INFO-F307 - Génie logiciel et gestion de projets
Ragnhild Van Der Straeten
Résumé du cours

Rodrigue Van Brande

13 avril 2015

Table des matières

1 Introduction au Génie Logiciel

1.1 Qu'est-ce que le génie logiciel ?

Le génie logiciel est le fait de développer des logiciels en suivant les meilleurs méthodes possibles.

1.2 Pourquoi le génie logiciel est très important ?

Par exemple la fusée Ariane 5 qui a explosé suite à un petit bug dans un programme. C'est encore aujourd'hui le bug informatique le plus chère de l'histoire.

1.3 Pourquoi un projet logiciel échoue ?

Les projets logiciels échouent pour une ou plusieurs raisons :
- Hors budget ;
- Retard, le client n'en aura peut être plus besoin ;
- Ne correspond pas aux exigences du client ;
- Qualité inférieure qu'initialement prévu ;
- Performances ne répondent pas aux attentes ;
- Trop difficile à utiliser.

1.4 Les quatre P

People Il y a plein de groupes de personnes qui interviennent sur le projet.

Product Le code, le produit compil, la doc, les tests, ...

Project Le planning, les processus de développement, les méthodologies utilisées.

Process Manière d'organiser les gens, la discipline, la structure.

1.5 Principes du génie logiciel

Les plus importants :
- La qualité est de première importance.
- Un logiciel de haute qualité est possible.
- Donner le produit aux clients le plus rapidement possible pour avoir leurs avis.
- Utiliser un processus de développement adapté.

1.6 Éthiques du génie logiciel

Il y a une part d'éthique. On ne copie/colle pas des bouts de codes sur des contrats différents, éthiques entre collègues, etc ...

2 Processus logiciel

2.1 Qu'est-ce qu'un processus logiciel ?

Un projet logiciel est composé d'activités (Planning, design, test, ...) et les activités sont organisés en différentes phases.

Un processus logiciel
- définit l'ordre et fréquence des phases ;
- spécifie les critères pour passer d'une phase à l'autre ;
- définie si un projet est livrable ou non.

2.2 Phases d'un processus logiciel

Lancement (Inception) Le produit logiciel y est conçu et défini.

Organisation (Planning) Initialise l'emploi du temps, les ressources et le coût déterminé.

Besoin d'analyse (Requirements Analysis) Structure et spécifie les besoins ("Quoi").

Conception (Design) Structure et spécifie une solution ("Comment").

Mise en oeuvre (Implementation) Construit une solution du logiciel.

Test (Testing) Valide la solution en fonction des besoins.

Entretien (Maintenance) Répare les défauts et adapte la solution aux nouveaux besoins.

2.3 Modèles de processus logiciel

2.3.1 Waterfall

Le processsus classique utilisé comme modèle est le développement logiciel étape-par-étape **waterfall**.
On analyse les besoins du client, on design l'application, on la code, on la lire et ensuite on la maintient.
On termine une étape avant de commencer la suivante.

C'est facile à mettre en place mais on ne recoit aucun retour du client. Donc si il y a une erreur dans la compréhension des besoins du clients ; on ne s'en rend compte que à la fin du processus et il est trop tard. On doit alors revenir à la première étape et tout refaire, le processsus devient donc long et peut être coûteux.

2.3.2 Itératif et incrémental

Le développement **itératif** est un waterfall avec un feedback du client entre chaque étape.
Le développement **incrémental** consiste à développer le produit fonctionnalité par fonctionnalité en livrant à chaque fois une version préliminaire du projet (Processus composé de mini-waterfall).
La **livraison incrémentale** est la même chose mais appliqué à la livraison.

2.3.3 Prototyping

Prototyping est le processus de création d'un modèle incomplet du futur programme logiciel qui peut être utilisé pour des tests, exploration ou pour valider une hypothèse.

2.3.4 Unified Process

2.3.5 Open Source

2.4 Documentation

3 Méthodes agiles

3.1 Développements agiles

Les deux gros désavantages du waterfall sont :
- Besoins doivent être parfaitement connu dès le départ.
- Le client ne voit que le produit fini.

La méthode agile se concentre sur **les gens**. Ils préfèrent un **logiciel qui marche** plutôt que de la **documentation**. La **collaboration avec le client** est mise en avant. Il faut savoir **répondre à des changements de besoins**.

3.2 Principes de l'agile

Les grands principes agiles sont les suivants :
- Donner au clients de morceau de logiciel le plus souvent possible.
- Accepter les changements de besoins. (Il y a toujours un code freeze quelques jours avant une livraison, où on n'ajoute plus de fonctionnalités et on corrige les bugs)
- Les gens du marketing et les développeurs doivent bosser ensemble.

- Engager des gens compétents et motivés et leur faire confiance.
- La meilleure manière de communiquer est en face-à-face.
- Un logiciel fonctionnel est la principale mesure de progression.
- Le rythme de travail doit pouvoir être maintenu sans se crever.
- Une attention constante à l'excellence technique et à un bon design.
- La simplicité est d'or.
- Les meilleures architectures et designs émergent naturellement d'équipes auto-organisées.
- L'équipe réfléchit régulièrement à comment se rendre plus efficace et ajuste ses procédures de travail de manière appropriée.

3.3 Le cycle de l'agile

Cycle : dure entre 1 et 6 semaines.
- On obtient les besoins du client pour l'itération.
- On refactorise le code pour que les besoins puisse être ajoutés.
- Nouvelles fonctionnalités et des test.
- Code nettoyé le plus possible, simplifié.

A la fin on a un logiciel qui marche et testé.

3.4 Intégration entre les processus agiles et non-agiles

Il faut faire un compromis sur l'intensité de l'application du processus. Tous les idéaux ne sont pas adaptés au projet en cours.

4 Extreme programming (XP)

4.1 Description de l'XP

L'extreme programming vise à supporter des changements réguliers des besoins.
⇒ incertitude des besoins.
Le principe de l'extreme programming est de fixer le temps, le prix et la qualité, et le paramètre variable sera l'étendue des fonctionnalités. Si le temps manque, au lieu d'éliminer les tests ou faire du code de mauvaise qualité, on supprime des fonctionnalités.

4.2 Valeurs basiques de l'XP

Communication Il faut oser dire qu'il y a des problèmes.

Simplicité Un code simple marche mieux.

Feedback Illustre échanges entre le client et les développeurs.

Courage Ne pas avoir peur de faire des choix.

Respect Entre les membres de l'équipe et le client.

4.3 Principes

Il y a les **pratiques primaires** qui font réellement partie de l'XP et sont nécessaires et les **pratiques secondaires** sont facultatives mais intensifient l'expérience.

4.3.1 Programmation par paire

Un des développeurs code, et l'autre dicte et explique ce qu'il doit coder. Ils font des rotations.

4.3.2 Histoire

Les **histoires** sont des besoins. Ce sont les seuls documents de besoins de l'XP. Elles sont très petites (+- une phrase, un titre et une estimation de durée). Elles doivent être simples et maintenues à jour. Elles ne comprennent rien de technique.

4.3.3 L'intégration continue

Le projet peut être découpé en plusieurs tâches, chacune faite par un développeur (ou un groupe de dèv.). La technique du **divide and conquer**. Il faut constamment intégrer ces morceaux.

4.3.4 Test-first programming

On code d'abord par coder le test, puis ensuite la fonctionnalité. Si le test est difficile à écrire, c'est que la fonctionnalité est mal spécifiée.

4.3.5 Conclusion

Le processus est incrémental (Le logiciel "grandit") et itératif (les dèv. apprenent pendant le développement.).

5 Gestion de projets

5.1 Introduction

5.2 Rôles d'un gestionnaire de projet

Les trois grandes activités du gestionnaire sont :
- **L'étude de faisabilité** : Il vérifie qu'un projet est possible et profitable.
- **Le planning** : Difficile car il doit commencer avant d'avoir le moindre requirement.
- **L'exécution** : Il tue.

5.2.1 Définir les objectifs

Les objectifs de manière textuelle, sous forme de post-conditions. Les objectifs doivent être SMARTS :
- **Spécifiques** : Concrets et bien définis.
- **Mesurables** : On doit pouvoir mesurer si un objectif a été atteint ou pas.
- **Réalisables**.
- **Pertinents**.
- Prendre en compte le **temps**.

Les **buts** ou **sous-objectif** sont les détails des objectifs. On rentre dans le technique. Contrairement aux objectifs, ils peuvent être associés à une personne.

5.3 La gestion de projet

5.3.1 Appréciation de l'état du projet

Il faut prendre des décisions, qui demandent des compromis. Généralement on fixe la date et le cout et la qualité en prend un cout. Le développement agile fixe le cout, la qualité et le temps mais joue sur les fonctionnalités.

5.3.2 Business case

L'étude de fiabilité peut servir de **business case**. Un business case est un document qui suit une structure bien définie. On commence par une introduction, une description du problème, des opportunités, puis la description du projet, le marché et l'infrastructure dans lesquels il s'insert, la demande pour le projet, ses bénéfices. On fini le document avec le plan d'implémentation, les couts, les aspects financiers, les risques et le plan de gestion.

5.3.3 Méthode de gestion de projet

Le gestionnaire de projet doit savoir quoi faire et quand. Une manière de faire est de suivre la méthode PRINCE 2, une méthode standard. Elle est adapté aussi bien aux petits qu'aux grands projets. On va étudier une méthodologie très proche et plus ouverte.

La première étape est bien sûr le choix d'un projet sur lequel travailler. Les deux étapes suivantes se font en parallèles : Identifier les objectifs et identifier l'infrastructure nécessaire. VIennent ensuite l'analyse des caractéristiques, l'identification des produits et activités, l'estimation de l'effort et des risques, l'allocation des ressources, puis la publication d'un plan d'action.

5.4 Estimation de l'effort

Un projet est un succès s'il est livré à temps, en respectant le budget, les fonctionnalités demandées et la qualité requise. Les efforts demandés doivent être estimés. C'est une tâche difficile.

Les stratégies d'estimation sont les suivantes :
- **Jugement par un expert** : Quelqu'un vient et se prononce. Peu cher mais peu fiable.
- **Estimation par analogie** : On compare différents projets et leurs estimations.
- **Loi de Parkinson** : Le projet s'étend pour remplir tout le temps alloué.
- **Pricing to win** : On fait son maximum dans le budget donné.

Les deux derniers points ne prédisent pas le coût ni le temps. Deux stratégies plus complexes, sont plus efficaces :
- **Décomposition** : On découpe le projet en morceaux dont on estime le coût.
- **Modélisation algorithmique du coût** : On se sert de données historiques, de faits, de statistiques et on calcule de manière objective comment ces données s'appliquent au projet à effectuer.

5.4.1 La décomposition

L'approche **top-down** commence au niveau du système complet, ensuite on découpe le système et on estime l'effort nécessaire au développement des différents composants.

L'approche **bottom-up** commence par tout découper, et on estime le coût de chaque composant. On les sommes ensuite pour obtenir le coût total du projet.

5.4.2 Le modèle COCOMO

Ce modèle est le plus utilisé. Il distingue trois types de projets :
- Les simples ;
- Les semi-détaché ;
- les inclus.

Chaque classe de projet possède des valeurs pour 4 paramètres : a,b,c et d.
Ces paramètres apparaissent dans :

$$\left| \begin{array}{lcl} Effort & = & a \times KLOC^b \\ Duree & = & c \times Effort^d \end{array} \right. \quad \text{(pour un simple } a = 2.4,\ b = 1.05,\ c = 2.5,\ d = 0.38)$$

Dans COCOMO 2, le modèle évolue un peu. L'effort est :

$$A(taille)^{scalefactor} \times em1 \times em2 \times ...$$

Dans COCOMO 2, la formule change avec le projet, certaines variables sont utilisées et pas d'autres. Les emX sont des nombres qui expriment la difficulté du projet.

5.5 Planning d'activités

Après avoir fait tout ça, il faut choisir des dates de début et de fin pour chaque activité. Les **réseaux d'activité** permettent de voir les liens entre les activités, et quelles activités peuvent être effectuées en parallèle. Pour identifier les activités, on prend le produit, on le découpe en morceaux, et ces morceaux sont découpés en activités. ⇒ Approche **produit**. Sinon on peut d'abord identifier les activités, et le produit sera leur réunion. ⇒ Approche **travail**.

5.6 Construction de réseau d'activité

Deux types de réseaux d'activité :

1. Met les activités sur les arcs, et qui relient des états qui n'ont pas de durée.

2. Activités sur les noeuds (qui ont la des durées), et les arcs ne sont que des relations de précédence.

Un réseau ne peut contenir de boucle. Une fois qu'on a les activités avec leurs durées, il faut calculer quand elles démarrent. On se base sur le **chemin critique**. Ce chemin doit être le plus court possible, celles qui ne peuvent accepter un retard sans retarder tout le projet.

Le **flottement** est la différence entre le départ le plus tard et le plus tôt d'une activité. Le chemin critique est simplement le chemin qui passe par toutes les activités qui ont un flottement de zéro.

Les **chemins sous-critiques** sont les chemins qui passent par des activités avec un flottement très faible.

5.7 Organisation de l'équipe

Trois types d'entreprises :

1. **Orientées projets** : Tout tourne autour du projet.
2. **Orientées fonctions** : département des finances, IT, marketing, ...
3. **Organisation matricielle** : mélange des deux.

5.7.1 Équipes de développement

Elle doit être la plus petite possible (entre 3 et 7 personnes). S'il y en a plus, trop de problème de communication. L'**egoless programming** se base sur l'idée que le groupe est plus important que l'individu. La **décomposition hiérarchique** est une organisation où chaque personne a un chef, et les chefs ont leurs chefs. Les équipres à **chiefs programmer** sont des équipes composées de programmeurs très réputés qui prennent toutes les responsabilités.

6 Condensé des questions

6.1 Software Engineering

6.1.1 Pourquoi le Software Engineering est important ?

Éviter des catastrophes ou les simples petites erreurs. Exemple : La fusée Ariane 5 qui explose à la suite d'un petit bug dans un programme.

6.1.2 En quoi ça consiste ?

Le génie logiciel est le fait de développer des logiciels en suivant les meilleures méthodes possibles.

6.1.3 Quel en est l'activité principale ?

6.1.4 Quels sont les principes du Software Engineering ?

Les plus importants :
- La qualité est de première importance.
- Un logiciel de haute qualité est possible.
- Donner le produit aux clients le plus rapidement possible pour avoir leurs avis.
- Utiliser un processus de développement adapté.

6.2 Qualité d'un Software Engineering

6.2.1 Quels sont les moyen pour une "bonne qualité" d'un Software Engineering ?

6.2.2 Quels sont les "défauts" dans les applications ?

6.2.3 Quel est la différence entre vérification et validation dans le développement du Software ?

6.2.4 Comment mesurez-vous la qualité d'un Software ?

6.3 Software Processes

6.3.1 Quel est la principale activité du Software Processes ?

6.3.2 Quel est le type du principal Software Processes ?

6.3.3 Comment les méthodes agiles sont apparu ?

6.3.4 Quel est le principe de l'agilité ?

6.3.5 Comment les processus agiles sont effectuées ?

6.3.6 Pouvons-nous combiner un processus agile avec un non-agile ?

6.4 XP

6.4.1 Pourquoi appliquer XP ?

6.4.2 Quels sont les valeurs de XP ?

6.4.3 Quel est la principale pratique de XP ?

6.4.4 Comment concevoir avec XP ?

6.4.5 Comment un projet de XP est effectuée ?

6.5 Qualité des Agile Processes

6.5.1 Quels sont les principes de gestion de la qualité ?

6.5.2 Comment bien concevoir pour une bonne qualité ?

6.5.3 Comment les méthodes agiles sont conçus pour une bonne qualité ?

6.6 Program Design

6.6.1 Quels sont les buts du Software Design ?

6.6.2 Comment améliorer la qualité du Software Design ?

6.6.3 Recevoir un problème : quel Design Pattern pouvez-vous appliquer ?

6.6.4 Quel est la conséquence principale en appliquant un Pattern spécifique ?

6.7 Tests

6.7.1 Quand et pourquoi faut-il (re)tester ?

6.7.2 Comment faites-vous les tests unitaires ?

6.7.3 Que sont les modules testing and integration ?

6.7.4 Que sont les systeme de test ?

6.7.5 Que sont les tests d'acceptation ?

6.7.6 Quels sont les éléments traditionnel d'une phase de test ?

6.8 Gestion de projet

6.8.1 Comment le Software Development Projects est oganisé ?

6.8.2 Quel est la taille d'une équipe approprié ?

6.8.3 Comment les équipes sont organisés ?

6.8.4 Comment les équipes sont géographiquement distribué ?

6.8.5 Quels sont les outils et techniques disponibles pour aider dans un projet ?