Génie Logiciel

Principes et Techniques

Document de travail dédié aux étudiants en 3 ème SI

Source : Pierre Gérard
IUT de Villetaneuse - Universit□e de Paris 13

Table des matières

I Processus de développement logiciel **3**

1 Motivations **3**
1.1 Qualités attendues d'un logiciel 4
1.2 Principes du Génie Logiciel 6
1.3 Maturité du processus de développement logiciel 7

2 Cycle de vie d'un logiciel **8**
2.1 Composantes du cycle de vie d'un logiciel 8
2.2 Documents courants 9
2.3 Modèles de cycle de vie d'un logiciel 10
2.4 Modèles de processus logiciels 12

Première partie

Processus de développement logiciel

1 Motivations

Matériel et logiciel

- Systèmes informatiques
  - 80 % de logiciel
  - 20 % de matériel
- Depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement
  - Le matériel est relativement fiable
  - Le marché est standardisé
- Les problèmes liés à l'informatique sont essentiellement des problèmes de **Logiciel**

Spécificités du logiciel

- Un produit immatériel, dont l'existence est indépendante du support physique
  - Semblable à une œuvre d'art (roman, partition...)
- Un objet technique fortement contraint
  - Fonctionne ou ne fonctionne pas
  - Structure complexe
  - Relève des modes de travail du domaine technique
- Un cycle de production différent
  - La reproduction pose peu de problèmes, seule la première copie d'un logiciel a un coût
  - Production à l'unité
  - Semblable au Génie Civil (ponts, routes...)

Un processus de fabrication original

- Le logiciel partage des propriétés contradictoires avec l'art, les technologies et le Génie Civil
- Les possibilités de réutiliser les savoir-faire des autres technologies sont (très) limitées
- Compte tenu du cycle de production, il faut bien faire tout de suite

« La qualité du processus de fabrication est garante de la qualité du produit »

La « Crise du logiciel »

- Etude sur 8 380 projets *(Standish Group, 1995)*
  - Succès : 16 %
  - Problématique : 53 % (bujet ou délais non respectés, défaut de fonctionnalités)
  - Echec : 31 % (abandonné)
- Le taux de succès décroît avec la taille des projets et la taille des entreprises

Le Génie Logiciel

- Conférence de l'OTAN à Garmish, Allemagne (1968)
  - L'informatique ne répond pas aux attentes qu'elle suscite

  - L'informatique coûte très cher et désorganise les entreprises ou organisations
- Introduction de l'expression « Génie Logiciel » *(Software Engineering)*
  - Comment faire des logiciels de qualité ?
  - Qu'attend-on d'un logiciel ? Quels sont les critères de qualité pour un logiciel ?

1.1 Qualités attendues d'un logiciel

Utilité

- Adéquation entre
  - Le besoin effectif de l'utilisateur
  - Les fonctions offertes par le logiciel
- Solutions :
  - Emphase sur l'analyse des besoins
  - Améliorer la communication (langage commun, démarche participative)
  - Travailler avec rigueur

Utilisabilité

- *« Effectivité, efficacité et satisfaction avec laquelle des utilisateurs spécifiés accomplissent des objectifs spécifiés dans un environnement particulier »*
  - Facilité d'apprentissage : comprendre ce que l'on peut faire avec le logiciel, et savoir comment le faire
  - Facilité d'utilisation : importance de l'effort nécessaire pour utiliser le logiciel à des fins données
  - Solutions :
    - Analyse du mode opératoire des utilisateurs
    - Adapter l'ergonomie des logiciels aux utilisateurs

Fiabilité

- Correction, justesse, conformité : le logiciel est conforme à ses spécifications, les résultats sont ceux attendus
- Robustesse, sureté : le logiciel fonctionne raisonnablement en toutes circonstances, rien de catastrophique ne peut survenir, même en dehors des conditions d'utilisation prévues
- Mesures :
  - MTBF : Mean Time Between Failures
  - Disponibilité (pourcentatge du temps pendant lequel le système est utilisable) et Taux d'erreur (nombre d'erreurs par KLOC)
- Solutions :
  - Utiliser des méthodes formelles, des langages et des méthodes de programmation de haut niveau
  - Vérifications, tests
  - Progiciels

Interopérabilité, couplabilité

- Un logiciel doit pouvoir interagir en synergie avec d'autres logiciels
- Solutions :
  - Bases de données (découplage données/traitements)

- « Externaliser » certaines fonctions en utilisant des « Middleware » avec une API (Application Program Interface) bien définie
- Standardisation des formats de fichiers (XML...) et des protocoles de communication (CORBA...)
- Les ERP (Entreprise Resources Planning)

Performance
- Les logiciels doivent satisfaire aux contraintes de temps d'exécution
- Solutions :
  - Logiciels plus simples
  - Veiller à la complexité des algorithmes
  - Machines plus performantes

Portabilité
- Un même logiciel doit pouvoir fonctionner sur plusieurs machines
- Solutions :
  - Rendre le logiciel indépendant de son environnement d'exécution (voir interopérabilité)
  - Machines virtuelles

Réutilisabilté
- On peut espérer des gains considérables car dans la plupart des logiciels :
  - 80 % du code est du « tout venant » qu'on retrouve à peu près partout
  - 20 % du code est spécifique
- Solutions :
  - Abstraction, généricité (ex : MCD générique de réservation)
  - Construire un logiciel à partir de composants prêts à l'emploi
  - « Design Patterns »

Facilité de maintenance
- Un logiciel ne s'use pas
- Pourtant, la maintenance absobe un très grosse partie des efforts de développement

	Répartition effort dév.	Origine des erreurs	Coût de la maintenance
Définition des besoins	6%	56%	82%
Conception	5%	27%	13%
Codage	7%	7%	1%
Intégration Tests	15%	10%	4%
Maintenance	67%		

(Zeltovitz, De Marco)

Maintenance corrective
- Corriger les erreurs : défauts d'utilité, d'utilisabilité, de fiabilité...
  - Identifier la défaillance, le fonctionnement

  - Localiser la partie du code responsable
  - Corriger et estimer l'impact d'une modification
- *Attention*
  - La plupart des corrections introduisent de nouvelles erreurs
  - Les coûts de correction augmentent exponentiellement avec le délai de détection
- La maintenance corrective donne lieu à de nouvelles livraisons (release)

Maintenance adaptative
- Ajuster le logiciel pour qu'il continue à remplir son rôle compte tenu du l'évolution des
  - Environnements d'exécution
  - Fonctions à satisfaire
  - Conditions d'utilisation
- Ex : changement de SGBD, de machine, de taux de TVA, an 2000, euro...

Maintenance perfective, d'extension
- Accroître/améliorer les possibilités du logiciel
- Ex : les services offerts, l'interface utilisateur, les performances...
- Donne lieu à de nouvelles versions

Facilité de maintenance
- Objectifs
  - Réduire la quantité de maintenance corrective (zéro défaut)
  - Rendre moins coûteuses les autres maintenances
- Enjeux
  - Les coûts de maintenance se jouent très tôt dans le processus d'élaboration du logiciel
  - Au fur et à mesure de la dégradation de la structure, la maintenance devient de plus en plus difficile
- Solutions :
  - Réutilisabilté, modularité
  - Vérifier, tester
  - Structures de données complexes et algorithmes simples
  - Anticiper les changements à venir
  - Progiciels

1.2 Principes du Génie Logiciel

Principes utilisés dans le Génie Logiciel
- Généralisation : regroupement d'un ensemble de fonctionnalités semblables en une fonctionnalité paramétrable (généricité, héritage)
- Structuration : façon de décomposer un logiciel (utilisation d'une méthode bottom-up ou top-down)
- Abstraction : mécanisme qui permet de présenter un contexte en exprimant les éléments pertinents et en omettant ceux qui ne le sont pas
- Modularité : décomposition d'un logiciel en composants discrets
- Documentation : gestion des documents incluant leur identification, acquisition, production, stockage et distribution

- Vérification : détermination du respect des spécifications établies sur la base des besoins identifiés dans la phase précédente du cycle de vie

1.3 Maturité du processus de développement logiciel

Objectif CCM 6 !

- Le Modèle de Maturité (CMM) du SEI
  1. Initial
  2. Reproductible
  3. Défini
  4. Maîtrisé
  5. Optimisé
  - 75% des projets au niveau 1, 25% aux niveaux 2 et 3 selon Curtis
- Pour maîtriser le processus de développement logiciel et assurer la qualité du logiciel, il faut :
  - Séparer le développement en plusieurs étapes
  - Organiser ces étapes et modéliser le processus de développement
  - Contrôler le processus de développement

Niveau de maturité 1 : Initial

- Chaotique : plans et contrôles inefficaces
  - Processus essentiellement non contrôlé, non défini
  - Le succès dépend des individus

Niveau de maturité 2 : Reproductible

- Intuitif : dépend encore des individus
  - Procédures de gestion utilisées, gestion des configurations et assurance qualité
  - Pas de modèle formel de processus

Niveau de maturité 3 : Défini

- Qualitatif : institutionnalisé
  - Procédures formelles pour vérifier que le processus est utilisé

Niveaux de maturité 4 : Maîtrisé

- Quantitatif : Processus de mesures
  - Gestion quantitative de la qualité

Niveaux de maturité 5 : Optimisé

- Améliorations retournées dans le processus
- Stratégies d'amélioration du processus

2 Cycle de vie d'un logiciel

Cycle de vie

« La qualité du processus de fabrication est garante de la qualité du produit »

- Pour obtenir un logiciel de qualité, il faut en maîtriser le processus d'élaboration
  - La vie d'un logiciel est composée de différentes étapes
  - La succession de ces étapes forme le cycle de vie du logiciel
  - Il faut contrôler la succession de ces différentes étapes

2.1 Composantes du cycle de vie d'un logiciel

Etude de faisabilité

- Déterminer si le développement proposé vaut la peine d'être mis en œuvre, compte tenu de attentes et de la difficulté de développement
  - Etude de marché : déterminer s'il existe un marché potentiel pour le produit.

Spécification

- Déterminer les fonctionnalités que doit posséder le logiciel
  - Collecte des exigences : obtenir de l'utilisateur ses exigences pour le logiciel
  - Analyse du domaine : déterminer les tâches et les structures qui se répètent dans le problème

Organisation du projet

- Déterminer comment on va développer le logiciel
  - Analyse des coûts : établir une estimation du prix du projet
  - Planification : établir un calendrier de développement
  - Assurance qualité du logiciel : déterminer les actions qui permettront de s'assurer de la qualité du produit fini
  - Répartition des tâches : hiérarchiser les tâches et sous-tâches nécessaires au développement du logiciel

Conception

- Déterminer la façon dont dont le logiciel fournit les différentes fonctionnalités recherchées
  - Conception générale
    - Conception architecturale : déterminer la structure du système
    - Conception des interfaces : déterminer la façon dont les différentes parties du système agissent entre elles
  - Conception détaillée : déterminer les algorithmes pour les différentes parties du système

Implémentation

- Ecrire le logiciel

Tests

- Essayer le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement
  - Tests unitaires : faire tester les parties du logiciel par leurs développeurs
  - Tests d'intégration : tester pendant l'intégration
  - Tests de validation : pour acceptation par l'acheteur
  - *Tests système :* tester dans un environnement proche de l'environnement de production
  - *Tests Alpha :* faire tester par le client sur le site de développement
  - *Tests Bêta :* faire tester par le client sur le site de production
  - *Tests de régression :* enregistrer les résultats des tests et les comparer à ceux des anciennes versions pour vérifier si la nouvelle n'en a pas dégradé d'autres

Livraison

- Fournir au client une solution logicielle qui fonctionne correctement
  - Installation : rendre le logiciel opérationnel sur le site du client
  - Formation : enseigner aux utilisateurs à se servir du logiciel
  - Assistance : répondre aux questions des utilisateurs

Maintenance

- Mettre à jour et améliorer le logiciel pour assurer sa pérénité
- Pour limiter le temps et les coûts de maintenance, il faut porter ses efforts sur les étapes antérieures

2.2 Documents courants

Cahier des charges

- Description initiale des fonctionnalités désirées, généralement écrite par l'utilisateur

Spécifications

- Décrit précisément les conditions que doit remplir le logiciel
  - Modèle objet : indique les classes et les documents principaux
  - Scénarios des cas d'utilisation : indique les différents enchaînements possibles du point de vue de l'utilisateur

Calendrier du projet

- Ordre des différentes tâches
- Détails et ressources qu'elles demandent

Plan de test du logiciel

- Décrit les procédures de tests appliquées au logiciel pour contrôler son bon fonctionnement
  - Tests de validation : tests choisis par le client pour déterminer s'il peut accepter le logiciel

Plan d'assurance qualité

- Décrit les activités mises en œuvre pour garantir la qualité du logiciel

Manuel utilisateur
- Mode d'empoi pour le logiciel dans sa version finale

Code source
- Code complet du produit fini

Rapport des tests
- Décrit les tests effectués et les réactions du système

Rapport des défauts
- Décrit les comportements du système qui n'ont pas satisfait le client
- Il s'agit le plus souvent de défaillances du logiciel ou d'erreurs

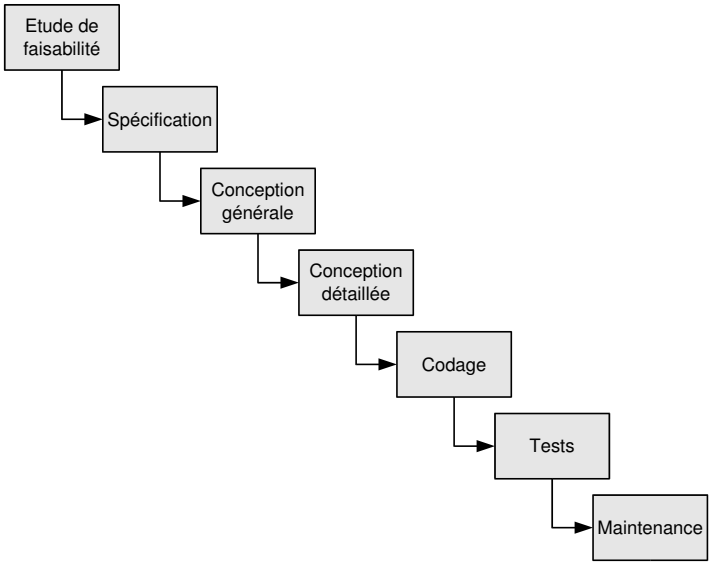
Documents produits dans le cycle de vie

Document	*Phase de production*
Manuel utilisateur final	Implémentation
Conception architecturale	Conception
Plan d'assurance qualité	Planification
Code source	Implémentation
Cahier des charges	Faisabilité
Plan de test	Spécification
Manuel utilisateur préliminaire	Spécification
Conception détaillée	Conception
Estimation des coûts	Planification
Calendrier du projet	Planification
Rapport des tests	Tests
Documentation	Implémentation

2.3 Modèles de cycle de vie d'un logiciel

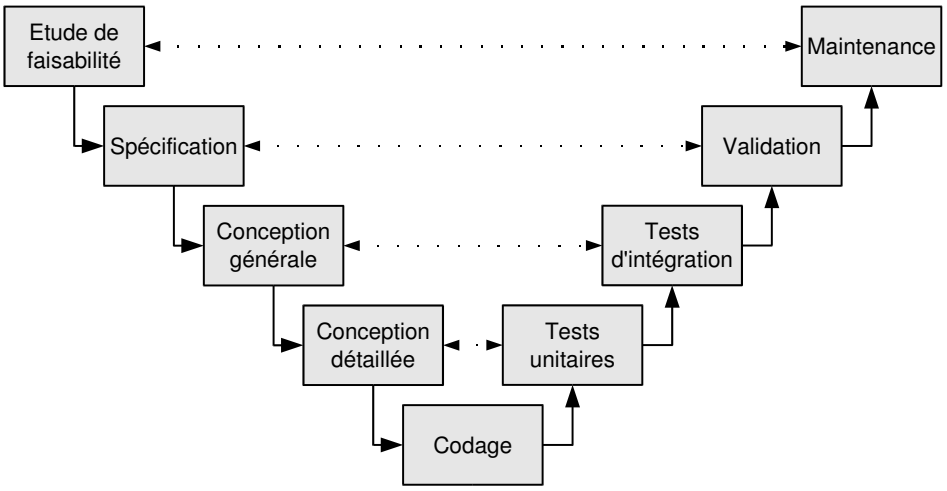
Modèles linéaires et incrémentaux
- Modèles linéaires
  - cascade
  - modèle en V
  - ...
- Modèles non linéaires
  - prototypage
  - modèles incrémentaux
  - modèle en spirale
  - ...

Le cycle de vie en « Cascade »



– Adapté pour des projets de petite taille, et dont le domaine est bien maîtrisé

Le cycle de vie en « V »



– Adapté pour des projets dont le domaine est bien maîtrisé

Le prototypage

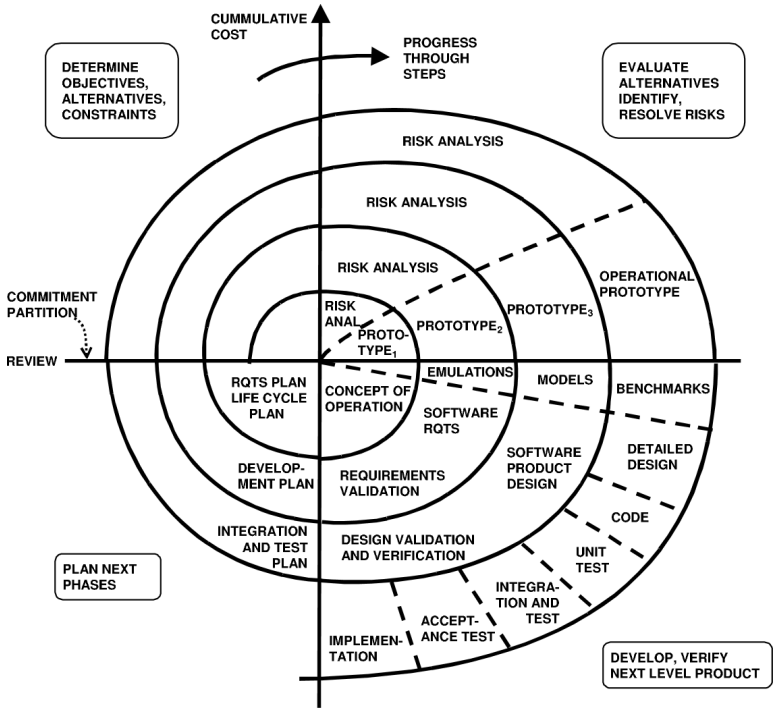
– Prototype : version d'essai du logiciel
  – Pour tester les différents concepts et exigences
  – Pour montrer aux clients les fonctions que l'on veut mettre en œuvre
– Lorsque le client a donnné son accord, le développement suit souvent un cycle de vie linéaire
– Avantages : Les efforts consacrés au développement d'un prototype sont le plus souvent compensés par ceux gagnés à ne pas développer de fonctions inutiles

Le modèle incrémental de Parnas

1. Concevoir et livrer au client un sous-ensemble minimal et fonctionnel du système
2. Procéder par ajouts d'incréments minimaux jusqu'à la fin du processus de développement
3. Avantages : meilleure intégration du client dans la boucle, produit conforme à ses attentes

Le modèle en Sprirale de Boehm

- Un modèle mixte
- A chaque cycle, recommencer :
  1. Consultation du client
  2. Analyse des risques
  3. Conception
  4. Implémentation
  5. Tests
  6. Planification du prochain cycle



- Avantages : meilleure maîtrise des risques, mais nécessite une (très) grande expérience

2.4 Modèles de processus logiciels

Modélisation des processus logiciels

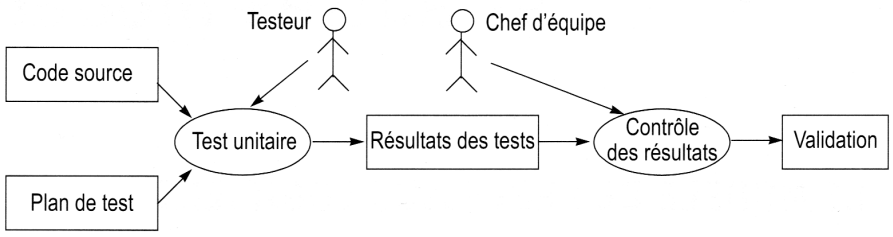
- Les modèles en V, spirale... sont des archétypes de modèles
  - On peut les adapter à des projets particuliers

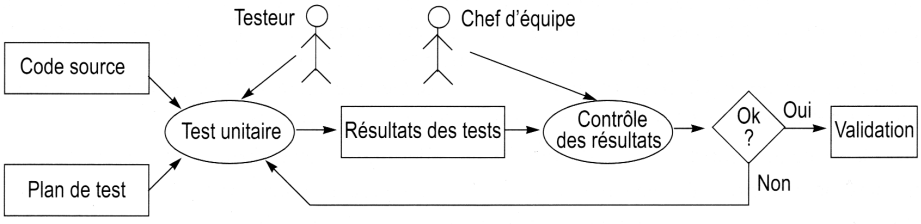
- On peut vouloir représenter le processus de développement (et ses parties) de manière plus fine
  - Une modélisation précise limite les risques d'ambigüité
- Il est nécessaire de se doter d'un formalisme pour représenter le processus de développement
  - Pour ordonner les tâches
  - Pour déterminer les échanges d'information entre les différentes tâches
  - Pour permettre aux jeunes recrues de mieux travailler

Modèle de processus logiciels

- Un modèle de processus logiciels décrit
  - Les tâches
  - Les artefacts (fichiers, documents, données...)
  - Les auteurs
  - Les décisions (facultatif)
- Règles à observer
  - Deux tâches doivent être séparées par un artefact
  - Une tâche ne peut être exécutée tant que ses artefacts d'entrée n'existent pas
  - Il doit y avoir au moins une tâche de début et une de fin
  - Il doit y avoir un trajet depuis chaque tâche jusqu'à la tâche de fin

Exemples de processus logiciels



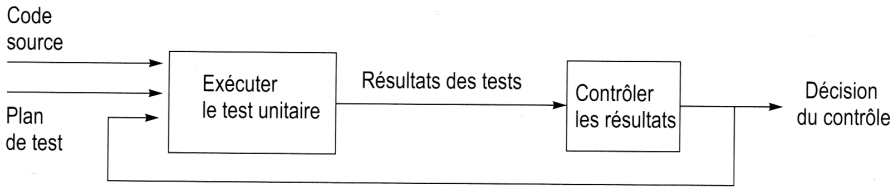


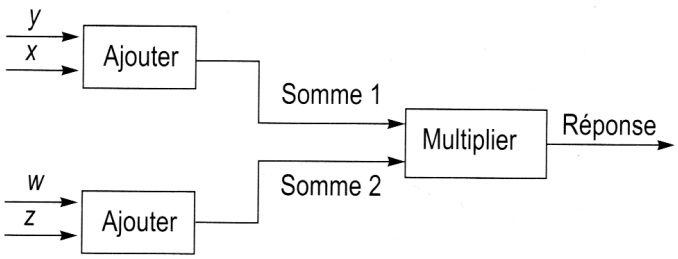
Diagrammes de flots de données

- Indique la circulation des données à travers un ensemble de composants qui peuvent être
  - des tâches
  - des composants logiciels...
- Règles à observer
  - Les processus sont représentés par des cases qui contiennent des phrases verbales

- Les flèches représentent des données et doivent être accompagnées de phrases nominales
- Un processus peut être une activité ponctuelle ou continue
- Deux flèches sortant d'une case peuvent indiquer
  - Soit deux sorties simultanées
  - Soit deux sorties exclusives

Exemples de diagramme de flots de données





pour représenter $(x + y) \times (w + z)$