



FACULTÉ DE
**GESTION,
ÉCONOMIE
& SCIENCES**



Éviter le legacy en optimisant la maintenance applicative

Mémoire de recherche présenté en vue de l'obtention du Master 2 Cyber (2022 / 2023)

Anthony Quéré

Tuteur : Lucien Mousin

REMERCIEMENTS

Je tiens à remercier toutes les personnes m'ayant aidé dans la rédaction de ce mémoire.

Dans un premier temps je voudrais remercier mon tuteur de mémoire, Lucien Mousin, pour ces précieux conseils et sa disponibilité qui m'ont permis de travailler efficacement.

Je souhaiterais aussi remercier l'ensemble du DavLab de Davidson SI Nord pour leur accompagnement et leurs retours d'expériences qui m'ont permis d'agrandir mon socle de connaissance et d'apporter de nouveaux éléments pour enrichir ce mémoire.

Je tiens à témoigner toute ma reconnaissance à Jules Spicht et Sylvain Lavazais, mes tuteurs d'alternance, pour s'être rendus disponibles pour les interviews m'aidant ainsi à compléter la recherche de données pour ce mémoire.

Je tiens également à remercier les nombreuses personnes m'ayant aidé à relire et corriger ce travail de recherche.

Enfin, je remercie mes proches, en particulier ma famille et mes amis pour leur soutien au quotidien.

ATTESTATION PLAGIAT

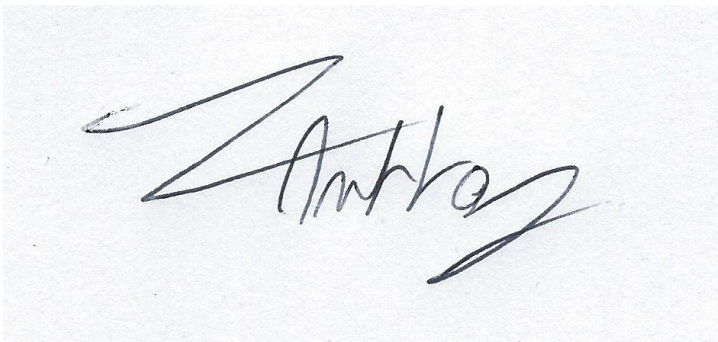
Je soussigné, Quéré Anthony, étudiant à la FGES, durant l'année universitaire « 2022/2023 » certifie que le présent mémoire de Master « Éviter le legacy en optimisant la maintenance applicative », est strictement le fruit de mon travail personnel, de synthèse et d'analyse.

Toute citation (articles, livres, mémoires, documents d'entreprises, sources Internet, ...) est formellement notée comme telle, explicitée et référencée dans le corps du texte et en bibliographie. Tout tableau ou modèle (photos et illustrations diverses) est dûment cité s'il est emprunté à un auteur ou cité en source s'il est adapté.

Tout manquement à cette Charte de non-plagiat entraînera la suspension de l'évaluation du mémoire, une notation égale à 0, et la convocation devant le conseil de discipline de l'école.

Fait, à Lille le 28/04/2023

Signature

A handwritten signature in black ink on a light gray background. The signature is stylized, starting with a large, sweeping 'Z' shape that loops around the first part of the name 'Anthony'. The name 'Anthony' is written in a cursive, flowing script.

ATTESTATION PLAGIAT.....	1
INTRODUCTION.....	4
Une tour qui part dans tous les sens.....	4
Langage de programmation.....	4
Entité.....	5
Framework.....	5
Paradigme de programmation.....	5
Programmation procédurale.....	6
Programmation orientée objet.....	6
Programmation fonctionnelle.....	6
REVUE DE LA LITTÉRATURE.....	8
Clean Code : La méthode de Martin.....	8
SOLID.....	8
SRP : Single Responsibility Principle.....	8
OCP : Open-Closed Principle.....	8
LSP : Liskov Substitution Principle.....	9
ISP : Interface Segregation Principle.....	10
DIP : Dependency Inversion Principle.....	10
Un code de bonne qualité : Nommage des entités.....	12
Une réflexion plus moderne.....	13
Anti-pattern : Les interfaces de classe.....	13
Les effets de bord en programmation orienté objet.....	14
Architecture hexagonale.....	16
Architecture.....	17
Adapters.....	18
Inspirations.....	18
Au-delà du code.....	19
You build it, you run it.....	19
Normes et définitions.....	20
MÉTHODOLOGIE DE RECUEIL DES DONNÉES.....	22
Cible des entretiens.....	22
Déroulé des entretiens.....	22
Introduction.....	22
Les applications dites legacy.....	23
Clean Code.....	24
Agilité.....	24
You build it, you run it.....	24
ANALYSE DES RÉSULTATS.....	26

La documentation.....	26
L'équipe.....	26
Maintenance opérationnelle et maintenance applicative.....	27
Une rigueur automatisée.....	28
Conception de l'application.....	30
DISCUSSION.....	31
La documentation.....	31
L'équipe.....	32
Maintenance Opérationnelle.....	34
Une rigueur automatisée.....	35
Conception de l'application.....	36
CONCLUSION.....	38

INTRODUCTION

Depuis les débuts de l'informatique moderne, les évolutions en termes de technologies et de bonnes pratiques n'ont cessé de s'accélérer de façon exponentielle. Dans ce contexte en constante évolution, il est en parallèle devenu important que les applications gagnent en fonctionnalités. Le problème majeur rencontré est que l'ajout des fonctionnalités d'une application est souvent complexe car une nouvelle fonctionnalité peut changer (directement ou indirectement) le comportement des autres créant ainsi des applications de plus en plus complexes à étendre. La plus simple des fonctionnalités qui aurait pu prendre quelques heures en début de projet finit par prendre plusieurs semaines à développer dans une application avec un plus fort historique.

Une tour qui part dans tous les sens

Voyez ici la construction d'une application comme une tour d'habitation dont l'achèvement ne sera jamais terminé. Vous ne savez pas s'il faudra qu'elle grandisse en largeur ou en hauteur, sur combien d'étages, quel type de fenêtre... Il est alors nécessaire d'établir une solution dans laquelle il sera simple d'ajouter ou de modifier les éléments sans voir tout s'effondrer et devoir recommencer à zéro. Ainsi vous pourrez facilement vous adapter aux nouveaux changements.

Avant de nous intéresser à comment rendre notre code plus adaptable aux modifications, il est important de préciser la nature de notre code.

Langage de programmation

L'internaute définit la notion de langage de programmation par « Un langage de programmation est un ensemble de règles et de symboles permettant de formuler des algorithmes et de produire des programmes ». Dans la fin des années 1940 furent créés les langages assembleurs permettant aux développeurs de ne plus devoir transcrire leur code en binaire. Puis furent créés d'autres langages de plus en plus simples d'utilisation comme le A0 puis le Fortran, le COBOL, le C... Jusqu'à aujourd'hui avec des langages

créés il n'y a même pas dix ans comme le Go ou le Rust (*Clean Architecture de Robert C. Martin, introduction de la deuxième partie*).

Entité

La notion d'entité représente tout élément manipulable. Une entité peut être une variable, une fonction, une classe, un groupement de classe (package ou composants par exemple) voire même des applications (dans le cas d'une architecture microservice, donc composée de plusieurs applications, par exemple).

Framework

La notion de Framework n'est pas approfondie dans ce mémoire de recherche mais elle peut aider à comprendre certains passages.

Des millions de programmes existent, nombreux sont ceux qui ont du code en commun. Pour épargner aux développeurs la tâche de devoir toujours tout refaire à la main, des outils ont été créés comme par exemple des bibliothèques permettant d'ajouter des entités externes dans les programmes sans avoir à réécrire le code. Au fur et à mesure certains outils ont commencé à être utilisés ensemble et ces ensembles furent appelés des frameworks.

Paradigme de programmation

Nous comptons aujourd'hui plusieurs méthodes de programmation principales que l'on appelle "paradigmes". Les paradigmes sont utilisés par des langages ou frameworks et permettent de garder une cohérence dans la logique d'un langage à un autre. Certains langages sont dit multi-paradigm, ce qui signifie qu'on peut les utiliser avec un ou plusieurs paradigmes comme le Javascript ou le Python. Nous pouvons identifier 3 paradigmes de programmation principaux.

Programmation procédurale

Premièrement la programmation procédurale, comme décrite dans l'article *Programmation procédurale - Définition et Explications de **Techno-Science.net*** est un

paradigme de développement basé sur l'utilisation des procédures. Une procédure est une suite d'instruction ordonnée, « incluant d'autres procédures, voire la procédure elle-même », nous parlons alors de récursivité. C'est un paradigme très commun dans les langages C ou Go par exemple.

Programmation orientée objet

Ensuite nous avons la programmation orientée objet souvent abrégée en OOP (Object-Oriented Programming) qui place l'objet au centre. Un objet est une brique logique représentant « un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. » *Programmation orientée objet - Définition et Explications, **Techno-Science.net***. Nous retrouvons très souvent ce paradigme en Java, en C++ ou en Dart par exemple.

Programmation fonctionnelle

Le paradigme de programmation fonctionnelle, décrit entre autres dans l'article *Programmation fonctionnelle - Définition et Explications de **Techno-Science.net*** dans lequel l'élément central est la fonction qui ici reprend la logique d'une des fonctions mathématiques. Il impose le principe d'immutabilité et rejette les changements d'état. La programmation fonctionnelle a été imaginée avant même le début de la programmation elle-même et est fortement basée par le lambda-calcul inventé par Alonzo Church dans les années 1930 comme décrit par **Robert C. Martin** dans le *chapitre 6 de Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Nous retrouvons ce paradigme dans des langages comme Clojure ou Haskell.

Dans ce contexte de nécessité d'évolution constante de nos applications, nous nous demanderons quels éléments permettent d'assurer la maintenabilité au long terme d'une application.

REVUE DE LA LITTÉRATURE

Clean Code : La méthode de Martin

Aujourd'hui, quand il est question de bonnes pratiques d'écriture de code, les ouvrages de **Robert C Martin** (ou Uncle Bob) reviennent systématiquement et plus spécifiquement *Clean Code* Et *Clean Architecture*. Le premier se concentre sur l'écriture du Code en elle-même en partant de règles de nommage des entités jusqu'aux scopes des fonctions. Le second a une approche plus haute pour se concentrer sur la structure des différents composants d'une application et les façons dont ils doivent communiquer.

SOLID

Les principes SOLID sont largement détaillés dans *Clean Architecture*. Ils ont pour objectif de créer une architecture de code tolérante au changement, simple à comprendre et utilisable dans n'importe quelle application.

SRP : Single Responsibility Principle

Le principe de responsabilité unique impose qu'une entité (une fonction, une classe, un composant, un package...) n'ait qu'une seule responsabilité, ce qui signifie qu'il ne doit y avoir qu'une seule raison pour laquelle cette entité serait modifiée (**Robert C. Martin, 2012**).

Prenons un composant qui manipule les utilisateurs d'une application. Ce composant ne doit être modifié que si la stratégie de gestion des utilisateurs est modifiée.

OCP : Open-Closed Principle

“A module is said to be open if it is still available for extension. For example, it should be possible to expand its set of operations or add fields to its data

structures. A module is said to be closed if it is available for use by other modules.”

Object Oriented Software Construction, Bertrand Meyer, 1988, chapitre 4

Ce principe impose qu'un ajout de fonctionnalité ne doit pas modifier le code existant d'un projet mais doit uniquement en ajouter. Pour ce faire, chaque entité de notre code doit être aisément extensible mais la signature de cette entité (par exemple, les méthodes proposées) ne doit pas être modifiée. Ainsi, on réduit au minimum le risque de changer le comportement d'une entité qui utiliserait le code que l'on modifie. (*Clean Architecture: A Craftsman's Guide to Software Structure and Design, Robert C. Martin, 2012*)

En programmation fonctionnelle, ce principe peut aussi s'appliquer en utilisant la composition de fonction et les "high-order functions" (fonction pouvant prendre une autre fonction comme paramètre et / ou retournant une fonction). (*Do the SOLID principles apply to Functional Programming, Patricio Ferraggi, 2022*)

LSP : Liskov Substitution Principle

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

Do the SOLID principles apply to Functional Programming? par Patricio Ferraggi

Le principe de substitution de Liskov dit qu'une entité doit pouvoir être remplacée par une sous-entité sans altérer l'exécution du programme. Bien que le lien avec le polymorphisme de la programmation orienté objet est flagrant, ce principe peut tout à fait s'appliquer en programmation fonctionnelle. Par exemple avec l'utilisation des paramètres génériques, ce qui est très courant en programmation fonctionnelle, les fonctions résultantes de la fonction avec les paramètres génériques se comportent toujours de la même manière sans nécessiter de changement sur le code résultant. (*Patricio Ferraggi, 2022*)

ISP : Interface Segregation Principle

Ce principe énonce qu'il est préférable d'utiliser plusieurs petites interfaces plutôt qu'une grande. Ces petites interfaces sont dites "client-specific" ce qui signifie qu'elles ne sont faites que pour un seul client. Ce qui implique en toute logique que le client définit les interfaces dont il a besoin. (**Robert C. Martin, 2012**)

En programmation orienté objet, il est courant de pouvoir implémenter plusieurs interfaces dans une même classe. Cependant le concept d'interface n'est pas propre à la programmation orienté objet et ce concept existe aussi dans les autres paradigmes de programmation même s'il n'est pas toujours explicite.

DIP : Dependency Inversion Principle

Le principe d'inversion de dépendance dit qu'une entité ne doit jamais dépendre directement d'une autre mais doit toujours passer par une forme d'abstraction. Pour schématiser, prenons :

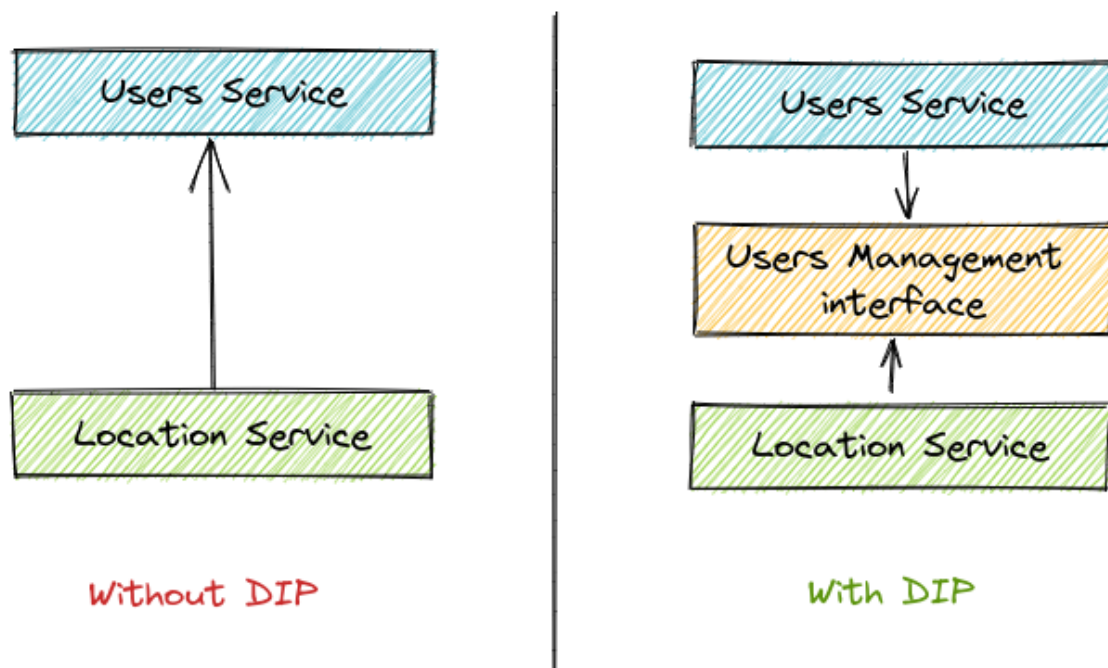


fig.1 - Exemple d'utilisation du Dependency Inversion Principle

Dans l'exemple de fig.1, nous avons deux composants : "Users Service" et "Location Service". "Location Service" a besoin des données des utilisateurs pour fonctionner, il va donc utiliser le "Users Service". Pour respecter le principe d'inversion de dépendance, le Location Service doit passer par une interface ici appelée "Users Management interface" implémentée par le "User Service". Cette interface sert de contrat et permet de rendre abstrait la logique du "Users Service".

Ce principe a un grand intérêt car il est très résistant au changement. Mettons que la gestion des utilisateurs soit déportée dans une autre application afin de rendre la gestion des utilisateurs globale au sein d'une entreprise. La logique du "Users Service" dans notre application de départ changerait alors complètement. Il faudrait donc implémenter une nouvelle entité qui serait chargée de communiquer avec cette nouvelle application. Tant que cette nouvelle entité implémente l'interface, il n'y a nul besoin de modifier le "Location Service".

Un code de bonne qualité : Nommage des entités

Après avoir abordé les fondamentaux de l'architecture d'application avec **SOLID**, il est maintenant sujet de prendre nos composants indépendamment et d'étudier les bonnes pratiques de construction. *Clean Code: A Handbook of Agile Software Craftsmanship* de **Robert C. Martin** traite un grand nombre de sujets mais nous prendrons le nommage des entités comme exemple car c'est le plus générique.

Le deuxième chapitre de l'ouvrage est concentré sur le bon nommage des entités dans le code. Ici quand il est question d'entité, il est question de tout ce qui peut avoir un nom dans le code d'une application. Par exemple une classe, une fonction, une variable, une constante, un attribut ou de manière générale, tout ce que le langage a besoin de nommer. Les règles principales quant aux choix des noms sont peu débattues :

- Un nom doit clairement exprimer le rôle de l'entité et ne pas porter à confusion
- Deux noms doivent être distinguables de façon logique. Par exemple, si on a une entité `Product`, on évitera d'en appeler une autre `ProductData` car ça n'apporte pas d'information sur la différence entre ces deux entités. Par extension, il vaut mieux éviter les mots tels que `Data` ou `Info` dans le nommage des entités car ils n'apportent pas d'informations. Toujours dans ce chapitre, ils sont qualifiés de *noise words*, pour signifier leur manque de sens.
- Un nom doit être prononçable et avoir du sens une fois prononcé, c'est pourquoi il faut éviter voir exclure les acronymes dans le nommage des entités.
- Chaque concept de l'application doit être identifiable par un mot et ne doit être identifiable que par ce mot. Un autre concept ne doit pas être identifié par ce même mot.
- Un nom doit être trouvable facilement si cherché dans la globalité du projet. Pour se faire, il ne faut pas hésiter à avoir des longs noms de variable

The length of a name should correspond to the size of its scope

Clean Code: A Handbook of Agile Software Craftsmanship, **Robert C. Martin**, 2008, Chapitre 2.

Une réflexion plus moderne

Bien que les ouvrages de **Robert C. Martin** restent des références sur le sujet, certains éléments sont encore source de débat.

Anti-pattern : Les interfaces de classe

D'après **Scott W. Ambler** dans *Process patterns: building large-scale systems using object technology*, un *anti-pattern* est une solution à un problème récurrent qui est souvent inefficace voire contre-productive.

Dans l'article *Explicit interface per class anti-pattern*, l'auteur, **Marek Dec**, énonce un anti-pattern souvent utilisé dans le Framework Java EE mais pouvant tout aussi bien s'appliquer dans d'autres langages ou Frameworks. Cet anti-pattern est l'utilisation des interfaces explicites de classe ou de façon plus concrète l'utilisation d'interfaces destinées à être implémentées par une seule classe. Il est expliqué que cette pratique remonterait du début de l'injection de dépendance dans laquelle elle était nécessaire pour la phase d'injection.

Nous retrouvons bien sûr la problématique du rôle de l'interface qui doit être définie par l'entité qui en a besoin puis implémentée par les autres entités et non l'inverse. Avoir une interface conçue uniquement pour une classe n'est donc pas utile car la classe pourrait être utilisée seule.

Afin de repérer cet *anti-pattern* et le supprimer, l'article propose plusieurs moyens :

- Le nom de l'interface contient le nom d'une technologie : Une interface devant être la plus générique possible, la dissocier des technologies utilisées pour ses implémentations est important car la classe utilisant l'interface n'a pas besoin de savoir quelles technologies sont utilisées derrière la classe.
- Les types de retours, de paramètres ou les exceptions sont associés à l'implémentation : Par exemple, si le type d'un paramètre est défini dans la

dépendance utilisée dans son implémentation. Les types utilisés se doivent d'être le plus générique possible afin de faciliter la création de nouvelles implémentations.

- Le nom de l'implémentation contient les mots « Impl », « Default » ou tout autre mot ne décrivant pas l'implémentation.

Là où l'auteur porte une idée contraire à **Robert C. Martin** est qu'à plusieurs reprises dans *Clean Code*, il est fait mention de cette pratique de création d'interface dédiée à une classe. Ce sujet est d'ailleurs une dualité dans les ouvrages de **Robert C. Martin**. Dans *Clean Architecture* il est exprimé qu'une interface doit être définie par le module en ayant besoin et non pas dans le module implémentant ce qui va dans le sens de l'article de **Marek Dec**. Cependant *Clean Code* casse à multiple reprise ce principe sans détailler pourquoi.

Les effets de bord en programmation orienté objet

En lisant *Clean Code*, il est surprenant de voir à quel point le livre se concentre sur la programmation orientée objet en omettant les autres paradigmes de programmation que ce soit pendant les exemples ou les appellations d'entités. Dans cette partie, il sera question de la notion d'effet de bord en programmation orientée objet.

Un effet de bord désigne tout changement d'état dans un programme se produisant en dehors du scope de la méthode ou de la fonction en cours d'exécution. On peut prendre comme exemple d'effet de bord la modification d'une variable globale, d'un fichier ou une modification en base de donnée.

Dans le cadre de la programmation orientée objet, la notion de scope est plus complexe. Prenons une classe `User` avec deux attributs privés `name` et `age`. Si dans cette même classe nous avons une méthode `birthday` ne prenant aucun paramètre et ajoutant 1 à l'attribut `age`. En Java, cela pourrait donner :

```

class User {
    private String name;
    private int age;

    public void birthday() {
        System.out.println("Happy Birthday " + name + " !");
        this.age += 1;
    }
}

```

fig.2 - Exemple de classe avec setter en Java

La situation présentée en fig.2 présente-t-elle un effet de bord ?

Il y a deux possibilités. Soit on considère que l'objet avec ses attributs et méthodes ne forment qu'une seule et même entité donc on exclut les effets de bords, tant que rien n'est modifié en dehors du scope de la classe. Soit on considère que c'est bien un effet de bord comme l'attribut **age** est définie en dehors de la méthode et que **birthday** n'est pas un *mutateur*.

Un *mutateur* ou *setter* en programmation orientée objet est une méthode dont le rôle est de modifier un objet en assignant une nouvelle valeur à un des champs de cet objet. Champ devant être explicité dans le nom du *mutateur*.

Ici **birthday** n'est pas un *mutateur* or il modifie l'attribut **age**. La méthode contient donc un effet de bord.

Le problème engendré par les effets de bord est qu'il est impossible pour l'utilisateur de la méthode de savoir ce qu'elle fait à moins de regarder le code de la méthode.

Dans *It's probably time to stop recommending Clean Code* par **Qntm**, l'auteur écrit en réponse à un code similaire écrit par **Robert C. Martin**, l'auteur de *Clean Architecture* :

At this point you might reason that maybe Martin's definition of "side effect" doesn't include member variables of the object whose method we just called. If we take this definition, then the five member variables [...] are implicitly passed to every private method call, and they are considered fair game; any private method is free to do anything it likes to any of these variables.

It's probably time to stop recommending Clean Code, Qntm, 2020

Il ajoute ensuite une citation de Robert C. Martin

Side effects are lies. Your function promises to do one thing, but it also does other hidden things. Sometimes it will make unexpected changes to the variables of its own class. Sometimes it will make them to the parameters passed into the function or to system globals. In either case they are devious and damaging mistruths that often result in strange temporal couplings and order dependencies.

Robert C. Martin, 2016

Il existe donc une dualité concernant les attributs privés d'un objet. D'un côté certains considèrent qu'ils peuvent être modifiés et d'autres au contraire considèrent qu'ils doivent être constants et que toute modification devrait entraîner la création d'un nouvel objet.

Architecture hexagonale

Nous avons abordé de nombreux éléments concernant le code tel qu'il est écrit, cependant une grande partie de la réflexion autour de la création d'une base de code concerne son architecture.

Le concept d'architecture hexagonal provient l'article *Hexagonal architecture* écrit en 2005 par **Alistair Cockburn**. Il commence par décrire le problème initial à savoir le déplacement de code métier dans des parties de l'application qui ne devraient

pas en avoir comme l'interface utilisateur. À cette solution, l'auteur propose d'isoler toute la logique métier d'une application du reste des éléments d'une application.

Architecture

La partie contenant la logique métier de l'application est appelée "Application" dans l'article mais le terme de "noyau de l'application" ("Application Core") existe aussi dans d'autres articles et évite ainsi une confusion entre l'application au global et l'application au sens de logique métier. Ce noyau de l'application communique avec le reste de l'application via des interfaces appelées "Ports". Chaque port peut être implémenté par des "Adapter".

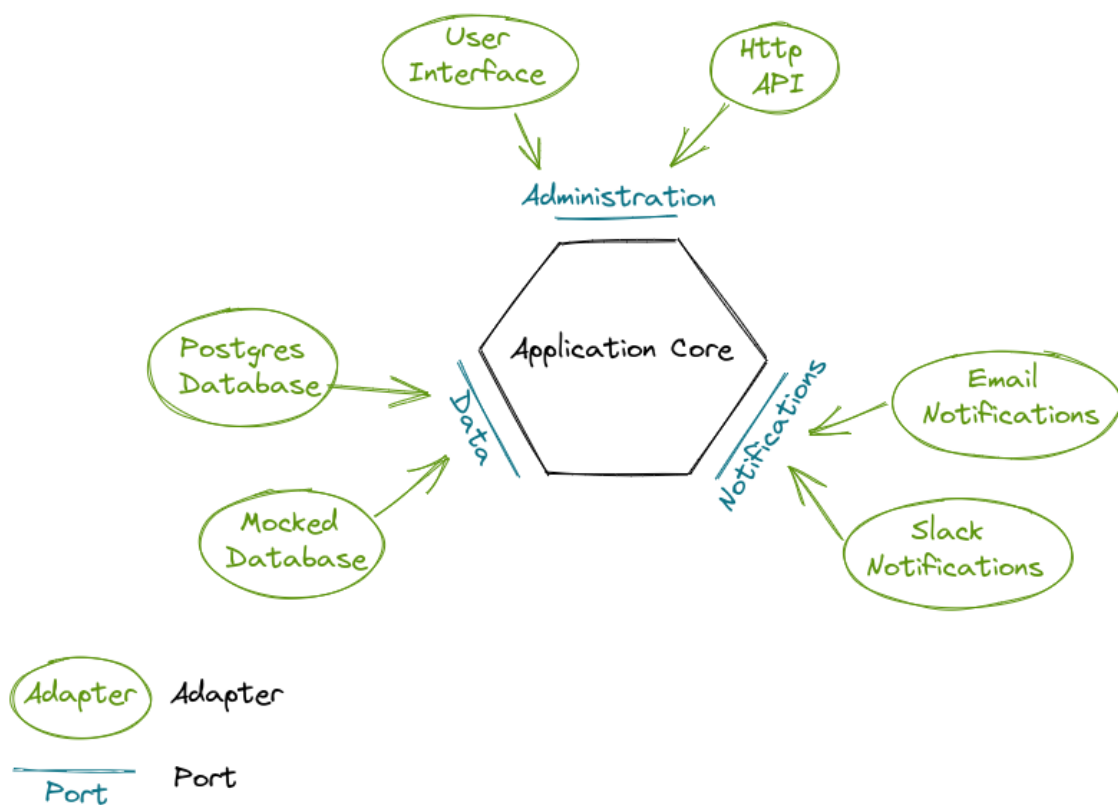


Fig.3 Exemple d'application utilisant une architecture hexagonale

Dans la figure ci-dessus (fig.3), notre application est composée d'un noyau contenant la logique métier de notre application, de trois ports concernant une couche de gestion de la donnée, des notifications et de l'administration de l'application. Chaque

port a ici deux adapters implémentant la même interface. Ces adapters sont interchangeables car leurs comportements ne sont pas connus du noyau de l'application.

Adapters

Le terme d'"Adapter" n'est pas anodin, il fait directement référence au design pattern du même nom.

“Convert the interface of a class into another interface clients expect.” The ports-and-adapters pattern is a particular use of the “Adapter” pattern.

Hexagonal architecture, Alistair Cockburn, 2005

D'après l'article concernant les Adapters sur le site Refactoring Guru, il décrit un objet dont le rôle est de faire communiquer deux interfaces incompatibles. L'exemple évoqué dans l'article présente une situation dans laquelle l'application récupère des données en XML et doit les fournir à une librairie sous format JSON. Les langages JSON et XML n'étant pas compatibles, il est nécessaire d'effectuer une conversion de l'XML vers le JSON, c'est le rôle de l'Adapter.

Dans le contexte d'une architecture hexagonale, l'Adapter va interfacer le service externe avec le Port correspondant. Par exemple, interfacer une base de données Postgres à un Port dédié à la gestion de la donnée.

Inspirations

L'article d'**Alistair Cockburn** peut directement être mis en parallèle avec les principes SOLID décrits par entre autres **Robert C. Martin** et notamment le principe d'inversion de dépendance. En effet ce principe est cité par l'auteur dans les références et il ajoute même une courte partie dédiée à expliquer le lien

Bob Martin's Dependency Inversion Principle (also called Dependency Injection by Martin Fowler) states that “High-level modules should not depend on

low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.” The “Dependency Injection” pattern by Martin Fowler gives some implementations. These show how to create swappable secondary actor adapters.

Hexagonal architecture, Alistair Cockburn, 2005

Dans cette citation, l'auteur explique que l'utilisation du principe d'inversion de dépendance permet de créer des Adapters remplaçables dans notre application.

Au-delà du code

Nous avons abordé différentes méthodes et points d'attention concernant les bonnes pratiques d'écriture de code dans la maintenabilité d'un projet applicatif cependant la maintenabilité d'un projet ne s'y limite pas.

You build it, you run it

Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.

Werner Vogels, 2006

C'est lors d'une interview de **Werner Vogles en 2006**, CTO d' Amazon que né l'expression "You build it, you run it", un modèle de développement et de maintenance

d'application dans lequel les développeur de l'application sont les mêmes développeurs qui la maintiennent (*Is 'you build it, you run it' living up to the hype?*, **Atlassian**). Le développement d'une application est souvent composé de deux parties, la partie développement ou le "build" et la partie maintenance ou le "run".

La notion de maintenance d'application peut être découpée en deux parties. D'un côté la maintenance applicative et d'un autre la partie opérationnelle comportant entre autres la partie déploiements de l'application et santé de l'application en production. Ici le model "You build it, you run it" concerne essentiellement la partie opérationnelle de la maintenance.

L'avantage de cette méthode est que les développeurs étant fortement impliqués dans la maintenance de l'application et notamment pendant le moment d'astreinte, ils font plus attention à la robustesse de l'application. Le développeur ayant créé l'application est de plus la personne la plus à même de résoudre ses problèmes en production. Et à l'inverse avoir la connaissance sur la méthode de déploiement de l'application permet d'anticiper dès la conception les problématiques éventuelles de production.

Cependant l'application de ce modèle est parfois complexe dans certaines équipes car cela implique de changer leurs structures et les modes de communications. (*Is 'you build it, you run it' living up to the hype?*, **Atlassian**).

Normes et définitions

La *norme ISO 25010* définit la maintenabilité comme étant le degré d'efficacité et de rendement dans l'évolution d'un produit ou d'un système dans un objectif d'amélioration, de correction ou d'adaptation aux changements d'environnement, et de besoin. Cinq éléments sont listés pour caractériser de la maintenabilité :

- **La modularité** : capacité d'un système ou d'un programme à être composé de composants de tel sorte que des changements apportés à un composant aient un impact minimal sur les autres composants.
- **La réutilisabilité** : capacité d'un élément à être utilisable dans plus d'un système
- **L'analysabilité** : degré d'efficacité et de rendement
 - de l'analyse de l'impact d'une modification sur une ou plusieurs parties d'un système.
 - du diagnostic des causes des problèmes rencontrés par le système.
 - de l'identification des éléments de l'application à modifier.
- **La modifiabilité** : capacité d'un produit ou d'un système à être aisément modifiable sans introduire de déficiences ou de régressions
- **La testabilité** : aisance à établir des cas de test d'un système et à leur mise en place

Cette norme ISO répond aux attentes déjà évoquées dans les parties précédentes, mais il reste à voir la problématique suivante : Quels éléments permettent de favoriser la maintenabilité à long terme d'une application ?

En effet, si la norme fixe un cadre, la gestion des détails permet une plus grande latitude au professionnel. Il sera donc de la responsabilité du professionnel de s'assurer que son application est pérenne sur le long terme, et surtout qu'elle puisse être reprise par d'autres personnes sans avoir besoin d'analyser toutes les lignes du code ou d'avoir connaissance de l'historique du projet.

MÉTHODOLOGIE DE RECUEIL DES DONNÉES

Une des problématique engendrée par les questions de maintenance est la difficulté d'avoir un résultat chiffré de la qualité de maintenance d'une application. Certains outils comme Sonar Cloud proposent une note de maintenabilité en se basant sur les "Code Smells" (les mauvaises pratiques identifiées par l'outil) et le temps nécessaire à leurs résolutions. Cependant la notion de "Code Smell" a été ajoutée arbitrairement par l'outil et certains cas sont assez largement débatables.

Il est donc complexe d'obtenir une analyse quantitative de ce qui favorise la maintenabilité d'une application. Je me suis donc dirigé vers une approche qualitative avec des entretiens destinés à établir des éléments récurrents entre différentes équipes ayant favorisé ou non la maintenance de leur application.

Cible des entretiens

J'ai privilégié des profils de Leader Techniques ayant pu travailler dans différentes équipes et différentes entreprises afin qu'ils puissent voir un maximum de projets et d'organisations différentes et donc avoir un bon recul sur le sujet. J'ai également privilégié des consultants en ESN pour ces mêmes raisons. Étant moi-même en ESN, j'ai pu profiter de mes connaissances afin d'obtenir les entretiens.

Déroulé des entretiens

Introduction

Quels sont pour vous les éléments les plus importants quand vous développez/travaillez sur une application ?

Afin d'introduire l'entretien, une première question générale est posée afin de démarrer le questionnement et de noter les différents points qui pourront être utilisés par la suite.

Les applications dites legacy

Les applications legacy (ou héritées) sont des applications informatiques n'étant plus destinées à être utilisées dans un système soit suite à une perte d'un besoin soit à la suite d'un remplacement.

Le legacy faisant partie intégrante du métier de nombreux développeurs et étant le signe de la fin de maintenance d'une application, il est souvent lié à un problème dans cette dite maintenance. Il est donc intéressant d'en tirer des leçons à appliquer ou non sur de nouveaux projets afin qu'ils soient utilisés à plus long terme.

Est ce que vous avez en tête une expérience marquante en lien avec la maintenance d'une application ?

Ici, il est question d'ouvrir cette partie sur le legacy en évoquant un événement marquant en lien avec la maintenance d'une application et de commencer à s'interroger ce qui aurait pu être différent pour améliorer l'efficacité de cette maintenance.

Quand vous travaillez sur de vieilles applications ou que vous reprenez le code d'une autre équipe, quels éléments ont tendance à complexifier votre travail de reprise ? Quels éléments auriez-vous mis en place pendant le développement afin d'éviter ces problèmes ?

Cette question permet d'ouvrir la discussion sur des éléments récurrents retrouvés dans du code legacy afin de déterminer leur impact sur la maintenabilité et des moyens de les éviter si besoin.

Clean Code

Connaissez-vous Clean-Code de Robert C. Martin ?

Cette question vise à établir si l'interlocuteur est sensibilisé aux ouvrages de Robert C. Martin qui sont très largement cités sur le sujet. En cas de réponse négative, nous pouvons directement passer à la question suivante.

Agilité

Quels corps de métiers sont selon vous responsables de la maintenance d'une application ?

Cette question permet d'introduire la partie agile dans laquelle le rôle des membres non-développeurs de l'équipe sont impliqués. J'attends des résultats comme Développeur, Quality Analyst, Product Owner ou DevOps mais c'est aussi l'occasion d'explorer le rôle d'autres métiers.

*Êtes-vous familier des méthodes agiles ? (Framework Scrum, Kanban)
Comment se déroule habituellement la gestion de projet dans vos équipes ?
Pensez-vous que les méthodes agiles sont adaptées à la maintenance d'une application ? Pourquoi ?*

Ces questions permettent d'établir un profil type d'équipe et d'ouvrir la discussion sur les limitations éventuelles de l'organisation sur la maintenance applicative ou au contraire des atouts qu'elle peut apporter.

You build it, you run it

Connaissez-vous le model "You build it, you run it" ? Si oui, l'appliquez-vous ?

Cette partie de l'interview se concentre sur la méthodologie You build it, you run it et le rôle des développeurs dans la maintenance applicative. Bien que l'on puisse différencier la maintenance applicative concernant les évolutions des applications de la maintenance opérationnelle qui sera plus en lien avec le suivi et le monitoring de l'application en production; ces deux parties sont très souvent liées.

En effet, un comportement non-souhaité présent dans une application peut avoir un impact sur son comportement en production nécessitant plus de maintenance opérationnelle. Une application bien maintenue permet de réduire ces événements.

Dans le cas où la personne interviewée ne connaîtrait pas le sujet, je lui présente en décrivant le système dans lequel l'équipe de développement d'une application est aussi responsable de son état en production.

Que pensez-vous de l'implication des développeurs dans les phases de maintenance applicative et opérationnelle d'un projet ?

Cette question permet d'avoir un retour d'expérience pour cette thématique et d'ouvrir la discussion sur le sujet.

ANALYSE DES RÉSULTATS

Deux interviews ont été réalisées dans le cadre de ce mémoire suivant le protocole exprimé précédemment. La première avec Sylvain Lavazais, Technical Leader chez Davidson Consulting, il a beaucoup d'expérience en développement d'application, notamment autour de l'écosystème JAVA. Le deuxième avec Jules Spicht, également Technical Leader à Davidson Consulting.

La documentation

La documentation a été évoquée à multiple reprise dans les deux interviews. Présenté comme point central dès l'introduction. Elle permet aussi de communiquer sur le projet de permettre la passation aux futurs membres de l'équipe. Jules Spicht précise toutefois que la documentation peut se trouver aussi bien dans le projet qu'à l'extérieur en fonction de la pertinence et de la cible de l'information documentée.

L'équipe

Une partie importante des interviews concerne la part de la maintenance applicative dans la vie de l'équipe. Sylvain Lavazais souligne qu'une des sources de ce qui rend un projet "legacy" est parfois le manque d'accompagnement des jeunes développeurs sur un projet. Dans les deux cas, chaque membre de l'équipe est considéré comme important dans le travail de maintenance. Que ce soit le Product Owner (ou PO), comme expliqué par Sylvain Lavazais par exemple, qui a pour charge d'identifier les sources de problèmes et de partager au mieux la vision du projet à l'équipe afin d'éviter les erreurs dues à l'incompréhension des demandées par les développeurs. Ou bien les développeurs qui concrétisent la vision du Product Owner à travers la réalisation des différentes tâches du projet.

Le rôle du QA, "Quality Analyst" ou "Quality Agent" en fonction des sources, a été beaucoup mentionné par Sylvain Lavazais au cours de son interview. Ce corps de métier est très présent dans les grandes entreprises, notamment Décathlon qu'il cite. Le rôle du QA est d'assurer la qualité autour d'une application, d'après Sylvain Lavazais, les équipes ont tendance à trop se reposer sur les QA alors que leur travail pourrait être

effectué plus efficacement par les développeurs. Ainsi, ce rôle est plutôt présenté comme celui d'un accompagnant de début de projet dont la finalité est de disparaître, une fois l'équipe autonome niveau qualité.

L'agilité comme point central de la gestion de projet informatique moderne, elle a bien sûr été mentionnée par les interviewés. Sylvain Lavazais décrit les deux frameworks principaux de l'agilité à savoir le Scrum et le Kanban mais définit le Kanban comme plus adapté à la maintenance applicative. Dans la vision Scrum, toute l'activité de l'équipe est découpée en périodes de durée fixe appelées "sprint" et au début de chaque sprint on définit les tâches à accomplir. À l'inverse la vision Kanban est de prendre les tâches au fil de l'eau sans avoir cette restriction du sprint. Ceci implique que les différentes tâches sont proposées avec notamment l'utilisation d'une "Fastlane", regroupant le suivi des tâches les plus importantes du projet comme la résolution de failles de sécurité. Le framework Kanban permet plus de réactivité que dans le Scrum ce qui permet d'identifier plus rapidement les problèmes et donc de les résoudre plus rapidement. Néanmoins, le Scrum reste plus répandu que le Kanban dans la plupart des équipes.

Maintenance opérationnelle et maintenance applicative

Étant deux sujets extrêmement liés, il a été demandé aux interviewés de parler du lien entre maintenance opérationnelle et maintenance applicatives. À plusieurs reprises dans les entretiens, l'importance du déploiement continu a été demandée. Cela fait partie de la "CI/CD" (Continuous Integration, Continuous deployment) qui décrit toute l'automatisation entre le moment où le code a été envoyé vers un gestionnaire de version comme Git au moment où il est déployé en production. Le déploiement continu est d'ailleurs un des premiers éléments évoqué par Jules Spicht dans son interview.

Sylvain Lavazais a parlé du programme "Accelerate" créé par Google pour identifier les écarts de performance entre différentes équipes via un questionnaire. Ce programme se base essentiellement sur le temps nécessaire au développement d'une fonctionnalité ainsi que du temps de résolution d'un problème présent en production. Le déploiement continu permet de grandement favoriser ces deux points en réduisant au

maximum le temps entre la résolution d'une tâche et son déploiement en production, permettant ainsi aux équipes de déployer leurs applications plusieurs fois par semaine voire même par jour.

La méthodologie "You build it, you run it" décrit un système dans lequel les développeurs s'occupent aussi de la maintenance opérationnelle. Cela peut se faire par exemple par la mise en place d'un déploiement continu comme vu précédemment. Sylvain Lavazais décrit cette méthodologie comme plus stimulante pour les développeurs qui sont ainsi forcés à s'impliquer dans ce que va devenir leur code en production. Jules Spicht ajoute à ces éléments que les problèmes en production sont plus faciles à résoudre, car les développeurs s'occupant de la production sont les mêmes qui pourront résoudre le problème, qu'il soit applicatif ou non. Même si Sylvain Lavazais comme Jules Spicht déclarent que ce n'est pas appliqué dans la majorité des entreprises du Nord de la France, c'est tout de même appliqué dans de grosses entreprises comme Adéo ou Norauto. Cette méthodologie est aussi intéressante dans l'autre sens car les développeurs connaissent mieux que quiconque leurs applications et donc sont plus à même de connaître leurs besoins en production comme décrit par Jules Spicht.

Une rigueur automatisée

La notion de rigueur s'est imposée comme point central de la maintenabilité applicative. Les interviewés ont décrit cette rigueur comme centrale afin d'éviter que l'application ne devienne "legacy". Les points les plus importants évoqués par les deux participants sont:

- Les tests et plus généralement la couverture de code permettant à la fois d'éviter les régressions dans le code mais aussi de simplifier la reprise du code par les nouveaux développeurs car une lecture des tests unitaires ou End-to-end d'une application permet souvent de mieux comprendre le comportement et le rôle des différents éléments de code.
- L'utilisation d'un outils de linting (comme Sonar Cube) pour assurer le respect des convention et de maintenir une homogénéité dans les règles utilisées dans l'écriture du code

- Le suivi d'un guide comme Clean Code, à utiliser comme référence pour trouver des pistes de résolution de problème et d'homogénéiser les bonnes pratiques de développement.
- Utiliser un outil de versioning comme Git
- La mise en place d'une CI/CD si possible
- Penser à l'architecture de l'application et savoir comment l'application va se mettre à l'échelle si elle est fortement utilisée.

À ces différents outils, Sylvains Lavazais en ajoute :

- Le besoin d'une convention de commit. Cela permet d'avoir un historique de commit plus lisible et d'ainsi pouvoir automatiser la gestion de version de l'application, toujours en lien avec le programme Accelerate, évoqué dans la partie sur "You build-it, you run it".
- La mise en place d'une Definition of Done permettant de définir les éléments nécessaires à la validation d'une tâche.
- La mise en place d'une Definition of Ready définissant les éléments nécessaires à valider pour qu'une tâche puisse être effectuée par un développeur.
- La mise en place d'une convention de co-travail permettant de définir comment les membres de l'équipe vont travailler ensemble. Par exemple, si une tâche ne doit être effectuée que par une seule personne ou si du Peer-Programming peut être effectué dans certains cas.

Enfin Jules Spicht ajoute d'autres pratiques :

- Avoir un référent technique connaissant bien la stack technique du projet, permettant ainsi de mieux anticiper le travail nécessaire aux différentes tâches.
- Avoir des outils de monitoring afin de contrôler l'application en production.

Bien que ces pratiques permettent de favoriser la rigueur, il ne faut pas aller dans l'extrême. Par exemple, Jules Spicht prend l'exemple de "Git Hooks", ce sont des scripts qui se lancent à chaque "Commit" sur l'ordinateur du développeur. Ils servent généralement à faire une vérification du bon fonctionnement des tests, de l'application du linter ou de la forme du message de commit. Le problème c'est que le développeur

peut se sentir frustré, de ne pas pouvoir faire ce qu'il veut en local. Il faut donc laisser une marge de manœuvre pour que le développeur soit rigoureux sans pour autant affecter ses performances.

Sylvain Lavazais ajoute la notion de "rigueur automatisée". La rigueur ne devrait pas être contrôlée par les autres développeurs mais par des automatisations. Ainsi, le développeur peut avoir un résultat plus fiable plus rapidement et donc être proactif si besoin. Cette automatisation peut se faire avec des outils de CI/CD comme GitHub Actions ou bien CircleCI.

Conception de l'application

Un point revenu à multiples reprises dans les deux interviews est l'importance de l'architecture des applications. Sylvain Lavazais parle d'homogénéité dans les choix d'architecture d'un projet. Jules Spicht considère que l'architecture est à réfléchir avant même le début des développements. Pris au sens large dans son entretien, l'architecture désigne ici aussi bien la structure de l'application, que la manière dont elle va être déployée mais aussi la façon dont elle va communiquer avec les applications. Il précise l'importance de juger la pertinence d'une application au global et de son intégration avec les autres applications de son écosystème afin d'éviter les redites. Cette question peut parfois amener à se poser la question du découpage de l'application en plusieurs micro-services la rendant ainsi accessible aux autres applications ayant besoin de ses fonctionnalités. Il a également parlé de la manière dont l'application doit être mise à l'échelle, en effet une application recevant beaucoup de requête se doit de se mettre à l'échelle et si l'architecture ne permet pas de le faire assez efficacement, des développements supplémentaires vont être nécessaires et donc affecter la maintenance applicative.

DISCUSSION

Les résultats précédemment présentés permettent d'identifier divers domaines permettant de répondre à la problématique.

La documentation

Largement mentionnée durant les entretiens, la documentation d'un projet fait partie des piliers les plus importants dans la réalisation d'une application. Il est important de distinguer les différents types de documentation et notamment la documentation utilisateur de la documentation technique car elles doivent être pensées différemment. Dans le contexte de la maintenance applicative, la documentation technique est celle qui aura le plus d'impact. Elle doit permettre à un développeur ayant peu ou pas de connaissance sur le projet de commencer à développer dans de bonnes conditions en comprenant les choix architecturaux et de technologies qui ont été réalisés. De plus, cette documentation peut recenser les schémas d'architecture de l'application une fois déployée, permettant ainsi de la même manière de faciliter le déploiement et donc de gagner du temps.

Comme expliqué par Jules Spicht durant son entretien, cette documentation peut aussi bien se trouver dans le code source qu'à l'extérieur. J'ajouterais néanmoins l'importance d'avoir une homogénéité de gestion de la documentation entre les différents projets d'une organisation. Cela permet de repérer plus les similarités et différences entre les différents projets et de profiter de l'expérience acquise durant le développement d'un autre projet sans nécessairement avoir de développeurs en commun. Ainsi, si une équipe développe une fonctionnalité et la documente, une autre équipe souhaitant développer une fonctionnalité similaire pourra s'inspirer de l'expérience de la première équipe pour ne pas reproduire les mêmes erreurs voire apporter des suggestions ou des questionnements de cette équipe. C'est d'autant plus vrai concernant la documentation de l'architecture de l'application déployée car dans une organisation, une homogénéité des méthodes de déploiement est souvent privilégiée afin de simplifier la maintenance opérationnelle.

L'équipe

L'équipe de développement et son organisation a également été un sujet mis en évidence. Afin de créer une application facilement maintenable, il est primordial que son organisation favorise ces principes :

- Favoriser la communication entre le Product Owner et le reste de l'équipe. Un des rôles principaux du Product Owner, comme le dit Sylvain Lavazais dans son interview, est de concrétiser le besoin client et de le transmettre au reste de l'équipe de développement. Si dans cette transmission, le besoin est altéré, la production de l'équipe ne sera pas en accord avec la volonté du Product Owner et du Client, des bugs peuvent alors être créés nécessitant donc une plus grande maintenance.
- Donner du temps aux développeurs. Développer en favorisant la qualité de l'application est un processus chronophage ne devant pas être négligé. Il est hélas courant de voir des projets où l'on considère une tâche terminée à partir du moment où elle est remplie fonctionnellement. Afin de favoriser la maintenance applicative, une tâche doit être prise dans son intégralité. Cela comporte sa réalisation fonctionnelle mais aussi l'application de toutes les contraintes que l'équipe a définie (dans une Definition Of Ready, par exemple)
- Penser l'organisation sur la durée. La maintenance d'une application est un élément à prendre en compte dès le début du projet pour des effets souvent visibles après beaucoup de temps. Il est de l'affaire de chaque membre de l'équipe de penser au futur de l'application et une application maintenable est une application qui a moins de chance d'être refaite de zéro dans quelques années.

Comme évoqué plus haut, la mise en place d'une Definition Of Done est un bon moyen de définir les éléments nécessaires pour considérer une tâche comme terminée. Cette charte peut comprendre aussi bien des éléments de qualité comme la réalisation de tests unitaires mais aussi des tâches de documentation. Il est important qu'elle soit connue, comprise et acceptée par l'ensemble de l'équipe afin de favoriser sa mise en place et son application. Il est cependant important, comme Jules Spicht et Sylvain Lavazais le mentionnent, de ne pas trop restreindre les développeurs. Les éléments

composants la Definition Of Done doivent laisser suffisamment de marge pour les développeurs afin qu'ils ne se sentent pas frustrés et qu'ils puissent être créatifs. L'exemple des hooks de pre-commit mentionné par Jules Spicht représente particulièrement cette situation dans laquelle, étant trop contraints, les développeurs désactivent les outils censés améliorer leur travail.

L'agilité au centre du développement moderne, c'est un outil très utile pour favoriser la maintenance de nos applications. Les deux frameworks agiles les plus présents sont le Scrum et le Kanban. Même si le Scrum est bien plus répandu et est le framework de référence quand il est question d'agilité, le framework Kanban semble être bien plus adapté pour favoriser une bonne maintenance applicative. En effet, le Scrum oriente le développement autour de différents sprints d'un temps défini. Au début de ce sprint, la composition de l'équipe de développement ainsi que les tâches sont définies.

Les développeurs sont donc limités à cette période de temps pour effectuer leurs tâches et bien souvent les rôles déterminant la finalité d'une tâche n'ont pas la vision technique du produit et donc se basent sur la partie purement fonctionnelle. Un cas que j'observe régulièrement est celui où un développeur doit développer une fonctionnalité, la fin du sprint arrive, il faut donc rendre compte de cette tâche. Étant fonctionnellement réalisée, elle est validée et déployée mais la partie test n'a pas été réalisée. Nous sommes donc dans une situation où la fonctionnalité est déployée mais n'a pas un niveau de qualité suffisant pour être facilement maintenable.

Par la suite, il y a trois possibilités :

- Une tâche dédiée est ajoutée au sprint suivant mais elle n'apporte pas de valeur ajoutée.
- L'équipe décide de ne pas réaliser ces tests dans l'immédiat et de le faire si un développeur n'a plus de tâche à la suite d'un sprint.
- Les tests ne seront pas réalisés tant que le code de la fonctionnalité ne sera pas de nouveau repris.

Pour pallier ça, l'équipe peut avoir la rigueur de ne pas déclarer la tâche comme terminée et de la reporter au second sprint mais cela demande une rigueur quasi-utopique aux équipes car la tâche n'apporte pas plus de valeur ajoutée au produit à la suite du sprint suivant. Un argument souvent évoqué suite à cette situation est la meilleure prévision du temps nécessaire à la réalisation des tâches. Cependant, l'agilité doit répondre aux imprévus d'un projet, hors le moindre imprévu peut décaler le temps de travail d'un développeur qui n'aura donc pas le temps de terminer sa tâche.

Le framework Kanban en revanche n'a pas cette notion de sprint. L'accent n'est plus sur le temps mais sur la priorisation des tâches. Ainsi, un développeur peut consacrer tout le temps nécessaire à sa tâche. Pour la même raison, le kanban est bien plus propice au programme Accelerate qui incite les équipes à déployer leurs applications en production plus souvent. Le framework Scrum impliquant un déploiement en production en fin de sprint, il est restreint par rapport au framework Kanban qui peut permettre un déploiement à la fin de chaque tâche.

Bien sûr, comme précisé par Sylvain Lavazais, il ne faut pas utiliser les frameworks agiles tel quel. Les préceptes suivis par l'équipe doivent s'adapter à leurs besoins. C'est la raison pour laquelle il est souvent question de ScrumBan, mélange entre Scrum et Kanban, pour parler de la gestion agile de certaines équipes. Cette capacité d'adaptation des équipes est clef pour favoriser sa productivité. Chaque équipe a ses problématiques, ses contraintes et sa maturation, c'est pourquoi il n'est pas pertinent que toutes les équipes d'une organisation soit organisée de la même manière point de vue agilité.

Maintenance Opérationnelle

La maintenance opérationnelle et la maintenance applicative sont des thématiques extrêmement proches. En effet, une application difficilement maintenable aura une plus grande chance d'avoir des bugs en production et donc d'impacter la maintenance opérationnelle. Une manière efficace de sensibiliser les développeurs aux enjeux de la maintenance applicative est selon moi l'application de la méthodologie

“You build it, you run it». En suivant cette méthodologie, l’équipe de développement est aussi impliquée dans la maintenance opérationnelle de l’application et notamment dans les phases d’astreinte. C’est la métaphore du bâton et de la carotte, une application difficilement maintenable applicativement créera potentiellement plus de problèmes en production et donc plus de charge aux développeurs. En faisant plus attention à la maintenabilité de l’application, le développeur aura moins de charge de maintenance opérationnelle et sera donc plus disponible.

Une rigueur automatisée

Sûrement le point le plus important de la maintenance d’une application, la rigueur est ce qui permettra de mettre en place tous les éléments favorisant la maintenance et assurera le suivi. Cette rigueur doit permettre de garantir un niveau de qualité suffisant.

Pour ce faire, Il existe de nombreux outils. Premièrement la rédaction de tests sur l’application. Aussi bien unitaires pour tester les composants de l’application indépendamment du reste de l’application. Mais aussi des tests d’intégration vérifiant le comportement de l’application entière en l’isolant du reste de son écosystème. Et enfin les plus importants, les tests End-To-End (ou e2e) où l’application tourne dans les mêmes conditions que l’environnement de production et où le comportement de l’utilisateur est simulé.

Pour garantir une rédaction efficace des tests unitaires, la méthodologie du Test Driven Development (TDD) peut être appliquée. Cette méthodologie contraint les développeurs à écrire leurs tests avant de commencer à développer les fonctionnalités. C’est une méthodologie assez largement détaillée dans *Clean Code* de **Robert C. Martin** mais elle n’impacte pas en tant que telle la maintenabilité de l’application. En effet, si le TDD n’est pas appliqué dans une équipe mais que les tests sont effectués après le développement des fonctionnalités, toujours en étant inclus dans les tâches, le résultat côté maintenance est identique. Il n’est d’ailleurs pas rare de voir des équipes dans lesquelles le Test Driven Development est appliqué par seulement une partie des développeurs, ça n’a alors aucun impact sur le reste du projet.

Afin d'éviter des problèmes dues à de mauvaises pratiques de code, des outils appelé Linter peuvent être mis en place. Ils ont d'ailleurs été largement mentionnés dans les interviews, en particulier Sonar Cloud qui est fortement implanté dans la région Lilloise. Un Linter est un outil qui va réaliser une analyse de code et remonter les lignes de codes ne respectant pas des règles définies dans la configuration de l'outil. Par exemple, une vérification du bon nommage des variables peut être appliquée. Il existe de nombreux Linter comme Sonar Cloud ou ESLint (pour le JavaScript), certains mêmes assemblent plusieurs Linter pour pouvoir assimiler une plus grande quantité de règles comme Super Linter. En plus de ces outils, il est utile que les développeurs partagent une base de références quant aux bonnes pratiques de développement. C'est là que peuvent intervenir des ouvrages tels que *Clean Code* de **Robert C. Martin** par exemple. Ainsi, devant une problématique, les développeurs ont une référence similaire, permettant d'accomplir leurs tâches de façon plus homogène.

Ce qui rend réellement cette rigueur efficace c'est son automatisation. Nous avons discuté plus haut de la Definition Of Done et de son rôle dans une équipe de développement. Afin d'assurer l'application des "guidelines" dictées par cette charte, leur application peut être vérifiée. L'idée est simplement de «cocher des cases» concernant la position de la tâche par rapport à cette Definition Of Done. Très souvent cela passe par une CI/CD (Continuous Integration, Continuous Deployment). Un CI/CD est un outil qui va la plupart du temps consister en une suite d'étapes liées ou non entre elles s'activant suite à une action du gestionnaire de version utilisé. Typiquement lors du push d'un commit sur un répertoire Git. Cette suite d'action va notamment comporter l'ensemble des vérifications à effectuer comme la validité des tests unitaires, le respect des bonnes pratiques de développement ou la non-régression de la fonctionnalité. Mais aussi la manière dont l'application va être déployée, évitant ainsi toute erreur humaine lors du déploiement de l'application.

Conception de l'application

La maintenance d'une application se prépare dès sa conception de l'application. En effet, une mauvaise décision à ce moment peut drastiquement augmenter la charge

de maintenance à l'avenir. Un élément évoqué par Jules Spicht pendant son interview concerne la reprise d'un code Legacy avec des technologies dépréciées ou plus maintenues. L'informatique se développant exponentiellement, il est fréquent qu'une technologie soit mise de côté pour une autre soit par la communauté, soit par les contributeurs. Une application sur une technologie vieillissante risque à terme de ne plus pouvoir être mise à jour, que ce soit suite à une fin de support ou bien à la difficulté de trouver des développeurs utilisant cette technologie. Il est impossible de prédire avec certitude l'avenir d'une technologie tant les facteurs pouvant la faire monter ou non en popularité sont importants. Il est cependant intéressant de s'intéresser à la tendance d'utilisation de la technologie. Des sites web comme npmjs.com permettent de se renseigner sur l'utilisation d'une technologie via l'évolution de son nombre de téléchargement. Cela peut donner un indice sur l'avenir de la technologie et la capacité de trouver des développeurs y étant formés.

L'architecture de l'application au sens interne du code est également un facteur important à prendre en compte. Comme le dit Sylvain Lavazais dans son interview, il est important d'avoir une certaine homogénéité dans le code source d'une application. Son exemple dans lequel l'accès à la base de données d'une application est réalisée via deux méthodes différentes est un exemple typique. Il est fondamental qu'un type d'action soit toujours réalisé de la même manière dans une application. Cela permet de faciliter l'arrivée de nouveaux développeurs sur le projet et de simplifier les diagnostics en cas de bug.

Il est également important que l'application soit pensée dans son contexte et donc par rapport aux autres applications de son organisation. Par exemple, si elle doit aborder des fonctionnalités similaires à d'autres applications, il est sûrement plus pertinent d'utiliser cette application pour cette fonctionnalité. De la même manière, si deux applications appartenant à une même organisation partagent une même fonctionnalité, il peut alors être intéressant de séparer cette fonctionnalité dans une application dédiée et ainsi éviter la duplication de code..

CONCLUSION

La maintenance applicative fait partie intégrante du métier de développeur, ce n'est cependant pas toujours la tâche la plus simple et la plus agréable. Afin de la simplifier, il est intéressant de se pencher sur les éléments permettant de réduire ou de simplifier la charge de travail.

Nous l'avons vu précédemment, cela demande une rigueur importante impliquant un travail humain de formation et de sensibilisation dans un premier temps. L'humain est la principal concerné dans le travail de maintenance applicative et reste au coeur de la réalisation d'un projet applicatif. C'est à lui de prendre les décisions en se concentrant sur l'avenir de l'application. Dans le cas d'une méthodologie trop frustrante pour lui, il risque de la contourner et d'être contre-productif. C'est pourquoi il est important de penser à l'humain à chaque décision du projet.

Un travail au niveau de l'organisation et de la gestion de l'équipe est aussi important. L'agilité montante dans les entreprises a montré des évolutions quant aux problématiques de maintenance. L'organisation et la gestion de l'équipe doivent prendre en compte les problématiques de maintenance dès la conception de l'application. Rendre au plus simple l'intégration et la formation de nouveaux arrivants est capital et du temps doit être consacré afin que l'équipe de développement puisse rester productive et homogène dans sa manière de travailler.

Enfin, un travail d'automatisation est également nécessaire afin d'assurer le respect des bonnes pratiques et de la maintenabilité du code. L'automatisation permet de décharger l'équipe de développement d'une partie du travail de qualité et de renforcer la détection de régression entre les versions d'une application.

Les recherches effectuées dans le cadre de ce mémoire sont limitées à la région Lilloise. Toutefois, dans une autre région où la culture du travail diffère, les problématiques autour de la maintenance d'application pourraient être différentes.

Afin de compléter les différents aspects de ce mémoire, il serait pertinent d'étendre cette étude au niveau international.

BIBLIOGRAPHIE

Définition de l'Internaute : Langage de programmation

<https://www.linternaute.fr/dictionnaire/fr/definition/langage-de-programmation/#:~:text=Un%20langage%20de%20programmation%20est,informatiques%20qui%20applique nt%20ces%20algorithmes>

Programmation procédurale - Définition et Explications de Techno-Science.net

<https://www.techno-science.net/definition/11446.html>

Programmation orientée objet - Définition et Explications de Techno-Sciences.net

<https://www.techno-science.net/glossaire-definition/Programmation-orientee-objet.html>

Programmation fonctionnelle - Définition et Explications de Techno-Science.net

<https://www.techno-science.net/glossaire-definition/Programmation-fonctionnelle.html>

Clean Architecture: A Craftsman's Guide to Software Structure and Design, Robert C. Martin (2012)

Clean Code: A Handbook of Agile Software Craftsmanship, Robert C. Martin, 2008

Object Oriented Software Construction, Bertrand Meyer, Prentice Hall, 1988, chapitre 4

Do the SOLID principles apply to Functional Programming, Patricio Ferraggi - 2020

<https://dev.to/patferraggi/do-the-solid-principles-apply-to-functional-programming-56lm>

Ambler, Scott W. (1998). Process patterns: building large-scale systems using object technology

Explicit interface per class' antipattern, Marek Dec

<https://marekdec.wordpress.com/2011/12/06/explicit-interface-per-class-antipattern>

It's probably time to stop recommending Clean Code, Qntm, 2020

<https://qntm.org/clean>

Hexagonal architecture, Alistair Cockburn, 2005

<https://web.archive.org/web/20180822100852/alistair.cockburn.us/Hexagonal+architecture>

Adapter, Refactoring Guru

<https://refactoring.guru/design-patterns/adapter>

Interview de Werner Vogles en 2006

<https://queue.acm.org/detail.cfm?id=1142065>

Is ‘you build it, you run it’ living up to the hype?, Atlassian

<https://www.atlassian.com/incident-management/devops/you-built-it-you-run-it>

Norme ISO 25010

<https://www.iso.org/fr/standard/35733.html>

ANNEXES

Retranscription de l'interview avec Jules Spicht, Technical Leader à Davidson SI Nord

Je te propose de commencer par une petite intro, donc déjà merci beaucoup Jules. Je te propose de te présenter, ton parcours, les endroits où t'as eu l'occasion de travailler, ce genre de chose.

OK Ben écoute concernant mon parcours donc moi j'ai effectué mon stage de fin d'étude chez Davidson donc là aujourd'hui ça fait 4 ans que j'y suis et en fait j'ai démarré par une mission client au Btwin village. Donc à Villeneuve-d'Ascq pendant

3 ans dans l'équipe RFID tags donc, en fait la RFID chez Décathlon, on s'en sert aujourd'hui pour l'encaissement, pour la logistique, et cetera. Et moi je suis plus sur la partie logistique. Enfin, j'étais. Où nous en fait, on se servait de la RFID pour tracer pour la traçabilité, donc en fait être capable de dire, tiens, ce composant ou ce produit est parti de telle usine tel jour, telle heure. Jusqu'à quel magasin ? Et en fait, grâce à ça, on pouvait justement analyser la production en temps réel de

chaque usine. Donc y avait quand même des millions d'événements par jour, donc c'était un petit challenge côté Big data. Et aussi à l'époque, en fait, ça tournait sur un gros monolithe. Je reviendrai là-dessus plus tard. Et après ?

Par la suite du coup, moi je suis comme tu le sais, je suis revenu en interne chez Davidson pour effectuer des missions à mi-temps en audit. Chez nos clients donc. Bah là c'était Décathlon, Croix en l'occurrence, pour une équipe qui s'appelait My Game. Et en fait le but de ma Game, donc c'est un outil qui va offrir des KPI donc des indicateurs de performance sur les magasins. Des pour les chefs de rayon pour les propriétaires des magasins justement. Donc de façon à avoir tiens, est-ce que tel jour, j'avais assez d'employés ? Est-ce que c'est ça qui a fait baisser mon chiffre d'affaires ? Dans quel rayon ? Et cetera. Donc y a vraiment plein de croisés et le problème de cet appli ? Bah

voilà, c'est que c'était aussi en un gros monolithe fait n'importe comment ou les composants étaient réparties, c'est pareil, et cetera.

Et puis après par la suite, Ben je suis revenu sur le centre de service à plein temps où là je suis un peu plus en mode avant-vente donc une casquette un peu plus commerciale. Pour justement remporter les projets, aller chez les clients, présenter nos équipes, présenter nos compétences. Et voilà, après ramener les projets sur le Lab. Et voilà. Après donc encadrement classique des nouveaux arrivants, tech lead sur le choix des technos. Alors en ce moment plus trop. Enfin tu vois, j'ai moins la main dedans sachant que je fais plus de commercial.

Mais globalement, je pense que j'ai fait le tour sur mon parcours, ouais ouais donc

globalement j'ai eu un gros client, tu vois après, ça a été vraiment des petites missions au compte goutte.

OK, merci beaucoup. La problématique principale de mon mémoire c'est "Quels éléments permettent de favoriser la la maintenabilité d'une application à long terme ? L'idée, donc, on va le découper en plusieurs parties. Et on va commencer avec petite question pareil, un peu en suite de l'introduction. Quels éléments sont, pour toi, les plus importants quand tu vas développer une application ?

Bah les éléments les plus importants, c'est déjà le choix des technos. Parce que si tu te mets sur une techno qui est vieillissante et qui va plus être maintenu final, tu vas le payer plus tard hein ? Après, au niveau des tests aussi, c'est pareil si tu te mets à faire les tests plus tard, bah ta qualité. Déjà dès le début, elle va pas être dans une bonne dynamique. La documentation. Ce qu'il faut savoir que c'était sur un projet en plus, qui bouge beaucoup de personnes, et cetera, qui a un un petit turnover, ça peut aider et même pour plus tard. En fait quand il va y avoir des Breaking changes en fonction des versions majeures, et cetera.

Après t'as la question du déploiement. Est-ce que le déploiement est complètement bancal ? Est-ce que t'as un gitops qui tourne derrière des choses comme ça ? Et puis après, j'ai envie de te dire que dans l'équipe, ce soit bien rythmée. Donc bah ça je vais, je vais partir sur les méthodes agiles hein au final. Parce que tes méthodes agiles, elles vont te permettre déjà d'être en contact assez constant avec le client et en fonction de ce que tu dois développer toi au moment donc tu vas rendre ton développement un peu plus prédictible. Et t'es aussi souple en fait, tu peux voir dès le début ce qui va pas dans ton ton. Flux de développement où ton équipe.

*Merci beaucoup, je te propose de commencer une partie sur le Legacy.
Et donc, est-ce que t'as en tête une expérience marquante en lien avec la maintenance applicative d'une application legacy ?*

C'était dans ma première mission. Du coup, Bitwin donc en fait pendant 3 ans. Quand je suis arrivé en stage, il m'a refilé le bébé. Si tu veux donc ma première mission, c'était de refaire les tests. Déjà, on m'a dit, refais les tests. J'ai dit y a un gros problème, refaire les tests, 382 tests exactement. Je me souviens du nombre que c'était ignoble. Et en plus donc c'est un gros monolithe qui faisait tout. Il faisait la gestion des usines, la gestion des utilisateurs, la gestion des événements. Quand je te dis qu'on tapait le million d'événements et que, par exemple t'avais un pauvre endpoint HTTP qui prenait les événements, y a pas de flux Kafka des choses comme ça.

Donc voilà, c'était assez mal fait, c'était pas documenté y avait énormément de duplication de code. Donc en fait tu as compté arrivé sur Legacy ? Bah forcément t'as tout le temps des surprises de genre hein les tests ils se faisaient en plus sur une base de données réelles voire limite en prod.

C'était assez choquant aussi. Euh, côté sécurité y avait énormément de problèmes aussi. Ça veut dire que t'avais un pauvre fichier Properties Springboot, qui se baladait dans le repo avec les identifiants de prod super. Pas de Secret, rien

du tout. Et alors cerise sur le gâteau, côté déploiement, on devait nous-mêmes créer sur notre poste un war. Et le balancer sur un Tomcat, qui était déployé en bare-metal ni managé, ni rien.

Très intéressant. De façon générale Quand tu vas reprendre du code soit du Legacy ou même juste un code que tu n'as pas écrit, qu'est ce qui va avoir tendance à complexifier ta reprise du travail.

Tout bonnement, si le code est déjà mal structuré, tu vois si tu prends les principes du Clean Code les principes SOLID et cetera. Donc c'est à dire si ta classe fait 1000 lignes, si ton composant React en fait 400, que la logique est pas centralisée que le non des variables et m'a choisie...

Après le fait qu'il y ait des tests, ça peut aussi t'orienter vers les logiques métiers que l'application enfin, auquel l'application doit répondre. Donc ça, ça peut te faire comprendre en fait, à quoi sert l'application donc c'est très important, tu en es si on n'a pas, bah ça va complexifier. Si y a pas de documentation, tout simplement, si tout le code est là comme ça claqué en mode j'ai écrit mon code et hop au revoir tout simplement. Et après, qu'est-ce qui pourrait complexifier ?

Je vais te dire c'est des trucs bateau, mais si je connais pas la techno ou si c'est une techno très vieillissante. Je prends, je prends l'exemple des flux JMS, des flux EBS. Des choses comme ça, qui sont assez vieux.

Merci. Aujourd'hui tu démarres une application toute neuve. Avec un budget et un temps illimité. Qu'est-ce que tu mets en place pour que ce soit un minimum frustrant pour les dev et pour les nouveaux arrivants et que ton application soit durable dans le temps ?

Bah alors déjà tu démarres un projet from scratch, donc avoir un référent technique qui va s'y connaître assez pour faire un choix de techno assez judicieux et pour encadrer justement les nouveaux arrivants qui ont pas forcément ce genre de

formation et démarrer tout de suite avec des méthodes agiles. La gestion du projet, ça c'est quasi obligatoire. De toute manière, après démarrer toute la partie testing donc déjà là bootstrapper un peu soit des des tests de métier ou alors du end to end. Mettre en place un storybook, pourquoi pas ça, je le fais pas systématiquement. En fait, ça dépend la l'approche de dev qu'on va avoir. C'est du Component driven Development ou ou après si on fait du Behavior Driven Development des choses comme ça, tu vois ? Et après donc mettre en place pourquoi pas un socle de documentation. Mais ça c'est pas indispensable non plus. Quand j'entends socle de documentation, c'est par exemple un outil externe ou une page github dédiés. Voilà, tu vois ça, ça dépend vraiment. Parce que la documentation peut se trouver dans le code directement.

Et par contre, chose à faire et à prendre dès le début. C'est par exemple prendre le réflexe de faire des pull request. Tout simplement pour voir si on répond aux principes clean code, si c'est clair et cetera. Parce que si tout le monde travaille sur sa branche, et est tout seul son travail. Bah en fait, on va avoir des surprises les mois d'après. Parce que personne ne code de la même manière et personne ne comprend de la même manière, hein aussi hein, c'est pas forcément un souci de compétence. Mais voilà, il faut quand même uniformiser un minimum le développement. Voilà, et puis après côté déploiement, moi je ferai du déploiement continu, ça c'est sûr. Ce qu'aujourd'hui a rien de plus simple. Donc tel que tu gitops, hein, dès que tu vas aller sur ta branche dès que tu push, derrière ça va faire le travail en CI/CD sur ton cluster et cetera.

Et pourquoi pas mettre des outils de maintenance ? Enfin des de monitoring, pardon pour voir si ton application est résiliente. S'il y a des contraintes de performance à tel ou tel niveau ?

Et et ensuite mettre en place pourquoi pas de l'autoscaling si c'est vraiment sur une application qui nécessite d'adapter sa charge en fonction de la charge utilisateur. Puis je pense que j'ai fait le tour hein. Après globalement, si t'as ça t'as déjà un bon socle pour démarrer une application sereinement.

Un truc dont tu as parlé tout à l'heure, c'est la couverture de test. Pour toi, est-ce qu'utiliser un outil comme par exemple un sonar est intéressant pour les nouveaux projets ? Est-ce-que c'est pertinent ? Ou est-ce que au contraire, ça peut être frustrant ?

Ouais, c'est vrai que je l'ai pas mentionné, mais c'est quelque chose que l'on fait et que je fais aussi. C'est bah tu mets en place un sonar, un linter aussi. En local. Après, je sais qu'il y en a, ils aiment bien mettre aussi des des hooks, donc tels que le pré commit et cetera. Moi, j'ai mon avis là-dessus, ça peut être assez frustrant pour le développeur à chaque fois de faire un commit. Le Linter se lance. Autant que ça se fasse côté CI parce que ça peut bloquer et être frustrant. Chaque tu fais

tes commandes. Moi j'ai déjà vu des équipes qui avaient mis en place ça et le reste de leur équipe faisait sauter les hooks en fait.

Le hook peut être intéressant. Par exemple, je prends exemple d'un code Python ou autre quand on a des des formateurs et des idéaux différents. On peut faire appel à un formateur comme Black en Python qui va uniformiser justement le format de code pour tout le monde. Et comme ça on n'a jamais de surprise qu'on va reprendre le code, que ce soit sur les PR et cetera, on aura toujours le même style, même style de code donc ouais mettre un sonore c'est intéressant. Après ce qui est dommage c'est vrai sur le marché.

J'ai pas poussé la recherche non plus, mais je trouve que sonar devient un peu le standard mais y a pas assez de concurrence à ce niveau. Ou alors j'en ai vu en fait si j'en ai vu mes qui est très très cher. Voilà niveau abordable pour une petite équipe. C'est sonar, c'est bien l'impression veut partir sur des trucs plus poussés.

Je te propose de commencer une autre partie sur le clean code et le code que t'as mentionné tout à l'heure. Est-ce que tu trouves que les percepts de clean-code sont toujours d'actualité et pourraient être utilisé comme une référence pour les devs en général ?

Bah c'est une référence, oui parce que il final, quand t'apprends à dev, autant on prend-on a mieux dev et avec une expérience de quelqu'un qui a eu des années derrière lui final le le mec qui a écrit Bien tu vois, c'est un peu le cas et. C'est surtout que en fait, tu peux, tout le monde peut développer, mais le souci c'est que tout le monde doit avoir un minimum une façon de penser équivalente. Après tout le monde ne va pas penser de la même manière et heureusement, sinon on aurait pas d'innovation. Mais tu vois dans le sens où clean-code Ça t'apporte quand même des bases. Bah tiens nomme pas ta variable comme ça, il faut qu'elle ait un rapport avec le contexte que t'es en train de manipuler des choses comme ça et notamment après au niveau du Split de ton code. Tu vois un débutant ? On peut se demander, tiens, je vais faire une grosse classe alors que au final tu pourrais faire une classe qui implémente une interface sur des choses plus propres au niveau du découpage ? Tu vois donc en fait c'est pour moi c'est un, c'est quelque chose, c'est important je pense. C'est une notion à avoir quand on commence le dev.

Après, on l'a peut-être pas dès le début, quand on commence le dev hein, Moi je l'ai pas eu. Tu vois par exemple quand je suis sorti, je me rappelle de de master Clean code. Je savais pas ce que c'était. Tu vois. Alors après, je codais à ma sauce, c'était pas du du code sale mais c'est du code aujourd'hui si je le regarde, bah oui forcément. Je peux faire mieux quoi ? Donc tu es en prenant du recul, mais je pense que clean code, c'est important pour chaque d'avoir cette notion dans sa carrière, que ce soit tôt ou tard, en fait faut à un moment il faut l'avoir.

OK intéressant, un peu dans le même style et du même auteur, est-ce que tu connais clean architecture ?

Alors ça un peu moins parce que déjà j'ai pas lu le bouquin, mais une vague idée de l'architecture de toute façon, c'est appliquer les mêmes principes au niveau de ton archi applicative. Après je sais pas si dans le bouquin il va au point de faire l'archi cloud où l'archi vraiment Ops tu vois des choses comme ça.

C'est très centré autour de l'application en général, ça ne va pas traiter du micro-service par exemple.

OK. Ouais bah alors va les la clé architecture ça serait important aussi ? Pas forcément à connaître par cœur hein encore une fois mais tu vas avoir une idée de bah tiens, comment je vais architecturer mon application en fonction de ce que le client me demande ? On fait pour moi un bon dev, ça va être quelqu'un qui va être capable en sortie de réunion client déjà avoir en tête un schéma d'application. Ou alors un schéma de base de données. Tiens, je vais faire tel, je vais faire telle relation, avoir une idée au minimum.

Parfait. Alors, il y a une une partie, qui me tient à cœur; c'est sur la place des devs, aujourd'hui dans le milieu de l'OPS. Jusqu'à présent, on on a beaucoup parlé de maintenance applicative, là l'idée c'est pas un peu aussi de du lien entre maintenance applicative et la maintenance opérationnelle. À travers le modèle "You build it, you run it". Pour commencer, est-ce que c'est quelque chose qui te parle ?

Le "You build it, you run it" ? Je t'avoue non. Tu me poses une colle.

D'accord, le principe du modèle "You build it, you run it" c'est tout simplement dire que c'est les mêmes personnes qui vont concevoir l'application, qui vont la développer donc la partie "build" qui vont aussi faire toute la partie run, donc la partie vraiment maintenance de l'application en production.

Est-ce que c'est quelque chose que t'as déjà pu appliquer dans des équipes ?

Après vaguement en ce moment, j'ai plus la casquette DevOps sur des projets clients, mais à l'époque je t'avoue que c'était plutôt quelqu'un d'autre qui s'occupait de cette partie. Là, je prends l'exemple de mon équipe ou je suis resté 3 ans. En fait, on avait un responsable d'application, donc qui était pas devops lui non plus au final, hein. Enfin au début qu'il est devenu qui a mis en place donc qui fait du DEV en fait ? Il a

sorti quelques microservices, du gros monolithe dont je te parlais et en plus de ça derrière il a mis en place un flot gitops pour faire du déploiement continu. Nous sommes notre équipe donc il a fait la gestion du cluster, la mise en place de flux, la mise en place des helm charts, des choses comme ça.

Personnellement, moi, ce genre de choses, je l'ai jamais monté de 0. C'était toujours quelqu'un d'autre dans une équipe.

Est-ce que tu penses que ce serait pertinent ? En tout cas que ce serait bénéfique que ce soit plus fait par quelqu'un au sein de l'équipe de développement voir l'équipe de développement en entière. Que par quelqu'un externe à l'équipe.

Ouais, c'est plus bénéfique que ce soit, alors tout dépend le modèle parce que je t'avoue y a des sociétés comme Décathlon Qui vont laisser ça à une équipe dédiée. Tu vois donc ça dépend. Le modèle de l'entreprise. Des fois tu peux pas avoir le choix, mais je dirais que idéalement, si t'as le choix, c'est mieux que ce soit en interne parce que dès que t'as un souci bah la personne est là. Y a rien de tel que se lever de son bureau et parler avec la personne du problème plutôt qu'envoyer un ticket sur un outil de support. Je ne parlerai pas de SMAX hein, bien sûr, mais mais voilà, tu vois.

Mais du coup ça peut rallonger les délais inutilement. Et en fait. Le bénéfice que tu l'as de l'avoir en interne aussi, c'est que la personne en charge de ça va aussi avoir conscience de ce que fait l'application et de comment elle doit tourner, tu vois ? Donc ça, ça peut être bénéfique, contrairement à quelqu'un qui va être externe et qui va dire grosso modo bon les requirements, on va vous mettre 2 gigas, on verra ce qui se passe. Tu vois dès le début on pourrait avoir des choses plus précises, plus nettes, qui se rapprochent du produit.

Très intéressant. Dans une équipe en en général pour toi, qu'est ce qui va être les corps de métier, en fait, vont vraiment avoir un rôle à jouer ou doivent en tout cas avoir une prise de conscience sur les enjeux de la maintenance applicative.

Ben niveau, maintenance applicative. Bah c'est les devs en premier lieu et puis après le tech-lead qui serait responsable de la qualité du produit au niveau technique. Il faut savoir que si admettons, il a pris les mauvaises décisions dès le début. Ben potentiellement voilà, on pourrait avoir des soucis au niveau maintenance applicative et bah tout simplement je dirais peut-être aussi le scrum master, des choses comme ça en agile parce que si t'es pas proche de ton équipe, si tu es pas assez de rétro des choses comme ça que si les tickets à la volée on les fait sans code review on les fait sans sans les estimer des choses comme ça. Au

fil du temps. Bah le développement va se faire un peu à l'arrache tu vois donc.

Au final, j'ai envie de te dire, c'est peut-être toutes les personnes qui composent les types qui ont cette responsabilité là ? Chacun a une petite partie du moins.

Est-ce que l'agilité pour toi c'est un rôle qui est bénéfique dans la partie pour créer une application qui soit plus facilement maintenable ?

Bah je te dirai oui parce que souvent, on retrouve des équipes qui sont quand même de l'ordre de plus de 3 ou 4 personnes qu'on va chez nos clients. Après, si t'es 2 à développer, faire de l'agile, voilà je veux pas te dire que c'est indispensable. Mais ouais, moi pour moi, quand t'es chez les clients avec des grosses équipes, l'agilité ça y participe grandement.

Merci beaucoup, j'ai posé les questions que je voulais, est-ce qu'il y a d'autres thématiques que tu souhaiterais aborder ?

Sur la maintenabilité de l'application, je sais pas si on sort du sujet mais on a déjà eu un peu le cas, tu vois si je te parle de DavIdentity, tu dis mais mince, on va répéter cette chose là dans plusieurs applications alors qu'on pourrait le sortir et revenir du coup sur le code ? On pourrait le sortir en microservices et du coup revenir sur les autres applications. Tu vois je pense qu'il serait important aussi du côté maintenance.

Maintenabilité applicative. Tu vois, ça serait d'être bien, réfléchir final, c'est de l'archi. Dès le début ce qu'on va utiliser comme ressources, et cetera, comment on va les manipuler pour, pourquoi pas y sortir ? Un micro service qui va ensuite nous éviter de se répéter au sein des autres applications. C'est des cas qui peuvent arriver en fait, dont on se rend compte plus tard. C'est ça qui est dommage. Donc bien penser, pourquoi pas en amont, autour d'une table. Tiens, mais là, faudrait mieux le sortir parce qu'on va en avoir besoin pour des activités futures.

Merci beaucoup d'avoir répondu à mes questions. Je te souhaite une excellente journée.

Retranscription de l'interview avec Sylvain Lavazais, Technical Leader à Davidson SI Nord

Je te pose de commencer par une petite introduction que tu puisses un peu te présenter, parler un peu des projets sur lesquels tu as l'habitude de travailler.

OK ça marche donc moi, Sylvain Lavazais, ça fait un peu plus de 10 ans que je travaille dans ce métier. En fait 10 ans précisément. J'ai commencé en alternance donc dans la même situation que toi en 2009 donc j'ai fait une licence chef de projet informatique. J'ai enchaîné sur un master en expert en système d'information. J'ai terminé en fait mon alternance ce début 2011 chose comme ça. Donc voilà 3 ans, 3ans que j'ai passé avec ma première boîte SQLI et c'était à Lyon à l'époque. Et j'ai commencé. En fait, pendant mes années d'alternance, je faisais du des projets en .Net et donc j'étais principalement développeurs, mais c'était des petits trucs, donc des petits projets, de la maintenance très très souvent de la TMA. Ça c'est assez récurrent quand on fréquente les SSII, nouvellement appeler les ESN.

Globalement je faisais du .Net donc j'ai fait .Net 2, .Net 3. Pas plus, pas plus que ça en fait. Concrètement, j'ai commencé du coup réellement ma carrière puisque si on peut considérer que, en sortant de d'alternance, c'est le début de carrière, j'ai commencé ma carrière. Finalement, en passant sur du Java. À l'époque, c'était du Java 4 et pas de framework, rien du tout. Je travaillais pour Groupama. Et donc vous avez de travailler sur du web sphère Portal à l'époque, donc c'était pas c'était pas ouf hein ?

Déjà à l'époque ça vendait pas du rêve. C'est la grosse grosse usine à gaz. Et puis après ça, j'ai enchaîné sur un changement de boîte où je suis parti pour Norcys, donc j'ai travaillé d'abord un an à Lyon chez Norcys, toujours sur du Java. Cette fois-ci, j'ai monté un peu en compétence sur des des frameworks autres comme du Struts à l'époque, struts 1 et 2. C'est un vieux framework. On utilise plus maintenant. C'était à l'époque on utilisait struts, notamment pour pouvoir gérer la couche de la persistance, on avait pas encore hibernate pour les clients qu'on avait.

Donc j'ai passé un an chez Norcys à Lyon, comme ça. J'ai ensuite déménagé à Lille et là je vais me retrouver à Lille à faire plus du Spring MVC 4 et 3. Pour pour le compte d'assurance ou d'experts en assurance. En l'occurrence, c'est beaucoup de l'assurance chez chez Norcys et donc à ce moment-là j'étais déjà à Lille, mais toujours en anglais. En sortant de ça en fait.

J'ai fait un an et demi dans un peu dans des petits projets comme ça donc pareil c'était beaucoup du CDS finalement. Donc très peu de régie et en sortant de ça en fait j'ai fait un an et demi là-dedans en sortant de ça je suis sorti, je suis parti chez Capgemini.

Et là j'ai fait 2 ans, chez Décathlon, donc sur le projet cube. Qui est quand même assez assez gros chez Decathlon et là c'était du java, donc toujours du Java 7. À l'époque sur du cube donc sur cube c'était Oracle commerce ATG, anciennement ATG donc c'est un framework assez gros. Un monolithe pour être exact.

Et puis après ces 2 ans là, je suis parti de de Decathlon mais surtout quitter Capgemini en faite, et je suis parti en fait travailler pour Davidson donc en faite c'est depuis 2018 que je suis chez Davidson donc c'est le poste que j'occupe actuellement et j'ai enchaîné sur une mission chez Norauto qui m'a un peu tout appris en fait sur ce qu'on fait maintenant, tout ce qu'on a fait chez Norauto en fait dans ma mission chez nous on le fait actuellement toujours donc ça a pas beaucoup de bouger depuis donc notamment c'était une mission donc en python.

On a fait du Golang, on a fait du Typescript. Donc des petits trucs sympas à l'époque déjà. Et puis, en termes d'infrastructures, on utilisait du déjà du Kubernetes. C'était un projet de product information, c'est à dire que c'est un catalogue de produits imprime ce qu'on appelle un PIM, une product information management.

Et là on a commencé, j'ai commencé à monter en compétence sur toutes ces technos que je maîtrisais pas vraiment. Et puis surtout moi j'étais jamais sorti de mon Java et c'est un peu là que j'ai découvert la vie, si je puis dire hein. Dès que dès qu'on

touche un peu à du Go ou du Python, voilà bon un Java 8 il est un peu déboussolé, donc c'était l'occasion.

Et puis, suite à ma mission, donc là on a eu le début du COVID suite à ma mission chez Norauto on je suis sorti de nos de chez Norauto, je suis parti chez Adeo. Travailler sur un projet de de magasin alors c'est pas un projet de magasin, c'est un projet de construction de cuisine virtuelle pour magasin.

Donc en fait c'est la construction de préparation, de construction d'une cuisine en fait chez chez Leroy Merlin. Et là, c'était sur du revenu sur du Java, un petit peu et par contre on avait bien du Kubernetes et cetera quoi. Une très courte mission et j'ai enchaîné ensuite ma mission pour revenir chez Décathlon. Et c'est là la mission que sur laquelle tu m'as connue, Anthony.

C'est du donc là, c'est du Java, du vieux Java, du vieux Legacy. Mais pareil c'est du produit d'information, donc c'était très intéressant pour moi. J'aime bien, j'aime beaucoup ce domaine puisque c'est quelque chose, c'est un domaine dans lequel on on tape de la donnée, on on mélange de la donnée, on manage de la donnée donc c'est très enrichissant. Et là là pour le coup, c'était donc du Java 8 en Legacy et on a aussi testé des des petites technos comme du Quarkus. Le Java 8 c'était du Spring MVC, c'est vieux en plus. Et on a testé du Quarkus en Java17. Si mes souvenirs sont bons donc plutôt cools et puis moi je me suis un peu continue un petit peu à m'éclater, à faire du Python.

J'aime beaucoup faire du Scripting, Python et faire des petits programmes Python pour moi. Et ensuite j'ai quitté cette mission il y a quelque chose comme 6 mois et maintenant je travaille chez Auchan en tant que expert devops.

Alors ça j'aime bien un petit peu dans mon domaine. C'est je suis plutôt développeur back. En fait j'ai des grosses affinités de back, mais j'ai j'ai au cours de ma carrière travaille sur du Front et maintenant j'essaie de trouver un petit peu mon enrichissement sur du devops. Et voilà où j'en suis.

Merci beaucoup. Le sujet de cette interview ça va être vraiment tout ce qui va être autour de la maintenance d'une application. Donc maintenant c'est plus pour la partie applicative, on ne va pas aborder la partie maintenance opérationnelle avec l'OPS, ce sera dans une partie dédiée donc on va au global on va se concentrer sur la partie applicative. Petite question pour se mettre dans le bain Quelles sont donc pour toi les éléments les plus importants, ou les plus gros points de vigilance, quand tu travailles sur des applications ?

Alors les plus gros points de vigilance, ça va être surtout l'architecture de la manière dont les choses sont goupillées. En tant que jeune dev, c'est quelque chose que je négligeais beaucoup. En tant que Senior dev, c'est quelque chose que je regarde en premier. En fait, ça me donne un bon indicateur Sur la forme et la manière dont on va pouvoir maintenir l'application.

Pour rien te cacher en fait la plupart du temps en fait les missions que sur lesquelles on fait intervenir, c'est principalement pour moderniser en fait des applications. Donc moi je m'intéresse beaucoup à ces aspects-là. Donc à savoir concrètement. Savoir si la structure qui est choisie en termes d'architecture applicative et à l'intérieur de l'application, donc, avec les couches et cetera. Donc si y a une régularité en fait et si ou si ya des irrégularités.

Donc c'était le cas par exemple chez Décathlon. On avait un Spring MVC avec du Hibernate pour pouvoir gérer la persistance et à côté de ça, on avait aussi un type d'accès à la donnée en JDBC, donc en accès direct, donc sans couche de persistance. Et typiquement le mélange des 2. Bah crée des problèmes évidemment et donc rien qu'on se disant rien qu'en prenant conscience que c'est hybride, on sait déjà qu'on va avoir des emmerdes donc et un bon indicateur tout simplement et. Pareil. Un autre indicateur qui pourrait intéressant aussi, c'est la couverture de test.

Alors, qui est souvent négligée, tout simplement, la couverture de tests unitaires à après que ce soit du test unitaire ou que ce soit du test end to end, peu importe si on

s'aperçoit que les tests unitaires c'est une plaie et une galère à changer, à faire évoluer, et cetera rien qu'en modifiant une classe, on peut très vite s'apercevoir qu'en fait ça va être l'enfer de gérer ou pas, hein ? Un projet. Pour être tout à fait honnête, c'est quasiment tout le temps le cas, c'est à dire que modifier ne serait-ce qu'une classe, on se dit non, je vais faire un petit fixe quelque part. Modifier le test unitaire : "Mais qu'est-ce qui teste ce truc ? Je comprends pas" et du coup on se retrouve à un petit peu à la place du normalement de ce que de ce que de ce qu'aurait dû faire le développeur à ce moment-là c'est Ah je vais devoir réécrire toute la toute la classe métier, toute la classe de production parce que la classe de test unitaire est impossible à mettre en place et donc normalement c'est là c'est ça qui est censé faire déclencher quelque chose quoi. Et donc ça, c'est un des indicateurs aussi.

Un autre indicateur aussi, c'est le la documentation. Typiquement pas de documentation, ça signifie, c'est pas très bon signe, ça veut dire que potentiellement il y a pas de y a pas eu d'études, de besoins corrects et quand on n'a pas pris le temps de documenter les choses, alors peut-être qu'il y a une étude de besoin correct. Mais à un moment on a pas pris du tout le temps de de d'écrire la documentation, c'est c'est quand même très dommageable parce que tout ce qui est tout ce qui est entretien de l'application dans le temps, elle va se faire parle biais de la documentation. Donc si personne ne sait comment elle fonctionne. Ça peut pas bien se passer, voilà typiquement. Ça, c'est donc, c'est des différents critères que je vois après ce que j'en vois d'autres. Non pas dans l'immédiat.

En fait, il y en a d'autres en fait, mais ils sont plus indirects, typiquement croisés les utilisateurs. Comprendre, comprendre si l'application est finalement douloureuse pour eux. Ce qu'elle est chiant à manipuler, ce qu'elle les empêche d'avancer dans leur travail. Donc c'est pas ça peut être un critère mais c'est pas directement en fait quand on croisé la première fois l'application, on n'a pas les utilisateurs qui sont juste à côté donc c'est pas quelque chose qu'on peut vraiment se baser.

> C'est hyper intéressant, on va partir sur une partie sur le Legacy. Tu en as un peu parlé, notamment avec ton expérience à Décathlon. Est-ce que tu aurais en tête un

projet ou une expérience qui vraiment t'a marqué sur l'applicatif ? Peut être tu peux reparler par exemple de ce que j'avais fait à Décathlon peut être un autre projet auquel tu penserais.

Alors quand tu dis marquer négativement où positivement ?

Peu importe; mais c'est plutôt en lien avec la maintenance, surtout du Legacy.

Ouais, d'accord, effectivement la mission, un Décathlon, la dernière mission que j'ai faite à Décathlon. C'est sur cette application Legacy qui était donc en Java 8 avec du Spring MVC avec Hibernate et donc là un des premiers trucs. Je crois que le découvrir la première semaine où je suis arrivé, c'est que la couche de persistance, la couche d'accès à la donnée en fait, elle est semi persistante. En gros y a une partie en accès direct et une partie en via la couche de persistance. Et ça ce que ça m'a marqué, oui en fait, on n'a jamais pu. On est partis sur le mood initial de se dire de rénover cette application et puis passer un cap en fait, à force de découverte macabre, découverte macabre, on s'est dit non, ça sert à rien et c'est souvent, malheureusement, si j'applique ça en fait à toute ma carrière, si je regarde un petit peu par rapport à ce que tout ce que j'ai pu croiser, je me rends compte qu'en fait il y a beaucoup d'applications que j'ai pu croiser Legacy parce que c'est souvent le domaine dans lequel j'interviens.

La rénovation du Legacy, finalement, il y a beaucoup de cas. En fait, on aurait pu laisser tomber rapidement quoi. Donc ouais, je dirais que le cas le plus évocateur c'est ça en fait, c'est symptomatique parce que c'est ça correspond à ce qu'on peut recroiser malheureusement trop souvent dans le domaine applicatif, du développement applicatif.

Je pense que c'est, c'est le phénomène est dû au fait que on est trop souvent confronté à des équipes qui changent beaucoup trop souvent. Qui emploient des jeunes développeurs non encadrés parce que on a le droit d'être jeune, on a le droit de faire des

erreurs, c'est normal, ça fait partie de du cycle de vie de développeur, mais il faut pouvoir être encadré correctement et ça c'est c'est très rarement le cas.

Tu en as déjà un peu parlé, mais peut-être que t'as d'autres éléments. Quand tu es amené à travailler sur une application en général donc là ça peut être du Legacy où ça peut tout simplement être une reprise du code écrit par quelqu'un d'autre. Qu'est ce qui va avoir tendance vraiment à complexifier ton travail ? Par exemple tu as parlé tout à l'heure de la couverture de test, là tu as parlé du Turnover, tu as peut-être d'autres choses entête.

C'est très souvent en fait très souvent. Ouais. Les tests unitaires, le manque de rigueur, en fait, d'une manière générale, sur l'application qui va vraiment me rendre mon boulot incroyablement compliqué. Et le pire, c'est là ce qui peut en ce qui peut arriver et ce qui est plus frustrant dans ce métier, c'est de se dire j'aimerais faire mieux, mais au-dessus on m'interdit et on on me dit J'ai pas le budget, on me dit C'est pas possible. Et donc c'est c'est difficile de joindre les 2 en fait et on y a une espèce de forme, de frustration un peu récurrente de tous les projets, on se dit, on vient, on est de bonne volonté, on apporte quelque chose, on vient moderniser quelque chose.

On vient apporter de la qualité et apporter de la rigueur, apporter quelque chose de neuf. Et derrière, on est très souvent confronté par "Ah oui, mais on ne sait pas. On ne connaît pas cette technologie", ça, c'est ce que j'ai pu rencontrer. Par exemple, chez Décathlon, on essaye une nouvelle technologie, ça leur fait très très peur, alors que en vrai elle est pas du tout effrayante. Elle est même très abordable.

Je prends le cas de Quarkus qui est beaucoup plus abordable que Spring Boot ça, c'est un un pas d'exemple qui qui peut exercer une espèce de forme de frustration alors que le fait d'expérimenter une nouvelle technologie ou un nouveau framework ou quelque chose de d'assez neuf permettrait en fait de résoudre des problèmes beaucoup plus facilement que de réinventer ou de réécrire la totalité de l'application.

Donc voilà, c'est des choses qui peuvent me frustré personnellement et qui m'empêche de faire mon travail correctement.

Merci, c'est hyper intéressant. Donc plus conception d'application plus dans la partie vraiment dans le code qu'est-ce que tu mettras en place aujourd'hui dans une toute nouvelle application ?

Ouais, ça c'est facile, ça, c'est le la partie la plus simple et c'est quelque chose qui, comment dire, est très stimulant quand on a la possibilité de enfin pouvoir faire quelque chose de propre. Tout dépend après de la technologie bien entendu, mais globalement dans les standards il faut absolument avoir un besoin.

Déjà parce que faire de l'application pour l'application, ça apportera rien. J'ai déjà fait un j'ai déjà participé à un projet de R&D où il y avait absolument pas de besoin. On nous demandait juste d'atteindre un but technique et c'était pas du tout intéressant pour le coup parce que on n'est pas du tout stimulé et pire du jour au lendemain.

Tout le code peut passer à la trappe, y a absolument pas d'enjeux derrière donc c'est vraiment très frustrant.

Première étape, déjà, avoir un besoin clair parce que si le besoin est pas clair, bah l'application va être mauvaise. Clairement, on va commencer à écrire du code, on va se dire "Ah oui, tiens, je vais, on va essayer de faire ça comme ça. On va essayer de goupiller cette application là avec cette application là, en passant par là, et cetera". Et puis quand le mec il passe enfin, quand le la personne qui demande qui a besoin de cette application passe par l'avenir ?

Oui, mais en fait moi je voudrais exactement ça. Ah mais fallait dire plutôt parce que ça correspond pas du tout à ce qu'on a dessiné. Un besoin clair, déjà, c'est un bon élément et c'est même un excellent élément. Et c'est pas forcément l'élément le plus simple à obtenir.

Ensuite, fixer ensuite une fois qu'on a quelque part décidé d'acheter quelque chose fixe, un cadre de rigueur, à savoir donc des des guidelines, ça peut passer par la DOD donc la "definition of done" mais également la DOR, la "definition of ready" pour savoir quel est besoin on sera, on sera en capacité de traiter quel est le niveau d'exigence qui nous permet, à nous de traiter dans équipe de développeurs. Je me situe plus dans une équipe de développeur. C'est très rarement moi tout seul qui fait tout le taf bien évidemment. C'est pas du tout mon genre et ça me plait pas du tout d'ailleurs, mais la définition of ready va nous permettre d'établir en fait des standards entrant dans l'équipe qu'on va pouvoir dire OK, qu'est ce que je suis capable, ce que je suis en capacité de traiter ? Quel est le niveau minimal que je peux prendre pour commencer à faire mon travail et la definition of done c'est quel est le but que je dois atteindre systématiquement à chaque fois que je j'accomplis une feature ? J'accomplis quelque chose en fait en sur l'application.

Et déjà ça en foutant une rigueur un petit peu, alors pas trop contraignante au départ parce que l'objectif c'est de prendre le rythme. OK, c'est à dire que si on met une tout de suite une définition d'un vrai dit une "definition of ready" hyper large hyper grosse, enfin même pas large mais plutôt très détaillé. Les développeurs vont être soit soûlés de répondre à toutes les exigences d'un coup, soit ils vont se dire OK on la, on la vire pas et puis vas y c'est parti, on on avance parce que il faut produire.

Et donc ça c'est c'est pas très bon, pas très bon et il faut pas mettre tous les œufs dans le même panier quoi. Il y a quand même pour laisser un peu de de liberté quand même au développeur même de, mais en gardant quand même une base de rigueur.

Dans cette base de rigueur qu'est ce que tu te verrais de mettre par exemple ?

Dans cette base de rigueur, typiquement, avoir une convention, une convention de codage, donc une convention de commit pour que le système de version puisse fonctionner correctement, qu'on soit tous d'accord sur les mêmes standards, tout

simplement. Aussi peut être une convention de travail de co-travail entre guillemets sur les tâches parce que il se peut que la stratégie ce soit une personne par tâche. Il se peut que la stratégie ce soit aussi également une tâche égale à un jour maximum ce genre de choses.

Donc mettre un peu de rigueur vis-à-vis de ça en fait dans l'équipe parce que c'est une organisation en fait qu'il faut qu'il faut réussir à monter. Et si pas tout le monde est d'accord sur la manière de fonctionner mais y avoir des tensions va y avoir aussi de la rébellion, peut-être ça arrive. C'est pas forcément quelque chose qui éclaté, mais c'est simplement un développeur qui s'en fou de ce qu'on a des fixe, ce qu'on a fixé comme standard donc du coup ils ne les respectent pas. Donc bien d'accord avec ça.

Et puis donc ça c'est un on va dire un des plus gros morceaux parce que après une fois qu'on passe à la partie technique, c'est la partie la plus simple, entre guillemets, on sait que tout le monde a à peu près le même niveau. Alors s'il y a des développeurs qui ont besoin de d'ajuster à ce moment-là, on ajuste. On essaie de les former, tout simplement. Ce qui nécessaire.

En général, c'est plutôt les gens sont recrutés pour faire un type de technologie et donc on sait qu'on va déjà partir sur ce type de technologie pour pouvoir accomplir l'application. Enfin créer l'application.

Voilà et puis, une fois que tout ça et tout ça est mis en place, il faut aussi des des outils qui nous permettent, à nous de surveiller, parce qu'on a une certaine rigueur, une certaine volonté d'atteindre un niveau de rigueur pour l'application, mais il faut qu'on puisse surveiller cette application et il faut que ce soit quelque chose d'autre que un humain qui surveille l'application.

En général, on utilise un linter on utilise un, on utilise un outil qui nous permet de surveiller la qualité de code, donc par exemple un sonar sonar fait pas tout. On on fonction des technos alors le sonar c'est parce que ça vient de surtout du monde Java. Mais dans l'absolu, n'importe quel outil qui fait le taf au niveau de la rigueur de code ou

du code style de linter et et aussi de la duplication decode y compris la couverture. Enfin y a plein de trucs à gérer, juste surveiller tous ces niveaux de qualité entre guillemets. Et puis qui nous permet, à nous d'en fait, de surveiller après, peu importe si on utilise quelque chose d'automatique, il permet de automatiquement délivrer le code automatiquement.

Créer une intégration continue ou alors, un déploiement continu, c'est ça. C'est en fonction du cas présent. Il y a des des clients qui n'utilisent pas du tout de CI, ni de CD et donc à ce moment-là on bah on fait avec ce qu'on a. Tout simplement. Donc, l'important est de pouvoir surveiller notre rigueur.

Et on en mettant en place toutes ces petits outils, nous, on va pouvoir commencer à développer et faire vraiment travailler dans le développement de fonctionnalités. Développement du corps, développement donc de la de la base, développement de des fondations de l'application, et cetera. Et puis avancer étape par étape en positionnant du coup chaque étape à un niveau, un niveau suffisamment bas au départ et puis après plus haut, de plus en plus haut, et cetera en fonction de en fonction des besoins. Voilà globalement.

Merci beaucoup Je te propose d'aller sur une partie un peu plus sur le Clean Code, en général plus tout ce qui est plus architecturé d'application donc déjà si je te parle de de Clean Code ou plus généralement des ouvrages de Robert C. Martin, est-ce que ça te parle ? Tu as déjà dû en entendre parler j'imagine ?

Clean code est censé être utilisé comme une Bible et non pas enfin comme une Bible, comme un un livre de codage, ça veut dire on code, puis on regarde de temps en temps le livre, et cetera. Il faut pas le lire debout en bout, entre guillemets, c'est ce qui est recommandé dans le livre. Et il a des bons exemples en fait à donner pour son expérience. Typiquement, je crois me souvenir. Ça fait longtemps que j'ai commencé à lire. Je crois me souvenir d'un cas qui ne citait très, très facilement et maintenant avec le. Avec le recul, je me dis oui, il a complètement raison là-dessus.

C'est une application qui est qui, qui a mis beaucoup de temps à être mis en place avec beaucoup d'argent, beaucoup de moyens, beaucoup de personnes. En fait, on voit très souvent ce genre d'application qui nécessite genre 300 personnes pendant 3 ans, ça fait plus de hommes que je ne peux compter, un budget de malade, plusieurs millions, et cetera pour à la fin se rendre compte qu'elle ne correspond plus aux besoins.

Et on la jette tout simplement. Et ça, avec le recul, tu vois, c'est vraiment quelque chose qui arrive plus fréquemment qu'on le pense. Donc oui, Clean Code est une référence pour moi, je l'utilise, alors je l'ai en plusieurs versions parce que y a j'ai eu des phases papier, puis j'ai eu des phases digitales. J'sais pas ce qui me prend ça c'est en fonction du besoin, j'ai envie de dire mais bon je je temps en temps, je ressorts ma version digitale sur mon téléphone et je regarde quelques trucs dessus. Pas beaucoup parce qu'en fait ya pas besoin de de voir énormément de choses pour comprendre quelle est la bonne marche à suivre pour pour la suite. Mais ouais c'est pour moi un livre de référence, tout simplement pour ne serait-ce que pour avoir un background d'expérience au tout départ.

Super, donc comme tu disais, ça reste une référence aujourd'hui et est-ce que c'est plutôt un livre que tu recommanderais ou tu penses que aujourd'hui on arrive à faire mieux ou autrement ? Pendant mes recherches, j'ai souvent trouvé des questionnements sur : Est ce que Clean Code, ou même Clean Architecture reflète réellement l'informatique d'aujourd'hui ?

Je vais avoir un point de vue très biaisé malheureusement, par rapport à ça. En fait, moi je pense que c'est toujours une liste de référence, mais c'est comme toujours. En fait, c'est si on devait comparer quelque chose donc Clean Code et Clean Architecture, ce sont des livres de référence pour se donner une idée quand on ne sait pas ce quel'on fait, il faut lire ce genre de livre, ne serait-ce que pour comprendre ce qu'il ne faut pas faire et ce qu'il faut faire. C'est juste des bons indicateurs, c'est pour ça que clean code est un livre de un livre de codage entre guillemets, c'est que le lire de

fond en comble, vous allez rien comprendre, tout simplement, ça ne sert à rien de lire de bout à bout.

Il faut le pratiquer, c'est juste une pratique et puis se faire sa propre expérience par rapport à ça. Et donc moi j'ai un point de vue assez biaisé parce que je pense que c'est toujours des bons livres. Maintenant je pense qu'il y a d'autres références, je suis pas, je suis pas un lecteur assidu par rapport à tout ça, j'essaie de rester conscient quand même de ce qui se passe, et cetera, de d'être un minimum en veille sur toutes les toutes les nouvelles formes d'architecture. On arrive plus ou moins à être au courant de ce qui se passe. Je pense peut-être pas aussi bien que si on travaille chez Google par exemple, mais au moins on a conscience de ce qui arrive et de ce qu'on peut faire, ce qu'on peut pratiquer tous les jours.

En fait, mon point de vue, il est un peu, ce sont des livres de références, mais il faut les prendre comme tels. Ce sont des livres de référence et donc les références ne s'appliquent pas forcément systématiquement telles qu'elles et surtout pas en fait, elle ne s'applique jamais telle qu'elle, il faut les réfléchir, il faut y penser. C'est censé inculquer une sorte de réflexion personnelle autour de des architectures et comment les pratiquer, forcer une architecture sur une application qui, même si ça pourrait passer.

Mais que personne n'a adhéré au passage, ça n'a aucun sens. C'est exactement les mêmes principes, les mêmes, on va dire problèmes qu'avec avec l'Agile. Le manifeste Agile, ce n'est qu'un manifeste, il est censé donner des méthodes et censé donner des choses. Il y a des frameworks agiles qui se qui se qui s'utilisent, on peut parler des 2, le plus utilisé comme le kanban et puis le scrum. Mais en réalité, ce qu'il faut faire vis-à-vis de l'agilité c'est l'adapter. En fait, utiliser de brutalement et sèchement une architecture de code. Oui, en fait juste sur le principe oui bien entendu, c'est très utile et ça peut même convenir à des équipes et cetera.

Et ça peut même accélérer les développements. Le problème c'est l'adhérence et l'adhérence pas que du côté de l'équipe de développeurs, mais également du côté du besoin. Si on s'aperçoit qu'en fait je prends l'exemple de de de l'architecture hexagonale.

Typiquement j'ai déjà eu le cas où on a essayé de la mettre en place. Le problème c'est que donc ça a un ça, nous ça nécessitait de bien séparer les couches entre elles et de veiller à ce que en fait il y ait pas de débordement entre les 2. Le problème c'est un concept assez nouveau comme c'est un concept assez nouveau pour nous à l'époque. On a eu beaucoup de temps en fait à le mettre en place et bah le client il attend et il dit mais comment ça se fait que ça prend autant de temps ?

C'est pas possible, ça correspond pas à mes standards et cetera. Donc en fait on avait commencé une application comme ça, on en a fait une. Et puis la 2e qui était censée la faire, on dit Ah non on y arrivera pas. On si ça prend le même temps que la première fois, ça va être horrible et donc en fait on avait déjà, on l'a, on l'a pratiqué complètement différent et un peu au feeling en disant Bon on va faire des trucs très simples, on va pas s'embêter et cetera donc c'est c'est un un vieux pieux en fait. De pratiquer et de mettre en place des belles architectures. Le problème, c'est d'avoir le temps et d'avoir les moyens de le faire. C'est juste ça en fait le problème, si on est dans une entreprise qui ne néglige pas ce genre d'aspect, qui nous autorisé vraiment. C'est si on tombe sur ce genre d'entreprise qui qui se dit, allez y carte blanche, faites ce que vous voulez, faites des trucs merveilleux, faites des trucs bien parfait. Allons y testons ça et en fait, testons même ce qu'on veut, testons des nouvelles choses, et cetera, et ça nous permet de grandir, nous et de aussi de également, de rendre service par la même intermédiaire. J'ai un bon exemple, en fait, vis-à-vis de tout ça. J'ai eu en fait au cours de ma mission chez Norauto. On, on a croisé en fait un un développeur qui venait de Zalando. Qui nous avait expliqué en fait que la la politique chez Zalando c'était maximum 2 mois pour un projet. Alors pourquoi un maximum 2 mois ? C'est qu'en fait tous les 2 mois, le projet pouvait être refacto, changer et donc il fallait maximum 2 mois maximum sur un projet pour que il se fasse de fond en comble et donc en fait les mecs ils avaient ça comme contrainte ils disent "Oh 2 mois, qu'est-ce que je peux faire en 2 mois ? Ah oui mais alors ce périmètre là il est trop gros ? Donc il faut qu'il passe en 2mois, donc je le découpe et cetera. Je passe en 2 mois, je fais en sorte que je puisse développer mon application en 2mois" et au bout des 2 mois en fait l'application elle peut être soit gardée soit refacto soit elle convient en besoin et on y touche jamais soit

elle convient plus au besoin et on peut la réfacto et ça ne coûte que toujours que 2 mois et cette règle était impressionnante.

En fait juste d'organisation et ça permettait direct de dire OK en 2 mois j'ai pas le temps de me consacrer à faire des trucs nouveaux. Par contre en 2 mois il y a des petites technos sympas qui nous permettent de remplir très rapidement le truc. Ça reste propre, c'est surtout ça le plus important, parce que si on doit le rédacteur en 2 mois, Ben il faut que ça reste propre hein minimum sinon ça va pas être rapide. Voilà, et donc tu vois, c'est un, c'est un un exemple qui vient en tête, c'est les entreprises qui ont conscience de comment ça pourrait être au niveau technologique. Se disent, bah chez nous on va fixer un standard, c'est pas du tout arbitraire. En fait les 2 mois hein c'est parfaitement. C'est la rigueur qu'ils ont vu, ils se sont aperçus qu'en fait, les projets qui qui faisaient où ils faisaient des refontes régulières dessus au bout de X années, ça, ça leur prenait plus ou moins 6 mois-1 an de un an de développement et ils ont dit non mais le mieux ce serait quand même de pouvoir dire OK, toutes ces tous ces blocs là chaque bloc ça vaut deux mois comme ça on saura facilement dire à quel moment on peut l'échanger, qu'est-ce que ça va nous coûter, et cetera.

C'est une maîtrise, une maîtrise qu'on n'a pas systématiquement en fait, dans toutes les entreprises quoi. Et je trouve que c'est un bon exemple typiquement de maîtrise et en termes d'architecture, ça permet aussi aux développeurs de se dire j'ai que 2 mois donc je peux choisir qu'un seul type d'architecture ou alors une, une architecture qui convient à toute la boîte qui est déjà mis en place partout. Si dans l'entreprise il y a déjà de l'architecture hexagonale par exemple. Et Ben. On peut l'appliquer, puisqu'on a l'habitude de le faire. On va pas s'amuser à tester notre architecture et cetera, donc c'est plutôt une bonne pratique au point de vue. Alors ça doit un petit peu quand même limiter l'inventivité des développeurs. Mais du coup, même une bonne pratique.

C'est intéressant, j'étais pas du tout au courant qu'ils travaillaient comme ça chez Zalando. Tu sais, c'est toujours d'actualité hein par hasard ?

Alors je sais pas du tout parce que mes informations datent de 2018. Mais je trouvais le concept intéressant et en fait à l'époque, on avait amorcé la discussion avec Zalando parce qu'on avait Norauto qui recrutait un de leurs développeurs et donc le développeur en question avait expliqué comment ça se passait tout simplement chez Zalando et notamment on avait besoin de comprendre son retour, son son background, son expérience. Parceque eux, ils avaient beaucoup travaillé, notamment avec Kafka et nous à l'époque, quand on essayait de mettre en place Kafka chez Norauto, on voulait comprendre un petit peu concrètement, comment est foutu Kafka. Et en fait ce coup de Kafka, contrairement à ce qu'on pourrait croire, c'est pas un bus d'événement standard, c'est en fait une armada, c'est la Rolls Royce des bases de données et typiquement c'est vraiment le summum ultime de la base de données on ou on place toutes les informations dedans structurées et après tu peux faire ce que tu veux avec tu les retraces, tu les tu les sorts tu peux requêter avec contrairement à un simple bus de données quite va te recracher simplement les messages tels qu'ils sont rentrés ou alors avec juste une stratégie bête de de sortie de message quoi.

Donc c'est c'est un un des trucs qu'on avait qu'on avait sur lequel on avait croisé. Du coup ce développeur qui venait d'Allemagne pour l'information et du coup c'était très intéressant, ne serait-ce que pour avoir son son background quoi tu vois.

C'est hyper intéressant en toutcas comme méthodologie. En tout cas, c'est assez facile d'imaginer comment ils en sont arrivés là, mais en tout cas en pratique ça doit être assez impressionnant j'imagine. Par contre, comme tu dis niveau créativité du développement tu disais tout à l'heure que pour faire des bonnes choses, il faut du temps et les moyen et là le temps on ne l'as pas donc ça complexifie les choses.

Ouais peut-être, mais je après tu vois j'étais pas au fait, il nous avait expliqué que c'était ça se passait comme ça chez Zalando, mais peut être que tu vois à côté de ça en fait tous les 2 mois en fait il fait une coupure et tu pourrais faire de la veille techno et tu et et connaître des enfin tu vois te renseigner ces nouvelles technologies afin de les ajouter aux aux aux entre guillemets,aux à la boîte à outils de l'entreprise.

Un peu comme une Slack Week ?

Ouais c'est ça, mais cette fois-ci un peu plus fort parce que 2 mois de développement quand tu sors d'un projet, tu dis peut être que tous les 2 mois en fait il change de projet les mecs ça aurait été de faire tout le temps la même chose, ce qui est plutôt pas mal et surtout très enrichissant puisque quelqu'un qui a travaillé sur le domaine information qui va se mettre à travailler sur du Big data. Ensuite elle va pouvoir apporter ces trucs à lui dedans et donc c'est très enrichissant je pense. J'imagine que le rythme de 2 mois est très symbolique et permet de se dire : On a une récurrence et on essaie de faire tourner les choses dans l'entreprise. Tout simplement, et je trouvais que c'était pas mal. Ton concept, c'est un concept d'ailleurs qu'ils ont chez Norauto, ils ont essayé de faire ça, mettre ça en place et puis finalement ça s'est effondré face à la volonté de toujours avancer, de voir développer des produits plus haut, plus loin, et cetera.

Et là pour le coup par exemple, pour le PIM qu'on a développé chez nous, on était pas à 2 mois, on était plutôt sur 3 ans.

Je te propose de parler un petit peu plus du lien entre maintenance opérationnelle et maintenance applicative. Les termes sont très vastement utilisés donc dans ce cas, la partie maintenance applicative, c'est "tu récupères une application et il faut continuer d'ajouter des fonctionnalités et de résoudre des bugs" et la maintenance opérationnelle c'est "mon application crash en prod comment elle revient ?". Tu as un modèle autour de ça, c'est le modèle "You build it, you run it" Est-ce que c'est quelque chose qui te parle ?

Oui, c'est quelque chose que je préfère. On va être honnête, on utilise les 3/4 du temps, le "you build-it, you run it". On est très loin, donc je sous-entends, je développe un peu ma pensée. Je sous-entends qu'en fait les développeurs sont pas du tout au contact même sont restreints d'accéder à la prod et n'ont pas du tout conscience dans quelles conditions vont être déployées leurs applications et donc c'est malheureusement

l'écrasante majorité en France, je pense. Alors évidemment, j'applique avec mon expérience, mais bon, j'ai pas connu tous les projets de France donc je peux pas me situer et cetera, mais de mon expérience en tout cas les 3/4 du temps c'est ça et c'est très dommageable.

En fait tout simplement, j'explique juste pourquoi puis après je passe sur l'autre l'autre partie en fait c'est très dommageable. Pourquoi ? Parce qu'en fin de compte on c'est un peu comme si on disait Ben fait ça. Mais ne t'inquiète pas de comment ça va être fait après. Et ça, c'est un petit peu comme déléguer sa responsabilité à quelqu'un d'autre. Ça fonctionne rarement. Et surtout les développeurs qui une bonne équipe de développeurs, qui a des idées, qui met en place des standards, ça les intéressera beaucoup plus en fait de participer jusqu'à la production.

C'est beaucoup plus stimulant. C'est beaucoup plus intéressant, tout simplement et un point ultime en fait, les idées qui ont été mises, qui ont été utilisées dans l'équipe de développement ne sont pas partagés et ne sont même pas partageables, tout simplement avec les Ops qui vont devoir mettre en place l'application. Et pour un Ops, pour le coup, je suis dans ce cas-là, chez Auchan, rentrer dans une application et essayer de comprendre comment ça marche, c'est l'enfer parce que on a pas idée de comment ça a été développé. On n'a pas idée avec qui ça a été développé, sur quel besoin, et cetera.

Très souvent, c'est très flou, on a très souvent qu'un titre d'application et puis c'est tout. Donc c'est vraiment très limité cette manière de faire et donc la maintenance opérationnelle donc le "You build it, you run it", ça donne un peu plus le pouvoir. Et même si ça donne la responsabilisation à outrance entre guillemets pour le développeur, c'est lui qui est responsable de l'application du moment où il commence à l'écrire jusqu'au moment où il va le mettre en production.

Et ça c'est essentiel. En fait, on on on dirait "Ah ouais mais c'est beaucoup de responsabilités, ça fait très peur. Comment je peux faire, comment je peux gérer ça ? Moi développeur", mais en fait, c'est merveilleux. Tout simplement parce que on est beaucoup plus stimulé. On est beaucoup plus engagé dans cette production. Dans dans

cette création d'applications, c'est pratiquement comme si c'était notre bébé, qu'on l'amenait jusqu'au bout et quelque part, c'est pour moi, c'est une bien meilleure pratique que de livrer du code à quelqu'un qui s'occupe de le mettre en production et pour l'avoir pratiqué chez Norauto c'était ça en fait la politique "you build it you run it" jusqu'à la prod.

On a failli le pratiquer chez Décathlon, ça s'est pas fait, mais c'était juste une question de temps. On commençait à gagner de la responsabilité. On avait pas complètement, mais c'était un peu le cas, mais je peux certifier qu'en tout cas chez Norauto et chez Adeo, c'est bien ce qui est pratique et donc c'est très très très intéressant et c'est beaucoup plus stimulant. Et en plus, c'est un gage de qualité puisque la très souvent donc mettons si on se positionne dans le cas de l'écriture d'une application de fond en comble, donc y a pas la première fois qu'on va mettre en prod. En fait il y aura personne dessus donc on est en mode un peu safe.

OK on l'a mis en prod ouais ça marche pas y a des trucs qui déconnent, un mince j'ai oublié un paramètre machin, pas de stress, y a personne dessus, y a personne dessus et pour l'avoir pratiqué et donc c'est chez Norauto c'était en mode cycle de développement en gitflow, c'est à dire 2 branches. C'est un peu l'enfer avec 2 branches, mais avec une seule branche, c'est le paradis. Pourquoi ? Concrètement ? Parce que le moment où on a livré le code ou tout le monde à review le code et on OK, on est bon on merge le code. La, l'application est immédiatement mis à jour en production et c'est beaucoup plus simple.

Le programme accelerate qui est absolument une merveille en fait ce qui est oblige les équipes de développement à livrer directement en production. Et concrètement, ça oblige en fait d'avoir ce que j'expliquais tout à l'heure. Si on est, on s'inscrit, on essaie de s'inscrire dans le programme Accelerate et qu'on a pas suffisamment de rigueur. Mais tout de suite, ça s'effondre. On va livrer que de la mauvaise qualité. On va livrer que des choses qui sont pas qualitatives en production et ça va être très, très dommageable. Par contre le justement pour correspondre tout de suite au au programme accelerate, le fait de se dire je veux correspondre absolument

à ce truc là, on est obligé de mettre de la rigueur, et mieux c'est de la rigueur automatique, c'est à dire qu'on a rien à faire, on n'a pas besoin de penser le tout. Le code est revu automatiquement, l'évaluation se fait automatiquement, les tests se font automatiquement. On peut aller quand même, c'est quand même assez fou comme mécanisme, ça peut aller jusqu'à créer des environnements éphémères pour tester des des, tester de la non-régression dessus, vérifier que tout se passe bien et si c'est pas bon, ça peut nous revenir dans la dans la tête et invalider notre pull request par exemple.

Et si c'est bon, bah ça part directement en prod parce que ça a été testé, ça fonctionne donc c'est de la rigueur automatique, c'est juste ouf quoi. Et en fait, dans l'écrasante majorité des cas, on peut se dire que ça pourrait pas forcément être un gage de qualité mais en fait au contraire. Le fait de mettre en place toute cette mécanique qui est un peu complexe à mettre en place au tout départ. Un me permet de mettre en fait de la rigueur sans euh, sans contrainte, en fait, sans sans douleur, entre guillemets pour le développeur. Et donc c'est tout simplement merveilleux de pouvoir se reposer sur un système automatique qui va faire quasiment tout le taf à ta place.

Qui va vérifier tout la totalité de ce que tu as pu commiter vérifier que tout se passe bien et t'es pas obligé, t'es pas obligé de te rappeler toutes les étapes puisque un système automatique lui va se va se souvenir de chaque étape de vérification avant de mettre en production. Donc ouais, c'est c'est de l'or en barre, ce truc c'est carrément mieux et c'est je je moi franchement si ça ne tenait qu'à moi, tous les projets fonctionnent comme ça quoi. C'est ça me semble illusoire en fait de continuer à faire du développement Manuel, de vérifier manuellement que tout ce qu'on a livré bah c'est bien, c'est qualitatif, c'est à cause de ça que alors c'est pas pour dénigrer le travail des des QA mais c'est à cause de ça qu'on a encore besoin de gens alors qu'en vrai ces gens-là pourraient faire du travail beaucoup plus intelligent. Ils pourraient apporter du besoin métier pourrait-il pourrait travailler beaucoup plus avec les besoins plutôt que de de travailler sur la qualité.

Toute cette partie qualité pour être automatisée pourrait être usinée, entre guillemets. Ça fait pas beau de dire comme ça, mais en réalité, c'est un véritable, c'est

du grand art et moi je considère que c'est de l'artisanat le fait de d'écrire ce genre de choses, d'écrire ce genre de mécanique. C'est absolument prodigieux.

Il reste une dernière partie sur l'agilité, que tu as déjà plusieurs fois évoqué d'ailleurs, déjà et en fait c'est plus généralement l'implication de chaque personne dans la maintenance applicative. Quels rôles dans l'équipe sont impliqués dans la maintenance applicative ?

C'est en premier lieu les développeurs, alors dans mes modèles personnels que j'aime beaucoup, sur lequel j'adore le plus, on va dire. Le développeur est au centre de tout parce que pour moi c'est la machine à tuer. Au niveau qualité quoi, c'est lui qui peut automatiser tout ça. En fait, toute cette mécanique dont j'ai cité, dont avec la méthodologie "You build it you run it", il y aurait dit, seuls les développeurs en fait ont cette capacité. Pour moi c'est c'est la force de frappe vraiment de l'équipe et on a trop souvent tendance à vouloir un petit peu les chouchouter. Je pense qu'il faut contraire, il faut un peu les bousculer justement pour que ils apprennent que ils peuvent faire des trucs merveilleux parce que je pense que ça fait grandir tout homme ou toute femme tout simplement.

Et après, en fonction du cas, on peut avoir le PO, donc le Product Owner, qui peut être sollicité ne serait-ce que pour comprendre ce qui a pu planter, qu'est ce qui a été loupé. Qu'est-ce qu'on aurait pu faire mieux, tout simplement, et et surtout très souvent, un bug est considéré comme un bug. Parce que le Product Owner ne l'a pas compris ou nous n'a pas su l'expliquer tout simplement. C'est souvent en fait de l'incompréhension mais mais, je comprends pas que le développeur dit au PO, "je ne comprends pas, je pensais que ça fonctionnait comme ça". "Ah oui mais tu as mal compris développeur", c'est pas du tout comme ça que ça fonctionnait et donc c'est un bug, ça ne fonctionne pas, ça ne répond pas aux critères, et cetera. Ouais mais moi j'ai des compris comme ça et donc si je suis pas forcément responsable alors c'est très souvent l'accusation quand il y a un problème hein, mais dans dans l'esprit pour moi il est important que le PO comprenne pourquoi les problèmes arrivent.

Si les problèmes arrivent potentiellement c'est que soit les développeurs n'ont pas tout le pouvoir et n'ont ne peuvent pas agir sur le cadre maintenance, soit le Product Owner lui même caché des choses. Et il peut y avoir plein de raisons de cacher les choses. Ça peut être multiple, ça peut être politique, ça peut être plein de chose. Mais voilà. Et puis il y a donc le cas des QA. Alors le QA le cas des QA alors c'est un rôle, Quality Agent, qui est de surveiller finalement toutes les toute la qualité autour de l'application. Mais pour moi, le rôle d'un QA, c'est de à terme faire en sorte que son poste n'existe plus. Pourquoi ? Parce que si on a besoin d'un QA, ça veut dire que potentiellement, on a toujours des problèmes de qualité et potentiellement, on ne répond jamais réellement complètement à la demande de qualité.

Normalement il doit disparaître à terme et si un QA est maintenu sur un projet, c'est pas très bon signe, ça veut dire que la qualité est en baisse ou qu'elle a baissé un moment donné et qu'il faut la redresser en permanence. Et ça, c'est pas bon pour l'avoir connu. Sur le cas de Cube. On a fait, donc j'ai travaillé pendant 2 ans et demi, dans lequel on avait plusieurs QA par équipe, donc une ribambelle de QA quoi. Et typiquement c'est difficile de comprendre pourquoi on a systématiquement besoin d'un QA pour moi c'est c'est pas censé être un rôle qui se ternisse, hein, qui se pérennise dans le temps, quoi, à la limite tu vois si on prend le cas du encore une fois d'un projet qu'on fait de fond en comble, tu prends je sais pas moi 6 développeurs, je lui dis voilà, j'ai besoin de 6 développeurs pour développer toute mon application et tu incorpores en fait dedans tout toute la gestion de la qualité au développeur.

Il va le faire, il va savoir le faire. Pourquoi ? Parce que les chaque développeur, sait exactement ce qu'il a écrit. Donc au moment où le développeur écrit la fonctionnalité, il l'a compris-il l'a testé en fonction de ce qu'il a compris. Donc après on peut partager très souvent les tests de qualité avec le Product Owner pour n'avoir un espèce de patchwork qui dise bah voilà comment je teste ma fonctionnalité et au moment où on dit cette phrase on va amener cette phrase, donc cette phrase fonctionnelle avec le Product Owner. Le Product owner peut dire non, c'est pas du tout comme ça que je l'explique, c'est pas du tout comme ça que je voulais te le transmettre

et donc y a une espèce de déjà de feedback, de look, de feedback qui se met en place directement avant même que le code soit écrit donc dans les bonnes pratiques c'est plutôt pas mal et on voit donc le les les développeurs plus le PO parce qu'il faut quand même faut quand même un product owner qui puisse aller chercher le besoin, travailler les choses, ça c'est un peu indispensable.

On peut pas demander aux développeurs de faire tout ça, on en fait en fin de compte on charge. Le développeur, là, on lui donne la charge de réalisation du projet. La concrétisation, finalement des idées. Et puis donc de en application. Mais le Product Owner lui va devoir en fait guider, entre guillemets, donner les bonnes, les bonnes pistes pour pouvoir amener tout ça dans la tête développeur donc c'est un petit peu, c'est un duo. Et on peut rajouter par-dessus tu veux un tech-lead bon pour l'organisation, juste pour pouvoir faire en sorte que les développeurs l'équipe de développement fonctionne très bien. Parce que c'est pas forcément évident.

Et est-ce que du coup tu penses que l'agilité, donc, que ce soit avec du Scrum avec du Kanban a un impact positif ou non au regard de la maintenance d'application ?

Oui, alors quand on parle de maintenance, il y a des rythmes qu'il convient, qu'il convient mieux d'utiliser. Personnellement, je connais beaucoup le scrum en fait on fait 3/4 du temps, on fait du scrum. J'ai déjà connu des organisations, des équipes qui fonctionnent en kanban. Je connais même des organisations, des équipes qui font du scrum-ban, donc du mélange un petit peu des 2. On en gros, ils prennent ce qu'ils veulent, moi ça va très bien. C'est le but de l'agilité. On pense ce qu'on veut, dans ce qu'on dans l'agilité, pour avoir expérimenté un petit peu les 2 pour avoir expérimenté le kanban.

Et puis le kanban correspond carrément mieux en fait, à la maintenance. Pourquoi ? Parce que on n'a pas de rythme de sprint, donc on n'est pas cantonné en fait à se limiter à un rythme de 2 semaines, 3 semaines ou le rythme du sprint. Tandis que sur un kanban, on fait les les tâches au fil de l'eau et en fonction de leur priorité. Donc à tout

moment, à chaque étape, une tâche de maintenance, donc un bug en l'occurrence n'a pas forcément besoin d'avoir une valeur. On sait juste qu'on doit traiter au maximum X bug ou on a une capacité en fait dans l'équipe de pouvoir traiter X Bug par étape OK et en fonction de la l'urgence de la chose, un bug peut passer devant un autre.

Donc pour une maintenance d'applications. C'est juste parfait et on peut même si on veut mettre en place des mécanismes tels que la Fastlane, donc la fastlane, c'est là une espèce de de passe droit. C'est un peu comme faire un autre fixe quoi, c'est de de de mettre en place tout de suite la correction d'application parce que c'est ultra urgent y a un problème grave et cetera. On le corrige tout de suite et donc elle supplante toutes les priorités et donc à utiliser à bon escient parce que je connais des équipes qui vont y mettre toute la backlog. Et hop, c'est parti, ça doit aller le plus vite possible. Bah non, c'est pas du tout comme ça que ça marche, c'est pas priorisé du tout.

Là l'importance dans le kanban, c'est de prioriser un maximum autant dans le scrum c'est pas très grave puisque en fait on s'est engagé dans le scrum à livrer donc c'est plus pour du build le Scrum. Finalement on s'est engagé à livrer toutes les X semaines une valeur, une valeur ajoutée à l'application et on livre au fil de l'eau sprint par sprint et donc l'engagement s'arrête au niveau du sprint c'est à dire que dans le scrum par exemple, on interdit que quelqu'un débarque en plein milieu du sprint, il me demande où ça en est l'application quoi tu verras bien à la fin du sprint.

Si c'était si tu me demandes quelque chose qui a été livré sur le sprint précédent, oui, tu peux demander où ça en est à priori, c'est livré ou pas, ça peut être en décalage. On est à ce moment, on essaie de communiquer au plus possible. Mais le kanban va correspondre pile-poil en finalement à ce qu'on attend d'une maintenance applicative et va nous permettre en fait, d'être réactifs tout simplement à ce qui pourrait se passer vis-à-vis de l'application. Donc dans le cadre d'une TMA à ou d'une d'une matière en condition opérationnelle ou le Kanban serait parfait, tout simplement.

On a fait le tour des questions que que je voulais te poser. Maintenant tu avais peut-être d'autres thématique que tu voulais aborder ?

J'ai pas d'autres thématiques. Après, pour être tout à fait honnête, moi j'ai beaucoup eu d'expériences de maintenance, c'est pas mon domaine préféré. Si je devais choisir, je préfère écrire quelque chose, de prendre from scratch. J'aime beaucoup en fait cette matière brute à travailler qui que sont les idées hein, tout simplement, mais ça s'applique beaucoup. Ben dans mon cas, ça s'applique beaucoup sur des projets personnels et j'aime beaucoup en fait travailler une matière brute. C'est un peu comme si tu veux faire une sculpture quoi, tu repars d'un bout de terre et c'est parti. On avance, on fait, on fait une belle statue, et cetera. Ben c'est un peu ça. L'idée j'aime bien, je préfère faire ça pour moi. Personnellement, j'ai retrouvé beaucoup plus de entre guillemets, de récompense que de m'occuper d'une application et de la rénover et puis de passer à autre chose, et cetera. Parce que y a y a beaucoup de considérations, il y a beaucoup de politiques.

Typiquement, c'est un petit peu comme si tu jouais avec le bébé d'un autre et c'est pas très agréable. En fait mine de rien donc c'est ça donne une mauvaise image mais c'est volontaire hein ? C'est vraiment ça, c'est pas très agréable de faire de la maintenance malheureusement c'est les 3/4 de ce que l'on fait. Pourquoi ? Parce que les idées en fait sont pas nouvelles et très très souvent en fait pour pour traiter un besoin en particulier, Ben client va avoir déjà quelque chose en place. Donc lui, ce qu'il va demander, c'est de ne pas payer double coût, il ne va pas redemander à ce qu'on rénove entièrement la chose. On ne peut pas demander à ce qu'on jette complètement le truc. Et donc c'est plutôt rare au niveau professionnel.

On a créé un outil chez Auchan. From scratch. Le besoin m'était apparu par Xavier, que tu connais. Il m'a expliqué son besoin. Il m'a dit, j'ai un problème, j'ai besoin de faire ça et en gros, il voulait qu'on que je crée un outil qui permette de faire de l'infrastructure as code et alors que l'outil lui-même que donc le AWX en l'occurrence, qui est donc pour information, AWX du coup, si personne ne sait, c'est la partie web qui permet de déclencher des scripts Ansible et typiquement Ansible ne permet pas de faire ce genre de choses, donc en fait créer un outil qui permet de faire l'IAC. Ça semblait

évident. Et on était très étonné de ne pas trouver ce genre d'outil sur le marché, donc le créer de from scratch, c'était très agréable, voilà.

D'accord, j'aurais une question par rapport à mon expérience. J'ai beaucoup travaillé en mode build mais à Décathlon je n'ai fait que du run. Et j'ai jamais eu ce moment où on se dit : "Le build est terminé, on passe sur du run". C'est vraiment un moment qui existe ? Ou à un moment il ya juste une réalisation où l'on se dit que l'on est plus sur une phase de run que de build ?

C'est plus fluide, c'est plus fluide. Alors chez Norauto j'ai fait pas mal de build quand même malgré tout hein je critique beaucoup la maintenance mais quand on crée un produit comme ça on se pose des questions typiquement, pour moi, les phases les plus intéressantes, c'est en début de projet, mais la phase dont tu parles, là de passer du build à du run, elle est quasi-transparente en fait, pour l'équipe qui mène à bien un projet. Pourquoi ? Parce que ça se fait pas d'un coup brut, ça se fait pas d'un moment distinctif. Ça se fait au fil de l'eau à dire les choses qui arrivent sur la production passent en maintenance dès qu'elles arrivent en production, entre guillemets et donc les feedbacks d'utilisateurs vont tomber après.

Mais pendant ce temps-là, on développe des nouvelles choses, et cetera, donc c'est très, c'est assez fluide, c'est pas marqué du tout, y a fatalement une période où il y a plus de maintenance que de build. Mais c'est une maintenance. Enfin, c'est une, c'est une période si tu veux ou qui est souvent légère, entre guillemets, quoi dire. Il y a beaucoup moins de taf, on limite fortement le taf de build. Pourquoi ? Parce qu'on sait qu'on va avoir des retours, et cetera quand c'est prévu. Souvent les clients se disent non j'ai pas de retour donc j'en ai pas donc j'en aurai pas. Puis quand ça arrive Ah mince y a beaucoup de retour. Bon bah fais du run alors la priorité au run. On dit souvent ça, "priorité au run" mais c'est le seul moment en fait, tout ça arrive.

En fait, contre les moments où les projets, à mon avis, de par mon expérience, des projets où on te demandera de faire décrire un produit de le concevoir from scratch, d'en faire toutes les études, de faire les POC, et cetera, de mener un bien. Le produit

de l'écrire, de le tu vois de l'amener tout ça. Ça prend une période assez assez longue suivant la taille du produit suivant le le besoin. Ça prend une période assez longue et donc on on sent pas venir le le run au final ou quelque part. Ça arrive, ça, ça vient. Typiquement, ce qu'on développe hier devient un un problème de maintenance de demain donc c'est pas un c'est pas quelque chose de distinctif, quoi, c'est vraiment fluide.

D'accord, merci beaucoup d'avoir pu faire cette interview.