INFORMATIQUE

LES PRATIQUES DE L'ÉQUIPE AGILE

2e édition

Définissez votre propre méthode

THOMAS THIRY

Préface de Marc Lainez

deboeck SUPÉRIEUR

LES PRATIQUES DE L'ÉQUIPE AGILE

Définissez votre propre méthode

INFORMATIQUE

LES PRATIQUES DE L'ÉQUIPE AGILE

2e édition

Définissez votre propre méthode

THOMAS THIRY

Préface de Marc Lainez

deboeck SUPÉRIEUR

Pour toute information sur notre fonds et les nouveautés dans votre domaine de spécialisation, consultez notre site web : **www.deboecksuperieur.com**

2e édition
Rue du bosquet 7, B - 1348 Louvain-la-Neuve

Dépôt légal :
Bibliothèque Nationale, Paris : septembre 2022
Bibliothèque royale de Belgique, Bruxelles : 2022/13647/116

ISBN 978-2-8073-3970-5

Sommaire

Partie III
L'entreprise

Préface

Par Marc LAINEZ, entrepreneur et professeur invité à l'UCL,
université catholique de Louvain, Belgique

Depuis des siècles, nos sociétés sont en pleine mutation. Les évolutions les plus marquantes de notre histoire ont été provoquées par de grandes découvertes scientifiques ou des progrès technologiques majeurs. Chacun de ces changements a créé de nouvelles manières d'interagir entre individus, provoquant une remise en question des modes de fonctionnement du monde du travail.

Certaines de ces pratiques, bien que d'un autre âge, restent encore bien ancrées dans la mémoire collective. Un des meilleurs exemples est l'héritage de la première révolution industrielle, dont les caractéristiques sont le mieux représentées par le Taylorisme. Taylor propose un modèle dit « scientifique », où l'organisation est à la recherche du « meilleur moyen possible de produire » et justifie le nom de cette approche par la rigueur qu'il appliqua dans ces tâches.

Taylor base ses théories sur une ultra-spécialisation, mais à l'époque dont nous parlons, ces tâches sont bien souvent limitées à visser un boulon ou porter une caisse d'un point A à un point B. Les études effectuées sont donc focalisées sur chacune de ces tâches, cherchant comment serrer un boulon plus vite, porter plus de caisses sur une journée, etc. Taylor introduit ce qu'on pourrait appeler aujourd'hui les premiers « managers », censés surveiller le travail des ouvriers et « optimiser » leur temps pour en maximiser le retour sur la production. Ces concepts sont ensuite repris par Henry Ford, obsédé par la vitesse, qui va élaborer une méthode travail basée sur le Taylorisme et introduire ce que l'on connait aujourd'hui comme le « travail à la chaîne » où chaque poste est une étape dans l'élaboration du produit final.

Bien qu'ayant leurs limites et pouvant être considérées aujourd'hui comme des approches inhumaines, le Taylorisme et le Fordisme sont, à leur époque, un moteur économique qui permit à nos sociétés d'évoluer vers d'autres innovations technologiques. La société de consommation telle que nous la connaissons aujourd'hui débute à cette époque en permettant l'émergence d'une classe moyenne.

Les premiers ordinateurs marquent le début d'une nouvelle ère et une accélération de son évolution. Ils sont apparus assez tôt dans l'histoire industrielle de nos sociétés,

mais l'innovation majeure qui a tout bouleversé est l'invention du transistor en 1947. Il permet de remplacer différents composants des premiers ordinateurs pour les rendre moins encombrants et surtout plus fiables. De 1947 à la fin des années 50, ces innovations, bien que capitales dans la conception des ordinateurs que nous connaissons aujourd'hui, n'ont pas impacté monsieur tout le monde. Ce n'est que plus tard, dans le milieu des années 60, que l'informatique explose et que les premiers mainframes apparaissent, changeant radicalement la manière dont les gouvernements et les entreprises traitent et stockent l'information. On voit alors apparaître des ordinateurs emblématiques comme le PDP-8 qui marquent le début de l'utilisation de l'informatique dans les laboratoires et les bureaux.

Cette arrivée de l'informatique sur le lieu de travail change les compétences requises et, par la même occasion, la nature du travail. Certaines tâches deviennent automatisées, d'autres nécessitent désormais de nouvelles compétences jusque-là inconnues. Le monde du travail connaît une mutation sans précédent et, dans la foulée, les habitudes des consommateurs changent pour plus de choix ainsi que plus de personnalisation des biens et des services.

Il aura fallu plusieurs décennies pour voir apparaître le Taylorisme et le Fordisme, il n'aura fallu que 20 ans pour entrer dans l'ère de l'informatique.

Plus d'ordinateurs signifie plus de demandes pour des programmes permettant aux machines d'accomplir des tâches rébarbatives. Les premiers métiers de l'informatique se démocratisent et créent de nouveaux défis aux entreprises. Cette explosion cambrienne de l'informatique crée des situations sans précédent pour lesquelles les entreprises doivent apporter des changements structurels dans leur hiérarchie et dans la gestion des relations entre collaborateurs.

L'arrivée des microprocesseurs dans les années 70 accélère une nouvelle fois ces changements en faisant exploser, de manière exponentielle, la puissance des ordinateurs pour les 20 années qui suivent, décuplant les problématiques rencontrées lors des débuts de l'informatique. Concevoir un logiciel reste encore aujourd'hui une tâche ardue. On ne produit pas un logiciel comme on produit une voiture.

Suivant cette évolution fulgurante, l'apparition d'internet ne fait que multiplier par cent l'impact des technologies sur nos vies. Permettant à plusieurs réseaux d'être interconnectés de par le monde, les opportunités qu'internet apporte changent une nouvelle fois, dans les années 90, notre conception de la gestion de l'information. Comme 30 ans auparavant, de nouvelles compétences sont désormais nécessaires, de nouveaux modes de travail et de nouvelles techniques d'interaction naissent. L'adoption massive de l'email, l'apparition de la vidéoconférence, les outils de collaboration à distance, changent nos entreprises et notre société une nouvelle fois. En conséquence, nos habitudes de consommation subissent une nouvelle transformation. C'est l'apparition des premiers sites populaires de commande en ligne et le début des réseaux sociaux. On peut désormais faire ses courses en ligne et payer dans la foulée. Nous basculons dans l'ère de l'instantanéité.

Au même moment, le téléphone mobile fait son apparition. D'abord local, il se transforme en appareil connecté début des années 2000 pour terminer par sa mutation la plus récente en « smartphone » lors de l'apparition de l'iPhone en 2007.
Il ne s'agit là que de quelques exemples. Mais la liste des mutations profondes de nos sociétés durant les 2 derniers siècles est bien plus longue. Une analyse approfondie de chacune de ces mutations nous permet de dire que celles-ci sont de plus en plus rapides et surtout, inévitables.
Avec ces changements technologiques, nos aspirations, nos désirs, nos modes de fonctionnement en tant qu'individu changent radicalement et s'adaptent à ces révolutions.
Si le changement est inévitable, pourquoi cherchons-nous toujours à l'éviter à tout prix dans nos entreprises, et plus particulièrement lors de l'élaboration de nos projets informatiques ? Nous avançons d'expérience en expérience en formulant sans cesse de nouvelles hypothèses pour repousser les limites de notre connaissance et de nos sociétés. En somme, nous appliquons la méthode scientifique et le bon sens à tout ce que nous entreprenons. Sauf parfois en entreprise
Nous sommes de nouveau au début d'une nouvelle ère. Provoquée par de nouveaux bouleversements technologiques : l'intelligence artificielle, la blockchain, l'explosion des périphériques connectés, la liste est longue. Nous ne pouvons pas continuellement appliquer d'anciennes recettes dans un monde où des changements radicaux arrivent à un rythme toujours plus rapide. Nous devons nous éduquer, nous former à d'autres approches, d'autres modes de penser le travail, revenir à cette culture de l'expérimentation et l'appliquer au sein de l'entreprise d'aujourd'hui et de demain.
Durant des années, l'agilité a été négligée par les cursus traditionnels. Bien que faisant une entrée timide dans nos écoles et nos universités, il est important de régulièrement dresser un instantané des différentes méthodes disponibles. Dans ce livre, j'ai pu retrouver toutes les approches et pratiques qui ont pu, au cours de ma carrière, m'être utiles. L'auteur y regroupe le savoir de nombreux membres de la communauté agile avec habileté et pose les bases d'une compréhension en profondeur des différentes approches issues de l'agilité. Il serait impossible de décrire en un seul ouvrage tout ce que ce mouvement apporte aujourd'hui au monde de l'entreprise, mais ce livre est une véritable carte qui vous aiguillera vers d'autres ressources qui vous permettront d'y voir plus clair et de trouver l'outil qu'il vous faut, quand vous en avez besoin.

Préambule

L'agilité a été inventée en grande partie par des développeurs en informatique pour se donner une alternative aux méthodologies basées sur la planification. Cette nouvelle approche permet de délivrer bien plus de valeur au client.
Elle nécessite cependant :

1. Une totale réorganisation du travail.
2. Un haut niveau de discipline.
3. Apprendre à programmer autrement.

L'ensemble des membres de l'équipe agile devront maîtriser ces nouveaux concepts pour obtenir un réel gain.

Vous trouverez dans ce guide un condensé d'information concernant l'agilité et les pratiques qui y sont liées de près ou de loin.
Le public visé est l'ensemble des personnes amenées à travailler dans une équipe agile ou avec elle. Que leur rôle soit de développer, de discuter avec le client, d'encadrer l'équipe, de gérer l'hébergement de l'application... tous auront besoin de réellement comprendre l'agilité pour pouvoir la mettre en œuvre avec impact dans leur quotidien.
Ce livre a été écrit avec une attention particulière à ne pas exiger de connaissance préalable dans le domaine de l'agilité ou même de l'informatique. Tous les concepts et le vocabulaire techniques sont expliqués dans les notes de bas de page.
Les différents chapitres sont regroupés en 3 parties. La première, *Le développement logiciel*, reprend de façon chronologique les évolutions de la gestion de projet en informatique et explique en détail et de façon concrète les principes agiles ainsi que les méthodologies scrum et extreme programming.
La seconde, *La chaîne de valeur*, étendra ces principes et ces pratiques à l'ensemble des acteurs impliqués grâce à DevOps et à la livraison continue. Lean aidera à voir la chaîne de production de valeur différemment et Kanban s'en inspirera pour donner encore plus de flexibilité et d'efficacité aux équipes.
Enfin, la dernière partie, *L'entreprise*, élargira encore le sujet sur l'organisation toute entière. Comment appliquer l'agilité à grande échelle tout en respectant ses principes

fondamentaux ? Quel cadre contractuel donner à cette nouvelle façon de délivrer le projet au client ? Comment Lean peut-elle aider les start-up à trouver un modèle économique durable ?

La plupart des chapitres se terminent par une conclusion qui reprend les éléments importants du chapitre sous la forme de questions à se poser pour vous encourager à faire le lien entre vos éventuels problèmes du quotidien et les solutions proposées dans le chapitre en question.

La gestion de projet et les méthodologies

Les projets informatiques coûtent cher. Très cher.

Dès lors, on comprend bien que les clients ont besoin d'un prix clair avant de se décider à acheter. Ils ont également besoin de savoir quand ils recevront ce qu'ils ont acheté. En tant que client, vous n'en attendriez pas moins de n'importe quel service, c'est normal.

Cependant, notre industrie souffre d'un grand manque de précision dans ces chiffres. En cause, les nombreux risques auxquels font face tous les projets informatiques.

Pour de multiples raisons, les dates de livraison prévues ne sont que très peu respectées. On ne peut pas s'y fier et on ne peut donc pas prévoir nos autres projets en conséquence. C'est même devenu une blague dans le métier : « *Livraison prévue dans 1 an, comptez donc le double !* » Plus personne n'y croit.

Mais un projet livré est un projet rescapé : certains n'en arrivent même pas là et sont tout bonnement abandonnés en plein milieu. Le budget est dépassé, la patience manque, le besoin n'est de toute façon plus d'actualité… Ce gâchis est bien plus fréquent qu'on ne le pense, en particulier pour des clients internes pour lesquels aucun contrat n'oblige à s'acharner face à un échec couru d'avance.

Un problème plus insidieux et présent dans tous les projets de développement est *l'augmentation du coût des changements*. L'effort nécessaire pour faire un changement après 1 an de travail n'est plus sans commune mesure avec ce qui aurait été requis au début du projet. Le client ne comprend pas et a l'impression de se faire escroquer.

Mais le pire pour lui est quand il obtient ce qu'il voulait et que ça ne marche pas. Les bugs l'empêchent de travailler et lui semblent innombrables. À quoi bon utiliser l'application ?

Cela nous mène au dernier risque. On ne mesure pas bien le gaspillage représenté par tout ce qui est développé, avec les coûts qu'on connaît, et qui n'est *jamais utilisé* ou n'apporte que très peu de valeur au client. C'est probablement le gâchis le plus stupide : on aurait mieux fait de ne rien faire et de s'occuper d'autre chose…

On l'a compris, les risques sont nombreux et obtenir la valeur promise ne sera pas si simple. C'est la raison d'être des méthodologies, agiles ou pas. L'organisation seule peut nous amener à réduire ces risques et à plus de sécurité. On pourra mieux définir

un coût à l'avance pour notre client. Maximiser ce coût et donc la rentabilité de nos développements. Cette stabilité réduira notre stress et augmentera donc notre bien-être au travail.

> Améliorez constamment tous les processus de planification, de production et de service, ce qui entraînera une réduction des coûts.
>
> *W. Edwards Deming*

Introduction

J'ai eu la chance de participer, au début de ma carrière, à plusieurs projets informatiques aux contextes extrêmement différents. J'en ai tiré une leçon qui m'a permis de voir mon travail d'une autre façon et qui m'a guidé dans le reste de mes expériences professionnelles.

À la fin de mes études, j'ai fait un stage de 3 mois et demi dans une entreprise de consultance en informatique. Je n'avais jamais entendu parler de méthodologie, de gestion de projet ou quoi que ce soit et cela ne m'intéressait de toute façon pas. J'allais développer des applications et c'était tout ce que je voulais.

J'ai donc commencé dans le département IT de la société. Compte-tenu de la taille modeste de la société, ce département était constitué d'un directeur, un responsable « réseaux et systèmes » très polyvalent et un développeur. Leur principale activité de développement consistait à fournir des applications web (ASP.Net) aux autres départements de la société et aux nombreux consultants. Cela allait de l'application de gestion des besoins des clients à l'encodage en ligne des timesheets[1] pour les consultants. Je ne le réalisais pas alors, mais aucune organisation stricte n'était en place et on parait systématiquement au plus pressé en trouvant des solutions ad hoc à nos problèmes.

Mon stage se passa, et, une fois mon diplôme en main, je me fis engager dans ce même département.

Sébastien

Les choses avaient un peu évolué depuis mon stage et on ne me proposa pas de poste en tant que développeur. Un deuxième programmeur avait déjà rejoint les rangs de l'équipe et mon nouveau manager, Sébastien[2], faisait alors le constat de mauvais résultats et d'une insatisfaction générale de nos clients internes. Il voulait que je m'occupe de cette problématique au lieu de simplement ajouter un produit à la quantité des logiciels écrits.

---

[1] Les *timesheets*, ou *feuilles de temps* sont les rapports de chaque employé sur l'emploi de son temps du mois ou de la semaine écoulée à des fins comptables. En l'occurrence cela permettait de facturer les clients sur base de ces chiffres.

[2] Tous les noms de ce chapitre sont fictifs.

À cette époque, quand un de nos clients internes – un comptable par exemple – avait besoin d'une nouvelle fonctionnalité, il entrait dans le bureau et s'asseyait à côté du principal développeur pour lui expliquer ce qu'il avait en tête. Le développeur en question prenait alors des notes et essayait ensuite de s'en occuper rapidement. Il devait parfois répondre à la douloureuse question du délai de livraison et s'en sortait en donnant quelques chiffres, mal à l'aise de dénoncer sa propre incompétence si le chiffre n'était pas assez bas. Il faisait revenir le comptable une fois qu'il avait terminé pour lui montrer le résultat sur son ordinateur et, si son avis était positif, le changement était déployé sur le site de production, en copiant les fichiers modifiés sur le serveur directement. Avec une application qui redémarre et recompile[3] automatiquement dans ce cas. Inutile de préciser que les utilisateurs éventuels qui perdaient alors leur connexion n'étaient pas très contents.

N'ayant moi-même aucune expérience, je ne voyais pas directement les problèmes. Ils étaient cependant nombreux. La qualité était très faible et les utilisateurs souvent mécontents. Nous n'avions aucun moyen de prioriser les demandes. Si le comptable insistait assez fort, le travail en cours à ce moment-là était interrompu pour s'occuper de sa demande. Si, le lendemain, le directeur demandait une amélioration de ses rapports, on s'arrêtait à nouveau pour se concentrer sur cette requête. Et il n'était pas rare de mettre celle-ci aussi de côté car un problème en production bloquait tous les utilisateurs. Bref, ce fonctionnement menait à beaucoup de travail inachevé, qui n'était au final pas réellement repris. Et même si le développement aboutissait, rien ne disait que cette demande était vraiment utile et prioritaire pour l'entreprise.

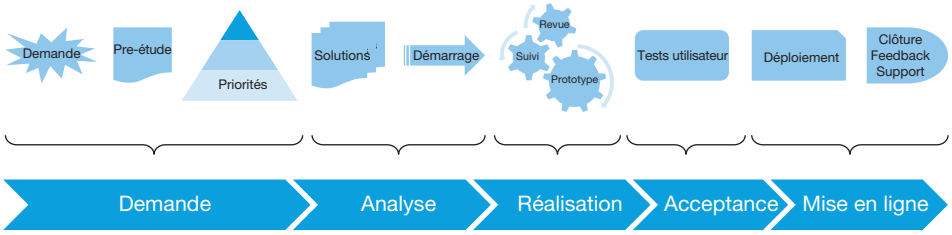
Au final donc, un fonctionnement peu efficace et un mécontentement général des utilisateurs et de la direction.

La première tâche que Sébastien m'a confiée a été de déterminer un processus idéal de fonctionnement qui nous permettrait d'éviter tous ces problèmes.

Aidé de mon manager, j'ai donc dessiné un processus où centralisation des demandes, priorisation, estimations, analyse, tests et validations venaient structurer notre approche. En appliquant cette nouvelle méthodologie, nous avons pu obtenir de bien meilleurs résultats et un retour de la confiance des utilisateurs ainsi que du directeur qui avait enfin la main sur les priorités et une vue sur l'avancement des projets. Les déploiements eux-mêmes causaient moins de problèmes grâce à une planification, une communication très claire et une application clairement indisponible pendant les opérations.

C'est à ce moment que j'ai compris que ***la façon dont on s'organisait dans l'équipe était primordiale si on voulait fournir des applications de qualités à nos clients pour les aider dans leur travail***.

[3] Compiler = exécuter un compilateur qui lit le code source et crée le programme correspondant.



Dessin du nouveau processus mis en place

Martine

Ce modèle fonctionna bien quelques temps, et après 2 ans, la direction se fit remplacer. Et avec elle, Sébastien également. Ma nouvelle responsable, Martine, avait une certaine expérience sur des gros projets dans de grandes entreprises de télécom et avait une vision très claire de comment un projet devait être géré. Avec sa nouvelle position, un objectif lui avait été donné : remplacer l'application de gestion des besoins des clients dans laquelle les commerciaux renseignaient leurs contacts.

Quand la direction lui demanda pour quand tout pourrait être prêt, elle répondit que tout serait livré 3 mois plus tard. Elle ne perdit ensuite pas de temps et engagea une nouvelle recrue pour réaliser l'analyse de la nouvelle application. Cette personne commença donc à rencontrer les commerciaux et imagina, dans Powerpoint, une nouvelle application.

En parallèle, Martine me chargea de réaliser un planning pour les 3 prochains mois et la livraison promise pour le 31 juillet. N'ayant de toute façon pas beaucoup de marge, je m'en suis tenu à diviser le temps accordé entre l'analyse, le développement et une phase de test pour tout valider les 2 dernières semaines de juillet.

Semaine après semaine, je mettais ce planning à jour pour correspondre à la réalité. Car celle-ci n'était pas vraiment rose. L'analyse a pris au moins un mois et le développement n'avançait pas comme prévu (ou comme espéré devrais-je dire). On a pourtant commencé la réalisation de certains écrans analysés sans attendre que les autres le soient, mais rien n'y fit : plus le temps passait, plus il devenait évident qu'on n'y arriverait pas.

L'inquiétude de Martine ne faisait que monter et, avec elle, la pression qu'elle nous mettait pour avancer. Début juillet il fallut se rendre à l'évidence : on allait droit dans le mur. Martine nous demanda alors de faire ce qu'on fait dans ces cas-là : venir travailler le week-end pour rattraper le retard.

Le vendredi suivant, on souhaita un bon week-end à nos collègues des autres départements et on s'apprêta à venir travailler le lendemain. L'ambiance du week-end fut particulière, mais motivée. On allait y arriver. Ce n'était finalement pas si terrible. Ce qui le fut, en revanche, c'est le lundi matin. Quand tous nos collègues revinrent reposés, nous, on n'était pas franchement motivés.

La semaine se passa et un matin, Martine nous annonça qu'il fallait travailler un week-end de plus. Pour rattraper le retard. Sinon on n'y arriverait pas.
On se regarda tous et, ne voyant pas vraiment d'alternative, on est venus travailler le week-end.
Ce mois de juillet-là, sur les 31 jours du mois, on en passa 28 au bureau à travailler.
Les semaines passant, le moral baissa toujours plus et, avec lui, la productivité. Le temps passé à corriger les bugs de plus en plus nombreux à cause de la fatigue de l'équipe grignotait le temps disponible pour les développements restants. C'était un cercle vicieux. Les journées se raccourcissaient à leur minimum contractuel en raison du manque d'énergie et de motivation de tout le monde.
Le week-end du 31 juillet arriva enfin. On avait réussi à assembler bon gré mal gré une application qui contenait encore de nombreux bugs qu'on se promettait de corriger plus tard. Un week-end de plus au bureau. Tout ne se passa pas comme prévu le samedi, et le dimanche servit à corriger les derniers gros problèmes. On pouvait rentrer chez nous, mission accomplie.
Lundi matin, l'application était en ligne et fonctionnait correctement. Je croisai des commerciaux dans les couloirs pour leur demander ce qu'ils en pensaient. Ils ne semblaient pas très intéressés par le sujet et se contentèrent de dire qu'ils regarderaient. Je réalisai alors tout doucement que l'intérêt pour notre projet était très limité. Six mois plus tard l'application serait entièrement remplacée.
Cette expérience reste une des plus mauvaises que j'ai vécues, au travail mais elle fut également une leçon marquante : ***la façon d'organiser l'équipe et de travailler pouvait me gâcher la vie et m'ôter toute envie d'aller travailler***. Ça valait la peine de s'y intéresser.

Étienne

À peine remis de nos émotions, un nouveau projet pointa le bout de son nez. Une intégration avec un système de recrutement et de gestion des besoins clients fournit par une start-up.
Dès le début du projet la même dynamique s'installa, un planning fut réalisé sur base de la date de livraison demandée.
Mécontents des mauvais résultats du projet précédent, les membres de la direction décidèrent de remplacer Martine par Etienne, un homme d'expérience qui trouva les mots pour les rassurer.
Son arrivée marqua une nouvelle fracture dans la façon de gérer nos projets. Il accorda ostensiblement sa confiance à l'équipe pour trouver et appliquer des solutions.
De mon côté, je m'intéressais depuis quelques temps à l'agilité. Je lisais tout ce que je trouvais sur internet sur le sujet, j'allais à des conférences et des formations. C'est ce qui m'amena à lui proposer, après la livraison du projet demandé par la direction, une nouvelle approche où notre application principale serait livrée à

intervalles réguliers, 2 fois par mois. Étienne nous soutint et nous encouragea dans cette voie.

Les mois qui suivirent furent riches en expériences et améliorations. Chaque matin, on regardait ensemble la liste des demandes prioritaires dans notre outil de suivi de tâches (Redmine) et on se répartissait le travail avec l'objectif de la release[4] à venir.

On essaya la technique du planning poker[5] pour estimer ce qu'on pouvait réaliser comme travail pour les 2 semaines qui suivaient et cela nous plut bien plus que notre façon de faire précédente.

On mit également en place une vérification systématique de chaque développement par un autre développeur. C'est-à-dire que quand un développeur finissait son travail, il cherchait un autre développeur prêt à essayer sa nouvelle fonctionnalité et à lui dire ce qu'il en pensait pour l'améliorer. On fut alors surpris du nombre de bugs découverts à ce stade-là...

Après un certain temps, les déploiements bimensuels (tous les mardis soir, c'était devenu un rituel) commencèrent à nous prendre du temps. On entreprit de scripter[6] de plus en plus d'étapes, jusqu'à l'application des scripts SQL sur base du numéro de version.

Dernière expérience marquante : le tableau et ses post-it pour suivre notre travail nous avait toujours semblé redondant avec Redmine, mais on fut malgré tout tentés de l'essayer. À notre surprise, on fut très vite convaincus par cette approche très concrète du travail restant, affiché en grand au mur toute la journée.

Fier de la qualité offerte à nos clients internes, Étienne décida de proposer nos services à des clients externes et de démarrer une activité de projets au forfait.

Il embaucha pour cela 2 personnes supplémentaires, et notre équipe de 8 personnes se lança alors dans la réalisation de projets pour diverses sociétés clientes.

Pendant cette période, j'ai appris à être fier de notre savoir-faire. D'avoir la reconnaissance de clients qui nous disaient ne pas être habitués à des solutions autant sur mesure, grâce à notre approche flexible et à un prix défiant souvent la concurrence. Ils en redemandaient, et nous aussi.

[4] Une release = la livraison d'une nouvelle version du logiciel.

[5] Planning poker = méthode d'estimation du travail (cf. *Planning poker* dans le chapitre *6. Autres pratiques agiles*).

[6] Scripter = écrire de petits morceaux de code, les scripts, pour automatiser une tâche.

Partie I

LE DÉVELOPPEMENT LOGICIEL

Cette première partie vise à expliquer précisément ce que l'agilité représente pour le développement logiciel.

Pour bien comprendre une pratique, il faut commencer par connaître son origine et comprendre les problèmes qu'elle résout. C'est pour cette raison que le premier chapitre synthétise l'histoire de la gestion de projets informatiques, des années 1960 à aujourd'hui.

Les chapitres suivants reprennent, dans l'ordre chronologique, chacune des méthodologies avec les solutions qu'elle apporte et les éventuels problèmes qu'elle provoque : *waterfall*, qui encadre une profession désorganisée par une pléthore de règles strictes et standardisées, suivi par de nombreuses méthodologies légères qui définiront ensemble le développement logiciel agile tel qu'on le connaît aujourd'hui. Cependant, seules les méthodologies les plus populaires sont effectivement expliquées en détail dans leur propre chapitre afin de se focaliser sur ce qui est le plus pertinent de nos jours.

Après les deux premiers petits chapitres plutôt théoriques (*1. Historique* et *2. Le modèle waterfall*), les deux chapitres suivants (*3. Méthodologies légères* et *4. Agile*) visent à bien comprendre les principes de l'agilité en les détaillant d'une façon qui se veut claire et pratique.

La première méthodologie agile qui vient à l'esprit de la plupart des gens est Scrum. Ses règles sont simples et le chapitre qui lui est dédié (*5. Scrum*) est agrémenté de conseils et d'anecdotes.

Le plus gros chapitre de ce livre est consacré à *Extreme Programming*, et cela pour une bonne raison : en plus des bonnes pratiques organisationnelles, cette méthodologie apporte une panoplie très riche de pratiques techniques qui sont indispensables à une application réelle de l'agilité, avec de réels résultats.

Cette partie se clôture par une technique supplémentaire qui vise à rapprocher les demandes des clients et la compréhension que les équipes de développement en ont : le *Behavior Driven Development*.

Chapitre 1

Historique

La gestion de projet informatique dans le temps

Pour comprendre l'agilité il est nécessaire de comprendre l'histoire de la gestion de projet.

Ce chapitre passe rapidement en revue les différentes périodes de l'histoire de l'informatique. Les chapitres suivants détailleront chacune des méthodologies correspondantes.

Les débuts

Dans les années 60, l'informatique fait son apparition dans les entreprises sous la forme de grands mainframes[7] fournis par des sociétés comme IBM à des prix très élevés. L'utilité de ces machines est généralement d'automatiser des tâches répétitives et de faciliter le travail administratif des différents départements de l'entreprise.

Pour cela, on a besoin de programmeurs capables de réaliser le logiciel sur mesure et ayant des connaissances dans des langages tels que FORTRAN, BASIC ou COBOL. Il n'existe alors bien-entendu aucun cursus académique en informatique et on recrutera donc souvent cette première génération d'informaticien dans d'autres départements de l'entreprise même. Pour ces tâches minutieuses sur ces ordinateurs coûteux on sélectionnera généralement des personnes expérimentées ayant déjà prouvé leur professionnalisme. Leur expérience passée dans l'entreprise va leur donner une connaissance du métier bien utile pour s'assurer de la valeur fournie par leurs logiciels.

Ces nouveaux travailleurs vont fonctionner dans un cadre relativement libre que chaque société va vouloir définir pour elle-même de façon à obtenir de bons résultats et cela de façon prévisible. En effet, derrière chaque demande/projet le management peut légitimement se demander quel budget et quelles échéances il faudra prévoir.

Cela fonctionne bien un certain temps, mais très vite les programmes deviennent un peu plus compliqués qu'un simple affichage d'une liste d'enregistrements. On rajoute

[7] Un mainframe = un ordinateur central très puissant mais très imposant utilisé dans les grandes entreprises bien avant l'avènement des *personal computers*.

de plus en plus de possibilités. D'autres programmeurs viennent se joindre au développement pour répondre aux demandes de plus en plus nombreuses.
Cette complexité (qui apparaît en réalité bien plus vite qu'on ne le pense) va entraîner de plus en plus de problèmes retardant la livraison d'un produit fini et utilisable par les employés. Le retour sur investissement est de moins en moins bon et il n'est pas rare de voir des projets compter leur retard en années.
Tout cela mène à un mécontentement général du management et à un désir d'établir davantage d'ordre dans ce fonctionnement anarchique où chacun fait à sa façon.

Waterfall

Le domaine s'élargissant, de nombreuses formations académiques en informatique voient le jour et les entreprises embauchent ces nouveaux informaticiens fraîchement diplômés. Leur profil jeune, inexpérimenté et impatient contraste fortement avec celui de la génération précédente.
C'est dans ce contexte que dans les années 70 de nombreuses méthodologies de gestion de projet voient le jour. Elles visent à définir les rôles de chacun et les processus à suivre pour cadrer les équipes de développement grandissantes.
Faute de modèle plus adapté, elles sont basées sur le fonctionnement en application dans l'industrie (manufactures) et le monde de la construction. Un des principes fondamentaux sera donc la planification pour maximiser l'efficacité. On donnera par la suite le nom de *modèle waterfall (modèle en cascade en français)* à la logique commune à ces méthodologies.
Celles-ci souffrent malheureusement toutes des mêmes défauts : un excès de bureaucratie et un manque de flexibilité qui engendrent une perte de motivation. Les résultats espérés en matière d'efficacité ne sont pas au rendez-vous et les projets sont tout autant en retard, si pas plus.
Mesurer ces résultats est difficile. C'est une tâche que le Standish Group (groupe d'experts en informatique) s'est donné en 1994 afin de rédiger ce qu'ils ont appelé le « chaos report » : une étude basée sur de très nombreux projets afin d'établir la proportion de réussites et d'échecs. Leurs résultats ont surpris une grande partie des professionnels du métier : seuls 16,2% étaient considérés comme des réussites[8]. Depuis, ils publient chaque année une nouvelle édition de ce rapport et analysent les évolutions.
Le modèle waterfall va cependant rester prédominent pendant de nombreuses décennies. Malgré sa diminution de popularité dans les discours, il reste aujourd'hui l'inspiration prépondérante dans l'organisation des projets de développement dans la plupart des grandes entreprises.

[8] Pour interpréter ces chiffres correctement il faut bien comprendre la méthode retenue pour analyser ces projets. En effet, la définition de « réussite » est ici prise dans le sens d'un projet qui est réalisé dans le temps et le budget prévus tout en délivrant ce qui était demandé. On peut regretter que la véritable valeur apportée au client ne rentre pas en compte pour définir si le un projet est une réussite ou non.

Méthodologies légères

Dans les années 90, de plus en plus de professionnels du développement réfléchissent à des alternatives au modèle waterfall. Deux décennies de bureaucratie à l'excès ont mené à une situation où le travail a perdu de son sens et où plus personne ne prend de plaisir à développer des applications.
Plusieurs méthodologies allant dans ce sens vont alors apparaître. Elles auront toutes certains points en commun. Elles seront souvent appelées *méthodologies légères* en comparaison avec les méthodologies plus lourdes et détaillées du modèle waterfall.
Tout d'abord elles ne tentent plus d'être prédictives, mais visent plutôt à être adaptives. C'est-à-dire qu'elles ne vont plus se baser sur un planning détaillé à respecter, mais qu'elles vont plutôt adopter une manière de fonctionner qui leur permettra de s'adapter aux changements quand ils surviendront, quels qu'ils soient.
Ensuite, elles vont se recentrer sur les gens et réfléchir à ce qu'il faut leur fournir pour les aider à atteindre les objectifs fixés.
Enfin, la documentation exigée va être grandement réduite (mais pas forcément supprimée) afin de ne garder que ce qui semble effectivement être utile.
Les plus connues de ces méthodologies sont citées ici, mais seule 2 seront détaillées dans leur propre chapitre, celles qui sont les plus populaires de nos jours (Scrum et Extreme Programming).

Crystal (1992)

Cet ensemble de méthodologies a été créé en 1992 par Alistair Cockburn[9]. L'idée est de ne plus appliquer un modèle unique à toutes sortes de projets mais bien d'avoir un modèle adapté à la taille et au contexte de chaque projet.
Elle est relativement facile à adopter.

DSDM (1994)

Un ensemble de vendeurs et d'experts s'est rassemblé pour organiser en 1994 le *DSDM Consortium*. Ils ont rassemblé leurs bonnes pratiques afin d'aider l'ensemble de la profession à s'améliorer.

Scrum (1995)

Jeff Sutherland et Ken Schwaber présentent en 1995 leur méthode *Scrum* à la conférence OOPSLA. Elle offre un cadre simple mais strict pour délivrer un produit par itération.

FDD (1999)

Feature Driven Development est une méthodologie développée par Jeff De Luca et Peter Coad basée sur des itérations de 2 semaines.

[9] Développeur américain ayant participé à la création du manifeste agile (cf. chapitre *4. Agile*).

Extreme Programming (1999)

En 1999 sort la première édition du livre de Kent Beck dans lequel il décrit les pratiques utilisées par lui et son équipe sur un projet chez Chrysler. Ces idées sont très innovantes et comptent aujourd'hui parmi les bonnes pratiques les plus importantes de la profession.

Agile

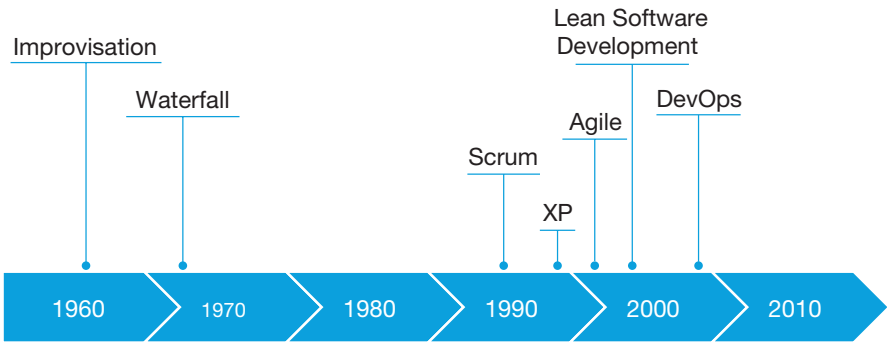
Les auteurs de différentes méthodologies légères se réunissent en 2001 et rédigent un document établissant les valeurs et les principes qui leur sont commun à tous : c'est le *manifeste agile*[10].
C'est également l'occasion pour eux de donner un nom à ce mouvement : *le développement agile*.

Lean software development

En 2003 sort un livre écrit par Mary et Tom Poppendieck : *Lean Software Development*[11]. Ils cherchent à appliquer les principes Lean[12] à l'informatique. Lean découle des pratiques de Toyota, reconnu depuis de nombreuses décennies dans le monde de l'industrie automobile pour son efficacité.

DevOps

Le mouvement DevOps démarre en 2009. Il rassemble toutes les initiatives visant à améliorer la collaboration entre les développeurs et les équipes opérationnelles de façon à envisager la chaîne de valeur dans son ensemble et y appliquer les bonnes pratiques agiles et lean de bout en bout.
En plus de changements d'organisation et de culture, de nombreux outils ont été créés par cette communauté qui se développe : *Chef, Puppet, Ansible, Docker*...



[10] Le manifeste agile est disponible en français à l'adresse http://agilemanifesto.org/iso/fr/manifesto.html
[11] Mary Poppendieck et Tom Poppendieck, *Lean Software Development*, Pearson Education (Us), 2003.
[12] Méthodologie de gestion de la production inspirée du modèle de Toyota (cf. chapitre *13. Lean*).

Chapitre 2

Le modèle waterfall

La structure et la normalisation comme solution à tout.

Objectifs

Comme on l'a vu, les méthodologies basées sur le modèle waterfall (ou « modèle en cascade ») visent à régler le problème des projets de développement qui ne délivrent pas dans un délai raisonnable à cause d'un manque flagrant d'organisation. L'idée est d'obtenir de meilleurs résultats au travers de l'application de la méthode scientifique, en s'inspirant du monde industriel et de la construction.

Les bonnes pratiques seront définies une fois pour toutes et applicables à toutes les entreprises et toutes les situations.

La logique principale est de planifier, de prévoir à l'avance un maximum de détails, et de respecter par la suite ce qui a été prévu en évitant tout écart.

Il sera alors nécessaire d'effectuer de nombreux contrôles tout au long du projet pour vérifier que tout se passe selon ce plan. Dans cette même optique de contrôle, on prévoira également une traçabilité exhaustive de toutes les actions prises au travers de documents, rapports et comptes rendus.

Ce modèle servira de base à de nombreuses méthodologies qui verront le jour dans les années 70 et 80. Elles proposent des variations sur les mêmes principes de base.

Elles vendent des manuels précis sur les rôles à définir, les processus à suivre, les formulaires et rapports à rédiger. Les nombreuses formations proposées constituent un business solide qui fonctionne à plein régime sur base de la promesse de projets enfin rentables.

Malheureusement, la réalité n'est pas toujours celle-là et de gros problèmes apparaissent.

Le plus visible étant l'énorme quantité de bureaucratie qui vient s'ajouter au travail des équipes. Chaque action doit être consignée, chaque changement doit faire l'objet d'une demande officielle, réalisée au travers de formulaires, validée par un comité...

Pour de nombreux informaticiens, dont la passion concerne la technique et non les procédures, le quotidien ressemble de moins en moins à leur métier et de plus en plus à un travail administratif. Cela entraîne naturellement une démotivation très importante chez ces employés.

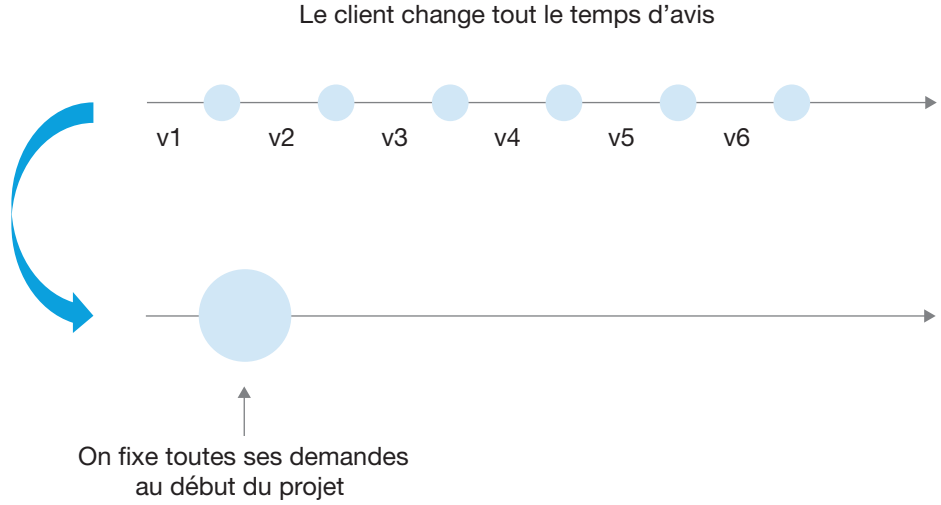
De plus, l'ajout de procédures impacte la rapidité de développement, et on ne constate pas forcément d'amélioration dans le respect des engagements de planning et de budget.
Tout cela passerait encore si, au final, la valeur fournie au client était améliorée par ces méthodologies. Or on constate au contraire que les clients qui reçoivent leur produit de nombreux mois (si pas années) après l'avoir demandé n'en ont pas (ou plus) l'utilité escomptée.

Raisonnement

L'approche waterfall semble à première vue censée, et, malgré les mauvaises expériences passées, on est vite tenté d'appliquer ces mêmes raisonnements dans notre quotidien sans leur donner le nom de waterfall. Pour ne pas tomber dans ce piège, Il est important de bien comprendre les raisonnements qui mènent à une logique de type waterfall.

Besoins du client

Tout d'abord, en ce qui concerne la récolte des besoins du client on constate systématiquement que celui-ci exprime une demande et change d'avis par la suite. Cela pose évidemment d'énormes problèmes d'organisation et impacte le planning ainsi que le budget. Une solution vient instinctivement à l'esprit : fixer toutes les demandes avec le client une fois pour toutes au début du projet et refuser les changements par la suite.



Cette façon de fonctionner facilite à priori les choses pour donner un prix au client avant de démarrer le projet.

LES PRATIQUES DE L'ÉQUIPE AGILE

Définissez votre propre méthode

L'agilité se base en grande partie sur le bon sens de chacun et propose des solutions en apparence simples. Il s'avère souvent difficile de changer réellement les choses dans son organisation et d'obtenir un véritable impact.

L'auteur pose les vraies questions et propose des réponses concrètes, illustrées par des situations vécues.

Ce livre vise à vous offrir une réelle compréhension des différentes méthodologies et pratiques liées à l'agilité afin que vous puissiez **déterminer par vous-même l'organisation** qui correspond le plus à votre contexte pour enclencher une transformation progressive et efficace.

Toutes les méthodologies liées à l'agilité sont expliquées de cette manière :

- Scrum
- Extreme programming
- Lean software development
- Kanban
- Mob programming
- Livraison continue
- DevOps
- Team topologies
- Lean startup
- Projets au forfait agiles
- L'agilité à grande échelle

Dans cette 2e édition, l'auteur a développé deux nouveaux chapitres consacrés aux méthodes de **mob programming** et **team topologies.**

Plus qu'un guide pour **organiser une équipe agile**, cet ouvrage détaille également l'**approche technique** à suivre pour obtenir de véritables résultats. Les nombreuses pratiques d'extreme programming y sont détaillées, telles que l'intégration continue, les tests unitaires ou le refactoring.

Ce livre s'adresse donc à toute personne amenée à travailler dans une équipe agile : scrum master, product owner, développeur, testeur, chef d'équipe, coach... afin que tous partagent la même compréhension complète des principes et pratiques de l'agilité et puissent la mettre en œuvre avec impact.

Thomas Thiry *a une expérience concrète de l'agilité et des pratiques DevOps au travers des entreprises qu'il accompagne dans leur processus d'amélioration.*
Il a également enseigné ces pratiques dans l'enseignement supérieur en Belgique (haute école).

ISBN : 978-2-8073-3970-5

deboeck SUPÉRIEUR

www.deboecksuperieur.com