Équipe FACT
Conception et développement d'applications

http://fact-team.github.io

Rédigé avec LaTeX
Version du 6 février 2015

Plan d'Assurance Qualité Logicielle

FactDev

Université Toulouse III – Paul Sabatier

— **F**lorent Berbie
— **A**ntoine de Roquemaurel
— **C**édric Rohaut
— Andriamihary Manan**T**soa Razanajatovo

6 février 2015

Table des matières

Avant-Propos

Le Plan d'assurance Qualité Logicielle a pour objectif la définition et le suivi des dispositions à prendre dans le cadre du projet *FactDev* afin d'en assurer la qualité, une bonne gestion et d'atteindre les résultats attendus.

À cet effet, le Plan d'Assurance Qualité Logicielle fixe les droits, devoirs et responsabilités de chaque partie prenante en vue d'assurer l'atteinte de ces objectifs.

Il constitue un outil de travail et un référentiel commun à tous les acteurs pour leur donner une vision similaire du projet, mais il constitue également le cahier des charges de la qualité, réalisé en collaboration avec le client puis approuvé par celui-ci. Il constitue enfin la définition des procédures à suivre, des outils à utiliser, des normes à respecter, de la méthodologie de développement du produit et des contrôles prévus pour chaque activité.

Ainsi, d'un commun accord sont déterminés ces différents aspects du projet qui constituent un contrat entre le titulaire et le client et toutes les autres parties prenantes. Ce contrat prend effet dès son acceptation par le client et les personnes concernées et peut être, si les circonstances l'obligent, amené à être modifié au cours du projet. Dans ce cas, toute évolution future sera soumise à l'acceptation du client car au terme du projet, le Plan d'Assurance Qualité Logicielle constituera l'un des documents de résultat du projet.

1 Description du Projet

1.1 Objet du projet

Le logiciel a pour but de faciliter la création de devis et la conversion de ces devis en factures.

Pour cela il sera possible d'enregistrer des clients dans une base de données et d'offrir une gestion de ces derniers. Un client peut avoir un ou plusieurs projets avec, pour chacun, un ou plusieurs devis ou factures.

1.2 Présentation des Parties Prenantes

1.2.1 Client : Antoine de bscRoquemaurel

Développeur Freelance, et membre de l'équipe de développement.

☎ 06 54 33 52 93
https://antoinederoquemaurel.github.io
✉ antoine.roquemaurel@gmail.com

1.2.2 Encadrant : Fédéric Migeon

Maître de conférence à l'Université Toulouse III – Paul Sabatier

☎ 05 61 55 (62 46)
✉ Frederic.Migeon@irit.fr
IRIT1 / Niveau 3, Pièce : 361

1.2.3 Responsable de l'UE Projet : Bernard bscCherbonneau

☎ 05 61 55 (63 52)
✉ Bernard.Cherbonneau@irit.fr
IRIT1 / Niveau 4, Pièce : 413

1.2.4 Titulaire : Équipe FACT

Étudiants en M1 Informatique Développement Logiciel à l'université Toulouse III – Paul Sabatier

Florent Berbie
☎ 06 85 31 92 90
✉ florent.berbie@gmail.com

Antoine de Roquemaurel
☎ 06 54 33 52 93
✉ antoine.roquemaurel@gmail.com

Cédric Rohaut
☎ 06 74 80 12 67
✉ rohaut@icloud.com

Manantsoa Andriamihary Razanajatovo
☎ 06 01 71 53 02
✉ manantsoa.razana@gmail.com

2 Production

2.1 Méthode de développement

2.1.1 Définition et pertinence de la méthode scrum

Le développement du projet se fera selon la méthode Agile *Scrum*, comme cela a été convenu avec notre encadrant.

Cette méthode, basée sur les stratégies itératives et incrémentale, permet de produire à la fin de chaque *Sprint* (incrément/itération) une version stable et testable du logiciel. Les différents événements associés à *Scrum* accroissent la communication grâce à des réunions quotidiennes aussi appelées *mêlées*. Ceci permet une meilleure cohésion, une meilleure coopération et une meilleure homogénéité du travail fourni par les membres de l'équipe. A cela s'ajoute la présence d'artefacts, c'est-à-dire des éléments à réaliser avec des ordres de priorité et qui contribuent à améliorer la productivité.

Dans le cadre de ce projet, la méthode *Scrum* s'avère être pertinente pour plusieurs raisons : Premièrement, dans la mesure où nous avons proposé un sujet et spécifié les fonctionnalités de celui-ci, la méthode *Scrum* se veut adaptée. En effet, les fonctionnalités que nous avons proposé permettent de définir les limites de notre première version *Release* livrable.

L'ajout de fonctionnalités en fonction des attentes du client pourront être implémentées au fur et à mesure des différents *Sprints*. Cela a pour avantage de fournir un travail continu, d'assurer un suivi avec le client pour répondre au mieux à ses besoins. De plus, la durée du projet étant relativement courte, il serait difficile de revenir sur notre conception préalable alors qu'ici chaque *Sprint* permet de s'assurer que le projet avance dans la bonne direction.

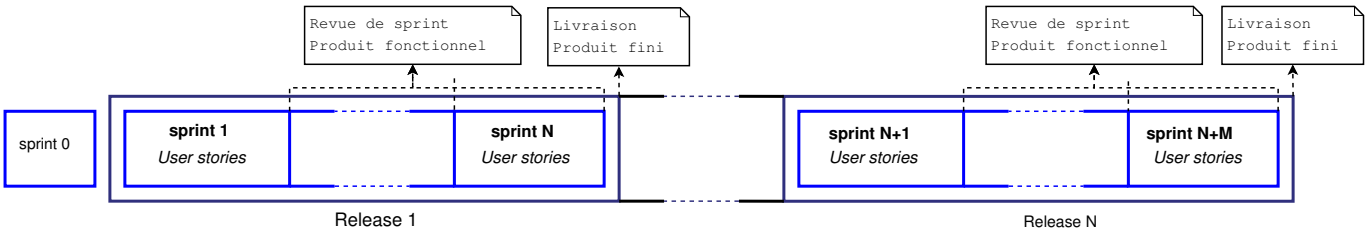

FIGURE 2.1 – Fonctionnement des *Sprints* et *Releases* de la méthode *Scrum*

Outre les avantages qu'apporte la méthode *Scrum* à ce projet, l'équipe avait la volonté d'évoluer vers une méthode qui diffère de celles qui ont pu être abordées en cours. Cette volonté est d'ailleurs

confortée par le désir de découvrir de nouvelles technologies (C++ et Qt jusque-là peu connus par la majorité du groupe).

2.1.2 Application de la méthode Scrum et critères de qualité

Dans le cadre de notre projet, la durée d'un *Sprint* a été défini à deux semaines. Un *Sprint* est constitué de *User Stories* et de *Technicals Stories* qui sont préalablement définies lors de réunions quotidiennes aussi appelées *mêlées*. Ces *mêlées* sont quotidiennes, nous profitons de notre temps libre dans l'emploi du temps pour les organiser. Pour les jours où ce n'est pas possible, nous avons mis en place un salon de discussion IRC.

La définition des *Technical Stories* indique brièvement la fonction que l'on doit réaliser. Les *User Stories* sont quant à elles toujours posées sous cette forme :

> **En tant que** `[Personne(s) utilisant le logiciel]`
> **Je souhaite** `[La fonctionnalité que je désire avoir]`
> **Afin de** `[Objectif de la fonctionnalité]`

Chaque *Story* possède un poids, c'est-à-dire une valeur indiquant sa complexité et/ou le temps nécessaire afin de la mettre en œuvre. Le poids de chacune des *Story* est déterminé durant les *mêlées* au moyen d'un « Planning poker ». Ceci permet à chacun d'exprimer la complexité d'une *Story* à réaliser. Chacun justifie le choix du poids qu'il a affecté et après un éventuel débat, on affecte cette valeur à la *Story*. Celle-ci ne peut être affectée qu'une fois que l'ensemble des membres de l'équipe se sont mis d'accord. Elle possède également une priorité précisant l'importance d'intégrer cette *Story* ans le *Sprint*. Cette priorité se fait au moyen de quatre niveaux d'importance :

- *Must* : La *Story* doit obligatoirement être réalisée lors du *Sprint*
- *Should* : La *Story* devra être réalisée (dans la mesure du possible)
- *Could* : La *Story* pourra être réalisée car elle n'a aucun impact sur les autres tâches
- *Would* : La *Story* ne sera pas nécessairement faite et sera alors reportée au prochain *Sprint*

Une *Story* est considérée comme finie lorsqu'elle est fonctionnelle d'un point de vue utilisateur c'est-à-dire :

- Les tests unitaires (pour la base de données ou pour les modèles) sont validés
- Les tests d'intégrations sont validés
- Lorsque chaque méthode est documentée

À la fin du *Sprint*, on présente à notre client, Monsieur Frédéric Migeon, afin qu'il le valide. Un *Sprint* fournit toujours :

- une démonstration des nouvelles fonctionnalités logicielles
- des tests unitaires
- une documentation du code (au format HTML et PDF)
- un manuel d'utilisateur à jour des nouvelles fonctionnalités

Notre projet sera réalisé en deux *Releases*, c'est-à-dire qu'il y aura deux versions livrables. Chacune de ces versions est composée de trois *Sprints* ayant chacun un poids moyen de 50 pour environ 10 *Stories*.

Les versions livrables du logiciel sont définies dans un carnet de *Sprint* (*Sprint Backlog*) qui référence l'ensemble des *Stories* des différents *Sprints*. Ce carnet est susceptible d'évoluer en fonction des besoins du client, de nouveaux choix de conception ou encore d'éléments n'ayant pas été prévu.

2.1.3 Organisation et rôles dans l'équipe de développement

La méthode *Scrum* possède des rôles qui lui sont propres : le *Scrum Master*, le *Product Owner* et l'équipe de développement.

2.1.3.1 Scrum Master : Florent Berbie

Le *Scrum Master* aura pour mission principale de guider les développeurs dans l'application de la méthode *Scrum*. Il veillera à ce que la méthode soit comprise de tous et appliquée de façon correcte. Il aura le rôle de meneur lors de phases importantes d'application de la méthode telles que le planning poker ou encore les mêlées quotidiennes.

2.1.3.2 Product owner : Antoine de Roquemaurel

Le *Product owner* est la seule personne responsable du carnet de produit et de sa gestion. Ce dernier comprend l'expression de tous les items associés à une priorité (l'importance pour le client). La compréhension de ceux-ci ainsi que la vérification du travail fourni est sous la responsabilité du *Product owner*.

2.1.3.3 Équipe de développement : Florent Berbie, Antoine de Roquemaurel, Cédric Rohaut, Andriamihary Razanajatovo

Afin d'assurer une plus grande cohésion entre les membres de l'équipe, un même niveau d'implication et un travail de plus grande qualité nous avons définis des rôles spécifiques :

2.1.3.4 Directeur qualité : Cédric Rohaut

Le responsable qualité sera garant de la transposition des exigences du client sous forme de solutions techniques au sein du logiciel final. En d'autres termes, il veillera à ce que le logiciel apporte une solution technique optimale pour le client. De plus il devra vérifier et valider la qualité du logiciel au travers de tests, du respect des conventions et de l'aspect général du logiciel.

2.1.3.5 Directeur documentation : Andriamihary Razanajatovo

Le responsable documentation sera chargé de la révision de l'ensemble des documents avant leur remise ou leur soumission aux parties prenantes concernées. Il veillera à la qualité des aspects fondamentaux, à savoir le fond et la forme, des documentations à fournir.

2.1.3.6 Directeur technique Qt, C++ : Antoine de Roquemaurel

Le responsable technique Qt, C++ sert de support à l'équipe en cas de problèmes techniques liés au développement sur la plate-forme Qt. De par ses connaissances acquises dans ce domaine, il sera le plus à même à aider les membres de l'équipe projet ayant des difficultés avec cette technologie. De plus, dans le but de minimiser les couplages lors de la conception, il veillera à ce que le patron de conception MVC [1] soit correctement utilisé. Ainsi, le code sera clairement découpé en trois parties : modèle, vue et contrôleur ce qui permettra de s'assurer des contrôles d'interactions entre nos composants logiciels.

2.1.3.7 Autres rôles et responsabilités de chaque développeur et organisation/fonctionnement général

Dans un souci d'assurance qualité, chaque responsable est en mesure de présenter les difficultés qu'il a eu à gérer, de présenter les différents solutions possibles à ce problème et de justifier le choix de la technique adoptée.

Lors de la réalisation d'un cas d'utilisation (*issue* sur *Github*), celui-ci est assigné à un développeur. Une fois que ce dernier considère sa tâche comme finie, il indique (via une *Pull Request*) aux autres membres de l'équipe que la tâche est soumise à la validation et à l'intégration. Un des autres membres vérifie que la fonction est conforme à sa description, que le code est facilement compréhensible et correctement commenté. Dans le cas ou la revue de code ne donne pleine satisfaction, le membre effectuant cette revue et les autres membres de l'équipe pourront ajouter des commentaires pour débattre de la fonctionnalité, de l'implémentation de la fonction ou des technologies employées pour y résoudre. Chacun pourra alors soumettre sa vision du problème et la manière avec laquelle il aurait résolu le problème.

2.2 Les Technologies utilisées

2.2.1 Architecture Logicielle

L'objectif du logiciel est d'éditer des devis et factures et de proposer une gestion de ses clients : le client possède un ou plusieurs projets auxquels sont associés un ou plusieurs devis et/ou factures. De ce fait, l'architecture logicielle comporte une base de données de type SQLite pour la gestion des clients, de ses projets, des devis et factures associés à chacun des projets et des prestations propres à chaque devis et facture. Nous utilisons un patron de conception MVC, avec un modèle qui correspond aux objets métiers Client, Projet, Factures, Devis, Prestations. Ces objets sont instanciés par les classes associées à la base de données qui réalisent ces tâches. Enfin nous générerons les devis et factures en LaTeX ou en PDF.

1. Modele, Vue, Contrôleur

2.2.2 Environnement de développement

Le logiciel est développé à partir du *framework*[2] Qt(version 5) en langage de programmation C++ (version 11) avec l'EDI[3] Qt Creator (version 3.3).

Les membres de l'équipe développent sur différents systèmes : MAC OS (OS X 10.10), Linux (Fedora 20 et Linux Mint 17.1). Bien que développé uniquement sur des systèmes UNIX, l'application sera compatible avec Windows.

Afin de mener au mieux ce projet, l'équipe utilise l'outil *Github*, un service web d'hébergement et de gestion de développement de logiciels.

2.2.3 Outils pour la Gestion de Projet Agile

La développement de notre logiciel s'appuie sur *Github* qui permet la gestion des différentes versions de notre logiciel. Cet outil est composé d'une branche directrice (appelée *Master*), sur laquelle se greffe de nouvelles branches : une branche par *Sprint*. L'ensemble des *User Stories* que nous avons défini sont représentées dans *Github* par des *Issues*. Chaque *Issue* fait l'objet d'une nouvelle branche provenant de celle du Sprint associé. L'avantage que propose cette gestion permet à chacun de développer de son côté sans entrer en conflit avec les autres membres de l'équipe. Une fois l'issue terminée et validée par la revue de code, l'on fusionne la branche de l'issue à celle du Sprint associée afin que l'équipe bénéficie de la nouvelle version, stable et fonctionnelle. On procède de la même façon en fin de *Sprint* en intégrant la branche de *Sprint* à la branche principale *Master*.

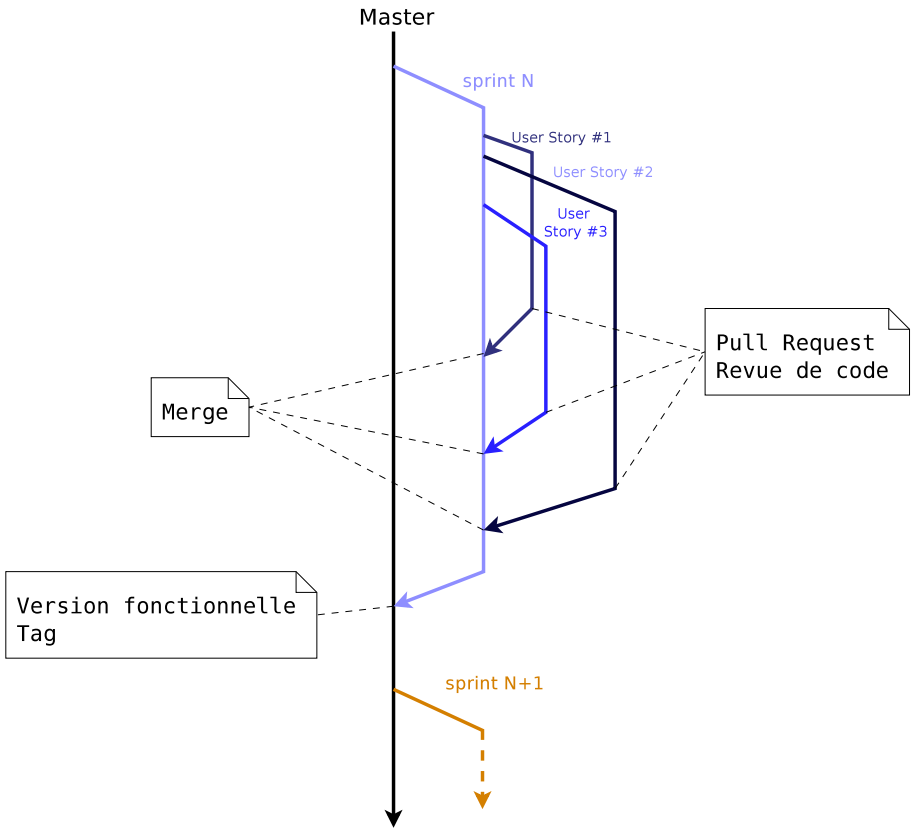

Figure 2.2 – Principe du « Git Branching Workflow »

2. Ensemble de composants logiciels structurels servant à créer les fondations d'une future application. Un framework est créé par des développeurs pour d'autres développeurs. Un framework peut être vu comme une boîte à outils.
3. Environnement de Développement Logiciel

Outre la gestion des différentes versions de notre code, *Github* permet un suivi de notre projet. Le site internet *Github* dédié au projet est accessible à l'équipe et notre client Monsieur Frédéric Migeon. De plus, *Github* contient un Wiki avec les règles de bonne conduite et les conventions à respecter pour assurer l'homogénéité de notre projet.

2.2.4 Documentation

La centralisation des documents se fait au moyen du Wiki de *Github* et de Google Drive. Le Wiki de *Github* contient : Tutoriel sur les méthodes de travail (Git, Scrum, C++) Convention de code à respecter (conventions de nommage des variables, des méthodes, d'organisation du code...) Tutoriel sur la documentation (*Doxygen*)

Sur Google Drive, on trouve l'ensemble des documents à rédiger en équipe. Cet outils permet notamment d'éditer un même document à plusieurs et d'être facilement accessible par tout le monde. Parmi les documents accessibles sur le Drive, on trouve :

- le manuel de l'utilisateur
- le plan qualité
- le Carnet des produits (« Product Backlog »)
- les graphiques d'avancement :
  - Burndown chart
  - BurnUp chart
- les Comptes rendus mensuels
- d'autres documents concernant le projet mais n'étant parmi ceux à rendre

2.2.5 Assurance de la qualité du code

Pour assurer une meilleure homogénéité, lisibilité et maintenabilité du code, nous utilisons l'outil *SonarQube*, un logiciel permettant de mesurer la qualité du code source en continu. //Celui-ci analyse et fournit diverses informations sur le code tel que :

- le pourcentage de code non documenté (ou pas assez documenté)
- la complexité générale du logiciel ou, au car pas car, des méthodes de notre programme
- le bon respect des conventions de codages (nommage des attributs/méthodes, indentation, ...)
- la couverture de code (via les tests unitaires) la duplication de code

3 Fournitures et Livrables

Durant le projet, l'équipe fournira plusieurs livrables pour l'ensemble des parties prenantes et des enseignants de l'UE Projet :

- Plan d'assurance Qualité Logicielle (signé par le client et déposé sur Moodle)
- Compte-rendu mensuel d'activité (envoyé par courriel au responsable de l'UE)
- Bilan de projet
- Graphiques d'avancement
  - Burndown chart
  - BurnUp chart
- Manuel d'utilisateur
- Carnet des produits (« Product Backlog »)
- Revues de sprint
- Builds[1]

Le resultat du projet sera presente par le Titulaire au Client lors de la recette, qui aura lieu fin avril 2014. Elle se fera en possession du cahier de recette qui contient la description des controles et tests de recette prevus.

Le deroulement du projet et les resultats obtenus seront presentes oralement devant un jury lors d'une soutenance en groupe du Titulaire, accompagnee de diapositives. Cette présentation orale d'une duree maximale de quinze minutes sera suivie par un temps reserve aux questions. Les temps de parole entre les differents orateurs seront convenablement repartis, de sorte que chacun ait un temps de parole d'une duree equivalente.

1. Un buid est un artefact logiciel autonome résultant de conversion de fichiers de code source en code exécutable

A Table des figures

Index