UNIVERSITÉ DU QUÉBEC
À MONTRÉAL

RÉDUCTION DES INEFFICACITÉS DANS LE DÉVELOPPEMENT LOGICIEL D'UNE COMMISSION SCOLAIRE

PAR
MAXIME PELLETIER

RAPPORT DE SYNTHÈSE PRÉSENTÉ À L'UQAM
COMME EXIGENCE PARTIELLE À L'OBTENTION DE
LA MAÎTRISE
EN GÉNIE LOGICIEL

MONTRÉAL, LE 29 NOVEMBRE 2017

PRÉSENTATION DU JURY

CE RAPPORT DE SYNTHÈSE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Mme Sylvie Trudel, directrice de projet,
Département d'informatique à l'Université du Québec à Montréal

M. Louis Martin, directeur de la maîtrise en génie logiciel,
Département d'informatique à l'Université du Québec à Montréal

M. Normand Séguin, professeur,
Département d'informatique à l'Université du Québec à Montréal

IL A FAIT L'OBJET D'UNE PRÉSENTATION DEVANT JURY ET PUBLIC

LE 29 NOVEMBRE 2017

À L'UNIVERSITÉ DU QUÉBEC À MONTRÉAL

REMERCIEMENTS

Afin de mener à terme ce projet de maitrise, j'ai pu compter sur la précieuse aide de ma directrice, Sylvie Trudel. Elle ne m'a jamais laissé tomber et m'a poussé à ne pas lâcher, à me rendre jusqu'au bout, tout au long de ces quatre dernières années où j'ai même dû changer de projet à trois reprises. Lors de nos nombreuses rencontres, j'ai pu apprécier à la fois ses connaissances très poussées du domaine, mais aussi son plaisir lorsqu'elle nous les partage.

Ensuite, je tiens à souligner le travail de toute l'équipe de développement de la commission scolaire des Affluents pour avoir permis la réalisation de ce travail. De ce groupe, la volonté du coordonnateur de l'équipe, Patrick Fortin, a été cruciale car elle a mis la table pour amorcer les travaux d'amélioration du processus de développement.

Finalement, j'ai pu constater l'impact qu'un tel chantier a sur une vie familiale et l'importance d'avoir le soutien de tous les membres de sa famille afin d'avancer et terminer le travail. Un merci spécial à Maude et les enfants, Alexis, Mariane et Lorie.

RÉDUCTION DES INEFFICACITÉS DANS LE DÉVELOPPEMENT LOGICIEL D'UNE COMMISSION SCOLAIRE

MAXIME PELLETIER

RÉSUMÉ

Il y a de plus en plus de pression sur l'équipe de développement pour développer de nouvelles applications et maintenir celles en place. Puisque l'ajout de personnel n'est pas possible dans un contexte d'austérité vécu par les commissions scolaires, il faut augmenter l'efficacité de l'équipe de développement pour faire plus avec les mêmes effectifs. Nous avons donc cherché à réduire le *rework*.

Nous avons tout d'abord analysé l'effort fourni pour un échantillon de 11 projets afin de classifier quelle portion était du *rework*, c'est-à-dire de l'effort pour refaire ou modifier quelque chose de déjà terminé. Cette analyse nous a permis d'identifier deux grandes sources de *rework*, soit les problèmes techniques, ainsi que les changements aux applications suite à leur premières utilisations par les utilisateurs.

Pour diminuer ces sources de *rework,* nous avons mis en place deux changements dans le processus de développement. Tout d'abord, nous avons ajouté une activité de prototypage afin de diminuer les changements découlant des premières utilisations. Dans un deuxième temps, nous avons remplacé le *framework* « ASP.NET Web Forms » par celui « ASP.NET MVC » afin de diminuer le nombre d'erreurs techniques lors de la phase de programmation.

Les changements ont été validés avec trois projets. Les résultats obtenus démontrent une amélioration notable quant à la diminution du *rework*, et laissent présager un impact très positif si les améliorations étaient appliquées à tous les projets futurs.

REDUCING THE SOFTWARE DEVELOPMENT INEFFICIENCIES OF A SCHOOL BOARD

MAXIME PELLETIER

ABSTRACT

There is more and more pressure on the development team to develop new applications and maintain the existing ones. Being in an austerity context, it isn't possible to hire new employees, so we must improve the team efficiency to do more with the same workforce. Thus, we have aimed to reduce the rework.

First, we analyzed a sample of 11 projects to classify which proportion of the effort was rework, meaning the effort to redo or modify something that has already been done. This analysis allowed us to identify two major sources of rework, namely technical problems, and changes to applications following their utilization by users.

To decrease these sources of rework we have implemented two changes in the development process. First, we added a prototyping activity to decrease the changes deriving from the utilization. In a second step, we replaced the framework "ASP.NET Web Forms" by "ASP.NET MVC" to reduce the number of technical errors during the programming phase.

The changes have been validated with three projects. The results show a significant improvement in the reduction of rework and suggest a very positive impact if the improvements were applied to all future projects.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CS	Commission scolaire
CSDA	Commission scolaire des Affluents
DIS	Développement, implantation, support
GUTI	Guichet unique des technologies de l'information
MEES	Ministère de l'éducation et des études supérieures
MVC	Modèle-Vue-Contrôleur
PO	Propriétaire de produit (de l'anglais *product owner*)
SAC	Service à la clientèle
SEI	Software Engineering Institute
STI	Service des technologies de l'information
TI	Technologie de l'information
UQAM	Université du Québec à Montréal

Glossaire

Défaillance	Rupture du service, qui peut se produire lorsqu'une faute se produit (IEEE, 2010).
Défaut	Présence d'un problème dans un logiciel, qui produit des résultats incorrects s'il n'est pas corrigé (IEEE, 2010).
Erreur	Une action humaine qui est la cause ou l'origine d'une faute (IEEE, 2010).
Faute	Manifestation d'une erreur dans un logiciel (IEEE, 2010).
Rework	Effort qui n'aurait pas à être fourni si le travail déjà fait aurait été correct, complet et cohérent (Fairley et Willshire, 2005).

INTRODUCTION

Ce rapport résulte d'un projet de fin d'études dans le cadre de la maîtrise en génie logiciel à l'Université du Québec à Montréal (UQAM). Ce projet a été réalisé dans l'équipe de développement logiciel de la Commission scolaire des Affluents (CSDA) au cours des années 2016 et 2017.

Le travail effectué vise à réduire les inefficacités de l'équipe de développement logiciel du service des technologies de l'information (STI) en analysant l'effort fourni en trouvant les sources de gaspillage. Pour ce faire, des changements seront apportés au processus de développement des applications afin de tenter de réduire les sources de gaspillage préalablement identifiées et, ainsi, augmenter l'efficacité de l'équipe.

Ce rapport est structuré comme suit. Le premier chapitre décrit le contexte dans lequel s'est déroulé le projet ainsi que la problématique à résoudre. Le second chapitre fait un résumé de la littérature en lien avec la problématique. Le troisième chapitre explique l'approche qui a été adoptée pour tenter de corriger la problématique. Le quatrième chapitre expose l'état de la situation initiale avant que des changements soient apportés. Le cinquième chapitre explique les changements apportés ainsi que les résultats obtenus suite à ces améliorations. Finalement, le sixième chapitre fait un retour sur les différents aspects du travail effectué et les apprentissages qui ont eu lieu.

CHAPITRE 1

CONTEXTE ET PROBLÉMATIQUE

1.1. Contexte

1.1.1 La Commission scolaire des Affluents

La commission scolaire des Affluents (CSDA) a été créée en 1998 suite à la fusion de deux commissions scolaires (CS). Elle regroupe 68 établissements d'enseignement, soit 50 écoles primaires, 14 écoles secondaires, 4 centres pour adultes, le tout pour un total d'environ 38 000 élèves. Cela en fait la cinquième commission scolaire en importance au Québec. On compte plus de 5 500 employés répartis dans les écoles et services administratifs. La CSDA est sous la gouvernance du conseil des commissaires, élus par la population. Le territoire desservi comporte 8 municipalités.

La CSDA a un budget annuel de 415 M$ (2017-2018) dont 11,4 M$ (2,7%) sont accordés à l'ensemble des services administratifs (Affluents, 2017).

La CSDA comporte 8 services administratifs :

- Direction générale
- Service du secrétariat général et des communications
- Service des technologies de l'information (STI)
- Service des ressources matérielles
- Service des ressources humaines
- Service des ressources financières
- Service des ressources éducatives
- Services éducatifs particuliers et complémentaires.

Chaque service est subdivisé en secteurs, relevant habituellement d'un gestionnaire. La figure 1.1 montre le détail de ces secteurs.

Commission scolaire des Affluents

Organigramme 2016-2017

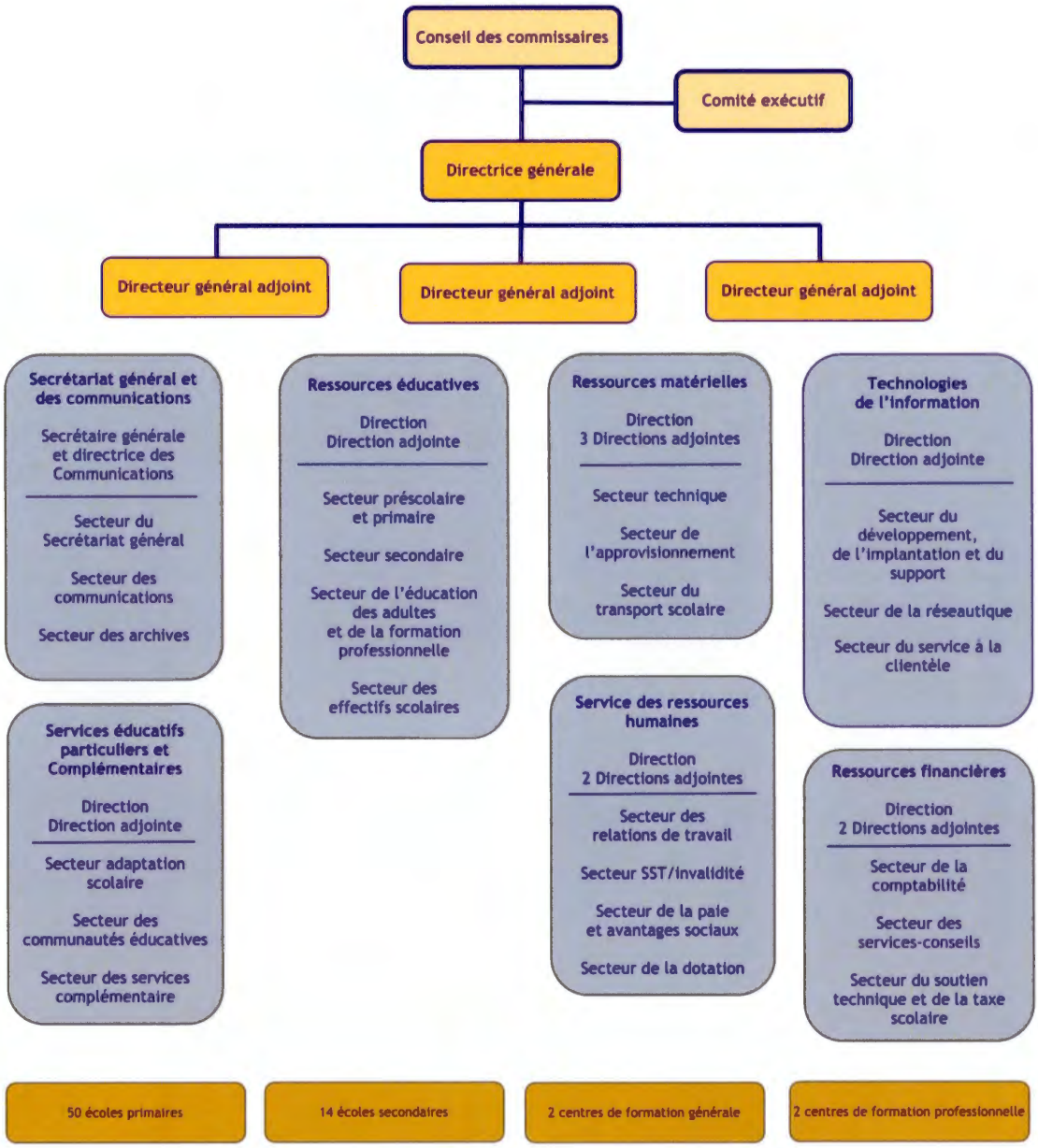

Figure 1.1 Organigramme de la CSDA

1.1.2 Le service des technologies de l'information

Le STI est divisé en trois secteurs, chapeautés par un directeur (voir la Figure 1.2 - Organigramme du STI de la CSDA).

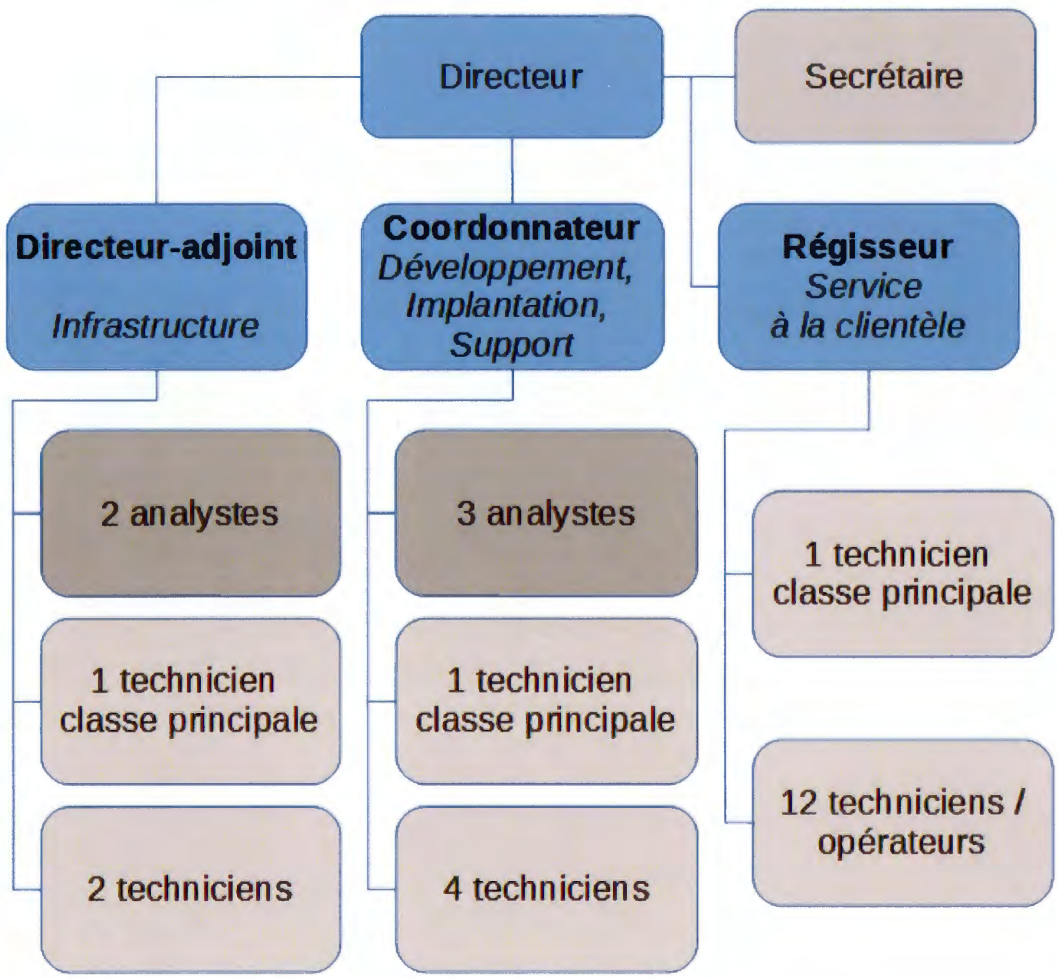

Figure 1.2 Organigramme du STI de la CSDA

Tout d'abord, il y a le secteur du service à la clientèle (SAC), qui est la première ligne face aux utilisateurs finaux (élèves, employés, parents), dans les écoles et les services

administratifs. Il s'occupe du soutien pour le matériel, pour l'installation de logiciels et pour les accès aux systèmes. Le guichet unique des technologies de l'information (GUTI) fait partie de cette équipe et est le point central du soutien aux utilisateurs. Quand un utilisateur a besoin de soutien, il doit contacter le GUTI, qui pourra ensuite soit traiter la demande, soit la rediriger au bon secteur ou à la bonne personne. Ce secteur comporte 1 gestionnaire (régisseur), 1 technicien classe principale[1] et 12 techniciens/opérateurs.

Le deuxième secteur est celui de l'infrastructure. Ses principales responsabilités sont le réseau (filaire et sans-fil), la téléphonie (filaire et mobile), les annuaires, le système de courriel et les serveurs. À cela s'ajoute aussi certains systèmes tels que Lotus Notes, Wordpress et Moodle. Ce secteur comporte 1 gestionnaire (directeur-adjoint), 2 analystes, 1 technicien classe principale et 2 techniciens.

Finalement, il y a le secteur développement, implantation, support (DIS). L'équipe de développement qui en fait partie voit au développement d'applications pour répondre aux besoins de ses clients. Une fois les applications développées, cette équipe est aussi responsable de leur maintenance (voir section 1.1.3 pour plus de détails sur cette équipe). L'équipe de support de ce secteur s'occupe de l'installation, des mises à jour et du soutien des applications de la société GRICS (voir section 1.1.4 pour plus de détails sur cette société). Cette équipe soutient aussi certains systèmes SQL, MS Access et *Reporting Service*s. Ce secteur comporte un gestionnaire (coordonnateur), trois analystes, un technicien classe principale et quatre techniciens.

---

[1] Un technicien classe principale est un technicien ayant une bonne expérience et des responsabilités accrues.

1.1.3 L'équipe de développement

La CSDA étant parmi les plus grosses CS du Québec, on y retrouve une équipe dédiée au développement. Cette équipe de développement, qui fait partie du secteur DIS, livre plusieurs nouvelles applications chaque année pour les écoles et les services administratifs de l'organisation, en plus d'assurer la maintenance de son catalogue d'applications déjà en place. Le catalogue contient plus de 70 applications maintenues par une équipe de six personnes, soit deux des analystes, le technicien classe principale et trois des techniciens[2].

1.1.4 Liens avec l'externe

Tout comme les autres commissions scolaires, la CSDA entretient des relations avec le Ministère de l'éducation et des études supérieures (MEES) pour diverses raisons administratives.

Un lien existe aussi avec la société GRICS pour l'acquisition ainsi que le soutien de nombreux systèmes informatiques. Cette dernière est un organisme qui a été mis sur pied pour répondre aux besoins des commissions scolaires du Québec. Elle est une entité indépendante des CS et du MEES, mais son conseil d'administration est constitué de membres des CS provenant de partout au Québec.

Certaines CS sont en contact avec la CSDA afin de s'entraider, autant au niveau de la pédagogie qu'au niveau administratif.

---

[2] Le troisième analyste et le quatrième technicien étant dédiés au soutien.

1.1.5 Politiques et contraintes opérationnelles

Les montants alloués par le MEES à une CS sont déterminés en fonction du nombre d'élèves sur son territoire. La seconde source de financement d'une CS est la taxe scolaire. Étant donné qu'une CS ne peut pas déclarer de déficit (article 279 de la loi sur l'instruction publique (LégisQuébec, 2017a)), elle doit tirer le maximum de son budget, sans jamais le dépasser.

Depuis la fin de l'année 2011, les CS du Québec sont aussi soumises à la Loi sur la gouvernance et la gestion des ressources informationnelles des organismes publics et des entreprises du gouvernement (LégisQuébec, 2017b). L'article 1 de cette loi stipule entre autres qu'elles ont le devoir « d'optimiser les façons de faire en privilégiant le partage et la mise en commun du savoir-faire, de l'information, des infrastructures et des ressources ».

Comme pour tous les employés de la CS, les membres de l'équipe de développement sont soumis à une convention collective. Un point important de la convention collective stipule que la semaine de travail des employés est de 35 heures. Les heures additionnelles travaillées sont considérées comme du temps supplémentaire. Cependant, elles doivent être approuvées au préalable en raison d'un budget restreint. Il n'est toutefois pas pratique courante d'y avoir recours pour du développement, à moins d'une urgence.

Finalement, l'ajout ou le retrait d'un employé permanent est un processus long qui peut être lourd de conséquences. Il est donc important de bien prévoir le surplus de travail, de même que les baisses de travail. Ce contexte exige donc d'être le plus efficient possible, en utilisant le personnel en place.

1.1.6 L'environnement de développement et ses caractéristiques

Étant donné qu'il était nécessaire d'avoir des environnements *Microsoft* pour l'utilisation des logiciels de la GRICS, le choix a été fait par le passé de privilégier les outils et les environnements de développement *Microsoft*. Tous les développeurs utilisent donc *Visual Studio*. Les applications Web sont développées avec le *framework* ASP.NET, avec une architecture Web Forms et le langage VB.NET, le tout accessible par des serveurs Web IIS, fonctionnant sur un système d'exploitation *Windows*.

Pour accélérer le développement, l'équipe utilise *Entity Framework* pour l'accès aux données, ainsi que la librairie *DevExpress* pour avoir des interfaces plus riches et développer plus rapidement.

Pour quelques anciens développements faits en PHP, une machine virtuelle avec *Zend Studio* est utilisée pour programmer. Ces applications fonctionnent sur des serveurs *Apache* et sur *Red Hat*, une distribution du système d'exploitation *Linux*.

La gestion du code source se fait avec *Team Foundation Server* de *Microsoft* (autant pour les applications ASP.NET que pour celles en PHP). Le suivi des demandes et du temps se fait avec l'outil *Redmine*. Il n'y a toutefois pas d'intégration entre ces 2 outils.

Pour chaque serveur de production, il existe un serveur de développement. Ceci permet de détecter un maximum de problèmes avant la mise en production des applications, en plus de les rendre disponibles à certaines personnes au besoin. Dans certains cas, une version de préproduction est installée sur les serveurs de production afin de faire des tests plus approfondis ou de rendre disponibles des environnements de formation.

Pour faciliter le partage de documents et d'informations entre les membres de l'équipe de développement, et aussi avec toutes autres parties prenantes, un espace *SharePoint* est créé pour chaque projet.

Bien que certains essais aient été faits par le passé, aucun outil de test n'est utilisé, ni d'outil de construction automatique (*build*).

Au fil du temps, une base de connaissances a été constituée sur un site utilisant le système de gestion de contenu *Wordpress*. De plus, certains éléments comme des normes de programmation sont déposés dans l'environnement *SharePoint*.

Pour accélérer le développement, un modèle de base d'application a été développé et est utilisé pour chaque nouveau développement. Les améliorations apportées à ce modèle sont reconduites aux applications existantes dans la mesure du possible.

Pour diminuer le couplage entre les systèmes, plusieurs services Web ont été développés pour les échanges d'informations entre les différents systèmes. Cela permet de réutiliser des méthodes déjà développées, car plusieurs besoins reviennent souvent d'une application à une autre, surtout pour celles d'un même client.

1.1.7 Processus de l'équipe de développement

Le STI souhaite tirer le maximum du nombre limité de personnel en développement à sa disposition pour répondre aux nombreuses demandes de sa clientèle. Pour atteindre cet objectif, l'équipe de développement s'inspire des pratiques Agiles (Beck *et al.*, 2001) qu'on retrouve dans la méthode SCRUM (Sutherland et Schwaber, 2013), mais cet exercice est complexe à faire dans le contexte de la CSDA. En effet, la présence de multiples clients et de multiples projets simultanés n'est pas toujours compatible avec les pratiques Agiles, quand les mêmes personnes sont affectées en même temps sur plusieurs projets. Plusieurs adaptations des méthodes Agiles ont donc été apportées au processus du STI en ce sens.

Lorsqu'une demande de changement est promue au développement, l'équipe de développement lance un processus inspiré de la méthode Scrum (voir figure 1.3). Le processus de développement décrit couvre les demandes qui nécessitent un

changement à une application existante ou encore la création d'une nouvelle application.

Dans un premier temps (sprint 0, soit l'étape #1), le chef de projet rencontre le client pour établir les bases du projet. Cela permet de s'assurer de la bonne compréhension du projet et de la portée de celui-ci. D'un autre côté, le client est informé des attentes de l'équipe de développement relativement à son implication pour favoriser le succès du projet.

Par la suite, les itérations de développement commencent (sprint X+1, soit l'étape #2). Une fois que le travail à faire est terminé, il est présenté au PO afin d'être validé. En collaboration avec le PO, le travail à faire pour le prochain sprint est décidé et le travail recommence. Cette boucle continue jusqu'à ce que le développement soit terminé. À ce moment, plusieurs tâches de fin de projet sont effectuées.

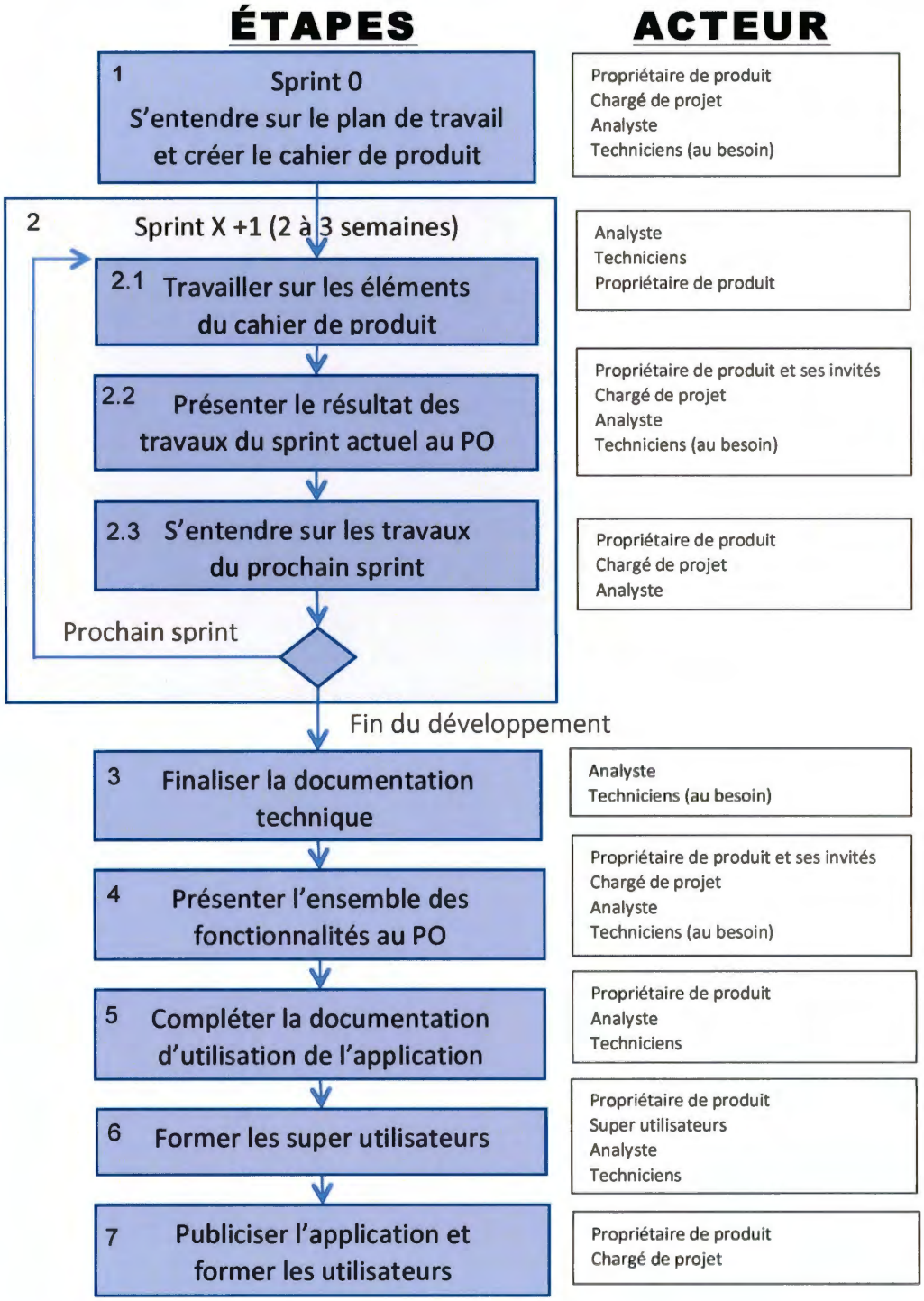

Figure 1.3 Processus de développement du STI

L'équipe de développement finalise d'abord la documentation technique du projet (étape #3). Ensuite, une présentation complète de l'application est faite au PO pour confirmer que le produit final répond aux attentes (étape #4). Après avoir élaboré les documentations pour l'utilisation de l'application (étape #5), les super utilisateurs sont formés (ceux qui feront le soutien de l'application) (étape #6). Finalement, la nouvelle application est publicisée lors de certaines rencontres de gestion et si nécessaire, des formations sont offertes aux utilisateurs (étape #7).

1.1.8 Rôles et responsabilités du processus actuel

Dans la présente section, nous allons présenter tous les rôles impliqués dans le processus de développement logiciel.

Gestionnaire de projet :

Veille au bon déroulement du projet et assure le lien avec le client. Supervise le travail de l'analyste. Responsable du démarrage du projet. Recueille tous les besoins auprès des services pour en faire l'analyse préliminaire. Propose les projets à la direction générale pour la planification annuelle. Il y a 1 gestionnaire de projet à la CSDA (coordonnateur).

Analyste :

Responsable de l'élicitation des besoins et de l'élaboration de la solution. Supervise les techniciens du projet. Programmation de la solution. Il y a 5 analystes à la CSDA, mais seulement 2 sont affectés au développement logiciel.

Technicien :

Programmation de la solution. Mise en production des applications. Suivi des applications dans leur environnement de production. Il y a 10 techniciens à la CSDA, mais seulement 4 sont affectés au développement logiciel, dont un comme technicien classe principale.

Client :

Manifeste ses besoins de développement auprès du STI et de la direction générale. Les clients sont habituellement les services administratifs de la CSDA ou, en quelques cas, des écoles.

Propriétaire de produit :

Représente le client et ses utilisateurs et soutient l'équipe de développement. Il y a un nombre très restreint de propriétaires de produit pour chaque client, c'est-à-dire que c'est souvent la même personne qui joue ce rôle pour plusieurs projets.

Utilisateur :

Les principaux utilisateurs des applications développées sont les employés de la CSDA, mais cela peut aussi être les élèves, les parents ou encore la population du territoire de la CS.

Tout au long du processus, les interactions suivantes se produisent entre les divers intervenants :

- Le gestionnaire de projet supervise les analystes;
- L'analyste supervise le ou les techniciens lors des projets;
- Le client informe le gestionnaire de projet de ses besoins de développement;
- Le propriétaire de produit exprime ses besoins au gestionnaire de projet au début du projet et à l'analyste lors de l'élicitation des besoins et lors du développement;
- Le propriétaire de produit valide le travail fait par l'analyste et les techniciens;
- L'utilisateur rapporte les fautes et les défaillances des applications au gestionnaire de projet;
- L'analyste et le ou les techniciens demandent des clarifications au propriétaire de produit.

1.1.9 Coûts des opérations du processus actuel

Les projets de développement ont un coût nul, étant donné que le salaire du personnel est déjà prévu au budget d'opération. L'impact est plutôt défini sur la productivité,

puisque si un projet prend beaucoup de temps, il reste donc moins de temps pour faire d'autres projets. Cependant, pour les projets avec une date butoir ferme, cela peut se traduire en pénalité ou en temps supplémentaire. Il n'y a habituellement rien de facturé aux clients pour un projet fait à l'interne.

Dans tous les projets, il y a un besoin de coordination et de collaboration avec le client. Dans le processus de développement de la CSDA (voir section 1.1.7 pour plus de détails), le propriétaire de produit, qui fait normalement partie du service client, passe beaucoup de temps à répondre à des questions et à valider les étapes terminées, ce qui représente en soit un coût non négligeable.

1.1.10 Facteurs de risques opérationnels

Le plus grand risque auquel fait face l'équipe de développement est de livrer un projet qui ne soit pas fonctionnel. Lors du développement, il n'est pas toujours possible de faire des tests qui représentent exactement l'environnement de production. Par conséquent, certains éléments peuvent ne pas fonctionner comme prévu lors de leur utilisation en production.

Un second risque est de livrer un projet en retard. En plus de demander plus d'effort que prévu, cela retarde les projets suivants. Du point de vue du client, il peut y avoir un impact financier si le projet est livré en retard. En effet, les programmes permettent souvent de diminuer certaines dépenses, donc les retards retardent les économies potentielles.

Un autre risque est la mauvaise analyse d'un projet. En effet, il est possible de trouver des besoins en cours de route qui n'avaient pas été exprimés au début du projet. Parfois ces besoins peuvent être ignorés, mais quand ceux-ci sont critiques à la viabilité du produit, il faut complètement revoir les échéanciers. Évidemment, tout cela a un impact important sur la planification de l'équipe de développement.

La majorité des applications développées dépendent des applications de la société GRICS. Étant donné que nous ne sommes pas propriétaires de ces applications, celles-ci peuvent changer n'importe quand et ainsi briser les dépendances qui existent. Dans le même ordre d'idées, certaines applications de la GRICS peuvent être complètement remplacées, ce qui peut demander des changements majeurs aux applications développées par le STI.

1.1.11 Performance

Il est possible de calculer l'effort passé sur chacun des projets pour connaître l'ampleur des développements effectués. Aucune mesure n'est cependant utilisée pour estimer la taille ou la complexité d'une application et pour faire une relation avec l'effort de développement.

Une CS étant un organisme relativement complexe qui a beaucoup de besoins mais des ressources limitées, les membres de l'équipe de développement sont souvent amenés à travailler sur plusieurs projets à la fois. Il est alors difficile de juger de la performance d'un projet, étant donné que plusieurs raisons peuvent biaiser les mesures recueillies. C'est pourquoi la CSDA a recourt à des sondages de satisfaction de la clientèle pour savoir s'ils ont bien performé du point de vue de ses clients.

1.1.12 Mesures de la qualité

La qualité du travail de l'équipe de développement est définie par plusieurs facteurs. Tout d'abord, la livraison des projets à temps est potentiellement le facteur le plus facile à vérifier. Par la suite, le nombre de défauts rencontrés après la mise en production est un indicateur important sur la qualité de l'application. Finalement, la satisfaction de la clientèle permet aussi de savoir si le projet est un succès ou non.

1.1.13 Modes opérationnels du processus de développement actuel

Il y a actuellement deux modes de fonctionnement principaux pour le développement : les nouveaux projets et la maintenance de projets existants. Dans le premier cas, on parle de développement pour un nouveau besoin, qui nécessite le développement d'une nouvelle application. Ces projets s'échelonnent habituellement sur plusieurs mois étant donné qu'il y a plus de choses à évaluer et à analyser, telles que la structure des données. Dans le deuxième cas, il s'agit d'apporter une ou plusieurs modifications à une application existante. Ces modifications peuvent modifier une partie déjà existante de l'application ou encore ajouter un nouveau module à l'application. Ces demandes s'échelonnent habituellement sur quelques semaines seulement, étant donné qu'une bonne partie du cadre de l'application est déjà définie. Cependant, plusieurs projets de maintenance ont la même ampleur que les nouveaux développements et s'échelonnent sur plusieurs mois. Il y a aussi d'autres demandes de maintenance de très petite envergure ne requérant que quelques heures de travail.

1.2. Problématique et opportunités

Avec les compressions budgétaires auxquelles font face les CS du Québec (IRIS, 2017; Radio-Canada, 2011, 2013) , les différents services se tournent vers les services TI afin d'informatiser et optimiser leurs processus d'affaire et ainsi réaliser des économies. Par conséquent, la pression est de plus en plus grande sur les services TI afin de produire plus d'applications, plus rapidement. De plus, avec l'arrivée de la Loi sur la gouvernance et la gestion des ressources informationnelles des organismes publics et des entreprises du gouvernement (LégisQuébec, 2017b), les CS sont tenues d'optimiser leurs façons de faire afin de réaliser des économies.

Dans ce contexte, le STI peut difficilement avoir recours à de nouvelles ressources. Il y a même certains cas où le STI doit lui aussi couper dans les dépenses. On se retrouve donc avec une équipe qui doit en faire plus avec moins.

Ceci nous amène à poser la question suivante :

Comment peut-on augmenter l'efficacité de l'équipe en tenant compte de son contexte?

Plusieurs facteurs du contexte viennent influencer l'efficacité de l'équipe ou son besoin d'améliorer son efficacité, dont:

1. Le développement dans un contexte multi-projets et multi-clients, qui oblige les membres d'équipe à changer de contexte de projet souvent, occasionnant des pertes de temps à chaque fois.
2. Le catalogue grandissant d'applications augmente le nombre de demandes de maintenance reçues à chaque année, ce qui laisse de moins en moins de temps pour les demandes de nouveaux développements.
3. L'élicitation initiale des besoins est parfois déficiente, ce qui résulte en une analyse incomplète. Les besoins initialement oubliés, identifiés en cours de

projets, retardent grandement les projets puisqu'ils nécessitent plus d'effort que s'ils avaient été identifiés en début de projet.

4. L'accessibilité limitée au PO, lors du processus de développement, ralentit le travail de l'équipe de développement, car cette dernière doit attendre avant d'effectuer certaines tâches ou doit faire des choix qui ne sont pas toujours les bons. Dans les deux cas, des pertes de temps peuvent être observées.
5. L'évolution des technologies permet de développer des applications de plus en plus intéressantes, mais cela nécessite un temps d'apprentissage. En effet, à chaque changement de technologie, un certain temps est nécessaire pour la maitriser. Donc, bien que le changement en question soit positif, il peut souvent avoir un impact négatif sur le temps de développement, au début du processus.
6. La faible documentation, qui rend certaines informations non disponibles lors des demandes de maintenance a pour conséquence de ralentir le travail. En effet, il faut dans ces conditions prendre plus de temps pour comprendre les explications de certaines décisions.
7. La perception d'une quantité importante de travail à refaire (*rework*), de sorte que si ce *rework* était diminué, l'efficience et l'efficacité seraient d'autant améliorées. Toutefois, ce *rework* n'est ni quantifié, ni sous contrôle.

1.3. Approche de notre projet

Nous avons choisi d'optimiser le processus de développement et de maintenance en cherchant à diminuer le *rework*. Nous pensons que cette approche permettra au STI d'améliorer son efficience, soit de diminuer l'effort nécessaire au développement et à la maintenance des applications. En d'autres termes, cela permettra de faire plus avec le même personnel et le même effort.

1.4. Portée et objectifs du projet

Les objectifs du projet sont les suivants :

- Objectif #1: Identifier et quantifier le *rework* initial de développement logiciel.
- Objectif #2: Améliorer le processus de développement afin de diminuer le *rework*.

La portée du projet inclut le processus de développement et de maintenance du STI. Le *rework* sera mesuré sur une dizaine de projets terminés pendant l'année scolaire 2016-2017, afin de nous permettre d'observer les plus grandes sources de *rework* et ainsi agir sur le processus aux endroits appropriés. Ensuite, nous déterminerons des améliorations au processus qui seront appliquées sur des projets pendant l'année scolaire 2017-2018 et pour lesquels nous mesurerons le *rework* et comparerons avec les données précédemment collectées et analysées.

CHAPITRE 2

ÉTAT DE L'ART

2.1. Qu'est-ce que le *rework*?

Au cours du développement d'un logiciel, le but est toujours de réussir à créer le logiciel qui répondra parfaitement aux besoins et ce, en n'ayant aucun défaut. Évidemment, ceci est très utopique, étant donné le grand nombre de sources d'erreurs possibles. Par conséquent, tous les projets vont tôt ou tard nécessiter du *rework*.

Dans le développement de logiciel, le *rework* représente l'effort additionnel nécessaire pour refaire un processus ou une activité qui n'avait pas été implémenté correctement la première fois, ou parce qu'il y a un changement dans les besoins du client (Huzooree et Ramdoo, 2015a, 2015b). Présenté autrement, « le *rework* est de l'effort qui n'aurait pas à être fourni si le travail déjà fait aurait été correct, complet et cohérent » (Fairley et Willshire, 2005).

Il est important de ne pas confondre le *rework* avec le *refactoring*. Ce dernier a pour but d'optimiser le travail effectué, tandis que le *rework* a pour but de corriger quelque chose qui ne répond pas aux besoins ou aux normes.

Il est possible de séparer le *rework* en trois grandes catégories : Évolution, Rétrospection, Correction (Fairley et Willshire, 2005).

Tableau 2.1 - Catégories de *rework*, tiré et traduit de (Fairley et Willshire, 2005).

Catégorie	**Définition**
Évolution	*Rework* causé par des facteurs externes, comme un changement de besoin, une contrainte de design, un élément de l'environnement du logiciel ou tout autre évènement impossible à prévoir.

Catégorie	Définition
Rétrospection	*Rework* pour améliorer la structure, les fonctionnalités, le comportement ou des attributs de qualité d'une version antérieure, afin de répondre aux besoins actuels, tout en développant la version actuelle.
Correction	*Rework* pour corriger un défaut découvert dans les versions actuelles et antérieures pendant les activités de revue, de test ou de démonstration de la version actuelle.

Le *rework* est un phénomène circulaire puisque celui-ci est causé par le travail de développement, mais est aussi corrigé par le travail de développement. Ainsi, le *rework* peut engendrer d'autre *rework*.

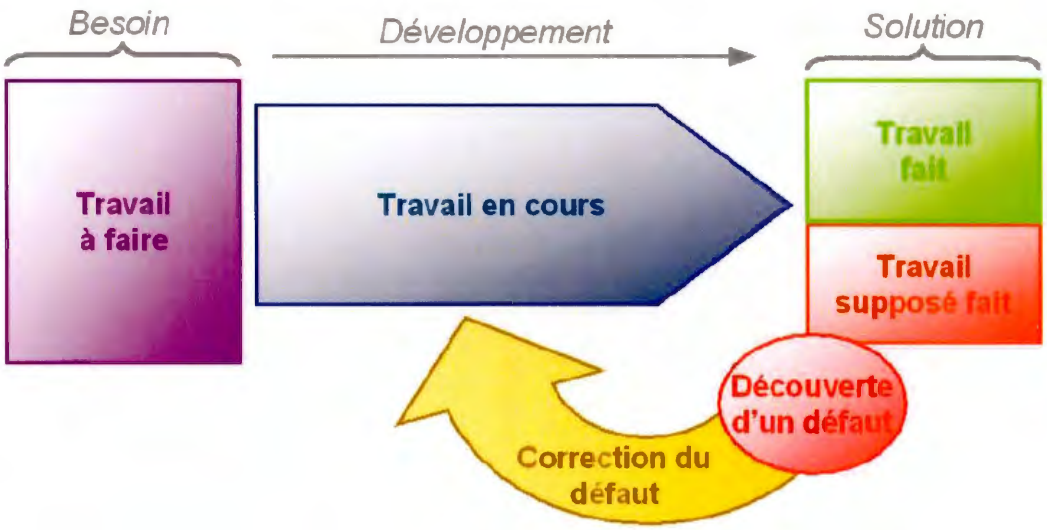

Figure 2.1 Le rework est un phénomène circulaire, tiré de (Tonnellier et Terrien, 2012)

L'impact du *rework* sur l'avancement d'un projet est doublement négatif, car il fait augmenter à la fois la durée du projet et l'effort total (Huzooree et Ramdoo, 2015a). Il n'est donc pas surprenant de voir que, parmi les approches pour réduire l'effort nécessaire au développement logiciel, la réduction du *rework* soit celle qui donne le

meilleur impact (Damm *et al.*, 2008). La raison principale qui explique cet impact est qu'il est moins couteux de trouver et corriger des défauts le plus tôt possible dans le processus (Damm *et al.*, 2008).

2.2. Causes du *rework*

Il n'existe pas de liste exhaustive de toutes les causes de *rework*. Cependant, il est possible de faire des grands regroupements.

Dans un premier temps, il est facile de comprendre que le manque de temps pour définir les besoins et l'architecture, ainsi que pour le développement, peuvent engendrer du *rework* à cause d'erreurs commises en voulant faire vite (Boehm, B. et Basili, 2001).

D'un point de vue plus général, une mauvaise communication entre les membres de l'équipe et avec le client, ainsi qu'un environnement de travail inadéquat, peuvent générer du *rework* pour un projet (Huzooree et Ramdoo, 2015a).

Des recherches ont aussi montré un lien avec le niveau de *rework* d'un projet et les techniques de modélisation, ainsi que l'usage ou non de bonnes pratiques de programmation (Huzooree et Ramdoo, 2015a).

2.3. Identifier et mesurer le *rework*

Bien qu'il serait préférable de n'avoir aucun effort de *rework* lors des projets de développement logiciel, il est généralement acceptable que cet effort représente entre 10% et 20% du projet, selon la complexité de celui-ci (Fairley et Willshire, 2005). En effet, le temps qui devrait être investit en amont afin d'éviter toute erreur pourrait dépasser le coût engendré par le *rework* lui-même. Cependant, certains projets de développement perdraient entre 40% et 50% de leur effort sur du *rework* évitable (Boehm et Basili, 2001).

La mesure du *rework* peut prendre plusieurs formes selon l'impact qui en résulte sur l'organisation. De fait, un défaut dans un environnement de production pourrait avoir des effets négatifs sur la rentabilité d'une entreprise, générer du gaspillage de temps et de ressources, ou encore encourir des délais. Cependant, une mesure fréquente est le cumul des impacts du *rework* à chacune des étape du processus de correction d'un défaut (Tonnellier et Terrien, 2012).

Puisqu'il existe une forte corrélation entre le moment où le défaut est trouvé dans le cycle de développement et l'effort nécessaire pour la correction, c'est un élément qu'il est préférable de consigner. En effet, un défaut peut prendre 100 fois plus de temps à corriger s'il est découverte tardivement que s'il l'avait été tôt dans le processus (Huzooree et Ramdoo, 2015a).

2.4. Résoudre le *rework*

2.4.1 Pratiques d'ingénierie logicielle menant à la qualité

Étant donné que le *rework* est une conséquence directe d'un manque de qualité, l'utilisation de certaines pratiques destinées à améliorer la qualité peut grandement aider.

- La **programmation en binôme** signifie que deux programmeurs travaillent ensemble à l'écriture du code. Ainsi, le travail effectué est continuellement revu par l'autre programmeur, ce qui a pour effet de réduire le taux d'erreur (Huo *et al.*, 2004; Pressman, 2010).
- Avec le ***refactoring***, le but est d'améliorer la structure interne d'une application sans modifier son comportement, afin de réduire les probabilités de générer des erreurs pendant le développement (Pressman, 2010).
- En mettant en place de **l'intégration continue**, les défauts peuvent être détectés dans le produit final rapidement. Cela a aussi pour impact de réduire le temps nécessaire à rechercher des défauts (Huo *et al.*, 2004).

- Les **tests** permettent de s'assurer que le produit fonctionne correctement et qu'il répond aux besoins. Ceux-ci peuvent être des tests automatisés ou encore des tests effectués manuellement (Michlmayr *et al.*, 2005).
- Les **tests d'acceptation**, effectués après les tests unitaires, permettent de s'assurer que le produit répond aux attentes du client. Cela permet aussi d'avoir une rétroaction du client rapidement si cela est effectué plusieurs fois au cours du projet (Huo *et al.*, 2004).
- Pour assurer un suivi efficace des défauts et des demandes de changement, un **outil de gestion des demandes** est requis. Ainsi, il est possible de garder une trace de tous les détails d'une demande et de s'assurer qu'elle sera traitée (Michlmayr *et al.*, 2005).
- Il est aussi important de faire un suivi des modifications apportées au code. Le **gestionnaire de version de code** répond à ce besoin, en plus de permettre à plusieurs personnes de travailler simultanément sur le même code, en limitant grandement les erreurs d'intégration (Michlmayr *et al.*, 2005).
- Que ce soit pour des documents ou pour le code, les **revues par les pairs** permettent d'examiner le travail effectué et ainsi déceler des défauts ou des omissions, vérifier que l'application répond aux exigences et s'assurer que les standards et pratiques sont respectés (Pressman, 2010).
- **L'utilisation de *framework* de programmation évolué** favorise la réutilisation et permet d'atteindre un plus haut niveau de productivité (Markiewicz et Lucena, 2001).

2.4.2 Compréhension commune et exhaustive des besoins

Il est impératif que les besoins d'une application soient bien compris à la fois par l'équipe de développement et par le client. Pour ce faire, il existe plusieurs techniques qui permettent de valider la compréhension mutuelle des besoins. En d'autres mots, il faut s'assurer que le client a exprimé tous ses besoins et que l'équipe de développement a bien compris les besoins exprimés par le client.

- La **présence du client sur place** lors du développement aide les développeurs à approfondir la compréhension des besoins, et même à y apporter des corrections (Huo *et al.*, 2004). Afin d'être efficace, le client doit être présent tout au long du processus de développement.
- En élaborant des **prototypes**, il est beaucoup plus facile de comprendre les besoins des clients, car ces derniers peuvent visualiser le produit final et plus facilement faire des liens entre l'application et leurs besoins (Boehm, 1987).

2.4.3 Stratégie de résolution du rework

D'entrée de jeu, il faut faire la nuance entre les activités pour éliminer le *rework* à la source et les activités pour détecter les défauts qui génèrent du *rework.* Du point de vue du produit final, les deux permettent d'obtenir une meilleure qualité, sauf que comme mentionné précédemment, plus les défauts sont éliminés tôt, meilleur sera l'impact sur la productivité.

Il y a plusieurs stratégies pour éliminer le *rework* à la source. On peut en voir quelques-unes dans la figure 2.2 portant sur l'amélioration de la productivité.

Bien que les exemples illustrés dans la partie droite de la figure 2.2 aient évolué depuis cette publication, les catégories au centre demeurent valables aujourd'hui. Donc, les stratégies permettant d'élimer le *rework* sont l'utilisation d'outils de développement performant, de bases de connaissances, de pratiques modernes de développement et de modularité, du développement incrémental, ainsi que du prototypage et la réutilisation de composants (Boehm, 1987).

Une fois les défauts introduits dans le système, plusieurs activités permettent de les détecter et ainsi réduire le plus possible l'effort nécessaire à leur correction en les détectant le plus tôt possible dans le processus de développement. Parmi ces activités, on retrouve entre autres l'analyse statique du code (Nagappan et Ball, 2005), les revues formelles techniques, les tests de vérification et de validation et la phases d'acceptation (Pressman, 2010).

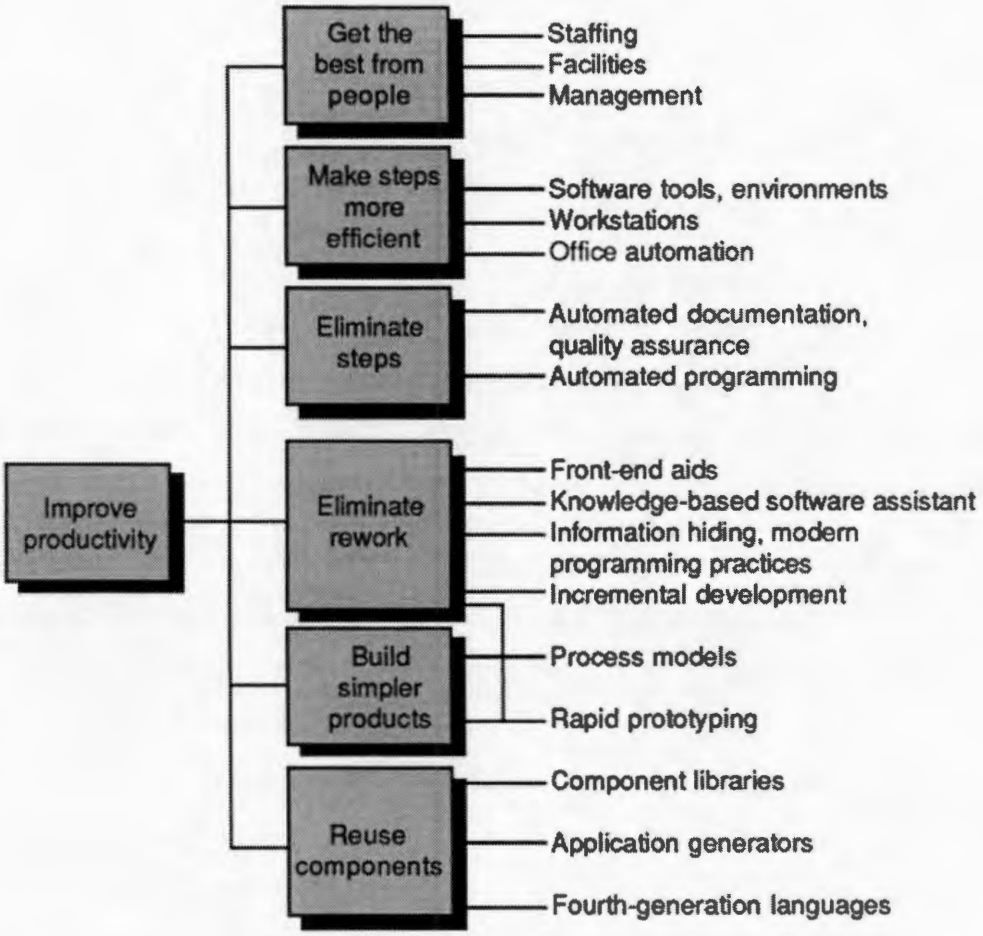

Figure 2.2 Stratégies d'amélioration de la productivité, tiré de (Boehm, B. W., 1987)

2.5. Amélioration des processus logiciels

L'amélioration de processus est composée d'un ensemble d'activités menant vers de meilleurs processus logiciels, ce qui en conséquence permet d'obtenir un logiciel de plus grande qualité, livré dans un temps plus court (Pressman, 2010). Cependant, afin d'améliorer un processus, il est nécessaire qu'un processus soit déjà défini et qu'il soit utilisé par l'organisation en question. Ce dernier point devient parfois

problématique dans les plus petites organisations où les processus sont plus informels et où il y a moins pratiques définies.

Lorsqu'on parle d'amélioration de processus, cela signifie nécessairement qu'il faut être capable de mesurer l'état d'une situation initiale et d'une situation finale. Ainsi, au final, il est possible de déterminer si la modification apportée est bel et bien une amélioration.

L'effort nécessaire à l'amélioration peut devenir assez important selon les objectifs définis au début. C'est pourquoi les mesures recueillies doivent permettre autant que possible de quantifier les retombées des améliorations pour ainsi définir quel est le retour sur l'investissement en temps, en effort et en argent (Pressman, 2010).

Afin de bien définir comment procéder pour mettre en place des améliorations, le Software Engineering Institute (SEI) a créé le modèle IDEAL, qui définit les cinq phases permettant de guider les organisations : initier (*initiating*), diagnostiquer (*diagnosing*), établir (*establishing*), agir (*acting*), et apprendre (*learning*) (Pressman, 2010). Plus de la moitié des efforts d'amélioration se soldent par un échec, ce qui démontre l'importance de prendre ce projet au sérieux si on souhaite en tirer des retombées positives. Parmi les éléments qui pourraient être sources de risques, on retrouve : le budget et les coûts, le contenu et les livrables, la culture, la maintenance future du livrable du projet d'amélioration, la mission et les buts, l'administration de l'organisation, la stabilité de l'organisation, les intervenants des processus et tous les éléments en lien avec le projet d'amélioration lui-même (Pressman, 2010).

L'amélioration des processus logiciels est quelque chose qui doit toujours être adapté à l'environnement où on la met en place. Plusieurs éléments tels que la culture, les types de produits développés, ainsi que les forces et faiblesses, doivent être considérés afin que l'opération soit un succès. De plus, afin de réussir la mise en place d'améliorations, cinq critères importants doivent être considérés : le soutien et l'engagement des gestionnaires, l'implication des employés, l'intégration et la

compréhension du processus, la personnalisation de la stratégie d'amélioration et une gestion rigoureuse du projet d'amélioration des processus (Pressman, 2010).

CHAPITRE 3

MÉTHODOLOGIE

3.1. Approche retenue

À la lumière de l'état de l'art sur l'amélioration de processus présenté à la section 2.5, l'approche retenue pour l'amélioration de notre projet consiste en 4 phases :

1. Le démarrage et référentiel : Lors du démarrage et référentiel, nous voulons connaitre la proportion de *rework* du processus actuel en obtenant des mesures sur les projets effectués par le passé.
2. L'analyse : Lors de l'analyse, nous voulons identifier les causes fondamentales du *rework* et ainsi expliquer pourquoi nous avons ce niveau de *rework.*
3. L'amélioration : Lors de l'amélioration, nous voulons mettre en place les améliorations et discuter de leur impact et/ou acceptation par les parties prenantes.
4. La validation : Lors de la validation, nous voulons comparer les résultats obtenus avec le référentiel pour ainsi confirmer si les améliorations ont eu un impact positif, soit la diminution du *rework.*

3.2. Démarrage et référentiel

3.2.1 Établir les critères de sélection des projets à analyser

Afin de sélectionner des projets pertinents à notre quantification du *rework*, soit d'établir un référentiel du *rework* du processus actuel, nous avons établi des critères de sélection. À partir de la liste des projets de l'équipe de développement, ainsi que

de l'effort fourni pour chacun d'entre eux, nous voulions déterminer ce qui caractérise ceux qui représentent des projets typiques de l'équipe de développement.

Tableau 3.1 Aperçu de la méthodologie

Phase	Intrants	Activités	Livrables	Retombées
Démarrage / Référentiel	• Portée • Effort réel [projets terminés] • Billets Redmine	• Établir les critères de sélection des projets à analyser (3.2.1) • Extraire les données des projets sélectionnés (3.2.2) • Établir le référentiel initial (3.2.3)	• Critères de sélection des projets (3.2.1) • Projets sélectionnés (3.2.2) • Référentiel initial de *rework* [par type de demande] (4.1)	
Analyse	• Référentiel initial de *rework*	• Analyser les données de *rework* (3.3.1) • Recenser des solutions potentielles (3.3.2)	• Causes principales du *rework* (4.2) • Solutions proposées (4.3)	Compréhension des causes du *rework* du processus initial (6.1)
Amélioration	• Processus actuel • Causes principales du *rework*	• Améliorer le processus (3.4.1) • Tester les améliorations (3.4.2)	• Liste d'améliorations appliquées au processus (5.1) • Observations sur la mise en œuvre des améliorations apportées (5.2)	Retombées de notre projet pour la STI et la CSDA (6.2)
Validation	• Processus amélioré • Effort réel [des nouveaux projets]	• Établir le référentiel de *rework* avec processus amélioré (3.5.1) • Comparer les données récentes avec le référentiel initial (3.5.2)	• Référentiel de *rework* mis à jour (5.3) • Analyse des écarts [entre les valeurs initiales et celles mises à jour] (5.4) • Limites de validité (5.5)	Compréhension des écarts (6.4)

Nous avons établi une première liste de critères que nous avons fait vérifier et valider par le coordonnateur. Les critères retenus sont:

1. **Projet de développement** : Nous voulions inclure uniquement des projets de développement ou de maintenance de logiciel et exclure les projets de rehaussement d'infrastructure ou d'implantation d'applications acquises à l'externe.
2. **Projet terminé** (en production) : Pour s'assurer d'avoir des données d'effort complètes, nous incluons seulement les projets terminés.
3. **Au moins 30 heures d'effort** : les projets trop petits n'étant pas significatifs, nous avons inclus seulement des projets dont le niveau d'effort représentait 30 heures ou plus.
4. **Actif en 2016-2017** : Étant donné que des changements avaient été apportés au processus dans le passé, nous ne voulions pas reculer trop loin dans le temps, afin de limiter les données aux projets assez récents, soit actifs au cours de la dernière année scolaire 2016-2017.

3.2.2 Extraire les données des projets sélectionnés

Nous avons appliqué les critères de sélection définis précédemment sur les 27 projets terminés au cours de l'année 2016-2017. Parmi ceux-ci, 11 projets rencontraient nos critères et nous avons recueilli la liste de toutes leurs tâches, avec l'effort mis sur chacune.

Le tableau 3.2 présente quelques informations sur ces 11 projets retenus.

Tableau 3.2 Liste des projets sélectionnés pour établir le référentiel

Id	Client	Effort (h)	Nombre demande	Langage	Type projet	En prod.
1	A	668,7	115	VB.NET	Développement	Oui
2	A	142,8	51	VB.NET	Développement	Oui
3	B	263,3	65	VB.NET	Développement	Oui
4	C	308,6	51	PHP	Développement	Oui
5	D	544,9	47	VB.NET	Développement	Oui
6	E	336,3	85	VB.NET	Développement	Oui
7	E	747,1	203	VB.NET	Développement	Oui
8	E	322,3	97	VB.NET	Développement	Oui
9	E	449,3	78	VB.NET	Développement	Oui
10	F	168,2	28	VB.NET	Développement	Oui
11	G	329,6	63	VB.NET	Développement	Oui

3.2.3 Établir le référentiel initial

Afin de pouvoir classifier l'effort de chaque projet, nous avons établi une liste de catégories représentant chaque type d'effort pouvant être effectué sur les projets. De ces catégories, certaines étaient considérées comme du *rework*. Il a ainsi été possible de quantifier l'effort de *rework* pour chaque projet. La liste des catégories est:

1. Exigence initiale du projet : Les exigences issues de l'analyse initiale du projet.
2. Mauvaise définition des besoins à l'origine (R) : Le besoin exprimé n'était pas le bon, il a été mal exprimé, ou encore la mauvaise information a été fournie à l'équipe de développement. Cela peut aussi être quelque chose qui avait été oublié, ignoré ou même inconnu lors de l'analyse initiale du projet.
3. Mauvaise compréhension entre le PO et l'équipe STI (R) : Le besoin exprimé par le PO et le besoin compris par l'équipe de développement ne concordent

pas. Potentiellement dû à une ambiguïté, un manque de connaissance du domaine ou à l'utilisation du mauvais vocabulaire.

4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (R) : Les utilisateurs identifient des lacunes dans l'application suite à l'expérimentation en préproduction, ou à l'utilisation en production.
5. Ajout de nouvelles exigences par PO : Amélioration d'une exigence déjà exprimée, optimisation. Représente souvent des « tant qu'à y être » et des « ça serait le *fun* que… ».
6. Ajout de nouvelles exigences par STI : Modifications à l'application demandées par le STI. Représente généralement une modification pour améliorer des éléments non-fonctionnels tels que la maintenabilité, la performance ou la facilité de configuration.
7. Autres : Tout autre type de demande provenant de l'interne : Utilisabilité, rencontre, présentation, soutien, accompagnement, …
8. Changements de technologies : Une technologie utilisée par l'application doit être mise à niveau ou encore remplacée par une autre technologie.
9. Changements de législation : Une des lois, des règles ou des politiques à la base des exigences de l'application change. Comprend aussi les règles de fonctionnement interne des services.
10. Problème technique (R) : Mauvais fonctionnement de l'application (exception, performance ou autres) causé par une erreur de programmation, une erreur de configuration ou tout autre élément technique.
11. Mauvaise connaissance de l'application (processus et données) (R) : Demande de vérification d'un problème qui s'avère être le fonctionnement normal de l'application.

Les 883 demandes des projets sélectionnés ont alors été analysées pour déterminer à quelle catégorie elles appartenaient (voir un échantillon de ces demandes catégorisées à la figure 3.1).

Il a ensuite été possible de faire la somme de l'effort associé à toutes les demandes de chaque catégorie. Nous avons aussi calculé le nombre de demandes pour chaque catégorie. Au final, nous avons pu connaitre quelle proportion de l'effort était du *rework* et qu'elle en était la cause.

#	Statut	Sujet	Assigné à	Temps passé	Type	Raison
5793	Fermé	[redacted]	[redacted]	3.00	Accompagnement fonctionnel	07. Autres (Utilisabilité, rencontre, présentation, soutien, accompagnement, ...)
5617	Fermé	[redacted]	[redacted]	3.00	Développement	10. Problème technique (R)
4903	Fermé	[redacted]	[redacted]	1.00	Développement	01. Requis initial du projet
3442	Fermé	[redacted]	[redacted]	3.00	Développement	01. Requis initial du projet
3415	Fermé	[redacted]	[redacted]	1.00	Développement	04. Modifications aux requis suite à l'expérimentation ou à l'utilisation (R)
3403	Fermé	[redacted]	[redacted]	0.50	Développement	04. Modifications aux requis suite à l'expérimentation ou à l'utilisation (R)

Figure 3.1 Échantillon des demandes catégorisées

Les données obtenues serviront de référentiel initial du *rework* pour le processus d'amélioration. Ce référentiel est présenté à la section 4.1

3.3. Analyse

3.3.1 Analyser les données de rework

En utilisant les données du référentiel initial de *rework*, nous avons analysé les chiffres obtenus pour faire ressortir les causes principales de *rework*.

La liste des principales causes de *rework* est présentée à la section 4.2.

3.3.2 Recenser des solutions potentielles

En se basant sur les causes de *rework* préalablement identifiées et en tenant en compte du contexte dans lequel évolue l'équipe de développement, nous avons recensé un certain nombre de solutions qui pourraient permettre de diminuer le *rework*.

La liste des solutions recensées, ainsi qu'une courte description et l'impact attendu, se trouvent à la section 4.3.

3.4. Amélioration

3.4.1 Améliorer le processus

Parmi les solutions recensées à la section 4.3, nous avons sélectionné trois d'entre elles pour les mettre en application dans le processus de développement.

La liste des améliorations qui ont été appliquées au processus se trouve à la section 5.1, ainsi que les raisons qui ont motivé ce choix.

3.4.2 Tester les améliorations

Les améliorations appliquées au processus ont été utilisées par l'équipe de développement pour certains projets pilotes, afin de voir si cela avait un impact sur le taux de *rework* du projet concerné.

Lors de la mise en place des améliorations, plusieurs observations ont été consignées et elles sont présentées à la section 5.2.

3.5. Validation

3.5.1 Établir le référentiel de rework avec processus amélioré

Une fois les projets pilotes terminés, les données sur les demandes des projets ont été consolidées pour permettre d'établir un référentiel du processus que nous avions amélioré.

Le référentiel de *rework* mis à jour avec le processus amélioré est présenté à la section 5.3.

3.5.2 Comparer les données récentes avec le référentiel initial

En prenants les référentiels des deux processus (initial et amélioré), nous avons comparé les écarts pour analyser l'impact que les changements ont eu sur la proportion du *rework* dans les projets de développement.

Le résultat des analyses des écarts est présenté à la section 5.4. De plus, les limites de la validité des résultats obtenus sont discutées à la section 5.5.

CHAPITRE 4

RÉSULTATS ET ANALYSE DE LA MESURE DU *REWORK*

4.1. Référentiel initial de *rework* [par type de demande]

4.1.1 Données brutes

Nous avons fait manuellement la classification de 883 demandes affectées aux 11 projets sélectionnés. Cela nous a permis d'établir le référentiel initial pour chacune des 11 catégories représentant les différents types d'effort. Du coup, nous obtenons le référentiel initial de *rework* présenté au tableau 4.1.

Tableau 4.1 Résultats du référentiel initial du *rework*

Catégorie	Nb heures	% heure	Nb demandes	% demande
1. Exigence initiale du projet	2 814,8	65,8 %	291	33,0%
2. Mauvaise définition des besoins à l'origine (R)	202,3	4,7 %	63	7,1%
3. Mauvaise compréhension entre le PO et l'équipe STI (R)	37,0	0,9 %	6	0,7%
4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (R)	578,1	13,5 %	193	21,9%
5. Ajout de nouvelles exigences par PO	47,2	1,1 %	26	2,9%
6. Ajout de nouvelles exigences par STI	57,0	1,3 %	26	2,9%
7. Autres	129,7	3,0 %	98	11,1%
8. Changements de technologies	0,7	0,0 %	1	0,1%
9. Changements de législation	0,0	0,0 %	0	0,0 %
10. Problème technique (R)	413,8	9,7 %	179	20,3%
11. Mauvaise connaissance de l'application (processus et données) (R)	0,0	0,0 %	0	0,0 %
Total général	**4 280,6**	**100,0 %**	**883**	**100,0%**

Le *rework* est constitué des éléments numéros 2-3-4-10-11 (surlignés en gris dans le tableau 4.1). Donc, pour l'ensemble des projets sélectionnés, nous obtenons un total de 28,8% d'effort attribué au *rework*, soit 1231,2 heures d'effort sur les 4280,6 heures recensées. De ce total, les deux principaux types de *rework* sont

- 4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (13,5% de l'effort total, représentant 47,0% du *rework*),
- 10. Problèmes techniques (9,7% de l'effort total, représentant 33,6% du *rework*).

Du côté du nombre de demandes, on constate que c'est une proportion de 49,9% (441/883) qui est attribuée au *rework*. Ce sont ici aussi les deux mêmes types de *rework* qui sont principalement en cause, dans une proportion très semblable :

- 4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (21,9% des demandes totales, représentant 43,8% du nombre de demandes considérées comme du *rework*),
- 10. Problèmes techniques (20,3% des demandes totales, représentant 40,6% du nombre de demandes considérées comme du *rework*).

4.1.2 Analyse des données du référentiel

4.1.2.1 Nombre d'heures moyen par demande

Le nombre d'heures passées pour chaque demande varie en fonction de plusieurs facteurs. En isolant cette variation par type de demande, nous sommes en mesure de déterminer la moyenne d'heures passées pour une demande de chaque type. Il est alors possible de comparer ces résultats à la moyenne générale de temps par demande. Au final, il sera possible de voir si le temps passé pour une demande d'un type donné est habituellement plus long que pour un autre type. La figure 4.1 nous montre le résultat de cette analyse.

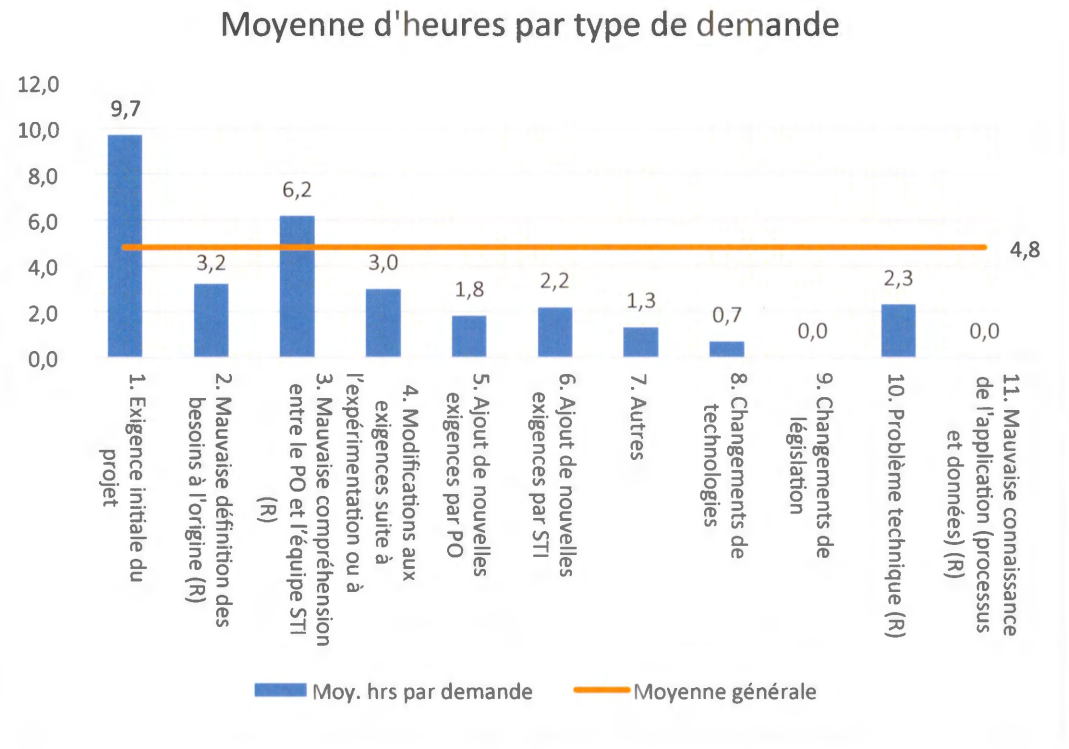

Figure 4.1 Moyenne d'heures par type de demande

Comparativement à la moyenne générale de 4,8 heures par demande, l'effort moyen de la catégorie « 1. Exigences initiales du projet » de 9,7 heures est beaucoup plus grand. On peut donc conclure que la proportion d'effort de cette catégorie est due au temps passé sur chaque demande et non au nombre de demandes. La catégorie « 3. Mauvaise compréhension entre le PO et l'équipe STI » est la seule autre au-dessus de la moyenne avec 6,2 heures. Cette information indique que ce type de demande est habituellement couteux en termes d'effort. Par contre, le faible échantillonnage (6 demandes) ne permet pas de tirer de réelles conclusions.

4.1.2.2 Client

Les projets sélectionnés proviennent de différents clients. Il est donc légitime de se demander s'il y a un lien à faire entre la proportion de *rework* et un client spécifique, étant donné qu'ils ont tous des réalités différentes et que leurs processus sont différents. Nous avons analysé les demandes par client à la Figure 4.2 et leur effort relatif à la Figure 4.3.

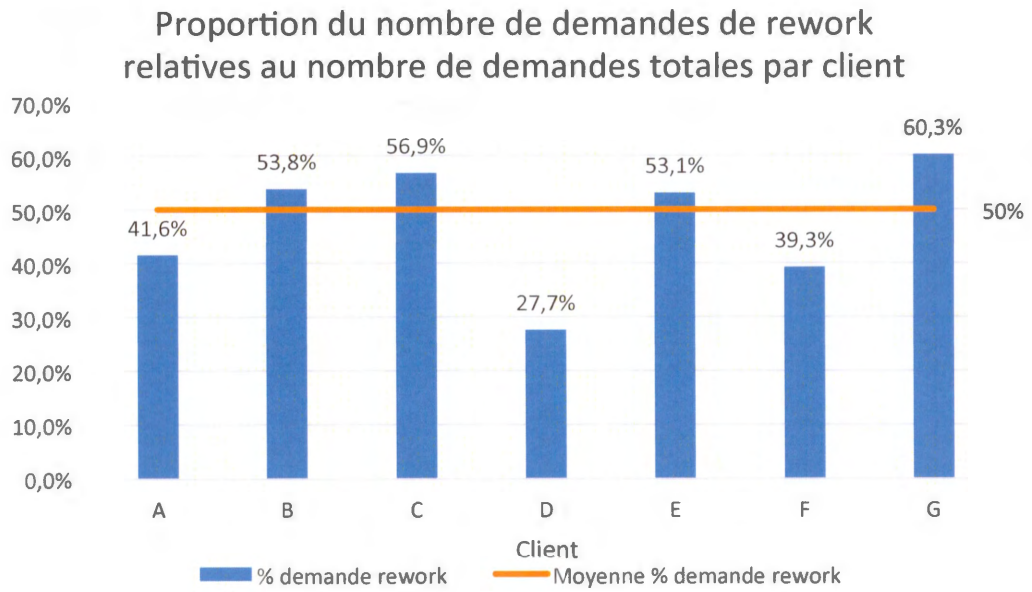

Figure 4.2 Proportion du nombre de demandes de *rework* relatives au nombre de demandes totales par client

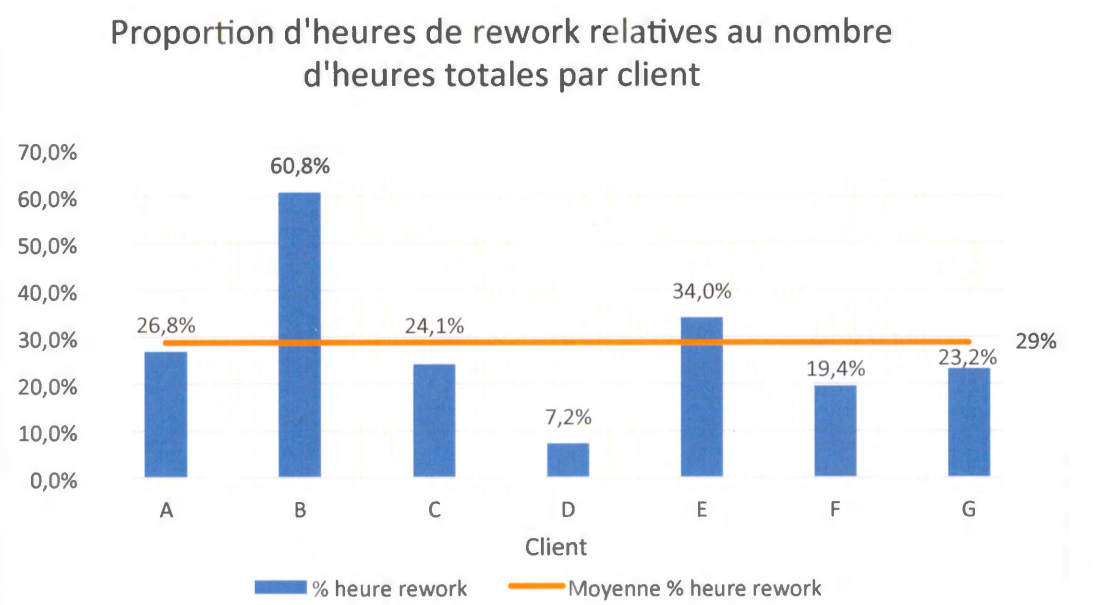

Figure 4.3 Proportion d'heures de *rework* relatives au nombre d'heures totales par client

En termes de nombre de demandes, il y a seulement le client D pour qui la proportion est significativement en dessous de la moyenne.

En termes de nombre d'heures, le client D est aussi bien en deçà de la moyenne. Cependant, le client B se retrouve très au-dessus de la moyenne. On comprend donc que le nombre d'heures passées pour ces demandes était aussi très au-dessus de la moyenne.

Avec le temps, il s'est bâti une perception selon laquelle le *rework* était plus présent pour le client E car ce dernier avait souvent des demandes de modification avec une très courte échéance. Il est vrai que les proportions sont plus élevées que la moyenne pour ce client, mais en considérant uniquement l'effort, l'écart n'est pas à la hauteur des perceptions négatives qui entourent ce client.

4.1.2.3 Langage de programmation

Dans les projets sélectionnés pour le référentiel, il y avait deux langages de programmation, soit ASP.NET et PHP. De par leur nature très différente (ASP.NET est compilé alors que PHP ne l'est pas), on pourrait s'attendre à une proportion de problèmes techniques plus grande en PHP. Nous avons effectué la comparaison dans la figure 4.4.

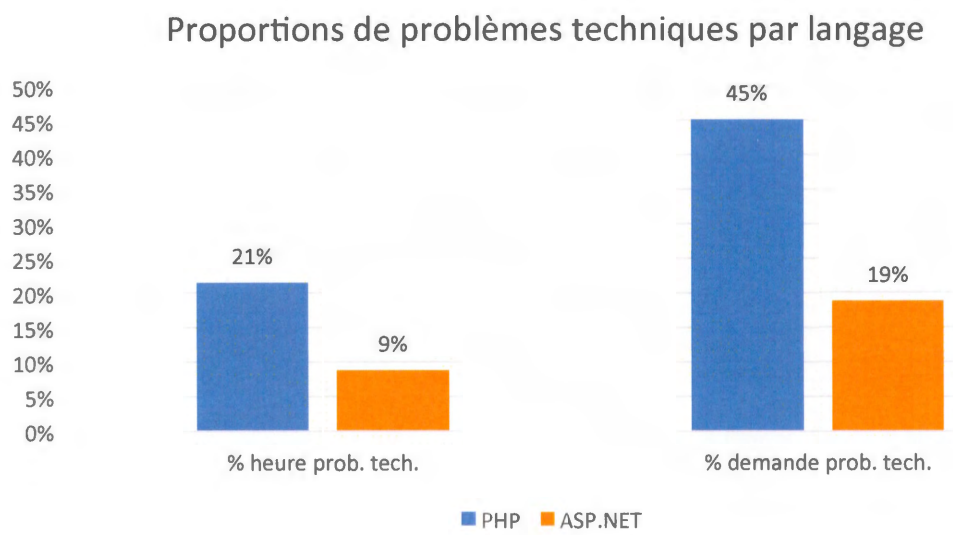

Figure 4.4 Proportions de problèmes techniques par langage

Lorsqu'on compare le poids des problèmes techniques dans les projets ASP.NET avec celui dans le projet PHP, on remarque une différence importante. Autant en termes d'heures qu'en termes de demandes, la proportion est plus du double dans les projets PHP. La problématique avec les applications PHP ne sera toutefois pas analysée, étant donné que tous les nouveaux développements sont faits en ASP.NET.

4.1.2.4 Taille du projet

Dans les critères de sélection de projet, nous avions établi que l'effort minimum pour un projet était de 30 heures. Cependant, certains projets ont un total d'heures beaucoup plus grand. Puisqu'il existe un écart aussi grand entre certains projets, nous avons regardé s'il existait un lien entre le nombre d'heures du projet et le nombre d'heures de *rework* dans le projet.

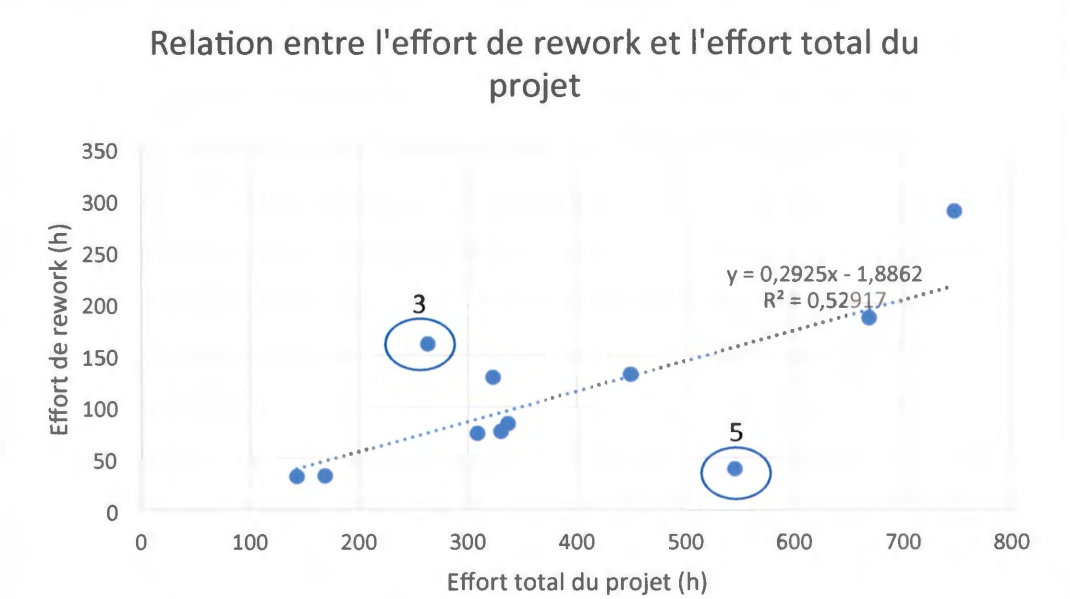

Figure 4.5 Relation entre l'effort de *rework* et l'effort total du projet

Donc, en croisant ces deux données, on observe qu'il y a une certaine corrélation, mais qu'elle n'est pas très forte (coefficient de corrélation de 0,5292). On observe aussi qu'il y a 2 projets qui semblent sortir du lot, soit les deux dont les points sont entourés dans la figure 4.5. Le projet 3 est un projet exceptionnel parce que c'était un projet de maintenance corrective composé de demandes de modification accumulées au cours de l'année scolaire précédente. Le projet 5 est aussi un projet exceptionnel mais pour une raison inverse, soit qu'il était un projet très répétitif, où un grand nombre de requêtes de validation devaient être mises en place de la même façon.

Tous les autres projets sont des projets typiques. Nous avons donc refait le même exercice, mais en excluant les deux projets exceptionnels. Le résultat se trouve dans la figure 4.6.

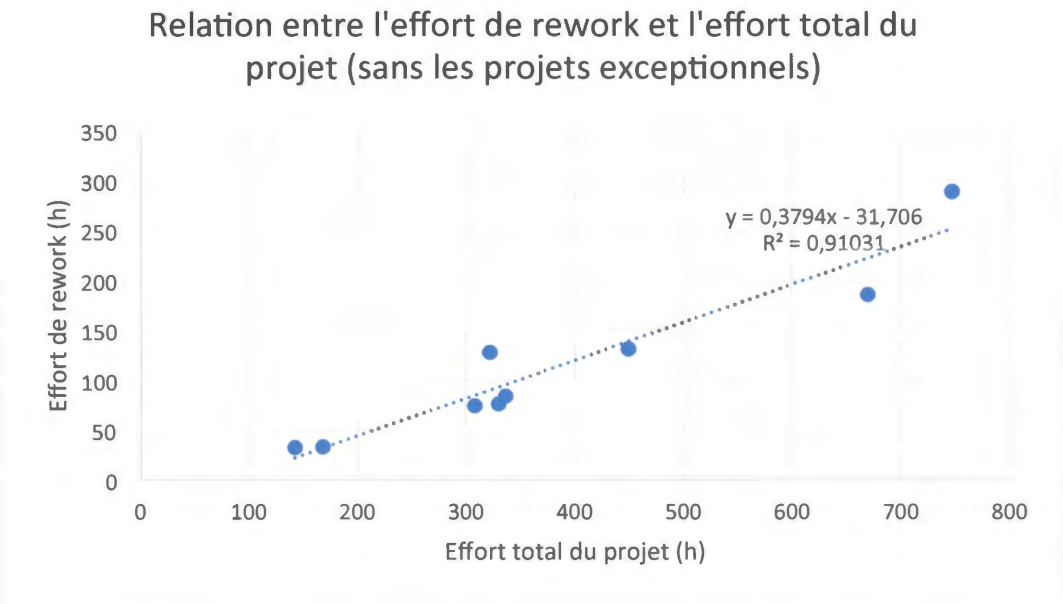

Figure 4.6 Relation entre l'effort de *rework* et l'effort total du projet (sans les projets exceptionnels)

On voit que la corrélation est maintenant très forte (coefficient de corrélation de 0,9103). Cela nous permet donc de conclure que le *rework* est semblable dans tous les projets, peu importe leur taille. Les améliorations apportées devraient ainsi avoir un impact sur n'importe quel projet typique futur.

4.1.3 Bonification des catégories

Au cours de notre projet, nous avons eu quelques situations où, suite à une plainte d'un client sur le fonctionnement d'une application, nous avons perdu du temps à chercher une cause probable alors que le problème s'avérait être une mauvaise

compréhension des règles d'affaires par ce client. Par exemple, une liste déroulante devait présenter uniquement les employés répondants à certains critères et pour laquelle le client se plaignait que tel ou tel employé n'y apparaissait pas. Après des recherches d'une heure et demie, il s'est avéré que l'application fonctionnait correctement. Cet effort est catégorisé comme étant du *rework* par le STI.

Pour tenir compte de ce type de *rework*, nous avons bonifié la liste des catégories de *rework* au cours de notre projet en ajoutant le type de demandes « 11. Mauvaise connaissance de l'application (processus et données) ». Cependant, nous ne sommes pas revenus sur les résultats du référentiel et seuls les prochains projets pourront cumuler des données de *rework* de ce type.

4.2. Causes principales du *rework*

Grâce au référentiel initial, nous avons identifié deux catégories de demandes qui étaient les plus grandes responsables du *rework*. De plus, l'analyse des données de *rework* ne nous a pas permis d'identifier d'autres facteurs qui auraient un impact important sur le *rework* des projets à venir. C'est donc vers ces catégories que nous avons dirigé notre analyse, sans égard au client, à la taille du projet ou à la technologie.

4.2.1 Modifications aux exigences suite à l'expérimentation ou à l'utilisation

Beaucoup de demandes de changement sont formulées lors des premières utilisations d'une nouvelle application par ses utilisateurs. Lors du développement, **certains choix sont faits alors que le PO ne comprend pas entièrement toutes les subtilités**, en grande partie parce que le tout est expliqué verbalement ou dans un texte trop simpliste, sans support visuel. Il arrive aussi que **l'équipe de développement croit répondre aux besoins du client alors que ce n'est pas le cas.**

C'est donc lors des premières utilisations que ces problèmes surgissent. Un cas assez simple se produisant souvent est le choix des colonnes à inclure dans un tableau de données. Lors du développement, on détermine ce que le tableau doit contenir, mais sans nécessairement tenir compte du contexte dans lequel se trouve l'utilisateur lorsqu'il consulte le tableau en question. C'est donc lorsque les données sont consultées par les utilisateurs dans l'application qu'ils réalisent ce qu'il manque et ce qui est inutile. Pour un des projets sélectionnés, il a été demandé d'ajouter un indicateur pour chaque transaction d'un tableau pour déterminer qui est l'auteur, parce que cela avait un impact sur le traitement à effectuer.

Dans certaines situations, l'expérimentation de l'application, avec des vraies données, permet d'avoir une meilleure compréhension de l'application. Ainsi, il sera plus facile de savoir si certaines informations peuvent être visibles ou non à certains utilisateurs, ou encore si certains cas ne sont pas prévus dans l'application.

C'est aussi lors de l'expérimentation de l'application que certains utilisateurs vont réellement comprendre le processus qui dicte le comportement de l'application. Ainsi, certaines étapes peuvent être ajoutées ou retirées.

4.2.2 Problèmes techniques

En analysant les demandes de problèmes techniques, il a été possible d'identifier certaines causes communes à plusieurs d'entre elles. Tout d'abord, la complexité des interfaces des applications rend le traitement des requêtes très complexe. Dans un tel contexte, la compréhension exacte du cycle des requêtes des applications du *framework* « ASP.NET Web Forms » demande des connaissances plus approfondies et plus de temps à développer. Par exemple, lors d'une requête, la fonction « dataBind() » est appelée plusieurs fois et tous les objets nécessaires ne sont pas initialisés à chaque appel. Il faut donc prévoir ces situations pour chaque appel, car cela peut générer une exception pour un pointeur nul (*nullPointerException*).

Dans plusieurs situations, des problèmes de performances sont rencontrés. En effet, le traitement nécessaire pour définir les valeurs d'un menu déroulant ou encore pour calculer le contenu d'une grille peut devenir lourd et résulter en des performances en deçà des attentes des utilisateurs. Par exemple, dans une page contenant beaucoup d'informations, si le contenu d'un menu déroulant est dépendant du choix fait dans un autre menu déroulant, alors le cycle « Web Forms » recalculera la page en entier au lieu de mettre à jour seulement le menu déroulant qui doit être modifié. Lorsqu'une telle situation se produit, il faut alors contourner le problème par l'utilisation de script Javascript dans l'application, ce qui vient augmenter la complexité de l'application étant donné que le traitement peut être déclenché à deux endroits différents.

Une autre cause est sans surprise l'erreur humaine. Dans l'élaboration du code, certaines erreurs humaines peuvent passer sous le radar et se retrouver dans le code de l'application. Par exemple, le nom d'un champs HTML peut contenir une erreur de frappe, ce qui empêche la donnée d'être correctement enregistrée dans la base de données, sans pour autant causer de défaillance à l'application. Ce genre d'erreur n'est pas capté par les compilateurs et est difficilement détectable par l'examen du code.

4.3. Solutions proposées

En fonction des sources de *rework* observées et de la revue de l'état de l'art, nous proposons des solutions à appliquer au processus pour lesquelles une diminution du *rework* serait attendue.

4.3.1 Modifications aux exigences suite à l'expérimentation ou à l'utilisation

Pour diminuer le *rework* du type « 4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation », nous proposons d'appliquer du prototypage et d'augmenter la présence du client au sein des projets.

Prototypage

Le prototypage sera appliqué pour chaque fonctionnalité pour laquelle il y a une interface utilisateur et ce, en début d'analyse. Un outil de prototypage comme le logiciel libre Pencil, pourra être utilisé à cette fin. Des prototypes de type *sketchy* seront créés, c'est-à-dire ayant une apparence de dessin et non pas l'apparence de l'application finale, ce qui permet de diminuer la confusion de ce qui est fait et ce qu'il reste à faire. Les prototypes seront discutés avec le client et approuvés par ce dernier.

En permettant au client de mieux comprendre quelle forme prendra le produit final, il est ensuite beaucoup plus facile de comprendre les besoins du client. Une fois les prototypes créés, le partage d'information est aussi beaucoup plus facile entre les membres de l'équipe et cela diminue le besoin de documentation. Certains aspects comme des critères de sélection des données ou les requêtes SQL ne sont toutefois pas couverts par cette technique. Il y a un investissement de temps nécessaire à la création du prototype par l'analyste, mais il n'y aurait pas d'effort additionnel requis

par le client comparativement au processus actuel. Toutefois, nous croyons que cet effort investi sera compensé par une diminution équivalente ou supérieure du *rework*.

Présence permanente du client

La présence du client sera requise en permanence avec l'équipe de développement afin de répondre immédiatement aux questions qui surgissent lors du développement. Puisqu'il y a parfois plusieurs projets du même client simultanément, ce dernier pourra répondre aux questions de tous les projets.

En évitant de mettre en place des choses qui ne répondent pas aux besoins du client, le produit final nécessitera moins de modifications. Tout au long du projet, les développeurs auront une meilleure compréhension des besoins et le client aura une meilleure compréhension de l'application. Cependant, l'investissement de temps de la part du client est majeur. Il faut aussi prévoir un lieu physique pour permettre de travailler au même endroit que les programmeurs. De plus, il pourrait être nécessaire de faire l'embauche d'une personne afin de combler la tâche habituelle du client, puisque ce dernier ne sera plus en mesure de la faire complètement avec le temps qui lui restera.

4.3.2 Problèmes techniques

Pour diminuer le *rework* du type « 10. Problèmes techniques », nous proposons d'instaurer un certain nombre de pratiques d'ingénierie logicielle menant à la qualité, soit des revues par les pairs, de la programmation en binôme, de l'analyse statique du code, des tests automatisés, de l'utilisation de techniques de programmation plus évoluées, de l'intégration continue et du *refactoring*.

Revue par les pairs

À chaque fois que l'application atteindra une étape importante (ex. : fin de sprint, livraison, fonctionnalité terminée), un ou deux autres membres de l'équipe de développement passeront en revue le travail effectué à la recherche de problèmes. Quand tous les membres auront terminé la revue, une rencontre sera organisée pour discuter de ce qui a été trouvé.

En regardant en détails le travail effectué, cela diminuerait le nombre de problèmes résiduels dans l'application. Du même coup, la revue permettrait de partager la connaissance de l'application développée avec d'autres membres de l'équipe. Par contre, il faut bien déterminer comment les revues doivent être conduites et quelles en sont les attentes. Il y a aussi un investissement de temps à faire pour réaliser cette revue. Cette activité de vérification ne permet toutefois pas de mettre en lumière les problèmes qui surviennent au moment de l'exécution.

Programmation en binôme

Lorsque le travail à effectuer n'est pas très simple, une seconde personne viendra assister le développeur pour mener le travail à terme. Ainsi, la seconde personne révise le travail au fur et à mesure qu'il progresse. De temps à autres, les deux personnes inversent les rôles.

Cette technique permettrait de déceler très rapidement des erreurs, au moment même où elles sont commises. En plus, cela permettrait d'avoir au moins deux personnes en mesure de faire efficacement la maintenance de l'application puisqu'elles gagnent toutes deux des connaissances de l'application. Bien que les prérequis techniques soient assez simples pour la mise en place, il ne serait pas toujours facile d'attitrer deux programmeurs à un même clavier.

Analyse statique du code

Un outil d'analyse statique du code serait installé sur les postes de tous les développeurs et, suite à l'archivage du code, ceux-ci devront corriger les défauts décelés par l'outil.

Les analyseurs de code sont souvent payants si l'on désire obtenir des résultats intéressants. La mise en place d'une telle pratique est habituellement assez simple car elle se limite à l'installation et la configuration de l'analyseur. Bien que cela permette de trouver facilement certains problèmes, les résultats seraient limités à certains types de problèmes bien précis.

Tests automatisés

Lors du développement d'une application, il faudra aussi développer des tests automatisés pour s'assurer que le code fait bel et bien ce qu'il est supposé faire. Ces tests seront exécutés manuellement avant l'archivage du code, afin de détecter les défauts.

Pour mettre en place cette pratique, il faudra former adéquatement tous les programmeurs, afin qu'ils soient en mesure de développer les tests automatisés. Un avantage de ce genre de test est qu'il est possible de détecter les fautes et les défaillances qui se produisent en cours d'exécution. Il est aussi possible d'obtenir une excellente couverture de test avec un effort minime.

Utilisation de *frameworks* de programmation plus évolué

Le développement des nouvelles applications sera fait en utilisant le *framework* de programmation ASP.NET MVC, qui utilise le patron de conception Modèle-Vue-Contrôleur (MVC). De plus, la bibliothèque « bootstrap » sera utilisée, ce qui permet de mieux contrôler le rendu visuel de l'application sur différentes plateformes. De son

côté, la bibliothèque « DevExpress » continuera d'être utilisée pour les contrôles utilisateurs (affichage et saisie d'informations).

Ces changements permettent de développer des applications plus facilement en rendant plusieurs tâches faciles à réaliser, tout en donnant un meilleur contrôle sur le comportement de l'application. L'utilisation d'un nouveau *framework* et de nouvelles bibliothèques nécessite un apprentissage important. De la formation est donc prévue et du temps en sus pour les premiers projets, afin de s'approprier ce nouvel environnement. Néanmoins, le résultat net est normalement une diminution de la complexité et une augmentation des possibilités et de la productivité.

Intégration continue

Il faudra mettre en place une infrastructure permettant de déployer l'application dans un environnement de développement à chaque modification du code. Ensuite, un certain nombre de vérifications et de tests seront automatisés pour savoir si l'application fonctionne correctement. Un rapport sera envoyé automatiquement en cas de problème.

Pour détecter les défauts rapidement lors de l'exécution du projet, l'intégration continue est très efficace. En plus de réduire le temps nécessaire à la construction de l'application, les problèmes sont détectés avant la mise en production. Pour avoir une telle infrastructure, beaucoup de temps de formation et de mise en place sera attribué, en plus des serveurs qui seront mis en place. De plus, un tel outil a un impact important sur les façons de travailler de l'équipe de développement.

Refactoring

Avant d'archiver leur code, chaque développeur devra prendre un certain temps afin de retravailler le code développé, pour le rendre plus simple et mieux structuré.

Toutefois, les modifications ne devront en aucun temps modifier le comportement de l'application.

En améliorant la structure interne des applications de manière régulière, les développements qui suivent et la maintenance sont plus faciles. On peut donc voir cette technique proactive comme un investissement, car le temps passé permettrait éventuellement de sauver du temps de développement. Le temps nécessaire pour la mise en place de cette technique est assez court mais demande du temps additionnel de manière récurrente.

CHAPITRE 5

AMÉLIORATIONS DU PROCESSUS DE DÉVELOPPEMENT

Lesson learned: "Anomalies for which no corrective action is taken often have a significant effect on the cost of rework".
(Damm *et al.*, 2008)

5.1. Liste d'améliorations appliquées au processus

Les propositions d'améliorations énoncées au chapitre 4 ont été présentées au coordonnateur pour déterminer lesquelles nous étions prêts à mettre en œuvre. Le choix final a été arrêté, pour le moment, à ces trois propositions:

- Prototypage, appliqué tel que proposé ;
- Utilisation du *framework* de programmation ASP.NET MVC, déjà en discussion dans l'équipe de développement avant notre projet d'amélioration de processus. Il a été décidé de le mettre en œuvre pour les prochains projets de développement ;
- Analyse statique du code, tel que proposé.

Dans la cadre de ce projet d'amélioration, il aurait été difficile de demander des changements majeurs auprès des clients, ce qui explique pourquoi la présence permanente du client a été rejetée.

La demande étant de plus en plus grande pour des développements, et ayant un nombre fixe de membres dans l'équipe, il a été choisi de ne pas sélectionner des nouvelles pratiques qui auraient un impact important sur l'effort additionnel nécessaire à leur mise en place. Donc, la revue par les pairs ainsi que la programmation en binôme n'ont pas été retenues, parce qu'il n'y avait pas de personne disponible pour en faire l'essai. Dans le même ordre d'idée, ayant déjà choisi l'utilisation de ASP.NET MVC, qui allait demander un temps d'apprentissage

non-négligeable, il a été décidé de privilégier les options avec une courte courbe d'apprentissage. Ainsi, le prototypage et l'analyse statique du code ont été préférés aux options restantes.

5.2. Observations sur la mise en œuvre des améliorations apportées

5.2.1 Prototypage

Lors du premier projet utilisant le prototypage, il a vite été constaté que c'était une activité qui prenait passablement de temps pour la création du prototype et pour laquelle il faut porter attention à plusieurs détails. Au fur et à mesure que les interfaces sont définies, nous rencontrons des problèmes qui nous font cheminer dans notre vision de l'application, ce qui nous fait souvent revenir sur le travail déjà accompli pour y apporter des ajustements.

Lorsque les premiers prototypes ont été présentés aux clients, ceux-ci ont été surpris de voir des interfaces graphiques si tôt dans le processus et ont posé quelques questions à savoir comment cela avait été fait.

Il est clair que le niveau de compréhension des clients était beaucoup plus grand qu'à l'habitude. En montrant la navigation dans l'application, ils ont pu comprendre les étapes qui allaient être effectuées par l'utilisateur. Aussi, en voyant les formulaires à remplir, ils ont tout de suite voulu savoir quel allait être le contenu des menus déroulants et ils ont posé des questions sur les interactions entre certains champs. Dans le même sens, lorsque des interfaces contenaient des tableaux, ils ont demandé d'ajouter des colonnes et aussi demandé quels allaient être les critères pour choisir les données à afficher.

Une fois validés avec les clients, les prototypes se sont avérés fort utiles pour les programmeurs. En effet, ces derniers pouvaient voir d'entrée de jeu ce qu'ils avaient à faire et pouvaient plus facilement voir l'ampleur du travail. De plus, les prototypes

ont rendu les programmeurs plus autonomes dans la mesure où ils n'ont plus à poser de questions sur la disposition des éléments dans la page. Toutefois, à quelques reprises, des interfaces qui avaient été construites ont été modifiées pour faciliter la programmation. Dans certains cas, l'information à afficher demandait en effet beaucoup de traitement et ralentissait considérablement l'application.

5.2.2 Utilisation du framework ASP.NET MVC

L'approche du ASP.NET MVC est très différente du ASP.NET Web Forms auquel les programmeurs étaient habitués. Les premiers pas ont nécessité beaucoup de lecture et de recherche. De plus, ce *framework* demande une utilisation beaucoup plus grande du langage Javascript, ce qui a ajouté à la courbe d'apprentissage. Évidemment, il a fallu développer ce premier projet à partir de rien plutôt que d'utiliser notre projet modèle. Du même coup, nous avons développé un tout nouveau style beaucoup plus moderne, avec lequel il est facile de travailler avec la librairie « bootstrap » et qui était donc compatible avec toutes les plateformes.

Avec l'utilisation du modèle MVC et du Javascript, uniquement certaines parties de la page peuvent être mises à jour. Le résultat est un gain de performance notable, car il n'est pas nécessaire de recalculer toutes les données. Il a été beaucoup plus simple d'utiliser des fenêtres modales dans l'application (superposition d'un cadre au-dessus de l'affichage actuel et qui simule l'effet d'une fenêtre), ce qui permet l'affichage rapide de plusieurs formulaires sans rafraichir la page. Puisque le traitement est découpé en plusieurs fonctions, le couplage entre les différents éléments d'une même page est beaucoup plus faible.

En contrepartie, la création des pages demande plus de code qu'en ASP.NET Web Forms. Lors de la création d'une nouvelle page, une bonne partie du travail consiste à recopier le code d'une autre page et à l'adapter pour les données à afficher.

5.2.3 Analyse statique du code

Deux outils avaient été identifiés pour faire l'analyse statique du code, soit celui fourni avec Visual Studio 2017 de Microsoft, ainsi que CodeRush Code Analysis de DevExpress.

Afin de les tester et de se familiariser avec ce genre d'outil, nous avons fait l'analyse sur deux projets déjà existants. Tel qu'anticipé, la mise en place des outils s'est faite rapidement et facilement. Toutefois, les résultats obtenus ont été extrêmement décevants. Il est vrai qu'un bon nombre d'irrégularités a été détecté par chacun des outils, mais aucune d'entre elles n'avait de réelle valeur ajoutée par rapport à la qualité de l'application. Parmi les éléments identifiés, il y avait entre autres : façon plus concise de déclarer des variables, ajout de paramètres optionnels lors des appels, entités non reconnues de certaines librairies alors qu'elles existaient.

Avec un tel résultat préliminaire, il a été décidé de complètement abandonner cette amélioration puisque de manière évidente, elle n'apportait aucune valeur ajoutée à quelque niveau que ce soit dans notre contexte.

5.3. Référentiel de *rework* mis à jour

Le processus amélioré a été utilisé en partie ou entièrement pour trois projets de développement présentés dans le tableau 5.1 avec le détail des améliorations utilisées.

Tableau 5.1 Projets ayant utilisés le processus amélioré

Id	Client	Effort (h)	Nombre demande	Prototypage	ASP.NET MVC	Analyse statique du code
12	F	937,7	150	Oui	Oui	Non
13	B	343,8	73	Oui	Non	Non
14	A	113,3	22	Non	Oui	Non

Les projets ayant utilisé le processus amélioré répondent tous aux critères de sélection du référentiel initial. Tel que mentionné à la section 5.2.3, aucun projet n'a utilisé l'analyse statique du code. Pour les deux améliorations restantes, le projet 12 a mis en place les deux améliorations, le projet 13 a seulement utilisé le prototypage et le projet 14 a seulement utilisé le *framework* ASP.NET MVC. Il serait donc possible de voir isolément l'impact des améliorations avec les projets 13 et 14.

À partir des demandes de ces projets, nous avons pu établir un nouveau référentiel, qui est présenté dans le tableau 5.2.

Tableau 5.2 Résultats du référentiel amélioré du *rework*

Catégorie	**Nb heures**	**% heure**	**Nb demandes**	**% demande**
1. Exigence initiale du projet	1012,6	72,6%	100	40,8%
2. Mauvaise définition des besoins à l'origine (R)	16,9	1,2%	9	3,7%
3. Mauvaise compréhension entre le PO et l'équipe STI (R)	8,0	0,6%	2	0,8%
4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (R)	91,4	6,6%	55	22,4%
5. Ajout de nouvelles exigences par PO	8,0	0,6%	7	2,9%
6. Ajout de nouvelles exigences par STI	118,0	8,5%	19	7,8%
7. Autres	36,1	2,6%	9	3,7%
8. Changements de technologies	55,7	4,0%	4	1,6%
9. Changements de législation	0,0	0,0%	0	0,0%
10. Problème technique (R)	48,1	3,5%	40	16,3%
11. Mauvaise connaissance de l'application (processus et données) (R)	0,0	0,0%	0	0,0%
Total général	**1394,8**	**100,0%**	**245**	**100,0%**

Le *rework* est constitué des éléments numéros 2-3-4-10-11 (surlignés en gris dans le Tableau 5.2). Donc, pour les trois projets, nous obtenons un total de 11,8% d'effort

attribué au *rework*, soit 164,4 heures d'effort sur les 1394,8 heures recensées. De ce total, les deux principaux types de *rework* sont

- 4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (6,6% de l'effort total, représentant 55,6% du *rework*),
- 10. Problèmes techniques (3,5% de l'effort total, représentant 29,3% du *rework*).

Du côté du nombre de demandes, on constate que c'est une proportion de 43,3% (106/245) qui est attribué au *rework*. Ce sont ici aussi les mêmes types de *rework* qui sont principalement en cause:

- 4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (22,4% des demandes totales, représentant 51,9% du nombre de demandes considérées comme du *rework*),
- 10. Problèmes techniques (16,3% des demandes totales, représentant 37,8% du nombre de demandes considérées comme du *rework*).

5.4. Analyse des écarts [entre les valeurs initiales et celles mises à jour]

Nous sommes maintenant en mesure de comparer les données du référentiel initial et du référentiel amélioré. Le tableau 5.3 contient les variations de l'effort entre les deux référentiels et le tableau 5.4 contient les variations en termes de nombre de demandes entre les deux référentiels. Les lignes en gris représentent les types de demandes considérées comme du *rework*.

Si l'on regarde l'effort attribué au *rework*, toutes les catégories sont en baisse, si l'on ignore la catégorie 11 pour laquelle nous n'avions pas de données au référentiel initial. Les catégories avec la plus grosse baisse sont la catégorie 4 avec une baisse de 7,0%, ainsi que la catégorie 10 avec une baisse de 6,3%. Pour contre-balancer ces baisses, c'est la catégorie 1 qui a augmenté le plus, soit une hausse de 6,8%.

Tableau 5.3 Variation de l'effort par catégorie de demande

Catégorie	% heure		
	Initial	**Amélioré**	**Variation**
1. Exigence initiale du projet	65,8%	72,6%	6,8%
2. Mauvaise définition des besoins à l'origine (R)	4,7%	1,2%	-3,5%
3. Mauvaise compréhension entre le PO et l'équipe STI (R)	0,9%	0,6%	-0,3%
4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (R)	13,5%	6,6%	-7,0%
5. Ajout de nouvelles exigences par PO	1,1%	0,6%	-0,5%
6. Ajout de nouvelles exigences par STI	1,3%	8,5%	7,2%
7. Autres	3,0%	2,6%	-0,4%
8. Changements de technologies	0,0%	4,0%	4,0%
9. Changements de législation	0,0%	0,0%	0,0%
10. Problème technique (R)	9,7%	3,5%	-6,3%
11. Mauvaise connaissance de l'application (processus et données) (R)	0,0%	0,0%	0,0%
Total général	100,0%	100,0%	0,0%

Pour ce qui est du nombre de demandes attribuées au *rework*, les catégories 2 et 10 ont des baisses significatives de 3,4% et 4,0% respectivement, alors que les catégories 3, 4 et 11 ont des hausses de moins de 1%.

En regardant les demandes de la catégorie 4 de plus près, on remarque que l'effort a diminué de 7%, alors qu'il y a eu une légère hausse de 0,5% du nombre de demandes. Ces données indiquent qu'il y a encore sensiblement le même nombre de demandes pour cette catégorie, mais que l'effort nécessaire pour chacune d'entre elles est moins grand.

Tableau 5.4 Variation de la proportion de demandes par catégorie

Catégorie	% demande		
	Initial	Amélioré	Variation
1. Exigence initiale du projet	33,0%	40,8%	7,8%
2. Mauvaise définition des besoins à l'origine (R)	7,1%	3,7%	-3,4%
3. Mauvaise compréhension entre le PO et l'équipe STI (R)	0,7%	0,8%	0,1%
4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (R)	21,9%	22,4%	0,5%
5. Ajout de nouvelles exigences par PO	2,9%	2,9%	0,0%
6. Ajout de nouvelles exigences par STI	2,9%	7,8%	4,9%
7. Autres	11,1%	3,7%	-7,4%
8. Changements de technologies	0,1%	1,6%	1,5%
9. Changements de législation	0,0%	0,0%	0,0%
10. Problème technique (R)	20,3%	16,3%	-4,0%
11. Mauvaise connaissance de l'application (processus et données) (R)	0,0%	0,0%	0,0%
Total général	100,0%	100,0%	0,0%

Le tableau 5.5 montre les variations d'effort par catégorie pour les trois projets du référentiel amélioré. Pour les deux projets ayant utilisé le prototypage, il y a eu une baisse significative pour la catégorie 4, soit 7,1% et 4,9%. Pour le projet n'utilisant pas de prototypage, il y a aussi eu une baisse importante de 11,7%. Cependant, cette variation s'explique en bonne partie par le fait que ce projet est une migration d'une application existante sous Lotus Notes vers l'environnement Web ASP.NET. Donc, d'une certaine façon, l'application existante a joué un rôle de prototype, aidant les clients à bien définir leurs besoins.

Tableau 5.5 Variation de l'effort de *rework* par rapport au référentiel initial

Catégorie	Projet 12 (Prototypage et ASP.NET MVC)	Projet 13 (Prototypage)	Projet 14 (ASP.NET MVC)
1. Exigence initiale du projet	6,5%	11,4%	-7,5%
2. Mauvaise définition des besoins à l'origine (R)	-3,8%	-2,4%	-4,7%
3. Mauvaise compréhension entre le PO et l'équipe STI (R)	0,0%	-0,9%	-0,9%
4. Modifications aux exigences suite à l'expérimentation ou à l'utilisation (R)	-7,1%	-4,9%	-11,7%
5. Ajout de nouvelles exigences par PO	-1,0%	-1,1%	5,3%
6. Ajout de nouvelles exigences par STI	10,1%	2,6%	-0,4%
7. Autres	-0,5%	0,6%	-3,0%
8. Changements de technologies	2,3%	0,3%	29,6%
9. Changements de législation	0,0%	0,0%	0,0%
10. Problème technique (R)	-6,5%	-5,6%	-6,6%
11. Mauvaise connaissance de l'application (processus et données) (R)	0,0%	0,0%	0,0%

La seconde catégorie de *rework* qui avait été ciblée, soit la catégorie 10, a aussi vu sa proportion diminuer. En effet, pour les deux projets utilisant le *framework* ASP.NET MVC, il y a eu des diminutions de 6,5% et 6,6%. Le troisième projet n'utilisant pas ce *framework* a quant à lui eu une diminution de 5,6%. Bien que cette baisse ne fut pas attendue, elle est potentiellement attribuable à la grande proportion de la catégorie 1 (11,4% de plus que le référentiel initial), où un effort plus grand qu'à l'habitude aurait été fait pour tester l'application.

Pour ce qui est de la catégorie 8 du projet 14, la variation de 29,6% s'explique par un changement pour l'appel de service Web à partir du client avec le *framework* ASP.NET MVC plutôt que du côté serveur avec ASP.NET Web Forms.

5.5. Limites de validité

Étant donné le nombre peu élevé de projets ayant appliqué le processus amélioré, nous ne pouvons pas généraliser les résultats pour le moment. Nous continuerons de quantifier le *rework* des projets de développement et de maintenance au STI afin de voir son évolution et de le garder sous contrôle.

Certains projets sélectionnés pour le référentiel initial étaient des projets de maintenance, comparativement au référentiel amélioré qui contenait uniquement des nouveaux projets de développement. Étant donné qu'il y a une phase de développement initial dans les nouveaux projets, la proportion d'effort lié à l'implémentation des exigences initiales est naturellement plus élevée. À l'inverse, les projets de maintenance ayant moins d'exigences initiales, voire aucune, la proportion de *rework* y est naturellement plus élevée. Ceci pourrait donc expliquer en partie l'écart entre le référentiel initial et le référentiel amélioré.

Les membres de l'équipe de développement n'ont pas tous les mêmes compétences. Il est donc possible que les taux de *rework* soient influencés par la personne travaillant sur le projet, particulièrement en ce qui a trait aux problèmes techniques. Nous avons volontairement écarté toute collecte et analyse de données en ce sens, pour des raisons d'intégrité professionnelle envers les collègues.

Pour les deux améliorations qui ont été mises en place, le manque d'expérience pourrait avoir un effet négatif sur le *rework*. En effet, certains problèmes techniques sont potentiellement dus au manque de connaissances du *framework* ASP.NET MVC. Dans le même ordre d'idée, le manque d'expérience avec le prototypage ne permet peut-être pas d'être aussi efficace que possible.

La saisie des demandes ainsi que la saisie de l'effort peuvent être une source d'erreur. En effet, le temps travaillé attribué à chaque demande n'est pas calculé de manière très précise car les membres de l'équipe peuvent être appelés à travailler sur plusieurs projets dans la même journée, ou encore sur plusieurs demandes. Quant à la saisie des

demandes, puisque les catégories de demandes sont un élément nouveau au STI, une demande pourrait être mal classée à cause d'une mauvaise compréhension.

Puisque nous n'avons pas utilisé de méthode d'estimation d'effort pour les projets évalués, nous ne pouvons pas déterminer si la proportion de *rework* est causée par une augmentation de l'effort total du projet.

Les demandes pour le référentiel initial n'étaient pas toujours facilement analysables et pouvaient parfois représenter deux catégories. Dans ce cas, nous avons classé la demande selon sa catégorie la plus plausible ou la plus importante. Il est possible que l'effort du référentiel par catégorie ne soit pas très précis.

CHAPITRE 6

DISCUSSION

Lesson learned: ''Classification by fault causes is hard since to obtain such a scheme with high interpretability, the categories easily become both product and process dependent''.
(Damm *et al.*, 2008)

6.1. Compréhension des causes du *rework* du processus initial

Grâce au travail effectué, nous savons maintenant d'où provient le *rework* et dans quelle proportion. En effet, en consignant toutes les demandes dans les différentes catégories, il est possible de savoir quelle est la cause du travail à faire, ainsi que de savoir s'il s'agit du *rework* ou non.

Avec de tels renseignements sur la provenance des demandes, il est maintenant possible de mettre en place des améliorations pour le processus de développement et ensuite d'en évaluer l'impact. Bien que les proportions de certaines catégories aient baissé avec le processus amélioré, il est encore possible de faire mieux. Il serait aussi possible d'apporter des ajustements en fonction de divers critères tels que le client, la nature du projet ou sa complexité.

6.2. Retombées de notre projet pour le STI et la CSDA

Tous les gestionnaires qui ont été en charge de l'équipe de développement ont souhaité améliorer l'efficience de l'équipe. Cependant, ils n'avaient pas sous la main de moyens pour comprendre où il y avait le plus de pertes et, du même coup, de référentiel pour connaître l'impact des changements sur les performances de l'équipe.

Bien que certains efforts avaient déjà été faits auparavant, la mise en place des catégories de demande a été la bougie d'allumage de plusieurs initiatives en plus du travail expliqué dans le présent document. En effet, en croisant les données de manière plus fine, il est possible d'identifier certaines problématiques (ex. mauvaise documentation des besoins à l'origine) et d'y confronter certaines pratiques de l'équipe de développement (ex. manque de documentation), des clients (ex. connaissances trop limitées de leur processus d'affaires) ou encore de la CSDA (ex. manque de communication).

L'arrivée de cette catégorisation a aussi démarré un processus d'amélioration continue par le coordonnateur, où chaque projet est analysé et comparé à la recherche d'opportunités d'amélioration.

Avec les retombées observées, le désir et le besoin d'améliorer les pratiques de génie logiciel se font sentir. Ainsi, des essais d'estimation d'effort de projet ont été faits avec des techniques d'estimation reconnues, afin de pouvoir éventuellement mieux prévoir l'effort de réalisation des projets.

6.3. Apprentissage lors des expérimentations

Lors de la création des prototypes, il ne faut pas sous-estimer l'importance de chaque détail. En présentant le prototype d'un écran, il n'y a pas de distinction à première vue entre les détails importants et les détails sans importance. Par conséquent, les clients et les développeurs ont à quelques reprises fait des commentaires sur des détails sans importance, en croyant que ceux-ci avaient une grande importance. Nous prenons donc soin de choisir chaque élément de la page comme s'il était important, ou encore d'identifier les éléments importants par rapport à ceux de moindre importance.

Un autre point par rapport aux prototypes est d'identifier des pages d'application existantes qui ont une apparence ou un comportement très similaire. Cela permet au

client de mieux comprendre le fonctionnement et au programmeur de réutiliser du code existant.

La classification des demandes n'est pas toujours simple. Dans tous les projets, il y a des cas limites ou des cas qui entrent difficilement dans les catégories. Mais puisqu'il faut toujours sélectionner une catégorie, il a été déterminé d'utiliser la catégorie « 7. Autres » si ce n'est pas du *rework*, ou la catégorie la plus plausible si c'est du *rework*.

Bien que les *frameworks* de programmation dictent une certaine ligne directrice, il y a tout de même une grande marge de manœuvre laissée au programmeur. Il est donc important de comprendre l'impact des décisions prises afin de ne pas nuire à la suite du développement. Par exemple, certaines fonctionnalités du *framework* ASP.NET MVC n'ont pas été utilisées, en croyant qu'elles ne répondaient pas à nos besoins. Cependant, il s'est avéré que ce choix était basé sur une mauvaise connaissance du *framework* et que cela aurait pu améliorer la productivité et simplifier le code.

6.4. Compréhension des écarts

À la lumière des résultats obtenus, il est clair que le processus amélioré a eu un effet positif sur la proportion d'effort de *rework*. En y regardant de plus près, nous observons que les variations d'effort représentent exactement les résultats espérés. Effectivement, la création des prototypes ainsi que l'appropriation de ASP.NET MVC ont contribué à faire augmenter la proportion d'effort lié aux exigences initiales, mais ont aussi permis de faire diminuer l'effort de *rework*.

Ce qu'il faut comprendre de ces écarts, c'est qu'il y a un coût à faire diminuer le *rework* et les problèmes techniques. En prenant plus de temps pour effectuer certaines activités ou pour utiliser des *frameworks* plus évolués, c'est comme investir dans un projet afin qu'il devienne de meilleure qualité et ainsi éviter d'avoir de mauvaises surprises plus tard.

Du même coup, les résultats démontrent que, du point de vue de la qualité, l'équipe de développement est plus performante avec le processus amélioré qu'elle l'était avec l'ancien.

6.5. Mes apprentissages

Le travail effectué pour améliorer les performances de l'équipe de développement m'a permis d'apprendre plusieurs choses. Dans un premier temps, la mise en place d'un référentiel initial est essentielle à tout travail d'amélioration et ce dernier doit permettre d'avoir une image concrète de la situation. Une fois que cette étape est franchie, une multitude d'opportunités s'ouvrent devant nous, puisqu'il est alors possible d'avoir une approche scientifique et de vérifier l'impact de chaque modification avec des faits, plutôt qu'avec des perceptions et des impressions.

Ensuite, j'ai compris l'importance de s'adapter à l'environnement dans lequel on travaille. Lors du choix des améliorations à mettre en place pour le nouveau processus, il y avait une multitude de choix très intéressants qui avaient un grand potentiel. Cependant, le contexte n'était pas nécessairement favorable à de tels changements. C'est pourquoi les améliorations choisies avaient un impact limité. Avec un premier pas d'amélioration franchi et la nouvelle possibilité de quantifier les améliorations, l'environnement devient de plus en plus favorable à des améliorations ayant un impact plus grand et qu'il n'aurait pas été pensable de mettre en place il y a six mois.

Finalement, j'ai réalisé l'importance des mesures afin d'y confronter les perceptions que nous avons à l'esprit avec le temps. Nous croyions qu'un de nos clients avait une proportion de *rework* beaucoup plus importante que les autres. Cependant, une fois le référentiel initial terminé, la différence n'était pas disproportionnée en comparaison des autres clients. C'était plutôt les échéances très courtes qui étaient responsables de cette perception.

CONCLUSION

Le développement étant toujours plus en demande à la CSDA, le besoin d'optimiser le temps de développement devenait de plus en plus grand. Dans ce contexte, il nous est demandé de faire plus avec le même personnel, puisque les embauches et la sous-traitance ne sont pas possibles. Nous nous sommes donc posé la question « Comment peut-on augmenter l'efficacité de l'équipe en tenant compte de son contexte? ». Pour y répondre, notre stratégie a été de quantifier le *rework* pour comprendre où en étaient les plus grandes sources, afin de pouvoir agir à les atténuer.

Notre premier objectif - Identifier et quantifier le *rework* initial de développement logiciel - a été atteint en quantifiant le *rework* à partir d'une catégorisation minutieuse de plus de 880 demandes de développement, réparties dans 11 projets. Cela nous a permis d'identifier les plus grandes sources de *rework*, pour ensuite prendre action pour les limiter le plus possible. Nous avons alors proposé neuf solutions possibles pour améliorer le processus de façon à diminuer le *rework*.

Quant à l'objectif #2 - Améliorer le processus de développement afin de diminuer le *rework* - nous l'avons atteint en appliquant trois des améliorations proposées (utilisation du *framework* ASP.NET MVC, analyse statique du code, ainsi que le prototypage) sur trois projets. De ces améliorations, nous avons observé une diminution marquée du *rework*. Il faudra toutefois un échantillonnage plus grand de projets pour tirer de véritables conclusions.

La prochaine étape sera d'appliquer ces améliorations aux projets à venir. Maintenant que nous sommes capables de quantifier indirectement l'efficacité de l'équipe, nous pourrons tenter d'améliorer à nouveau les processus en regardant le contexte particulier multi-projets et multi-clients dans lequel évolue l'équipe de

développement. De plus, il serait intéressant de confronter l'effort total des projets avec des estimations formelles, afin de quantifier l'impact des améliorations.

BIBLIOGRAPHIE

Affluents, C.s.d. (2017, 2017-06-27). *Budget 2017-2018*. Récupéré de https://www.csda.ca/static/media/uploads/budget-csda-2017-2018.pdf

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mallor, S., Shwaber, K. et Sutherland, J. (2001) *The Agile Manifesto*. Récupéré le 2017-07-10 de http://agilemanifesto.org/iso/fr/manifesto.html

Boehm, B. (1987). Improving software productivity.

Boehm, B. et Basili, V.R. (2001). Software Defect Reduction Top 10 List. *Computer, 34*(1), 135-137.

Damm, L.-O., Lundberg, L. et Wohlin, C. (2008). A model for software rework reduction through a combination of anomaly metrics. *Journal of Systems and Software, 81*(11), 1968-1982. doi: http://dx.doi.org/10.1016/j.jss.2008.01.017 Récupéré de http://www.sciencedirect.com/science/article/pii/S0164121208000186

Fairley, R.E. et Willshire, M.J. (2005). Iterative rework: the good, the bad, and the ugly. *Computer, 38*(9), 34-41. doi: 10.1109/MC.2005.303

Huo, M., Verner, J., Zhu, L. et Babar, M.A. (2004). Software quality and agile methods. Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International (p. 520-525). : IEEE

Huzooree, G. et Ramdoo, V.D. (2015a). Strategies to reduce rework in software development on an organisation in Mauritius. *International Journal of Software Engineering & Applications, 6*(5), 9-20.

Huzooree, G. et Ramdoo, V.D. (2015b). A Systematic Study on Requirement Engineering Processes and Practices in Mauritius. *International Journal, 5*(2).

IEEE. (2010). *Systems and software engineering -- Vocabulary ISO/IEC/IEEE 24765:2010(E)* : IEEE.

IRIS. (2017) *Éducation primaire et secondaire | Observatoire des conséquences des mesures d'austérité au Québec.* Récupéré le 2017-09-16 2017 de http://austerite.iris-recherche.qc.ca/education-primaire-et-secondaire#contenu

LégisQuébec. (2017a, 2017-06-01) *Loi sur l'instruction publique* Publications Québec. Récupéré le 2017-09-18 de http://legisquebec.gouv.qc.ca/fr/ShowDoc/cs/I-13.3

LégisQuébec. (2017b, 2017-06-01) *Loi sur la gouvernance et la gestion des ressources informationnelles des organismes publics et des entreprises du gouvernement* Publications Québec. Récupéré le 2017-09-18 de http://legisquebec.gouv.qc.ca/fr/ShowDoc/cs/G-1.03

Markiewicz, M.E. et Lucena, C.J.P.d. (2001). Object oriented framework development. *Crossroads, 7*(4), 3-9. doi: 10.1145/372765.372771

Michlmayr, M., Hunt, F. et Probert, D. (2005). Quality practices and problems in free software projects. Proceedings of the First International Conference on Open Source Systems (p. 24-28).

Nagappan, N. et Ball, T. (2005). Static analysis tools as early indicators of pre-release defect density. Proceedings of the 27th international conference on Software engineering (p. 580-586). St. Louis, MO, USA : ACM

Pressman, R.S. (2010). Software Engineering: A Practitioner's Approach, 7/e.

Radio-Canada. (2011). Les commissions scolaires dénoncent les compressions budgétaires. *Radio-Canada.ca*(2011-05-02). Récupéré de http://ici.radio-canada.ca/nouvelle/514115/commissions-scolaires-coupures-quebec

Radio-Canada. (2013). 65 millions de moins dans les commissions scolaires. *Radio-Canada.ca.* Récupéré de http://ici.radio-canada.ca/nouvelle/609763/compressions-commissions-scolaires-quebec

Sutherland, J. et Schwaber, K. (2013). The scrum guide. *The Definitive Guide to Scrum: The Rules of the Game. Scrum. org.*

Tonnellier, E. et Terrien, O. (2012). Modélisation et quantification du Rework.