Équipe FACT
Conception et développement d'applications

http://fact-team.github.io

Rédigé avec LaTeX
Version du 19 avril 2015

Bilan du Projet

FactDev

Université Toulouse III – Paul Sabatier

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

Rédigé par
Équipe FACT

Approuvé par
Frédéric Migeon

19 avril 2015

Table des matières

Introduction

FactDev est un logiciel de devis et de facturation réalisé dans le cadre de l'UE Projet.

Ce projet s'est fait en réponse à un problème de l'un des membres du groupe : Antoine De Roquemaurel. En effet Antoine, développeur *Freelance*, rédigeait pour ses clients les factures et devis « à la main ». La tâche était répétitive et le risque d'erreurs humaine important :

— erreur dans le calcul des montants
— perte de facture
— plusieurs factures différentes pour un même projet et donc risque de ne pas faire le bon travail demandé par le client

Face à ces difficultés, Antoine a soumis le projet devant répondre aux spécifications suivantes :

— Gestion des clients
— Gestion des projets associées aux clients
— Calculs des tarifs
— Génération de documents
— Recherche

Les autres membres ont vu dans ce projet l'opportunité de développer de nouvelles compétences, aussi bien sur le plan technique qu'organisationnel. D'un point de vue technique avec l'utilisation du *LaTeX* et du *C++* accompagné du *framework Qt*. Sur le plan organisationnel avec la mise en place de la méthode Agile *Scrum*.

Ce document fait état des parties prenantes, des outils et des méthodologies de développement appliquées au projet. Il fait le point sur les résultats obtenus au niveau de la méthodologie, du logiciel et en terme de respect du plan qualité.

1 Le projet

1.1 Le logiciel FactDev

Le logiciel FactDev a pour but de faciliter la création de devis et la conversion de ces devis en factures.
Il répond aux exigences définies lors de l'introduction en permettant l'enregistrement d'un nouveau client dans la base de données et de projets associés à ce clients. Par exemple, un client X peut demander la réalisation d'un logiciel pour le management de son entreprise et un site web pour promouvoir son activité. On a donc deux projets distincts qui peuvent cependant faire l'objet d'une seule facture. De plus, une facture est fréquemment précédé d'un devis, c'est pourquoi il doit être facile de transformer un devis en facture. Enfin le logiciel devra tenir de certaine réglementation «légale » tel que l'impossibilité de modifier une facture ayant été payée.

1.1.1 Présentation des Parties Prenantes

1.1.1.1 Client : Antoine de Roquemaurel

Développeur Freelance et membre de l'équipe de développement.

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

1.1.1.2 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

1.1.1.3 Responsable de l'UE Projet : Bernard Cherbonneau

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

1.1.1.4 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

1.2 Les outils

2 La méthodologie

2.1 La méthode Scrum

Cette méthode, basée sur les stratégies itératives et incrémentales, permet de produire à la fin de chaque Sprint (incrément/itération) une version stable et testable du logiciel. L'avantage par rapport aux méthodes plus « classiques » (cycle en V,...) se situe principalement dans l'absence d'effet tunnel durant le développement. De par les nombreuses réunions avec les différentes parties prenantes, le ou les clients peuvent donner un retour (feedback) sur l'incrément qui leur est proposé et ainsi s'assurer que le produit final correspondra parfaitement à leurs attentes.

2.1.1 Définitions

2.1.1.1 Le Sprint

Un sprint correspond à un incrément dans la méthode *Scrum*. Il peut durer entre quelques heures et un mois (pour se situer un peu, nos sprints duraient deux semaines). Le Sprint se déroule toujours de la même façon :

Des *mêlées* quotidiennes durant laquelle sont décidées les différentes *User stories* et *Technical stories* au travers d'un *Planning Poker*.

Une revue de Sprint durant laquelle le logiciel est présenté au client.

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

2.1.1.2 Les mêlées

Les *mêlées* sont des réunions quotidiennes durant lesquelles sont définies les différentes *User stories* et *Technical stories* à accomplir. Ces dernières sont toutes décidées à travers un *Planning Poker*.

2.1.1.3 Le Planning Poker

C'est au cours du *Planning Poker* que sont décidées les différentes *User stories* du *Backlog* de produit qui vont être accomplies. Pour chaque *User story* est défini un niveau de priorité :

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

2.1.1.4 User story «finie»

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

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

2.1.1.5 Le Backlog

Le *Backlog* du produit correspond à une liste de toutes les *User stories* et *Technical stories* des différents *Sprints*. Chaque élément du *Backlog* produit représente une fonctionnalité, un besoin, une amélioration ou un correctif, auquel sont associés une description et une estimation du poids de l'*User story* ou la *Technical story*.

2.1.2 Les différents rôles

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

2.1.2.1 Le Scrum Master

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.

2.1.2.2 Le Product Owner

Le Product owner est la seule personne responsable du carnet de produit et de sa gestion. Ce carnet 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.

2.2 L'intégration continue

2.3 La revue de code

La revue de code représente une démarche que nous avions mis en avant dans le plan qualité. L'objectif visé est de tendre vers un projet dont l'intégralité du code a été revu.

La revue de code se fait au moment de l'intégration, c'est pourquoi toutes intégrations nécessitent la création préalable d'une *Pull Request*. Pour cela, une fois le travail correspondant à une *User story* est fini [1], le développeur crée une *Pull Request* via l'outil *Github*. Cette *Pull Requests*'accompagne d'une description plus technique de la *User story* à laquelle elle est liée. La liste des *commits* associés et le code source ajouté et/ou modifié est accessible comme l'on peut le constater ci-dessous.

Figure 2.1 – Exemple de *Pull Request* du projet *FactDev*

À partir de cette *Pull Request*, les autres membres de l'équipe reçoivent une notification pour indiquer qu'ils doivent procéder à la revue de code. Au moins l'un deux doit s'assurer que le code est valide. Un code est dit valide lorsqu'il :

est lisible Le code doit être facile à lire.

est compréhensible Le code doit être facilement compréhensible, avoir un niveau de complexité minimum.

respecte les conventions d'écritures Respect des conventions d'écritures (convention de nommage et de mise en forme).

Cette vérification est aisé via l'outil *Github* qui permet de rajouter des commentaires « *inline* » c'est-à-dire d'ajouter un commentaire aux lignes précise de code à modifier. Ainsi, sans toucher au code, l'on sait précisément l'endroit où l'on doit procéder à des changements.

1. selon nos critères définis lors de la présentation de la méthode *Scrum* 2.1

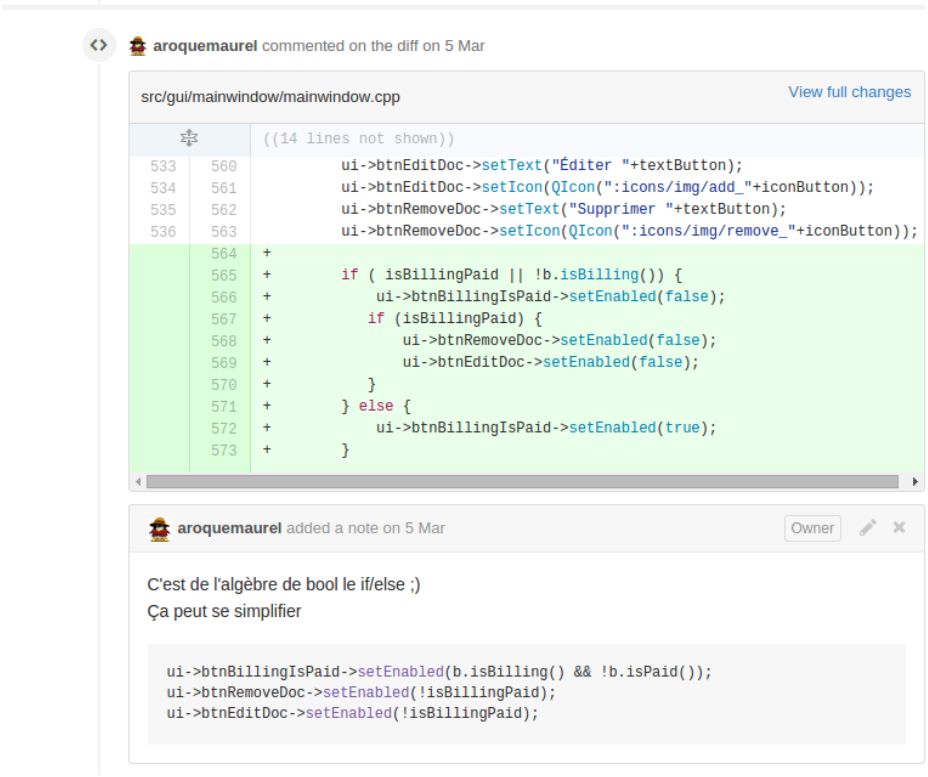

FIGURE 2.2 – Ajout d'un commentaire « inline » lors d'une *Pull Request*

L'on procède également à la vérification de la documentation. Chaque méthode et attribut doit être documenté. Là aussi, il faut que la documentation respecte les conventions d'écritures.

Une fois les remarques faites sur le code et sa documentation, l'on passe aux tests fonctionnels. On se rend donc sur la branche *Git* en question et l'on vérifie que la fonction répond bien à la *User story* et qu'il n'existe aucun bug. Bien entendu, on s'assure que ça n'a pas entraîné de régression sur d'autres parties du logiciel. C'est aussi le moment de proposer des modifications sur le plan ergonomique si besoin est.

Enfin, avant d'intégrer, l'on vérifie que les outils (*Travis CI* et *Coveralls*) ne s'y opposent pas c'est-à-dire que :

— le *Build*[2] passe, c'est-à-dire qu'il compile sans erreur.
— la couverture de code n'a pas régressé

2. Un *Build*est un artefact logiciel autonome résultant de conversion de fichiers de code source en code exécutable

3 Les résultats

L'UE projet était l'occasion de se positionner dans une situation très proche de celle que allons rencontré en milieu professionnel. Dans ce cadre là, il était nécessaire de prendre des mesures organisationnel afin de parvenir aux résultats escomptés.

3.1 La méthodologie

La qualité du logiciel *FactDev* est principalement dû à la rigueur dans l'application de la méthodologie de développement.

Elles fûrent définies en même temps que la conception du logiciel durant le *Sprint 0*. Cela comprend les attitudes à adopter par les membres de l'équipe sur le plan organisationnel et techniques avec le respect de conventions qui ont aussi été défini à ce moment là.

3.1.1 Le respect de la méthode Scrum

Au niveau de la gestion du développement par la méthode *Scrum* nous avons tenu un *Backlog* A. Dans celui-ci se trouve l'ensemble des *User stories* et *Technical stories* que nous avions à réaliser durant le projet. Ces *Stories* ont été déterminée lors de *Planning Poker*durant les mêlées. C'est durant ces séances que nous déterminions le poids attribué à chacune des *Stories* et comment les répartir sur les six *Sprints* des deux *Releases*.

La répartition des *Stories* s'est faite en fonction de leur poids mais aussi en tenant compte de l'équipe et des événements pouvant survenir. Ainsi le premier *Sprint* été plutôt léger car il tenait compte du temps d'apprentissage du langage C++ et du framework Qt. Les autres *Sprints* sont en revanche équilibré. Les deux derniers sont également plus léger sur le plan technique car nous voulions consacrer du temps à la rédaction des documents et à la préparation de la soutenance. De plus, certaines tâches possédaient un fort taux d'inconnu et il était donc difficile de lui attribuer un poids.

Lorsque nous ne pouvions nous voir pour assurer les mêlées quotidiennes, nous nous retrouvions sur une zone de chat *« #irc »* que nous avions créé pour la circonstance. Ainsi, nous faisions le point sur l'évolution du projet et débattre sur des choix de conception.

L'assiduité et le respect de la méthode *Scrum* est visible sur le *Github* de notre projet. En effet, le nombre de *Pull Request* (95), d'*issues* (145), de branches (91) et de *commits* (près de 1200) en est une preuve. Les *Pull Request* mettent également en avant la communication entre les membres de l'équipe.

Autre preuve de notre implication dans le respect de cette méthode, le *BurnUp Chart* ci-dessous :

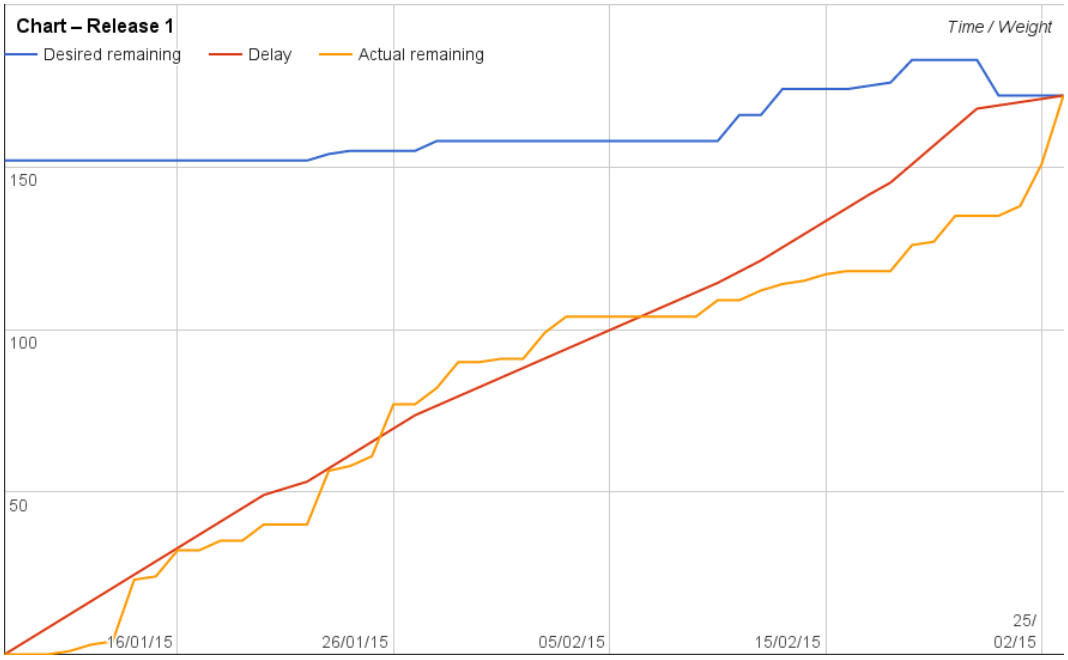

FIGURE 3.1 – Burn Up chart de la première release

La courbe rouge représente l'évolution théorique du projet durant les *Sprints* de la première *Release*. On constate que notre avancé (courbe orange) suit globalement la courbe théorique. Cependant, l'on observe une baisse de notre activité durant la semaine du 5 février qui correspondait à notre période de vacance. De plus, on note également que lors du *Sprint* 3 nous nous sommes trouvés en dessous de l'objectif. Cela s'explique par une augmentation du nombre de *Stories* (courbe bleu) qui correspondait à des modifications ergonomiques où à la résolution de bugs. Certaines d'entre-elles étaient sujet à discussion et ont donc été reportées au *Sprint* d'après. Néanmoins nous sommes parvenus à finir nos *Sprints* dans les temps.

Chacun des *Sprints* a été réalisé dans les temps et fut l'objet d'une démonstration auprès de M. MIGEON.

3.1.2 Les outils de qualité du code

La qualité du code peut être mesurer grâce à différents outils.

Dans notre cas nous avons utilisé plusieurs outils tel que *Travis CI* pour l'intégration continue, *Coveralls* pour la couverture de code ou encore *SonarQube* qui fournit des informations diverses sur le code (nombre de lignes de code, respect des convention de nommage, niveau de complexé, duplication du code).

Travis CI nous a permis de s'assurer que le logiciel compile sur une machine tierce, que les tests unitaires passent et de mettre à jour nos documents (manuel d'utilisateur ou *Doxygen*). Associé à *Github* l'on sait après chaque *commit* si le *Build* passe. C'est également lui qui fournit à *Coveralls* notre code à analyser pour vérifier la couverture de notre code. De plus, le fait que l'on ne peut pas intégrer notre code si la couverture de code régresse nous oblige à faire des tests unitaires sur les nouvelles fonctionnalités que l'on a implémenté. Ainsi, à la fin de notre projet, la couverture de code est de 90%. Cette continuité dans l'implémentation des tests unitaires nous a permis de

faire du *Refactoring* régulièrement sans crainte de provoquer des régressions du logiciels. Cela a été particulièrement efficace lors de l'ajout d'un second système de gestion de base de données où l'on devait s'assurer que les méthodes fonctionnent aussi bien sur un système que sur l'autre.

3.2 Le logiciel

Conclusion

Nous tenons tout d'abord à remercier toutes les parties prenantes au projet : notre encadrant M. Migeon qui nous apporter son expérience dans le développement du logiciel et une vision neutre sur le travail fourni ; M. Cherbonneau pour le suivi de l'ensemble des élève dans le cadre de cette UE.

Le projet *FactDev* nous a permis de nous placer en situation professionnel : nous avons du apprendre à nous adapter aux exigences de notre client (Antoine) ou de notre encadrant (M. Migeon) ainsi que mettre en place des moyens de communication et d'organisation pour parvenir à nos objectifs.

Il représente une expérience valorisante que l'on peut mettre en avant dans nos CV et lors de futurs entretiens. Sur le plan technique, nous pouvons citer le C++ (et le framework Qt), le LaTeX, les outils *Travis CI*, *Github* et *Coveralls*. Sur le plan organisationnel avec l'application strict de la méthode *Scrum* et le respect de la méthodologie mise en place pour assurer de la qualité du logiciel.

Bien que l'ensemble des fonctionnalités n'ont pu être implémentés et une difficulté à assurer nos *mêlées* quotidiennes le résultat qui en ressort est très positif. Le logiciel *FactDev* est fonctionne, distribué sous licence GPL et est déjà utilisé par Antoine.

A Backlog product

Release	Sprint	Statut	Vélocité Poids	Vélocité Nb Storie	Poids	Prio	#Issue	Type Story	En tant que	Je Souhaite	Afin de
		Done			2	Must	1	User Story	Utilisateur	Créer un nouveau client	Ajouter ses informations et prochianement le lier à un devi
		Done			3	Should	2	User Story	Utilisateur	Editer un client	Modifier les informations d'un client
		Done			2	Would	3	User Story	Utilisateur	Supprimer un client	Corriger une erreur, un client ajouté par mégarde
		Done			5	Must	11	User Story	Utilisateur	Afficher la liste des clients	Pouvoir prochainement afficher leurs factures
		Done			5	Must	12	User Story	Utilisateur	Renseigner mes données	
		Done			8	Could	14	User Story	Utilisateur	Afficher les informations d'un client particulier	Obtenir des informations détaillé sur le client
		Done			5	Could	20	User Story	Utilisateur	Chercher un client à partir de son nom	Filtrer l'affichage des clients
		Done			1	Would	22	User Story	Utilisateur	M'informer sur le développement du projet	
		Done			1	Must	16	Technical Story	-	Création de la fenêtre d'ajout/modification d'un client	
	Sprint 1	Done	40	10	8	Would	15	User Story	Utilisateur	Afficher un menu contextuel	*Editer, Ajouter, Supprimer...*
		Done			5	Must	13	User Story	Utilisateur	Créer un nouveau projet pour un client	pouvoir l'associé à un client
		Done			13	Must	7	User Story	Utilisateur	Créer un nouveau devis	estimer le coût d'un développement
		Done			5	Must	30	User Story	Utilisateur	Afficher les devis d'un projet	Avoir une estimation du coût du projet
		Done			8	Must	44	Technical Story	-	Récupérer un devis dans la bd avec son id	
		Done			1	Should	24	User Story	Utilisateur	Editer un projet existant	modifier ses informations
		Done			3	Must	25	User Story	Utilisateur	Lister les projets d'un client	Afficher la liste des projets d'un client particulier
		Done			5	Must	34	Technical Story	-	User Manual	
		Done			8	Must	37	User Story	Utilisateur	Créer une prestation	
		Done			8	Would	47	Technical Story		Sécurité des champs	
		Done			0.5	Could	50	Anomalie		Titres de fenêtres	
		Done			0.5	Could	49	Anomalie		Noms docks	
		Done			0.5	Could	48	Anomalie		Affichage Fax	
		Done			0.5	Must	52	Technical Story		créer un jeu d'essais de devis et projets	
		Done			3	Must	65	Technical Story		Insérer dans tripatte	
	Sprint 2	Done	64	15	3	Should	26	User Story	Utilisateur	Supprimer le projet d'un client	
		TODO			1	Must	8	User Story	Utilisateur	Créer une nouvelle facture	
		TODO			1	Must	39	User Story	Utilisateur	Afficher les factures d'un projet	
		TODO			1	Must	41	User Story	Utilisateur	Editer une facture existante	
		TODO			2	Should	45	User Story	Utilisateur	Signaler une facture comme payée	
		TODO			3	Could	27	User Story	Utilisateur	Editer un devis existant	le générer de nouveau
		TODO			3	Would	26	User Story	Utilisateur	Supprimer un devis	supprimer un devis d'un client
		TODO			3		40	Technical Story	-	User Manual	
		TODO			5		42	Technical Story	-	Plan Qualité	
		TODO			13		43	Technical Story	-	Préparer soutenance	
		TODO			5	Would	33	User Story	Utilisateur	Rechercher un devis ou une facture	Accéder facilement un devis ou une facture spécifique
		TODO			2	Could	35	User Story	Utilisateur	Editer une prestation	Pouvoir l'ajouter à un devis ou une facture
		TODO			8	Could		Technical Story		Sécurité des champs	
					5	Should	28	User Story	Utilisateur	Générer le fichier .tex d'un devis	pouvoir l'avoir ensuite en .pdf
Release 1	**Sprint 3**	TODO	59	13	20		38	Technical Story	-	Déploiement	
Totaux	3		163	38	163						
Moyennes			52	14	4.35135						

B Table des figures