Nous travaillons avec la méthodologie Agile Scrum dans ce projet et nous prenons connaissance des différentes adaptations lors des dailys. Donc à ce moment-là, on a soit un nouveau ticket avec l'explication, soit une réunion en direct ou on nous explique que voilà, il y a ces éléments-là qui changent ce que ça implique sur le code et à ce moment-là, à nous d'aller voir dans le code pour s'assurer qu'on n'oublie rien qui sera touché par ces

modifications-là. Ou alors, quand c'est des trucs un peu plus conséquents, il arrive qu'il y ait un petit POC qui soit fait dans le cadre des bilans. Par exemple, on avait un simulateur assez basique mais qui permettait de prendre en compte toutes les fonctionnalités. Mais quand il y a eu un re factoring, il a d'abord été adapté dans le simulateur avant d'être intégré au code, au code principal

Speaker 1: OK. Est-ce que tu dirais que l'utilisation du tactical pattern, donc le fait de décrire des entités, des value objects, des agrégats, etc. T'aide dans ta compréhension du métier du monde des huissiers? Est-ce que cette découpe influence ta compréhension?

Speaker 2: En partie dans le sens où le fait d'avoir justement des values, objects, des agrégats, etc ; permet d'avoir une meilleure vision de l'importance des différents éléments. En fait, si toutes les entités sont simplement des objets en DB, on a moins de facilité à voir la hiérarchisation dans l'importance de ces objets-là par exemple. Il est clair que dès que l'on voit un aggregate root, on voit tout de suite que c'est quelque chose de central, d'extrêmement important, qui ne peut pas être manipulé sans précautions. Tandis qu'une Value Object, c'est beaucoup plus restreint, ça a beaucoup moins d'impact sur le code. Chaque élément du tactical pattern a un niveau d'exigence différent et sera utiliser de manière différente. Donc ça aide vraiment à structurer. Je trouve que ça donne de l'importance aux éléments.

Speaker 1: Justement, quand tu parles de structure, où est ce que tu penses que l'utilisation des bounded context ou context map qui est un terme un peu plus général, ça t'aide également dans la compréhension du métier?

Speaker 2: Je ne sais pas si ça aide dans la compréhension du métier parce que des fois ça empêche parfois de faire des liens, puisqu'on développe vraiment de façon à en faire un bounded context un peu indépendant. Ensuite on développe le suivant, mais je pense que ça aide pour ne pas se perdre dans tous les liens non plus. Ça nous permet de garder une vue high level de l'application, ou en tout cas du module sur lequel on est en train de travailler. Dans le cadre de notre projet, on travaille sur le contexte des dossiers, ça permet de rester concentrés sur les dossiers parce qu'il y a énormément de connaissances finalement à travers le projet, dans tous les modules, et ça permet de se concentrer sur les dossiers et sur ce qui touche aux dossiers pour ne pas ne pas se perdre dans les connaissances ou confondre avec une value object qui a à peu près le même nom dans un autre contexte. On reste, je trouve, concentré sur l'élément sur lequel on travaille et c'est pour moi un peu plus facile dans un projet de cette envergure-là, de ne pas se confondre d'un élément à l'autre. Même si la découpe peut sembler complexe dans un premier temps, au final elle nous sert à faire la différence entre tous les éléments de notre application et surtout à mieux cibler nos actions.

Speaker 1: Justement, tu parles d'un projet quand même d'une taille assez importante. Est-

transcript

ce que tu as l'impression que les objectifs à court, moyen long terme sont clairs pour toute l'équipe? Et est-ce que vous êtes tous conscients du résultat attendu?

Speaker 2: Alors le résultat attendu en tant que tel? Pas exactement. On a une vision, je dirais à moyen terme, des modules qui vont être développés, des modules qui vont être ajoutés justement pour cette V1, la première version commercialisable. Après, il y a eu déjà des discussions pour des modules à intégrer en V2. En V3, on a une idée de ce qu'ils vont faire, mais on n'a peut-être pas encore ça. Les analystes, certains analystes s'en vont en discuter avec le client. Là, ce serait intéressant d'ajouter ça. Donc là, on n'a pas forcément la vision à long terme complète, je dirais, mais par contre, à court terme, on a quand une idée assez précise. Après je ne pense pas qu'en tant que développeur j'aie besoin de cette vision, contrairement à un analyste ou un architecte.

Speaker 1: Donc quand je parle de manière générale, comment est-ce que tu estimes les performances de l'équipe ou la vélocité de celle-ci? Ça peut très bien varier du début du projet jusqu'au bout maintenant.

Speaker 2: L'approche DDD au début de projet a peut-être un petit peu ralenti l'équipe parce qu'il y a beaucoup de règles, beaucoup de mise en place à effectuer. On a mis en place énormément de de tests ArchUnits, donc ils vérifient que certains principes sont respectés tout le long du projet. Et pendant que les développeurs mettaient en place l'architecture et la configuration du projet, les analystes avaient le temps de prendre un peu d'avance sur les analyses et les discussions avec le client. Sans cette avance je pense qu'on les aurait rattrapés beaucoup trop vite et qu'on se serait retrouvé bloqué. Donc je pense qu'au début ça a peut-être un peu ralenti, mais une fois que le rythme est pris et que le sprint initial est passé ça forme une structure qui permet justement d'accélérer un peu la vie du projet. Parce que voilà, on va rajouter un élément, on a X guidelines à respecter, on a ces tests qui vont vérifier qu'elles sont bien respectées, donc qui évite par la suite de se retrouver avec un test qui saute parce que on n'a pas ajouté cette annotation sur tel ou tel élément. Ben première fois qu'on fait tourner les tests, il nous dit tout de suite à tel type d'entité doit être à noter de telle façon pour respecter les règles qu'on a mis en place. Et à ce moment-là, on évite des fois de chercher des erreurs qui n'en sont pas, simplement parce qu'il manque une annotation quelque part. Donc oui les performances ne sont probablement pas à la hauteur lors des premières semaines, mais nous y gagnons lors des phases de régressions, de tests ou encore de modifications. Tout devient plus claire pour tout le monde parce que on sait qu'on a une base solide et commune à tous.

Speaker 1: En l'état actuel du projet, selon toi, qu'est ce qui a bien fonctionné? Et à contrario, qu'est ce qui n'a pas fonctionné? En gardant en tête justement cette approche très découpée, cette approche proche du métier, Qu'est ce qui serait potentiellement à refaire ou qu'est-ce qu'il faut absolument garder?

transcript

Speaker 2: Je pense qu'ici, dans notre cas, on a peut-être parfois été un peu trop précis avec les découpes. On se retrouve avec des tickets qui font parfois 4 h, parfois 2 h et des fois il y a des tickets qui pourraient être groupés. On se retrouve, on essaie et en général voilà. Si je prends le premier ticket, que le deuxième est quasiment le même, je vais faire le deuxième ticket. Mais c'est vrai qu'il arrive que quelqu'un d'autre prenne le ticket parce que oui, et en découpant trop précis, des fois, on perd un peu cette capacité à réutiliser ce qu'on vient de faire dans l'élément suivant je trouve.

Speaker 1: Et justement parce que c'est plus découpé qu'à l'accoutumée, est ce que ça ne va pas justement te faire prendre conscience ou en tout cas augmenter ta compréhension de ce qui est réellement demandé? Ou alors peut être que tu vas perdre aussi en abstraction et qu'on ne va pas être à çà et là aussi.

Speaker 2: Oui et non. Parce que des fois voilà que d'un point de vue documentation, s'il y a deux tickets, en général il y a deux éléments de documentation. Mais des fois, ces deux éléments qui sont tellement liés que je trouve que c'est un peu perdre du temps de de dire voilà, dans le cadre des dossiers, par exemple, on a un ensemble d'activités, par exemple, les activités d'ouverture d'un dossier, eh bien, il y avait plusieurs types, parce qu'il y avait ouverture et ouverture d'une autre type, etc. Et celui qui va faire l'ouverture et au moment ou faire une réouverture, va se retrouver avec un élément d'élément qui peut réutiliser de ce qu'il a déjà fait. Avoir aussi la possibilité de factoriser le code. Donc des fois, si on découpe par types d'activités les activités d'ouverture, je pense qu'on peut gagner un peu de temps par rapport à une découpe vraiment plus précise. Il serait nécessaire de faire plus de lien entre les différents tickets. Cette découpe hyper précise est une bonne chose si ça reste claire dans l'ensemble du projet et cohérent. Ça demande d'être plus rigoureux qu'à l'accoutumé.

Speaker 1: D'accord. Et qu'est-ce qu'il aurait été par contre que tu gardes?

Speaker 2: Ben justement, je pense que en effet, la mise en place de tests archi uniques qui vérifient l'infrastructure de manière générale prennent un peu de temps au début, mais ça aide vraiment par après à éviter des bugs qui des fois sont extrêmement dur à trouver. Parce que voilà, ça engendre un mauvais lien dans la DB qui ne se voit pas directement. Mais au moment où on va chercher une entité, il y a un élément qui se met à annule parce qu'il n'a pas été chargé de la DB ou parce que le cache n'a pas été rafraichi. Et tout cette mise en place, tous ces tests et cette structure qui est quand même très précise, permettent d'éviter énormément de tests et énormément de problèmes. Et souvent, ce qui prend le plus de temps, c'est de trouver. Ce n'est pas spécialement de faire le code, c'est de tester, de déboguer. Et je pense que ça permet de réduire ce temps-là de manière assez conséquente. La documentation, même si je l'ai un peu critiqué précédemment reste un point positif pour

moi car elle nous permet d'être tous alignés. Et si quelqu'un rejoint le projet en cours de route il ne devrait pas avoir de soucis à s'y retrouver rapidement.

Speaker 1: Ok, en tant que développeur, comment est-ce que les informations métier vous sont transmises?

Speaker 2: Donc nous on travaille avec Confluence, donc c'est un site web dans lequel tous les éléments du projet sont décrits et formaliser à l'aide de l'approche DDD dans notre cas. On a vraiment une page consacrée au projet qui contient un ensemble de sous pages pour chaque bounded contexte. Plus précisément comme nous utilisons la méthodologie agile nous avons aligné chaque bounded contexte sur les Epic du projet. Dans ce bounded contexte, les différents agrégats ont leur point et on a vraiment pour chaque fonctionnalité ou chaque élément, une page décrivant les prérequis, les entrées et les sorties, la façon, la compréhension, la description métier et les liens minimum vers ce qui va être mis en code. En général chaque aggregat représente une User Story, voir plusieurs si les analystes estiment que la découpe n'est pas assez granulaire. Puis quand la story est définitivement finalisée, celle-ci est encore découpée en plusieurs chapitres. Certains plus pour nous le dev et d'autres plus pour les analystes. Mais comme l'architecte a la volonté d'utiliser le langage des huissiers nous pourrions sans problème lire la partie des analystes et eux la nôtre.

Speaker 1: On a déjà à moitié répondu, mais est-ce que tu juges que cette découpe justement en value object, est ce que c'est réellement pertinent pour un développeur? Est-ce que ça a une plus-value pour vous?

Speaker 2: Dans un projet comme ça, c'est large avec une grosse part de règles métier et d'informations métier? Je pense que oui. Après, j'ai travaillé sur des projets où c'était finalement des gros CRUD avec des règles de validation, etc. Mais qui est beaucoup plus simple, avec beaucoup moins de liens entre les éléments. Je pense que ce n'est pas forcément nécessaire parce que voilà, c'est assez lourd à mettre en place. Il faut un projet je pense assez conséquent, avec assez d'éléments pour que ça en vaille la peine. Mais dans le cas d'une application complexe, avec plusieurs niveaux dans le métier, là ça devient vraiment justifier. Dès que l'on rencontre tel ou tel type d'entité on sait directement ce que l'on peut faire avec ou pas. Si nous avons à faire à une entity nous savons qu'elle aura forcément un Id et que celle-ci sera mutable. Contrairement à la value object qui sera immutable et remplacée à chaque édition. Cette découpe influence notre code et nous force à prendre conscience, de comment, euh, bien faire les choses je dirai. Ne pas se retrouver avec un gros plat de spaghetti au final.

Speaker 1: Ok. Et est-ce que tu juges justement que cette découpe augmente la lisibilité du code que vous produisez?

transcript

Speaker 2: Euh, je pense que oui. Quelqu'un d'extérieur ne devrait pas avoir de difficulté à rentrer dans le code, pour peu qu'on lui explique le fonctionnement de l'approche mise en place. Mais pour nous c'est difficile de s'en rendre compte quand on est le nez dans le code. Mais je pense que ça en tout cas rendra plus facile pour gérer les problèmes pour quelqu'un qui n'a pas vu le code puisque comme les bounded contexte sont très bien définis. Un problème va être facile à localiser sans devoir se tracasser de toute l'architecture du projet. En fait, on sera fait facilement dire dans tel bounded contexte on a un problème et je pense que pour quelqu'un qui n'a pas vu le code, on aura quand même plus facile de trouver la source du problème. Chaque chose a sa place et est définit dès le départ. Par contre encore une fois, ça nécessite une rigueur et du temps au début du projet pour tout mettre en place. On ne le ferait pas pour n'importe quel projet.

Speaker 1: C'était ma dernière question justement est ce que si tu devais reprendre du code que tu n'as pas écrit toi-même et qui respecte la découpe du tactical pattern. Est-ce que justement cette découpe-là va faciliter ta lecture et ta compréhension de quelque chose que tu n'as pas écrit toi-même?

Speaker 2: Maintenant que je connais le tactical pattern, oui, je pense que pour quelqu'un qui n'a jamais vu, qui n'a jamais fait de DDD et qui n'a jamais vécu ce pattern-là, il faudrait d'abord prendre la peine de comprendre le pattern avant de se plonger dans le code pour vraiment y avoir un intérêt. Sans ça c'est la catastrophe assurée mais si tu prends le temps de bien comprendre toute l'approche du DDD et surtout du tactical pattern pour un dev ça devrait réellement te faciliter les choses. Mais à nouveau c'est un investissement car il faut que l'ensemble de l'équipe soit aligné sur cette approche et sur comment la mettre en application.

Speaker 1: Donc oui, je suppose qu'il faut avoir une compréhension assez exhaustive de l'approche DDD que pour pouvoir l'appliquer de manière correcte, on ne le fait pas à moitié.

Speaker 2: Oui, c'est ça. Je pense que si on le fait à moitié et qu'on s'autorise des largesses par rapport à certains grands principes du DDD, on va vite se retrouver à avoir un truc qui est moins clair que si on n'avait pas appliqué le DDD du tout. En fait, je pense que ça doit être fait correctement ou pas fait du tout.

Speaker 1: Est-ce que ce serait alors une approche qui aurait également une plus-value pour des nouveaux venus dans la société pour justement qu'eux prennent en main plus rapidement le projet et comprennent plus vite ce qu'on attend d'eux?

Speaker 2: Il y aura une peut être une petite période d'apprentissage sur le DDD qui n'aurait pas forcément. Il ne serait pas forcément nécessaire si on n'utilise pas le DDD, forcément.

transcript

Mais je pense que la plus-value, il y aura quand même une plus-value sur la compréhension du code, donc je pense que ça équilibrera largement le temps perdu à apprendre le DDD. Je pense qu'au total ça restera quand même bénéfique.

Speaker 1: C'est top! Je te remercie.

transcript

Annexe IV - Entretien SD

Moi: Donc interview du 30 novembre de SD de frère développeur backend. Stéphane, est ce que tu peux me dire sur quoi tu travailles actuellement? Donc pour le.

Stéphan: Pour le moment je travaille sur le projet Gerico donc qui est un projet pour développer un logiciel de gestion pour les études de boîtiers et qui vise à remplacer Diogène, donc qui est le programme de gestion des études de huissiers le plus utilisé en Wallonie, il me semble, peut-être même dans toute la Belgique. Et personnellement j'occupe le poste de développeur backend depuis un peu de 2 ans.

Moi: Et voilà quel est l'état d'avancement du projet actuellement.

Stéphan: Pour le moment, c'est quand même plutôt bien avancé même si nous n'avons toujours pas une version qui est commercialisable. Mais il me semble que ça fait quand même plusieurs semaines qu'on approche assez rapidement du but. Donc moi j'espère que d'ici un gros mois on a une première version commercialisable. Même s'il reste encore quelques développements à réaliser, et surtout des tests à exécuter.

Moi: Ok, Comment est-ce que tu définirais la complexité du domaine métier de ce projet, à savoir au milieu des huissiers?

Stéphan: Alors moi ça me semble très compliqué. Donc dans un premier lieu, on a travaillé surtout sur la partie qui traite des tiers d'un dossier qui n'était vraiment pas très compliquée. Voilà, c'était simplement des liens entre tiers, donc il n'y avait pas de forte complexité métier là-dedans. Maintenant, une fois qu'on a commencé à arriver sur dossier, donc avec toute la gestion des activités propres à un huissier de justice, là ça a commencé quand même à beaucoup se complexifier. Par exemple, il y avait toute des parties traitant des communications entrantes/sortantes qui devait être généré automatiquement par des events. Ou encore la gestion de la vie d'un dossier, ce qu'il pouvait y avoir comme prescrits légaux. Puis, à titre personnel car c'est spécifiquement là-dessus que je travaille depuis plusieurs semaines, moi c'est vraiment quand je suis arrivé sur Bilan où je ne comprenais vraiment rien. Heureusement nous avons pris le temps de nous préparer et nous avons réalisé un POC sous forme d'un simulateur Python. Parce que même si l'analyse était ultra complète sur le sujet, étant donné que nous essayons de coller un maximum au vocabulaire de métier, ce n'est pas toujours évident pour les non-initiés. Je peux comprendre que dans la plupart des autres modules c'est une plus-value dans la compréhension commune du projet mais là j'avoue avoir un peu décroché car ça touchait principalement à de la comptabilité. Donc en fait, oui le métier est compliqué, mais certaines parties plus que d'autres.

transcript

Moi: Ok, mais justement, comment est-ce que tu décrirais ton niveau de compréhension à toi? Donc hier, il n'y a pas trop de difficulté, mais de manière générale, pour les dossiers par exemple, est ce que tu considères que ton niveau de compréhension est quand même assez bon?

Stéphan: Maintenant oui, étant donné que j'ai suivi le projet depuis le début, je pense que je vois bien toutes les interactions qu'il y a eu. Il y a peut-être un ou deux modules dont je n'avais pas la charge et que je connais un peu mais ça n'empêche pas la compréhension globale de l'application. Maintenant, le bilan, il y a quand même encore des parties qui restent assez floues.

Moi: Et vis à vis de tes collègues, est ce que vous, vous avez plus ou moins le même niveau de compréhension?

Stéphan: Oui et non… je pense que nous avons tous la même compréhension globale mais parfois à des niveaux différents selon les modules sur lesquels nous avons travaillé. Par exemple je ne crois pas que Laurent comprendrait quoique ce soit au module Bilan dont je m'occupe actuellement. Enfin, je pense qu'il arriverait à y effectuer des modifications si nécessaire car nous travaillons tous sur base des mêmes canvas de code ou sur la même analyse. Mais il risque de s'arracher les cheveux. Oui par exemple lui, il a développé tout le module Tâches. Moi, je ne sais pas ce qui se passe dedans. Maintenant, je pense quand même que si on me demandait de débugger des trucs dans tâche, j'y arriverais. Mais ça me demandera plus de temps que si c'est Laurent qui devait le faire par exemple. Enfin je dirais que le module Bilan est assez à part. Par contre le product owner ou l'architecte ont probablement une bien meilleure compréhension que les autres membres du groupe. Ils sont à l'origine des découpes métiers et étaient présents lors des interactions avec les huissiers. Mais pour rester dans le général je pense que oui, nous avons tous une compréhension plus ou moins similaire.

Moi: Globalement ok. De manière générale, comment est-ce que vous découvrez les changements qui sont apportés au projet en cours de route?

Stéphan: Alors très souvent, c'est l'architecte qui arrive dans le bureau et qui dit on a changé des trucs. Afin ça c'est la manière un peu franche de faire les choses et officieuse. En réalité on a quand même une demande un peu plus officielle sous forme de ticket JIRA et on nous en parle lors des dailys. Mais bon voilà, Renaud (l'architecte) aime bien nous le faire savoir avant la cheffe de projet. C'est Renaud qui va créer un ticket pour dire il y a ceci à changer dans tel fichier, et bien quelqu'un se charge de prendre ce ticket-là, quoi. Et à chaque fois, il met quand même bien le lien vers l'analyse où par exemple, si c'est pour bilan, il a fait des modifications dans le simulateur sur tel commit, il met le lien du commit.

transcript

Moi: Est-ce que tu estimes que l'utilisation du tactical pattern, donc l'utilisation de Value Object Entities d'agrégat peut t'aider dans la compréhension du métier, de l'importance de certaines valeurs du métier?

Stéphan: Oui, ça ça je pense. Maintenant, bon, je n'ai que peu d'autres expériences que le tactical pattern, mais il me semble que donc la découpe vraiment en objets métier permet vraiment de se rendre compte déjà de à quoi sert cet objet et de connaitre quelles sont toutes ses implications, toutes les interactions que ça avec les autres objets du domaine. Moi je pense que c'est une bonne chose qu'on utilise ça ici en tout cas, en tout cas au moins dans le projet. Comme le sujet est déjà assez complexe, le fait de découper tout en Value Object ou entité par exemple nous aide déjà à comprendre l'utilité de ces objets. Est-ce que l'on peut modifier ceci ou cela ? Est-ce qu'il réécrire l'objet ? On sait directement quoi faire avec l'objet. C'est comme avec les aggregate root. Comme ils collent à la découpe du projet tu sais toujours où effectuer tes modifications et ce que cela va impacter.

Moi: Comment cela se déroulait-il sur tes anciens projets s'il y en a ?

Stéphan : ben on n'avait jamais appliqué le DDD sur mes précédents projets. C'était beaucoup plus simple, beaucoup plus scolaire. Mais je ne pense qu'il aurait fallu appliquer pour autant le DDD dessus. C'est un peu comme aller chercher sa baguette chez le boulanger en F16 plutôt qu'à pied, ce n'est pas adapté… C'est vrai qu'au début ce n'est pas évident à comprendre. C'est une mécanique qui demande un temps d'investissement conséquent, et pas seulement d'une personne mais de l'ensemble de l'équipe. Mais après sur de gros projet je pense qu'on est gagnant. Par contre sur un projet simple avec juste quelques appels CRUD en base de données ça n'en vaut clairement pas la peine.

Moi: Ok. Et l'utilisation des context map est bounded context. Donc le fait de découper les gros modules et de les séparer de manière claire, est ce que ça vous aide aussi dans la compréhension?

Stéphan: Oui, déjà rien que le fait de nommer le module en fonction du métier, de faire des frontières très clairs ça aide beaucoup. On sait directement dire que telle fonction ou telle entité appartient à tel module. Le fait d'avoir tout découpé correctement nous permet aussi d'avoir une manière vue high level et de bien différencier chaque partie de l'application. On sait toujours clairement sur quoi on travaille, il n'y a pas d'ambiguïté. On implémente vraiment chaque module 1 à 1 et on peut voir l'application évoluer.

Moi: Comment cela se déroulait-il sur tes anciens projets ?

Stéphan : C'était développé un peu tout en même temps. Il y avait quand même une découpe mais ce n'était pas vraiment réfléchi en amont. On suivait nos tickets et on se disait au fur et à mesure qu'il faudrait peut-être séparer tel ou tel concept. Mais c'était quasi

toujours sur le moment même. Il y avait quand même des schémas d'architecture mais il ne prenait pas en compte le métier et les difficultés qu'on peut rencontrer avec le domaine. Mais de nouveau, c'était des projets moins complexes, donc on le ressentait probablement moins.

Moi: Est-ce que tu considères que toi et l'ensemble de l'équipe vous avez un objectif commun et clair et que le résultat qui est attendu, attendu au final est également clair pour vous?

Stéphan: Oui, parce que, nous avons des réunions trimestrielles avec les patrons pour nous parler des objectifs globaux, des échéances etc. Puis nous avons les dailys qui se déroulent tous les jours et donc on parle presque quotidiennement des objectifs. C'est juste que ça varie un peu en fonction du contexte. Les dailys ça reste fort sur le court-moyen terme alors que la grosse communication c'était vraiment sur l'objectif final.

Moi: Comment est-ce que tu définirais la performance de l'équipe actuellement en termes de vélocité?

Stéphan: Je crois qu'on est assez efficace… De manière général les tickets sont terminés en temps et en heure pendant le sprint. Je pense que c'est en autre dû à l'analyse qui est assez complète et juste dès le départ. Pas besoin de faire beaucoup d'allers retours, de faire du refacto sur ce qui a déjà été développé. Je ne dis pas que ça n'arrive pas mais c'est assez rare. Et puis on est rarement bloqué. Par exemple si je pars en vacances ou je suis absent, mes collègues pourront facilement reprendre mon travail en l'état. Ce n'est pas une boite noire pour eux. Par contre on sent que c'était un peu plus laborieux au début du projet. Tout le monde n'avait pas je pense bien compris la façon de fonctionner, moi le premier. On a dû me réexpliquer plusieurs fois le Tactical Pattern ou encore le langage commun des huissiers. Et puis ce n'était pas facile à mettre en place directement. Mais une fois les premiers mois passés c'est devenu étonnement facile. Et on arrive à tenir les délais, ce qui est plutôt rare dans les projets informatiques.

Moi: Comment cela se déroulait-il sur tes anciens projets ?

Stéphan : C'est difficile à dire parce qu'on fonctionnait en mode Agile classique. On avait nos objectifs du sprint et on arrivait souvent à les réaliser. Par contre on se rendait compte sur le long terme qu'il y avait du refacto à faire parce qu'on n'avait pas toujours bien pris le temps de faire les choses correctement. Donc on avait de nouveaux tickets pour retravailler sur d'anciennes fonctionnalités déjà développées. C'est le chef de projet qui avait la vision complète mais je pense que 90% des sprints étaient complétés à temps mais que la durée du projet dans sa totalité a été augmenté.

Moi: En reprenant en tête l'approche des DDD pour toi, qu'est ce qui fonctionne en

appliquant cette approche et qu'est ce qui ne fonctionne pas a priori? Qu'est-ce que tu garderais? Et à l'inverse, qu'est-ce qu'il faudrait éviter?

Stéphan: Donc, d'un point de vue purement technique, nous on utilise la librairie Hibernate pour faire tout le mapping de base de DB en base de données, et Hibernate n'a pas du tout été pensé pour justement fonctionner avec du DDD. Donc je me souviens encore les trois ou quatre premiers mois où j'ai travaillé ici on a eu mais énormément de soucis pour arriver à faire du DDD avec Hibernate. Et encore maintenant, il y a y a pas mal de trucs où on a mis de tout petit fix. On s'est dit on essaiera de résoudre ça après. Qui sont toujours là. On n'a toujours pas vraiment de solution. Enfin de vraies solutions. Si on veut faire du DDD, eh ben on est obligé de fonctionner comme ça quoi. Ça rejoint un peu ce que je disais précédemment avec la mise en place qui est plus longue et compliqué au début. Maintenant, je ne sais pas si ta question c'était vraiment lié pour la technique.

Moi: Non il n'y a pas de soucis ! et pour les aspects un peu moins techniques ? Tu as un avis ?

Stéphan : Ben j'ai trouvé les interactions entre analystes, dev et métier beaucoup plus claire que ce que j'ai connu précédemment. On avait l'impression de parler de la même chose et donc les discussions avec nos collègues étaient beaucoup plus simples. Etant donné que ce sont les analystes qui définissent les éléments du tactical pattern ou encore les frontières des bounded context, et que nous, développeurs, les gardons tels quels dans le code, on sait directement de quoi on parle quand il y a des questions ou un souci. C'est rare mais c'est même arrivé une fois que je doive participer à une réunion avec un huissier et j'ai été étonné de voir que je comprenais ce qu'il racontait.

Moi : Comment est-ce que les informations métier vous sont transmises?

Stéphan: Alors ce dernier temps, ça a souvent été on nous assigne à ticket, on lit le ticket. Ensuite le ticket fait toujours référence à une partie de l'analyse. Après ça dépend, soit on comprend directement le contenu soit c'est un peu plus compliqué et on demande des explications à l'analyste qui a rédigé cette partie. En fait ça dépend un peu du module dans lequel on est et de ça complexité. J'avoue que pour Bilan je suis régulièrement allé voir dans le bureau d'en face. Par contre le module tiers ne demandait pas d'explications en plus, ou alors très peu. Et de toute façon les analyses sont découpées en respectant les approches DDD et donc en respectant la manière dont on va découper notre code. Tout le monde est gagnant car je ne me perds pas dans l'analyse, même si celle-ci est assez conséquente, et l'analyste peut comprendre une partie du code étant donné que les noms des objets viennent de lui.

Moi: Est-ce que tu juges que cette découpe du tactical pattern en V.O. Entity etc. Est pertinente pour le développement?

transcript

Stéphan: En tout cas ça m’aide à mieux structurer mon code. Le Tactical Pattern utilisant une hiérarchie entre les objets assez précise je peux plus facilement positionner ceux-ci dans mon projet. Ce n'est pas vraiment quelque chose que l'on apprend à l'école, ou en tout juste les grandes lignes sans rentrer dans une application pratique comme c'est le cas dans le monde du travail. Et je suis certain que mon voisin fera la même chose que moi donc au moins on est aligné. Après je manque un peu d’expérience pour pouvoir comparer avec les patterns existants mais par rapport à ma petite expérience je pense que c’est bénéfique.

Moi: Ok. Et est-ce que tu juges que ça aide à la lisibilité du code de manière générale?

Stéphan: Oui, parce que nous on structure vraiment notre code en bounded context. Donc ça veut dire qu'on a chaque fois à la racine des projets le nom du contexte. Ensuite on a donc plusieurs sous fichiers, donc avec l'application services, le domaine, notamment les repositories, les services, etc. Et dans le domaine, on va retrouver à chaque fois les différents agrégats. Dans chacun de ces sous dossiers toutes les values objects qui sont liés à ces aggregate roots, etc. Donc en lisant les analyses, on peut facilement savoir où tel fichier se trouvera. Et donc oui, il me semble que ça aide pas mal, parce que si tu as une idée de l'analyse en tête, ben rien qu'en regardant la structure, enfin l'architecture plutôt du dossier, tu sais où se trouve tel fichier et quelles interactions ça avec les dossiers demandés. Donc oui, il me semble que ça aide beaucoup à la lisibilité de l’application.

Moi: Et alors? Enfin, est ce que tu considères que ça t'aide à facilement comprendre lire par exemple le code de tes collègues? Est-ce que tu y arriveras plus facilement à reconnaître ce que tu dois faire, réaliser, modifier potentiellement et voir même, est-ce que ça pourrait aider un nouveau collaborateur qui arrive à plus rapidement comprendre ce qu'on attend de lui?

Stéphan: Oui bien sûr. Comme déjà dit je pense que s’échanger nos tâches ne se fera pas facilement si le module est compliqué mais ça restera totalement lisible car nous appliquons la même démarche et que c’est très structuré. Pour un nouveau collègue on a déjà le cas et ça s’est bien passé. Par contre on a du vraiment prendre le temps de le former à l’approche DDD. Ça ne sert rien de le lancer directement dans le code ou avec une moitié de compréhension. Soit il ne s’y retrouvera tout simplement pas, soit il modifiera les mauvais attributs dans les mauvais modules.

Moi: Ben voilà c’est fini, je te remercie.

transcript

Annexe V - Entretien MVA

Moi: Alors. Interview numéro quatre du 7 décembre 2022. Marion, est ce que tu sais me dire sur quoi tu travailles actuellement?

Marion: Je suis dans la fonction de l'analyste business et fonctionnelle et je travaille sur les analyses du projet Gerico depuis la rédaction, la réflexion préalable et aussi bien les analyses business que fonctionnelle. Je travaille sur ce projet depuis environ 2 ans, donc pratiquement depuis le lancement de celui-ci.

Moi: Est-ce que tu sais me dire en deux mots en quoi correspond le projet Gerico?

Marion: Gerico est un logiciel de gestion pour dans un premier temps les huissiers de justice afin qu'ils puissent gérer leurs dossiers et le suivi de ceux-ci, aussi bien en phase amiable dans un premier temps et ensuite dans une phase judiciaire. Mais Gerico est quand même voué à évoluer et à pouvoir s'appliquer à d'autres secteurs que le métier d'huissier de justice. La roadmap nous a été communiqué récemment et l'on sait que l'on va devoir approcher des avocats et des sociétés de recouvrement.

Moi: Selon toi, quel est l'état d'avancement actuel du projet?

Marion: Nous parvenons à une fin d'une première version, d'ici encore un ou deux mois en tout cas, et on devrait pouvoir commencer la deuxième. En tout cas la version deux pour ce qui est des analyses, c'est-à-dire la partie judiciaire du métier d'huissier.

Moi: Comment est-ce que tu définirais la complexité du dossier métier sur le projet Géricault élevé?

Marion: Ça fait appel à beaucoup de termes techniques, judiciaires, légaux et donc ce qui implique que la compréhension n'est pas toujours facile, sans l'aide des acteurs métier en tout cas. Heureusement nous avons fait pas mal de réunion en amont des analyses. Ça a permis de pas mal démystifier les concepts et nous a aidé à définir les premières intentions métier. Et puis nos contacts chez les huissiers restent disponibles pour nos questions pratiquement tous les jours donc nous sommes rarement bloqués. Donc oui pour en revenir à la question je dirai que le métier est très complexe mais nous sommes bien entourés.

Moi: Et ton niveau à toi de cette compréhension que tu estimes qu'il est plutôt bon.

Marion: Il évolue au fur et à mesure du temps. Donc oui, ça fait un an et demi que je travaille

transcript

sur ce projet-là, donc ça me permet d'avoir un peu plus de connaissances et d'être plus imprégné par le métier. Mais il y a encore plein de choses au niveau légal et pour le judiciaire pour lesquelles je dois encore creuser. Mais ce sont des modules qui interviendront dans une deuxième version du logiciel donc nous devons encore avoir des réunions avec le métier pour en discuter.

Moi: Et si tu devais comparer par rapport à tes collègues, est ce que vous estimez que vous avez tous plus ou moins le même niveau de compréhension? Non.

Marion: Certains collègues avec plus d'expérience en analyse et peut être sur ce sujet là aussi, poussent les réflexions plus loin. Étant donné qu'ils ont déjà plus d'expérience en tant qu'analystes mais aussi sur la méthode employée. Et puis ça dépend aussi des tâches qui nous sont données. La découpe en Epic ne rend pas spécialement obligatoire la compréhension de tous les modules. On doit pouvoir naviguer sans problèmes entre eux bien sûr mais pas tout connaitre sur le bout des doigts. Personnellement je me suis occupé du module dossier et je pense le connaitre en profondeur, contrairement au module tâche qui n'a pas du tout été de mon ressort. Mais comme ces deux modules doivent interagir entre eux, il est nécessaire d'avoir une compréhension high level de l'application. Et je pense que c'est pareil pour tous mes collègues en fait, développeurs ou analystes. A l'exception peut-être de l'architecte et du product owner qui eux on une connaissance assez exhaustive.

Moi: De manière générale, quand il y a des changements à apporter au projet et donc plutôt analyse dans ton cas, comment est-ce que tu les découvres? Il y a quel le format? Comment est-ce que ça t'est amené?

Marion: Il y a deux aspects principaux je vais dire, soit quand le travail a déjà été effectué par nous-mêmes, puis le développeur prend le point en charge et on se rend compte qu'il y a des choses qui sont restées incohérente. En général ces problèmes sont rapidement détectés car ils sont dû directement à la cohérence entre modules. Par exemple on essaie d'effectuer des commandes entre différents modules, différents agrégats plutôt que de propager l'infos via des évènements. A priori c'est assez rare car l'analyste évite ce genre de soucis mais, si ça arrive quand même, la deuxième ligne, les devs, le repère immédiatement car c'est eux qui implémentent formellement les entités. La deuxième chose, et c'est probablement la seule qui apporte le plus de modifications, c'est quand on discute avec des acteurs du métier, les huissiers ou autres. On leur propose nos solutions qui ne sont pas toujours les plus adaptées à la réalité du métier. En général on essaie de tacler un maximum de points lors de nos réunions d'analyse, on pose vraiment beaucoup de questions pour coller un maximum avec leurs besoins. Puis on essaie de réfléchir à la problématique, à ce que l'on a relevé et la semaine suivante ou à la prochaine réunion on leur propose nos idées.

Moi: Ok, dans le cadre de l'approche des DDD, on préconise le tactical patterns. Donc pour

rappel, c'est l'utilisation de radio objet. Donc tu utilises d'agrégats. Est-ce que tu penses que cette découpe t'aide dans ta compréhension du métier d'huissier?

Marion: Oui, parce que ça permet de structurer toutes les informations que l'on peut recevoir dans le cadre de ces missions-là. Donc ça permet de les structurer en fonction parfois de dépendances, les unes par rapport à l'autre, parfois par le contenu vraiment de ces informations. Donc ça permet une structure plus claire de toutes ces données. Ça nous aide à mieux comprendre certains concepts métiers étant donné que les différentes structures du Tactical Patterns ont des objectifs différents. On sait par exemple que les entités seront réservées pour les concepts métier plus important, comme un tiers ou un dossier. Les agrégats c'est encore plus important car ils délimitent la découpe et les actions au sein de l'application. Les Value Object par contre définissent de plus petits objets, concepts, qui n'ont pas besoin d'identifiants par exemple. Et je pense que de manière générale ça aide également les développeurs à mieux comprendre les enjeux du métier de huissiers car c'est eux qui implémentent tous ces objets.

Moi: Est-ce que cette découpe en context map bounded context, est ce que ça t'aide dans la compréhension du métier?

Marion: La même chose ça permet vraiment de structurer les idées et de se concentrer que sur certaines parties qui sont indépendantes l'une de l'autre pour pouvoir mieux, mieux gérer et mieux structurer toutes les données en fonction du bounded context auquel ça appartient. Au début du projet c'était même hyper important car ça m'a permis de comprendre les premiers concepts du métier. Pour une mathématicienne, l'univers des huissiers est plutôt obscur au premier abord mais grâce à cette découpe j'ai pu mieux me représenter les différents enjeux. C'est également sur base de cette structure que nous avons pu présenter les premières infos aux développeurs. Nous voulons autant que possible que les devs comprennent les spécifications du métier et pas juste qu'ils suivent bêtement l'analyse. Je pense que la découpe les a aidés dans cette compréhension.

Moi: Ok, Est ce que tu estimes que les objectifs du projet, le résultat attendu est énoncé de manière claire et est ce qu'il est pour toi compris de la même manière par tout le monde?

Marion: L'objectif principal qui est donc de développer une application de gestion, Il me semble qu'il est bien compris par tout le monde après la manière d'y arriver, peut-être pas toujours.

Moi: Qu'est-ce que tu entends par la manière d'y arriver qui serait différente?

Marion: Le projet est assez jeune, ainsi que l'équipe dans sa majorité. On vient tous

d’horizons différents, avec des expériences diverses et ça se ressent dans la manière d’appréhender le projet. Certains sortent juste de l’école et n’ont jamais eu de projet aussi complexe alors que d’autres ont plusieurs années d’expériences et un avis déjà assez arrêté sur ce qu’il faut faire et comment y arriver. Réussir à accorder les violons de tout le monde n’est pas évident. Mais on essaie de toujours prendre le temps, surtout au début du projet, pour s’aligner un maximum. Ça prend peut-être un peu de temps au début mais c’est une plus-value pour la suite du projet.

Moi: D'accord, Selon toi, comment est-ce que tu estimerais la performance de l'équipe? Donc on peut calculer ça en termes de vélocité par exemple, la rapidité à laquelle on va analyser puis développer un point précis. Est-ce que tu estimes que nous sommes performants à ce niveau-là?

Marion: Il me semble que nous sommes assez performants, parfois plus qu'à d'autres moments, en fonction du sujet traité. Mais comme dit précédemment ça n’a pas toujours été le cas, le début a été compliqué, assez lent. Il nous a fallu plusieurs mois pour arriver à cerner le métier et commencer un semblant d’analyse. Je pense d’ailleurs que ça a fait un peu peur aux patrons au début, ça prenait trop de temps. Mais maintenant on est content d’être passé par là car ça nous évite pas mal de soucis rencontrés habituellement dans les projets Agile. Il y a beaucoup moins d’allers-retours, moins de bugs détectés, moins de questions de la part des devs. Mais voilà, ça ne s’est pas fait sans difficultés. L’approche DDD est assez lourde et compliqué à appréhender, il faut être certain que tout le monde a bien compris comment l’utiliser et ce que signifie les différents patterns qu’elle utilise. Sinon c’est probablement pire que de ne pas l’utiliser tout court. Mais après c’est efficace.

Moi: Justement, dans l'approche actuelle, qu'est-ce que tu estimes qui fonctionne plutôt bien? Et à contrario, qu'est ce qui ne fonctionne pas et qu'il faudra améliorer?

Marion: Ce qui fonctionne bien, c'est maintenant plus qu'avant la communication entre les équipes. On ne craint pas d'aller l'un vers l'autre discuter d'un point, ça va mieux maintenant. C'était aussi un point qui n'allait pas avant. Et lors de gros problème principal, c'est que certaines personnes sont indispensables sur beaucoup de points, ce qui peut bloquer parfois au niveau des analyses ou des développements quand ces personnes-là n'ont pas le temps de tout faire en même temps. Heureusement comme nous avons pris le temps de nous aligner, quelqu’un qui n’a pas directement travaillé sur un module pourra quand même y appliquer des modifications. C’est sûr que ça prendra un peu plus de temps pour lire et comprendre l’analyse sur ce point précis, mais il ne sera pas perdu.

Moi: Combien de temps est ce qui vous a été laissé pour effectuer les premières analyses du métier avant de commencer réellement à réaliser cette application? Ou si tu n’étais pas là au début du projet? Combien de temps est ce qu'on vous laisse alors pour des gros modules pour avoir le temps de les analyser avant qu'il ne passe réellement au développement?

transcript

Marion: De base je pense que nous avons eu presque 4 mois, à raison de 2-3 workshops par semaine pour arriver à capter un maximum d'infos. Entre ces réunions nous avons essayé de formaliser une première ébauche d'analyse et nous l'avons présenté régulièrement aux experts métiers afin d'avoir leur ressenti, être certain que nous allions dans la bonne direction. Cette base était super importante et allait conditionner la suite de l'analyse donc nous voulions être certain d'être dans le bon. Et pendant ce temps là ça laissait le temps aux devs de créer le projet et de comprendre comment intégrer le DDD à celui-ci. Je ne m'y connais pas assez mais je pense que ça a été assez compliqué pour eux, tant dans la compréhension théorique du concept DDD que dans son implémentation concrète. Mais nous avions heureusement le temps.

Moi: Questions directement liées Est ce que vous communiquez régulièrement avec les experts métier?

Marion: Oui, en tout cas, tout ce qui concerne les analyses a priori, on communique avec eux pour leur avis au début de chaque gros module. En tout cas un module métier, pas un module technique, pour savoir comment on peut mener l'analyse de ce module. Et par la suite on va régulièrement, parfois plus que d'autres vers eux pour voir si ce qu'on a fait tient la route. Nous avons aussi organisé des Events Storming avec les huissiers afin de capter les grandes idées du métier et pouvoir déjà procéder à une pré-découpe dans l'analyse.

Moi: Alors de manière générale, suite à ces entrevues avec les experts métier, est ce que tu estimes que vous avez construit ensemble ce qu'on appelle l'Ubiquitous langage? Pour rappel, c'est le fait de créer un langage commun aussi bien par le métier que par les analystes qui seraient également répercutés dans le front. Ce qui fait que tout le monde parle d'un concept métier, possède les mêmes termes en fait, et les mêmes mots pour exprimer celui-ci. Est-ce que si l'expert métier jette un coup d'œil au code, il reconnaîtra directement les mots qu'il utilise dans son métier?

Marion: En partie. Pour les modules plus métiers c'est même certain. On voulait que tout le monde comprenne bien les implications métiers, les analystes bien sûr mais également les développeurs. Donc oui il y a bien un langage commun, en tout cas on essaie un maximum. Par contre dès que le module s'éloigne un peu du métier, devient un peu plus technique, ça n'a plus d'intérêt et on laisse les devs s'en occuper à leur façon, selon leurs bonnes pratiques.

Moi: Ok. Ensuite, les deux dernières questions, mais tu as déjà à moitié répondu. C'était autre chose à ajouter étant donné que ce sont les analyses qui vont déterminer les objets

qui les synthétisent, etc. En amont, est ce que tu trouves que c'est pertinent pour un analyste justement de les déterminer et pas le développeur? Et en quoi est-ce que ça pourrait être une plus-value pour toi?

Marion: Euh, au niveau des analyses, ça pourrait être deux façons de voir les choses différentes. Premièrement c'est nous les analystes qui rencontrons majoritairement les experts donc pour coller au mieux au métier ça me semble important que ce soit nous qui définissions les entités et VO. Mais d'un autre coté ça a quand même un aspect un peu plus technique que dans une analyse classique. Par exemple j'ai appris ce que signifiait les termes mutable et immutable et l'importance qu'ils avaient pour les développeurs. Et Justement pour les devs c'est exactement l'inverse, ils avaient déjà cet aspect technique et ils ont dû monter en connaissance avec la partie métier que nous avons défini. C'est gagnant-gagnant je trouve.

Moi: D'accord. Et dernière question est ce que tu as déjà plus ou moins répondu aussi? Est-ce qu'il y a des corrections régulières qui sont faites de l'analyse déjà réalisée et si oui, à quelle fréquence est-ce quelque chose qui arrive vraiment régulièrement?

Marion: Pour le moment ça arrive un peu plus souvent qu'à l'habitude car nous sommes dans la dernière ligne droit de la V1, les derniers tests. On essaie de donner les derniers coups de tournevis mais c'est beaucoup de détails, pas de gros changement. Par exemple nous ne définissons pas explicitement dans l'analyse certains points de design et c'est seulement quand nous testons que nous nous rendons compte d'incohérence. Donc là on essaie de préciser un peu plus dans l'analyse et on reteste. Par contre il y a peu d'allers-retours en ce qui concerne le backend. Souvent quand il y a des points bloquants, ils sont pris directement au moment où ils surviennent et sont assez rapidement réglés car nous savons toujours déterminer où se trouve le souci. Avec la découpe granulaire qui a été faite en amont c'est assez facile, surtout que celle-ci est la même entre l'analyse et le développement. Mais ça reste assez rare à part dans les cas que j'évoquais plus haut.

Moi: Ok, je te remercie.