



UNIVERSITÉ DE GENÈVE

Extreme Programming - XP

Une méthode de développement
par *Kent Beck*

Le cycle de vie d'un système ...

Analyse de besoins

Spécification

Design (conception)

Codage

« Déboggage »

Mise en service

Maintenance

*Le modèle de la
« chute d'eau » -
La vision traditionnelle*

Temps t



..... mais la maintenance coûte !

Coût de modification

*La vision traditionnelle:
Un système devient plus
difficile à modifier avec le
temps*

Age du système

Confirmé par la bogue de l'an 2000 ?

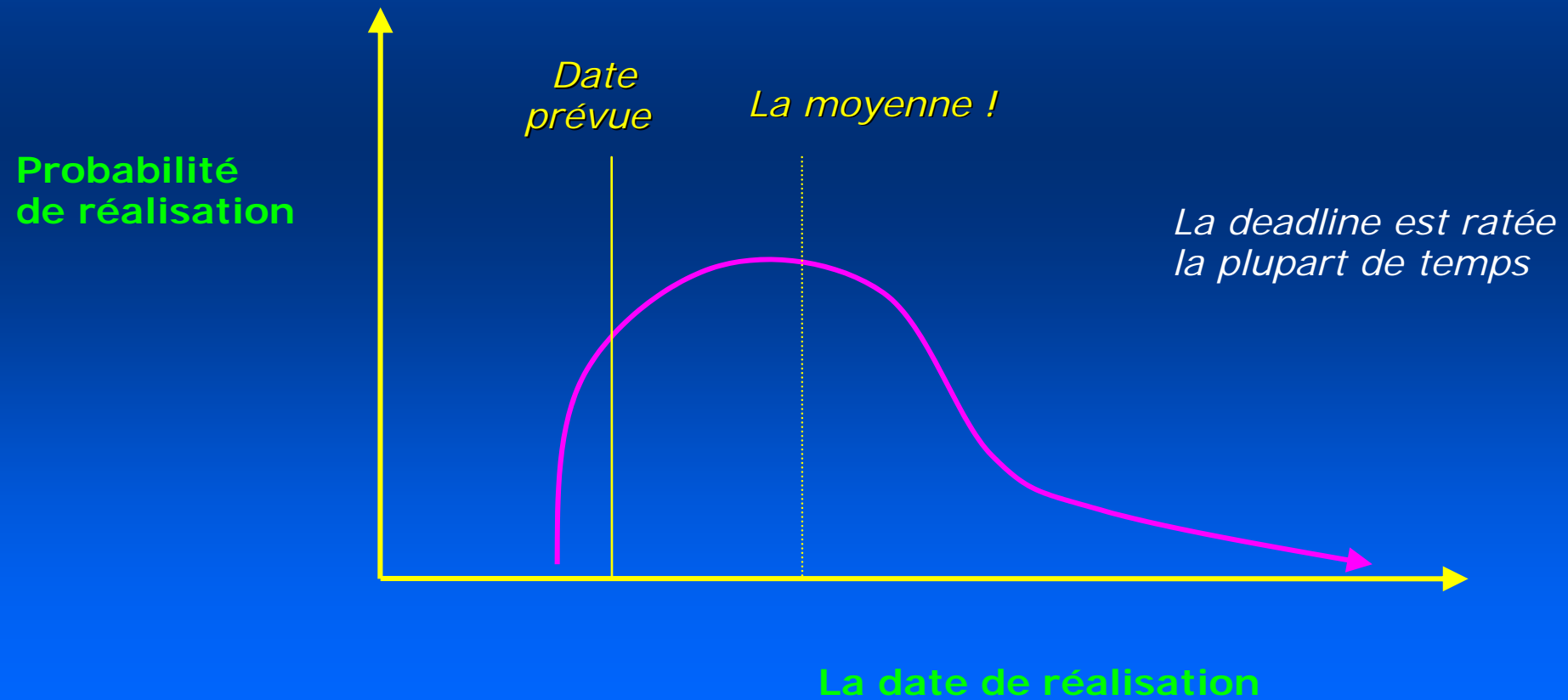
La crise de logiciel

- ◆ Une courbe exponentielle du coût de la maintenance
 - L'obligation de dépenser des moyens après la livraison
- ◆ Un manque de qualité
 - La qualité se définit par la validité, la sécurité, la tolérance aux erreurs, l'extensibilité, ...
- ◆ Une pénurie de *bons* programmeurs
- ◆ Les besoins des clients changent jusqu'à 25% pendant le développement
 - et sont rarement clairs au début

La crise de logiciel

- ◆ La plupart des projets sont livrés en retard
 - S'ils ne sont pas arrêtés avant
- ◆ Les « *moyens accordés au projet sont trop peu nombreux* »
 - Les programmeurs sont-ils de mauvais gestionnaires?
- ◆ Le mouvement *open source* indique un manque de confiance dans les grosses entreprises de logiciel (?)
- ◆ L'enseignement de la programmation est difficile
 - On n'apprend que par l'expérience
 - L'apprentissage nécessite le développement d'une discipline

La crise de logiciel



Extreme Programming - XP

- ◆ XP est un processus de développement qui a pour but
 - De produire du logiciel de qualité
 - Le développement ne coûte pas cher
 - Les systèmes ne sont pas livrés en retard
- ◆ XP consiste en un ensemble de mesures qu'une équipe de développement doit suivre
 - P.ex.: les tests, la programmation par pair,

Extreme Programming - XP

- ◆ XP est une *discipline*
 - La rigueur est une qualité primordiale du programmeur et de l'équipe de programmation
- ◆ La discipline est *extrême* dans le sens qu'il faut vraiment suivre toutes les mesures proposées
- ◆ L'approche est de *s'attendre à des changements*
 - dans les besoins de clients
 - dans la formation de l'équipe
 - dans les outils de développement

Vers XP

- ◆ XP nécessite une connaissance des risques
 - Les retards ou l'annulation du projet
 - Mauvaise compréhension de l'aspect Économique
 - C-à-d: les besoins des clients, l'orientation du marché
 - Changement dans l'orientation de l'Économique
 - Changement de personnel dans l'équipe
 - L'équipe perd l'intérêt pour le système
 - « *False Feature Rich* »
 - Trop d'effort dépensé dans le développement des propriétés qui ne sont pas les plus importantes

La philosophie de l'XP

- ◆ Pour rapidement trouver et corriger les bogues
 - On code par pair, tout le monde est co-propiétaire du code, et on écrit beaucoup de tests
- ◆ Pour gérer les risques associées à l'Économique
 - Le design est fait en itérations successives
 - Les modifications sont testées et puis intégrées rapidement
 - On garde les designs aussi simples que possible
 - On est toujours en train de réviser ses propres designs
- ◆ Pour gérer les problèmes de personnel
 - Tout le monde est co-propiétaire du code

XP marche, mais pourquoi ?

- ◆ Une meilleure prise de connaissance des risques vis-à-vis de l'Économie
 - On développe dans l'itération courante ce qui est le plus important (pour le client) actuellement
 - Pas de paris sur l'évolution de l'Économie
- ◆ L'effondrement de la courbe de coût de modification
 - La courbe devient linéaire !!! (le coût de modification d'un système ne change pas avec le temps)
 - À causes des tests
 - À cause des propriétés de la programmation objet

Les 4 variables du développement

- ✉ Le *coût*, le *temps* (la durée), la *qualité* et la *portée* (de la spécification)
- 1. Les variables sont fixées par l'économie, et ont une influence l'une sur l'autre
- 2. XP: La *qualité* ne doit pas être traitée comme étant une variable
- 3. XP: On varie la *portée* en fonction du *coût* et du *temps*

Les 4 valeurs du développement

- ◆ La *communication*

- Entre les programmeurs, dans une équipe entre le Développement et l'Économique,
- entre l'équipe et le clientèle

- ◆ La *simplicité*

- Pour la révision de code et le développement rapide
- Pour minimiser le risque de bogues
- Pour pouvoir comprendre son code par la suite

Les 4 valeurs du développement

- ◆ Le *feedback*
 - Sur le système (via les tests), mais aussi sur la performance de l'équipe (p.ex. la *vélocité du projet*)
- ◆ Le *courage*
 - Il faut pouvoir jeter un développement s'il ne marche pas avec les tests ou si l'Économique le rend inutile
 - De proposer des idées bizarres

Les mesures de l'XP

Les principes pour concrétiser les valeurs :

- ◆ Le *feedback* rapide
 - Les programmeurs ont une tendance à oublier leur code avec le temps
 - Comprendre tout de suite les implications d'une modification
 - XP : les tests, les mesures, l'intégration rapide
- ◆ Chercher la simplicité
 - On code un composant aussi simplement que possible

Les mesures de l'XP

- ◆ La modification par incrémentation
 - Un changement à la fois
 - dans l'équipe, la spécification, les outils, etc.
- ◆ Attendre des changements
 - On découvre toujours des moyens de mieux faire les choses
 - XP : la révision du code
- ◆ Qualité
 - Il faut savoir varier la variable de portée
- ◆ Apprendre (et noter)

Les mesures de l'XP

- ◆ Un investissement initial minimal
 - Il y a trop d'incertitude au début d'un projet
 - On ne dépense pas de ressources avant qu'on en ait vraiment besoin
- ◆ Jouer offensivement
 - Il faut créer un cadre et une mentalité où l'équipe peut progresser
 - P.ex: il est plus important d'écrire des tests que des rapports
- ◆ Des expériences concrètes
 - Ce sont les tests qui indiquent si le système marche ou pas

Les mesures de l'XP

- ◆ La communication honnête
 - Si le code a besoin d'être révisé, alors il faut le dire
- ◆ Accommoder les intérêts des gens
 - Un co-équipier n'est pas un valet; il faut qu'il ait un moyen d'avancer (apprendre, contribuer, ..)
- ◆ La responsabilité est prise, et jamais donnée
- ◆ Minimum de bagages
 - On achète les outils lorsqu'on en a vraiment besoin
- ◆ Il faut toujours vérifier la vélocité du projet

La mise en œuvre de l'XP

- ◆ La planification
 - Une équipe comprend une partie Économique et une partie Développement
 - L'Économique spécifie les besoins
 - Le Développement fait les estimations de coût et de durée
 - Puis l'Économique décide sur la portée
- ◆ Des versions incrémentales
 - Il vaut mieux avoir une nouvelle version du système tous les 2 ou 3 mois que tous les 6 mois (mais qui comprend plus de modifications)

La mise en œuvre de l'XP

- ◆ Une métaphore
 - Décrit le rôle (ou la vision) du système
 - On s'en sert pour guider le développement et pour évaluer son progrès
 - Ceci remplace la « spécification » du système
- ◆ Un design simple
 - Tous les tests marchent
 - Pas de duplication de code
 - Le nombre minimal de méthodes et des classes

La mise en œuvre de l'XP

- ◆ Les *tests*
 - Les tests sont écrits par les clients et par les programmeurs
 - Les tests servent également à préciser le *design*
 - XP parie que le temps « perdu » à écrire les tests est largement compensé par les gains en temps de développement
 - Il faut favoriser les tests automatisés
 - p.ex.: *JUnit*

- ◆ *Le testing reste l'élément du génie logiciel le plus important*

La mise en œuvre de l'XP

- ◆ La révision de code et du design
 - On change le code d'un composant dès qu'on voit une manière plus simple de l'implanter
 - Plus de travail dans le court-terme, mais
- ◆ La programmation par pair
 - On change les pairs fréquemment au sein de l'équipe
 - On range son bureau pour faciliter l'XP
- ◆ La propriété commune
 - Tout le monde est co-propiétaire du code du système
 - Donc, il est au courant du fonctionnement de chaque partie

La mise en œuvre de l'XP

- ◆ Les semaines de 40 heures
 - Il ne faut pas faire des heures supplémentaires !!
 - Une équipe débordée est une équipe dont le processus ne fonctionne pas !
- ◆ Intégration rapide
 - Un nouveau composant est vite intégré et testé
 - Si les tests ne marchent pas, alors on jette le composant
- ◆ Un représentant du client doit être toujours disponible
- ◆ Il faut adopter des standards de programmation
 - pour la propriété commune et l'intégration continue

Les rôles au sein d'une équipe

Une personne peut avoir plusieurs rôles :

- ◆ *Programmeur*

- Il code
- Et il propose des estimations pour le processus de planification

- ◆ *Client*

- Il achète et il écrit des tests et il prend des décisions sur la portée

- ◆ « *Testeur* » : il écrit et fait tourner les tests

Les rôles au sein d'une équipe

- ◆ « Traceur »
 - Il archive les données de performance de l'équipe
 - P.ex.: la vélocité du projet, le nombre de tests
 - Si la vélocité est trop élevée, alors on ne fait pas suffisamment de révisions de code

- ◆ *Coach*
 - un programmeur qui guide l'application des principes de l'XP
 - Il n'y a pas de chefs en XP
 - qui sont généralement de personnages trop éloignés du codage

Bibliographie

- ◆ *Extreme Programming Explained*
 - Kent Beck, publié chez Addison-Wesley
- ◆ <http://www.xp.com>
- ◆ <http://www.junit.org>
 - Pour les tests automatisés

Quelques choix personnels

◆ 1. La simplicité d'abord

- Un programme sans bogue vaut 1000 fois plus qu'un programme rapide, ou avec une jolie GUI
- Un langage de programmation qui permet l'évolution du système (la programmation modulaire)
 - P.ex: Java par rapport à C
- Un algorithmique simple d'abord
 - Quand le système marche, on peut optimiser
- On peut mettre autant de commentaires que de code dans les prototypes

Quelques choix personnels

◆ 2. Prototype d'abord

➤ P.ex: une version du système Linda faite à l'université

→ 1^{ère} version : un espace avec les opérations **in** et **out** pour les programmes Java (2500 lignes de code)

- ◆ Le système gère les données dans l'espace et comprend une technique de chiffrement pour protéger ces données

- La protection est le but principal du système

→ Mais il faut encore

- ◆ Optimisations des algorithmes de recherche des données dans l'espace pour la performance, persistance des données (sauvegarde des données sur disque), la distribution (accès aux espaces à distance)
- ◆ Cela nécessitera 20 000 lignes de code en plus !
 - Mais l'essentiel (la protection) est déjà fait

Quelques choix personnels

- ◆ 3. Pour programmer il faut toujours avoir un **stylo à la main**
 - Le terminal est mauvais pour les yeux et pour le dos
- ◆ 4. Un **bogue doit être documenté** dans un fichier
 - Un bogue a tendance à revenir
 - En lisant le code, on trouve toujours des bogues !
 - Tout le processus de développement doit être documenté
 - On doit souvent faire face aux problèmes qu'on avait déjà rencontrés
- ◆ 5. La programmation est un exercice de **sous-estimation de temps**
 - À cause des bogues de programmation imprévus ou difficiles à trouver

Quelques choix personnels

- ◆ 6. La notion d'équipe est primordiale
 - Les meilleurs joueurs ne font pas forcément la meilleure équipe
 - Il faut cultiver une bonne ambiance
 - Bière, ballon de foot, DVD
 - Tout le monde doit être au courant de ce que font les autres