Analyse, conception et validation de logiciels

1. Notions de génie logiciel

Mathias Péron

ENSIMAG 2A - année 2005-2006

Introduction

Projets logiciels célèbres pour

- leurs dépassements de budget
- leurs délais
- leurs cahiers des charges non respectés

Développement logiciel

- activité complexe
- loin de se réduire à la programmation

Nécessité de méthodes de développement

- permettent d'assister une ou plusieurs étapes du développement.

Approches objet

- issues de la programmation objet, mais ne se réduisent pas à la programmation objet
- basées sur une modélisation du domaine d'application.

Introduction

UML *Unified Modeling Language*
Langage de modélisation qui permet d'élaborer des modèles objets, indépendamment du langage de programmation, à l'aide de différents diagrammes.

Objectifs du cours

- Introduction au génie logiciel
- Présentation des principaux diagrammes UML
- Modélisation, analyse et conception objet de logiciels

Plan du cours

Constats 1/2

Selon une étude du gouvernement américain, réalisée en 1979 sur le coût des logiciels :

logiciels	*coût*
payés, jamais livrés	3.2 M$
livrés, jamais utilisés	2.0 M$
abandonnés ou recommencés	1.3 M$
utilisés après modification	0.2 M$
utilisés en l'état	0.1 M$

Uniquement 5% du coût est « acceptable ». On a parlé de « crise du logiciel ».

Encore Dijkstra (EWD340) !

Constats 2/2

Selon une étude réalisée en 1995, aux États-Unis (rapport Chaos du *Standish Group*) :

logiciels	*coût*
abandonnés ou jamais utilisés	31%
coûts et délais largement dépassés facteur 2 ou 3	53%
réalisés suivant les plans	16%

Pour les grandes compagnies le taux de succès tombe à 9%
La crise du logiciel n'est pas terminée . . .

Nature des problèmes

- Complexité (taille, algorithmes)
- Difficulté à établir et stabiliser les besoins des utilisateurs
- Difficultés de communication
- Malléabilité du logiciel
- Documentation insuffisante
- Difficulté à visualiser un logiciel
- Élargissement des domaines d'application
- Chute du prix des machines
- Croissance de la puissance des machines

Génie logiciel

Définition

« Le génie logiciel est l'ensemble des activités de conception et de mise en œuvre des produits et des procédures qui tendent à rationaliser la production du logiciel et son suivi. »

ou

« Le génie logiciel est l'art de produire de bons logiciels, au meilleur rapport qualité prix. »

Le génie logiciel inclut l'étude

- de notations
- de méthodes
- de langages de programmation

Critères de qualité des logiciels 1/2

Point de vue de l'utilisateur (critères externes)

- fiable (donne le résultat attendu)
- robuste (ne « plante » pas)
- efficace (donne le résultat en un temps acceptable)
- convivial (facile à utiliser)
- documenté (manuel utilisateur, cours)

Critères de qualité des logiciels 2/2

Point de vue du développeur (critères internes)

- documenté (documentation de conception)
- lisible (facile à lire et à comprendre),
- facile à maintenir (on peut le corriger ou le modifier facilement)
- portable (on peut facilement le transférer d'une machine à une autre)
- extensible (on peut facilement ajouter de nouvelles fonctionnalités)
- réutilisable (on peut facilement l'adapter pour d'autres applications)

Étapes du cycle de vie du logiciel 1/2

Analyse et définition des besoins

- cahier des charges
- étude de faisabilité

Analyse et conception

- spécification (description des fonctionnalités)
- conception architecturale (structure)
- conception détaillée (composants, algorithmes)

Mise en œuvre (programmation)

Étapes du cycle de vie du logiciel 2/2

Validation

- consiste à s'assurer que le logiciel est correct
- analyse statique (typage, conventions ?)
- preuve (peu utilisée)
- revue de code (efficace)
- tests (indispensables)

Évolution et maintenance

- maintenance corrective (correction d'erreurs)
- maintenance adaptative (portage du logiciel)
- maintenance évolutive (évolution du logiciel)

Répartition de l'activité

(hors analyse et définition des besoins)
Place de la maintenance

étape	*pourcentage*
développement initial	40%
maintenance	60%

Place de la programmation

étape	*pourcentage*
analyse et conception	40%
mise en œuvre	20%
validation	40%

Conséquence : la programmation ne représente que 8% de l'effort total.

Modélisation du processus de développement

But

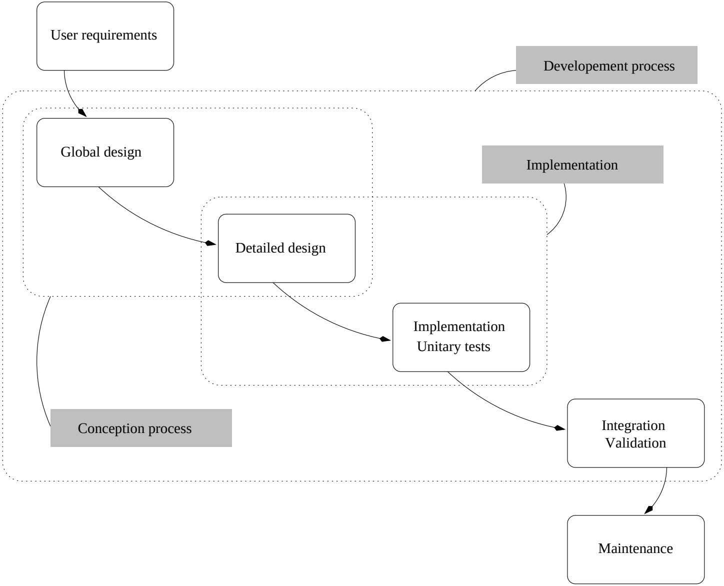
Contrôler le processus de développement afin que le logiciel :

- soit livré dans les délais
- respecte le budget
- soit de qualité

Il s'agit de « plans de travails », qui doivent permettre la planification du développement.
On parle de « modèles du cycle de vie du logiciel »

Parlons aussi de Royce !

Modèle en cascade 1/2



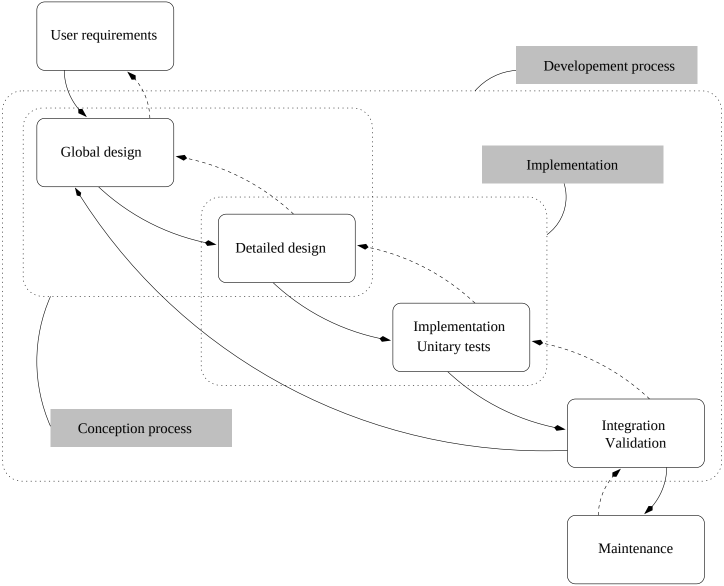
Modèle en cascade 2/2

Les interactions ont lieu uniquement entre étapes successives.
Exemple : un test ne doit pas remettre en cause la conception architecturale.

Modèle incontrôlable Si à chaque étape on s'autorise un retour sur n'importe quelle étape précédente, le modèle pert son sens.

Modèle en V

Modèle incrémental 1/2



Modèle incrémental 2/2

Le modèle incrémental est un modèle itératif, qui procède par incréments successifs (ou versions).
Un incrément est un petit nombre de fonctionnalités analysées, programmées et testées.

Intérêts du modèle incrémental :

- il permet de révéler certains problèmes de façon précoce
- on a rapidement un produit que l'on peut montrer au client
- le client peut effectuer des retours sur la version livrée
- ce modèle fournit plus de points de mesure pour apprécier l'avancement du projet.

Modèle du cycle de vie en spirale 1/3

Modèle itératif, dans lequel la planification de la version se fait selon une analyse de risques.

Idée S'attaquer aux risques les plus importants assez tôt, afin que ceux-ci diminuent rapidement.

Risques liés au développement de logiciel

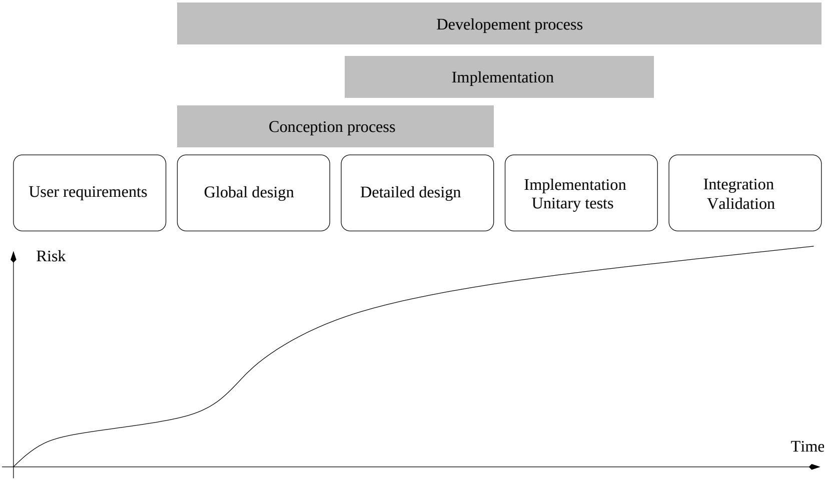
- risques commerciaux (placement du produit sur le marché, concurrence)
- risques financiers (capacités financières suffisantes pour réaliser le produit)
- risques techniques (la technologie inadaptée ou pas éprouvée)
- risques humains (défaillance de personnel, manque de compétence)
- risques de développement (évolution incontrôlée des besoins)

Modèle du cycle de vie en spirale 2/3

Exemple de risque : Manque de compétences de l'équipe dans l'écriture d'interfaces graphiques.
Solution préconisée : Prévoir rapidement un prototype d'interface graphique dès les premières itérations, afin d'acquérir rapidement une compétence sur le sujet.

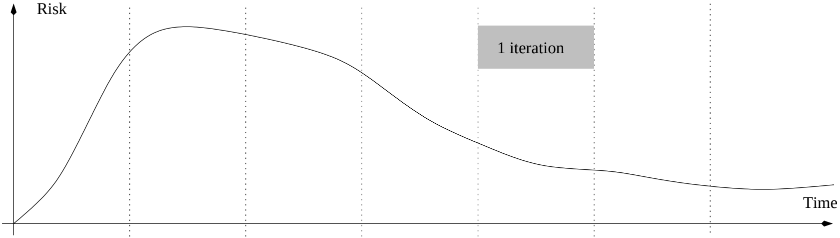
Les risques dans le modèle en cascade
Les activités qui comportent le plus de risques ont lieu à la fin, lors de la mise en œuvre et de la validation.

Modèle du cycle de vie en spirale 3/3



Risques et cycle de vie en spirale

Les activités les plus risquées sont réalisées au début, ce qui réduit le risque lors des dernières itérations.



Méthodes de développement

Une méthode définit une *démarche* en vue de produire des *résultats*. Exemple : recettes de cuisine

Une méthode permet d'assister une ou plusieurs étapes du cycle de vie du logiciel.

On distingue les méthodes fonctionnelles, basées sur les fonctionnalités du logiciel, et les méthodes objets, basée sur différents modèles (statiques, dynamiques et fonctionnels).

Méthodes fonctionnelles 1/2

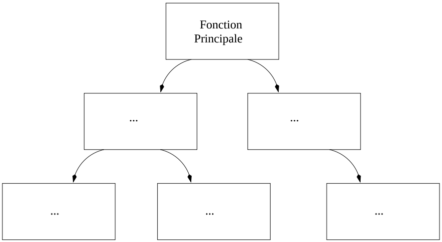
Origine programmation structurée.
Cette approche consiste à décomposer une fonctionnalité (ou fonction) du logiciel en plusieurs sous fonctions plus simples.

- Conception « top-down », basée sur le principe « diviser pour mieux régner ».
- L'architecture du système est le reflet de cette décomposition fonctionnelle.

Programmation

- soit à partir des fonctions de haut niveau (développement « top-down »),
- soit à partir des fonctions de bas niveau (développement « bottom-up »).

Méthodes fonctionnelles 2/2



Inconvénients de cette méthode L'architecture étant basée sur la décomposition fonctionnelle, une évolution fonctionnelle peut remettre en cause l'architecture.

- Cette méthode supporte donc mal l'évolution des besoins.
- Cette méthode ne favorise pas la réutilisation de composants, car les composants de bas niveau sont ad-hoc.

Méthodes objets

Origine programmation à objets
L'approche objet est basée sur une modélisation du domaine d'application à l'aide d'objets.

Modèle Un modèle est une abstraction de la réalité, c'est-à-dire une vue subjective, mais pertinente de la réalité.
La modélisation :

- réduit la complexité
- aide à visualiser un système
- permet de spécifier la structure ou le comportement d'un système
- fournit un guide pour la construction du système
- documente les décisions prises lors de la construction du système.

Aspects d'une modélisation

On distingue trois aspects :

- un aspect statique, où on identifie les objets, leurs propriétés et leurs relations ;
- un aspect dynamique, où on décrit le comportement des objets, en particuliers leurs états possibles et les événements qui déclenchent les changements d'état ;
- un aspect fonctionnel, qui, à haut niveau, décrit les fonctionnalités du logiciel ou, à plus bas niveau, décrit les fonctions réalisées par les objets par l'intermédiaire des méthodes.

Intérêts de l'approche objet

- approche « naturelle » car basée sur le monde réel, qui facilite la communication avec les utilisateurs
- approche qui supporte l'évolution des besoins
- la modélisation est plus stable
- les évolutions fonctionnelles ne remettent pas en cause l'architecture du système
- approche qui facilite l'intégration de constituants du système
- approche qui facilite la réutilisation des composants (qui sont moins ad-hoc que dans les approches fonctionnelles).

Méthodes adaptatives

Méthodes prédictives

- Cycle de vie du logiciel en cascade ou en V
- Planification très précise et très détaillée
- Peu d'évolutions possibles

Méthodes adaptatives (ou agiles)

- Cycle de vie itératif
- Changements inévitables (des besoins, de l'architecture, de la conception, etc.)
- Livraison de fonctionnalités utiles au client
- Moins de documentation

Deux exemples de méthodes adaptatives : Le *processus unifié* et la *programmation extrême*

Processus unifié 1/2

Cycle de vie itératif, basé sur quatre cycles :

- Etude d'opportunité (vision, étude, analyse)
- Elaboration (analyse, architecture)
- Construction (conception, développement, validation)
- Transition (transfert, installation, formation)

Chaque phase est réalisée par une suite d'itérations, chaque itération comportant plusieurs étapes du cycle de vie.

Processus unifié 2/2

Programmation extrême 1/3

L'*eXtrem Programming* une méthode de développement légère

- Logiciels de qualité
- Minimum de règles
- Minimum de document à produire

La clarté du code prime sur la documentation.

« La conception c'est le programme et le programme c'est la conception »

Règles

- Communication (entre développeurs, avec les utilisateurs)
- Simplicité (conception et programmation simples et claires)
- Retour (tests, livraisons régulières, prise en compte des remarques des utilisateurs)
- Courage

Programmation extrême 2/3

Pratiques

- Travail en équipe, avec un « client sur site »
- Planning, réunions
- Petites livraisons
- Conception simple et « refactorisation »
- Programmation par paires, propriété collective du code et conventions de programmation
- Développement piloté par les tests
- Intégration continue
- Métaphore
- Rythme supportable

Programmation extrême 3/3

Limites pour des projets de grande taille :

- les programmeurs ne peuvent pas maîtriser une grande partie du code
- intégration continue impossible
- ré-exécution quotidienne des tests impossible
- le développement peut être multi-sites