

Génie Logiciel II

Dr FERRAHI

Intitulé de l'UE : UEF5.1

Nombre de crédits : 5

Coefficient de la Matière : 3

Volume horaire : 1h:30mn de cours, 1h:30mn de TD ET 1h:30mn de TP

Prérequis

- ▷ Connaissance de base en génie logiciels

Objectifs du cours

- Maîtriser les aspects de conception de logiciels
- Acquérir les bases de la conception de qualité
- Utiliser les patrons de conception

Evaluation du cours (Interrogation écriteTD+ Test de TP + examen final)

2

Plan

- **Chapitre 1:** Notions de base et principes de la conception de logiciels
- **Chapitre 2:** Conduite de projet
- **Chapitre 3:** Métriques de GL
- **Chapitre 4:** Les patrons de conception
- **Chapitre 5:** Principe SOLID

3

1

Notions de base et principes de la conception de logiciels

4

Rappel



5

Logiciel: C'est quoi?



6



Définition d'un logiciel

- C'est un **ensemble de séquence d'instructions** interprétables par une machine (Ordinateur, smartphone, tablette, console de jeux, ...etc.) et d'un jeu de données nécessaires à ces opérations.

7

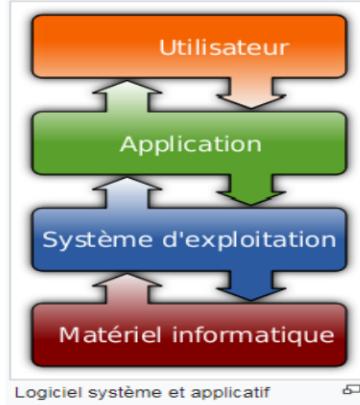
Logiciel: quels sont les classes d' logiciel?



8

Logiciel: quels sont les classes d' logiciel?

- Logiciels système
- Logiciels applicatif



9



Logiciel: quels sont les classes de logiciels?

■ Logiciel système

- ▷ C'est un Le logiciel un ensemble de programmes informatiques et de bibliothèques logiciels qui fournit un environnement permettant de créer et d'exécuter des logiciel applicatifs.

■ Exemple de logiciel système



10

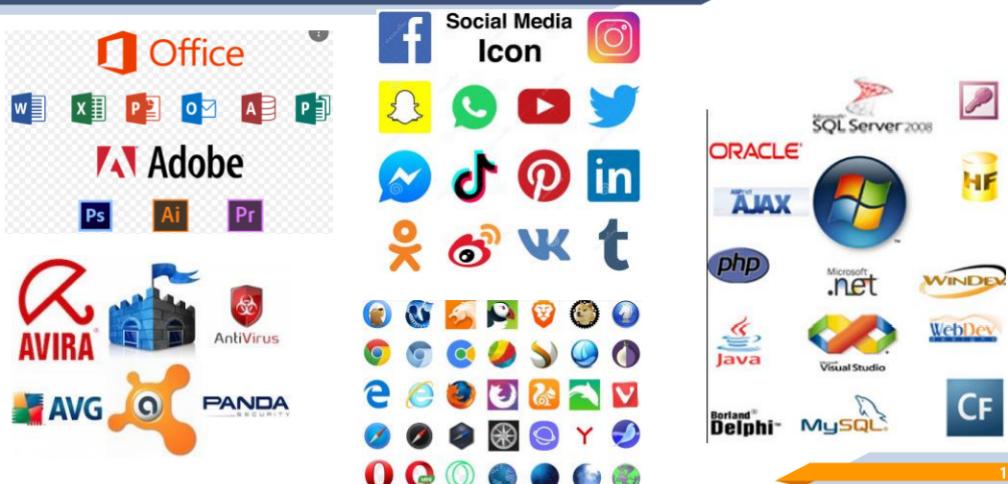
Logiciel: quels sont les classes de logiciels

■ Logiciel applicatif

- ▷ Une application est un programme (ou un ensemble logiciel) directement utilisé pour réaliser une tâche, ou un ensemble de tâches élémentaires d'un même domaine ou formant un tout.

11

Exemple de logiciel applicatif



12

Logiciel: quels sont les classes de logiciels?

- Un progiciel



13

Logiciel: quels sont les classes de logiciels?

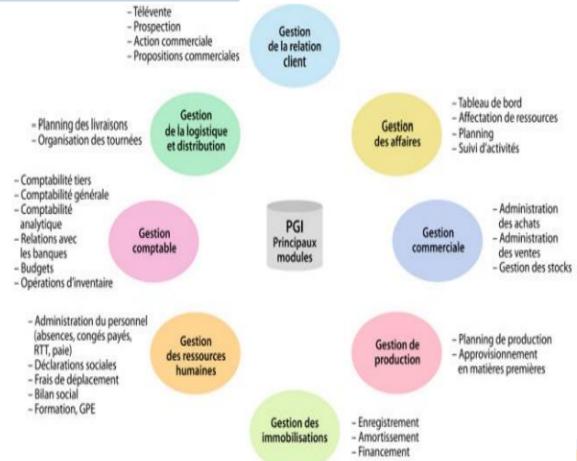
- **Un progiciel** désigne un logiciel applicatif généraliste aux multiples fonctions, composé d'un ensemble de programmes paramétrables et destiné à être utilisé par une large clientèle.

14

Logiciel: quels sont les classes de logiciels?

Exemple de progiciel

- ▷ Progiciel ERP (Entreprise Resource Planing)
- ▷ Projet de gestion Intégré (PGI)



15

Logiciel: une autre classification

- Les logiciels commerciaux;
- Les logiciels libres;
- Les shareware;
- Les freeware.

16

Débat

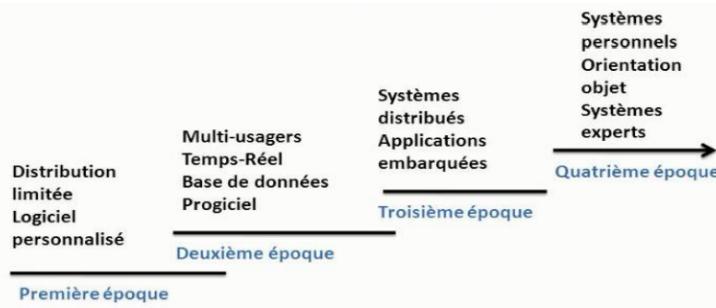
- Développer un logiciel de gestion de la scolarité de l'école

17

Crise du logiciel

Crise du logiciel

- À partir des années 1960, l'informatique conquiert de nouveaux champs d'application. Initialement centrée sur le calcul scientifique, elle s'impose dans:



19

La crise du logiciel

- Au cours de cette période:
 - la taille des systèmes informatiques connaît une croissance exponentielle.
 - Leur complexité croît en proportion.



20

Crise du logiciel

- Cette crise est apparue avec l'avènement des ordinateurs de la 3ème génération (années 60) et s'est accentuée avec l'ère de la micro-informatique (4ème génération).
 - ▷ **leur coût et leur puissance** ont permis la prolifération de l'informatique dans beaucoup de domaines avec la réalisation de grands systèmes logiciels.
- Les premières expériences de construction de ce type de systèmes ont montré que les méthodes de développement de l'époque **n'étaient pas adéquates**.
- Donc, il n'était plus possible de se contenter d'adapter les techniques applicables à de petits systèmes
 - ▷ Les problèmes rencontrés dans le développement des gros systèmes logiciels ne sont pas directement comparables à ceux relatifs aux petits.

21

Crise du logiciel

- **La complexité** des petits programmes peut être appréhendée par une seule personne qui peut avoir à l'esprit tous les détails de la conception et de la réalisation.
- Leurs spécifications peuvent rester informelles et les effets de toutes modifications sont en général évidents.
- En revanche, la complexité des grands systèmes est telle qu'il est impossible à une seule personne d'avoir présent à l'esprit tous les détails du projet et de les mettre à jour.
- Ce sont des équipes d'analystes, de concepteurs, de développeurs et de testeurs qui doivent collaborer pour parvenir à réaliser ce type

22

Exemple de Logiciels complexes

Notation :

- ▷ **HA** : Homme-Année (unité de coût représentant l'effort nécessaire pour développer un logiciel)
- ▷ **HM** : Homme-Mois
- ▷ **Kls** : un millier de lignes de code source (unité de mesure de la taille d'un logiciel)
- ▷ **Ls** : une ligne de code source.

Remarque : La productivité de développement peut être exprimée comme étant le rapport entre la taille du logiciel et son coût. Cet indicateur dénote la difficulté de fabrication du logiciel.
(Unité de mesure = Ls/hm)

23

Nature du Logiciel	Coût en HA	Volume	Remarque
Compilateurs :			Délais :
- Pascal	10 ha	20 à 30 kls	1 à 2 ans
- COBOL, FORTRAN	80 à 100 ha	100 à 200 kls	2 à 3 ans
- ADA	150 à 200 ha	>300 kls	> 3 ans
SGBD Relationnel (ORACLE, DB2, ...)	300 à 500 ha	300 à 600 kls	Délai de 3 à 5 ans incluant une 1 ^{ère} version.
Grands systèmes temps réel :			
- Navette Spatiale	>1000 ha	2200 kls	Délai 6 ans, écrit en HAL
- SAFEGUARD (1)	5000 ha	2260 kls	Délai 7 ans, écrit en p1
- SABRE (2)	955 ha	960 kls	Délai 10 ans pour un MTTF (3) de 55 h.
Systèmes constructeurs : VMS, GCOS7	2500 à 5000 ha	5000 à 10000 kls	Durée de vie 15 à 20 ans dont au moins 5 ans de délai pour une 1 ^{ère} version.
Systèmes industriels : GPAO, MRP, ...	150 à 500 ha	500 à 1000 kls	Bases de données importantes.

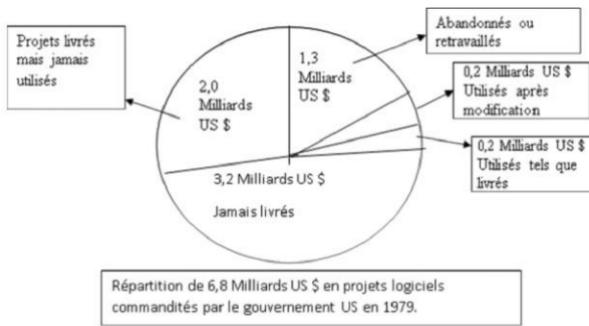
(1) Système de défense antibalistique des USA (années 70).

(2) Système de réservation d'American Airlines réalisé par IBM.

(3) Temps moyen de bon fonctionnement (mean time to failure).

Crise du logiciel

- La crise du logiciel dont on ne cesse de parler depuis les années 60 peut être illustrée par le schéma suivant :



25

Crise du logiciel

- Un constat: les projets souffraient de :
- ▷ **Retards** dans les livraisons (parfois des années).
 - ▷ **Surcoûts** considérables (dépassant largement les budgets prévus).
 - ▷ Produits qui ne répondent pas **aux besoins des usagers** (peu fiables, peu performants, difficiles à maintenir, etc.).
 - ▷ Produits **peu sûrs** (comportant des erreurs dont le nombre et l'emplacement sont souvent inconnus et ayant des répercussions graves sur la mission du système).
 - ▷ Ces problèmes s'apparentent avec une croissance exponentielle de la demande de logiciels et avec la croissance dans la complexité des systèmes construits dans un contexte caractérisé par la diminution des prix du matériel et le gonflement incontrôlable des prix de logiciels

26

Crise du logiciel

Coût moyen d'indisponibilité

Secteur industriel		Production et distribution d'énergie	2,8	Millions d'Euros par heure perdue
	Production manufacturière	1,6		
	Institutions financières	1,4		
	Assurances	1,2		
	Commerce	1,1		
	Banques	1		

Coût annuel des défaillances informatiques

Estimation compagnies d'assurance (2002)	France (secteur privé)	USA	Royaume Uni
Fautes accidentelles	1,1 G€	4 G\$	
Malveillances	1,3 G€		1,25 G£

Estimation globale | USA : 80 G\$ | EU : 60 G€

Coûts de maintenance

Logiciel embarqué de la navette spatiale : 100 M \$ / an

Coût logiciels abandonnés (défaillance du processus de développement)

USA [Standish Group, 2002, 13522 projets]	Succès	Remise en question	Abandon
	34%	51%	15%
~ 38 G\$ de pertes (sur total 225 G\$)			

27

Prise de conscience Naissance du génie logiciel



Naissance du génie logiciel

- Ce terme est né en 1968 (7-11 octobre 1968) en pleine période de crise de logiciel sous le nom anglo-saxon : Software Engineering et sous le parrainage de l'OTAN.
- Afin d'augmenter la productivité des équipes de développement et d'améliorer la qualité des produits, des procédés de fabrication de ces différents logiciels ont été proposés sous le nom de Génie Logiciel afin de s'assurer que :

29

Définition du génie logiciel

- ce qui est fabriqué par le **maître d'œuvre** répond aux besoins de celui qui les a formulés :
 - ▷ le **maître d'ouvrage** ou le représentant du client final
- Les **coûts** et les **délais** de réalisation restent dans les limites fixés au départ
- Le **contrat de service** sera effectivement respecté (performance, sûreté de fonctionnement, sécurité, ...) lors de l'exploitation future du logiciel.
- Aussi, s'assurer de l'**aptitude d'évolution** lors de l'apparition de nouveaux besoins :
 - ▷ caractéristique importante ayant un effet direct sur la durée de vie du système
 - ▷ d'où une répercussion directe sur son coût d'amortissement

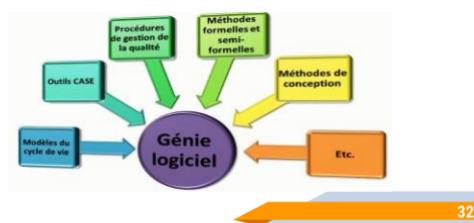
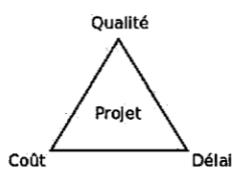
30

Génie logiciel



Génie logiciel

- Ensemble de méthodes techniques et outils pour produire un des **logiciels de qualité** à moindre **coût** et dans des **délais raisonnables**
- Le génie logiciel (software engineering) représente l'application de principes d'ingénierie au domaine de la création de logiciels. Il consiste à identifier et à utiliser des méthodes, des pratiques et des outils permettant de maximiser les chances de réussite d'un projet logiciel.



32

Qualité attendues d'un logiciel

- a. Utilité (correspond au besoins de l'utilisateur)
- b. Utilisabilité
- c. Fiabilité
- d. Interopérabilité (interaction avec les autres logiciels)
- e. Performance
- f. Portabilité
- g. Ré-utilisabilité
- h. Facilité de maintenance

33

Qualités attendues d'un logiciel:

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

34

Qualités attendues d'un logiciel:

b. Utilisabilité

- ▷ Facilite d'apprentissage : comprendre ce que l'on peut faire avec le logiciel, et savoir comment le faire.
- ▷ Facilite 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

35

Qualités attendues d'un logiciel:

c. Fiabilité

- ▷ Correction, justesse, conformité : le logiciel est conforme à ses spécifications, les résultats sont ceux attendus
- ▷ Robustesse, sûreté : 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 (pourcentage du temps pendant lequel le système est utilisable) et Taux d'erreur (nombre d'erreurs par KLOC)

36

Qualités attendues d'un logiciel:

■ c. Fiabilité

■ Solutions :

- ▷ Utiliser des méthodes formelles, des langages et des méthodes de programmation de haut niveau
- ▷ Vérifications, tests
- ▷ Progiciels

37

Qualités attendues d'un logiciel:

■ d. Interopérabilité (interaction avec les autres logiciels)

- ▷ 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)

38

Qualités attendues d'un logiciel:

e. Performance

Les logiciels doivent satisfaire aux contraintes de temps d'exécution

Solutions :

- ▷ Logiciels plus simples
- ▷ Veiller à la complexité des algorithmes
- ▷ Machines plus performantes

39

Qualités attendues d'un logiciel:

f. Portabilité Un même logiciel doit pouvoir fonctionner sur plusieurs machines

Solutions :

- ▷ Rendre le logiciel indépendant de son environnement d'exécution (voir interopérabilité)
- ▷ Machines virtuelles

40

Qualités attendues d'un logiciel:

■ **g. Ré-utilisabilité** 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 >

41

Qualités attendues d'un logiciel:

■ **h. Facilité de maintenance**

Un logiciel ne s'use pas. Pourtant, la maintenance absorbe un très grosse partie des efforts de développement (67%)

■ Objectifs

- ▷ Réduire la quantité de maintenance corrective (zéro défaut)
- ▷ Rendre moins couteuses les autres maintenances

■ Enjeux

- ▷ Les couts 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

42

Qualités attendues d'un logiciel:

h. Facilité de maintenance

Solutions :

- ▷ Réutilisable, modularité
- ▷ Vérifier, tester
- ▷ Structures de données complexes et algorithmes simples
- ▷ Anticiper les changements à venir Progiciel

43

Qualités attendues d'un logiciel:

e. Facilité de maintenance

Solutions :

- ▷ 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 => suivre une méthode décrivant clairement les étapes à suivre pour élaborer un logiciel.

44

Principes Utilisés dans le Génie Logiciel

Principes Utilisés dans le Génie Logiciel

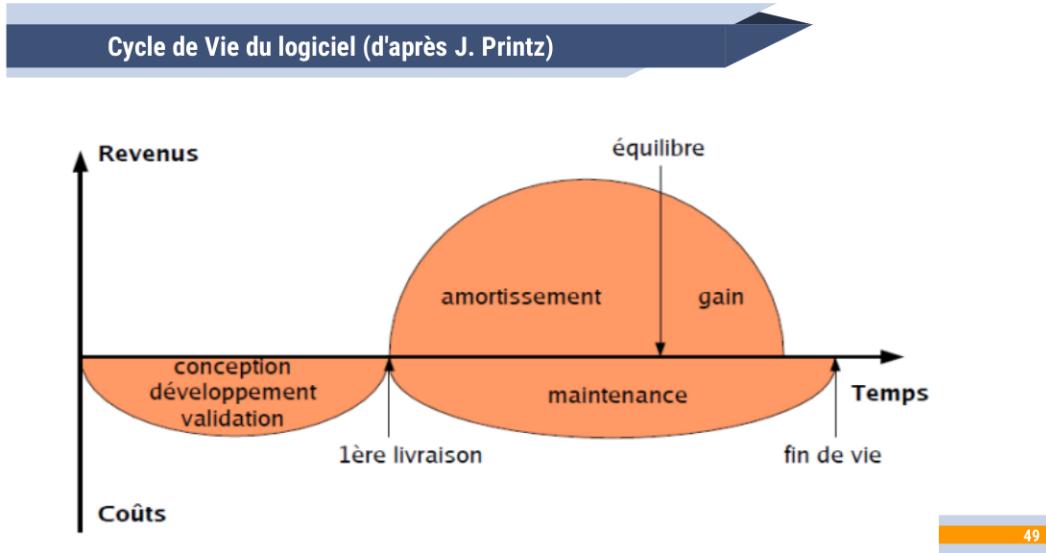
- **Généralisation** : regroupement d'un ensemble de fonctionnalités semblables en une fonctionnalité paramétrable (généricité, héritage)
- **Structuration** : façon de décomposer un logiciel (utilisation d'une méthode bottom-up ou top-down)
- **Abstraction** : mécanisme qui permet de présenter un contexte en exprimant les éléments pertinents et en omettant ceux qui ne le sont pas.

Principes Utilisés dans le Génie Logiciel

- **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

47

Cycle de vie d'un logiciel



Cycle de Vie du logiciel

Pourquoi se préoccuper d'un « cycle de vie » ?

- C'est un processus
 - ▷ **phases** : création, distribution, disparition
- But du découpage
 - ▷ maîtrise des risques
 - ▷ maîtrise des délais et des coûts
 - ▷ contrôle que la qualité est conforme aux exigences (→)
- En fait, problématique plus générale
 - ▷ mais spécificités relatives aux logiciels

Quelques chiffre

- Logiciel de réservation aérienne d'United Airlines
 - ▷ 56 millions de dollars
- Faille : nombre d'instructions par transaction
 - ▷ 146.000 au lieu de 9.000 prévues
- Inutilisable par manque d'analyse
 - ▷ besoins, étude de faisabilité
- Aurait pu être évité
 - ▷ par des inspections et revues intermédiaires

51

Cycle de vie du logiciel

- Définition des besoins (cahier des charges)
- Analyse des besoins (spécification)
- Planification (gestion de projet)
- Conception
- Développement (codage, test, intégration)
- Validation
- Qualification (mise en situation)
- Distribution
- Support
- Maintenance

52

- Phase - Définition des besoins

(on dit aussi « expression des besoins »)

■ Activité (du client, externe ou interne) :

- ▷ consultation d'experts et d'utilisateurs potentiels
- ▷ questionnaire d'observation de l'usager dans ses tâches

■ Production : cahier des charges (les « exigences »)

- ▷ fonctionnalités attendues
- ▷ contraintes non fonctionnelles
 - ▷ qualité, précision, optimalité, ...
 - ▷ temps de réponse, contraintes de taille, de volume de traitement, ...

53

- Phase - Analyse des besoins / Spécification

(on dit aussi « définition du produit »)

■ Productions :

- ▷ dossier d'analyse
- ▷ spécifications fonctionnelles
- ▷ spécifications non-fonctionnelles
- ▷ ébauche du manuel utilisateur (interface attendue)
- ▷ première version du glossaire (être bien compris)

■ À l'issue :

- ▷ client et fournisseur OK sur produit et contraintes

54

- Phase - Planification / Gestion de projet

■ Activités :

- ▷ tâches : découpage, enchaînement, durée, effort (→)
- ▷ choix : normes qualité, méthodes de conception

■ Productions :

- ▷ plan qualité
- ▷ plan projet destiné aux développeurs
- ▷ estimation des coûts réels, devis
- ▷ liste de dépendances extérieures (risques)

55

- Phase - Conception (globale, détaillée)

■ Activités :

- ▷ architecture du logiciel
- ▷ interface entre les différents modules

■ Productions :

- ▷ dossier de conception
- ▷ plan d'intégration
- ▷ plans de tests
- ▷ planning mis à jour

56

- Phase - Codage et tests unitaires

■ Activités :

- ▷ chaque module codé et testé indépendamment

■ Productions :

- ▷ modules codés et testés (→)
- ▷ documentation de chaque module
- ▷ résultats des tests unitaires
- ▷ planning mis à jour

57

- Phase - Intégration

■ Activités :

- ▷ modules testés intégrés suivant le plan d'intégration
- ▷ test de l'ensemble conformément au plan de tests

■ Productions :

- ▷ logiciel testé
- ▷ jeu de tests de non-régression
- ▷ manuel d'installation
- ▷ version finale du manuel utilisateur (→)
- ▷ Tutoriel: introduction pédagogique, exemples d'utilisation

58

- Phase - Qualification

■ Activités :

- ▷ test en vraie grandeur dans les conditions normales d'utilisation

■ Production :

- ▷ diagnostic OK / KO

59

- Phase - Maintenance

■ Activités :

- ▷ maintenance corrective : correction des bugs
- ▷ maintenance évolutive : adaptative ou perfective

■ Productions :

- ▷ logiciel corrigé
- ▷ mises à jour, patch
- ▷ documents corrigés

Maintenance = jusqu'à 2/3 du coût total !

60

- Phase - Maintenance

■ Coûts

- ▷ maintenance = jusqu'à 67% du coût total
- ▷ dont 48 % consacrés à réparer des défauts
- ▷ 60% des défauts correspondent à des erreurs de spécification et de conception

61

Modèle de cycle de vie

Modèle de cycle de vie

■ Cycle de vie

- ▷ Ensemble des étapes par lesquelles passe un logiciel depuis sa phase de spécification jusqu'à sa maintenance une fois en phase d'exploitation

■ Modèle de cycle de vie

- ▷ Organiser les différentes phases de cycle de vie pour l'obtention d'un logiciel fiable, adaptable et efficace
- ▷ Guider le développeur dans ses activités techniques
- ▷ Fournir des moyens pour gérer le développement et la maintenance
 - ▷ Ressources , détails avancements etc

63

■ Le GL perçoit la fabrication des programmes comme **un processus à étapes** ordonnancées selon une certaine **logique d'enchaînement**

■ Le cycle de vie du logiciel est constitué de l'ensemble des étapes successives de la fabrication depuis la commande du logiciel par le maître d'ouvrage jusqu'à sa mise en exploitation et lancement de la maintenance.

■ A chaque étape, une ou **plusieurs activités de développement** sont appliquées.

■ Il y a différents **modèles des cycles de vie** qui diffèrent au niveau de la logique d'enchaînement des étapes.

■ **Remarque :** Un modèle est une représentation abstraite qui se focalise sur les caractéristiques importantes

64

- On distingue dans tout cycle de vie deux sous cycles internes :
 - ▷ **Cycle de développement** comprenant les activités suivantes : Analyse, Conception, Codage et Tests,
 - ▷ **Cycle de maintenance** comprenant les activités suivante: Exploitation et maintenance (corrective, adaptative et évolutive).
- Quant aux modèles de cycles de vie, on distingue trois classes, chacune se distingue des autres selon sa logique d'enchaînement des étapes et la participation des clients au processus de développement :
 - ▷ Modèles linéaires
 - ▷ Modèles évolutifs
 - ▷ Modèles Hybrides.

65

Modèles linéaires de cycles de vie

- **Modèles linéaires** : le système logiciel est développé dans sa totalité en appliquant les étapes de manière séquentielle. Le client intervient uniquement au début et à la fin du cycle de développement.
- Exemples : Modèles en Cascade et en V.
- **Avantage** : ils permettent de développer des logiciels avec un bon ratio qualité/coût, quand les besoins sont claires, stables, faciles à cerner.
- **Inconvénient** : Les erreurs de définition des besoins sont détectées tardivement par le client. Quand les besoins du client sont ambigus, difficiles à cerner et volatiles, ce taux est très élevé qui induisent des coûts et des délais de correction exorbitants, conduisant parfois à l'abandon du projet

66

Modèles évolutifs de cycles de vie

- **Modèles évolutifs** : pour pallier le problème de définition des besoins, le projet est réalisé en appliquant plusieurs cycles de développement, en faisant intervenir le client pendant au moins un cycle de développement.
- **Exemples** : Modèles par prototypage, par incrément, en spirale.
- **Avantage** : Les erreurs de définition des besoins sont détectées par le client pendant les premières étapes de développement. Cela permet de développer des logiciels de bonne qualité, quand les besoins du client sont ambigus, difficiles à cerner et volatiles.
- **Inconvénient** : Ces modèles sont coûteux en termes de délais et budgets. De plus ils nécessitent un personnel de développement qualifié afin d'éviter des risques liés.

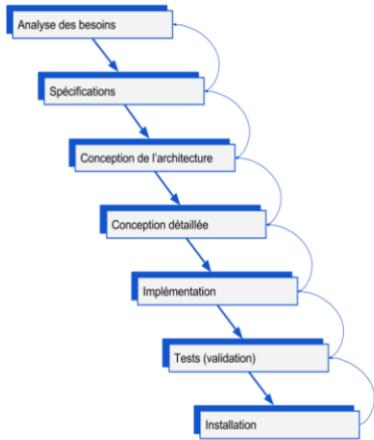
67

Modèles Hybrides de Cycles de Vie

- Dans l'approche évolutive du développement :
 - ▷ Les produits logiciels sont souvent de meilleure qualité.
 - ▷ Cependant, le coût est toujours très élevé.
 - ▷ Le contrôle et l'intégration du changement, surtout dans le cas des grands systèmes, sont souvent plus difficiles.
- La meilleure solution pour les grands systèmes, peut être composée de plusieurs approches
- En se basant sur l'analyse du risque, on choisit l'approche appropriée pour chaque sous-système :
 - ▷ Utilisation du prototypage pour les sous systèmes ayant des spécifications à hauts risques.
 - ▷ Modèle linéaire pour les parties bien connues (besoins stables et clairs)

68

Modèle en cascade date des années 70

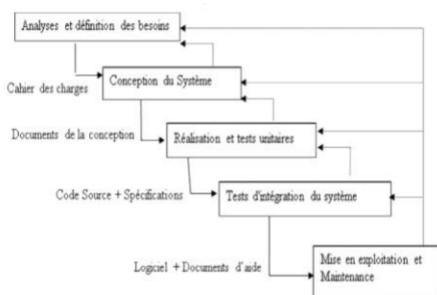


Caractéristiques

- Une étape ne peut pas être débutée avant que la précédente ne soit achevée
- La modification d'une étape du projet a un impact important sur les étapes suivantes.
- Chacune de ces phases doit produire un ou plusieurs livrables définis à l'avance et à une date d'échéance fixée.

69

Modèle en cascade date des années 70



Avantages du modèle en cascade :

- ▷ Le planning est établi à l'avance.

Inconvénients du modèle en cascade :

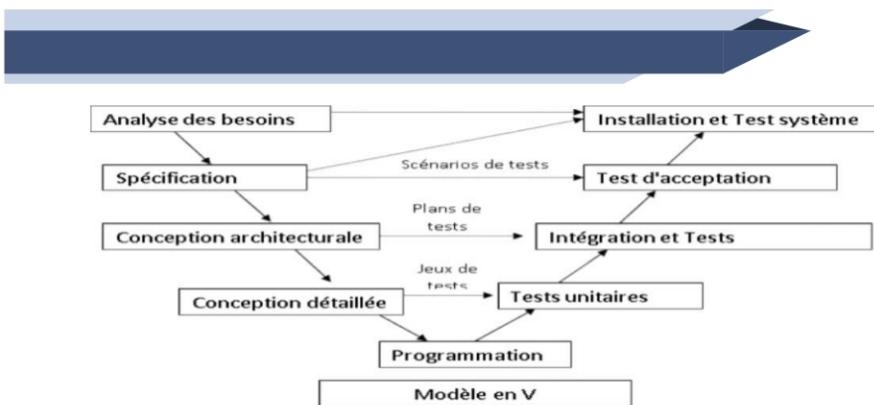
- ▷ ils sont assez nombreux mais le principal **inconvénient est la très faible tolérance à l'erreur.**
- ▷ Sensibilité à l'arrivée de nouvelles exigences
- ▷ Validation trop tardive
- ▷ Dure trop longtemps

70

Modèle en V

- Dans ce modèle, les premières étapes du développement du logiciel, doivent préparer, pour les étapes finales, les données liées aux activités de validation et de vérification (V&V).
- L'idée de ce modèle est qu'avec la décomposition, la reconstitution doit être décrite et que toute description d'un composant est accompagnée des tests qui permettront de s'assurer qu'il correspond à la description.

71



- les flèches continues reflètent l'**enchaînement séquentiel** des étapes du modèle de la cascade.
- Les flèches discontinues représentent le transfert d'une partie des résultats de l'étape de départ vers une étape d'arrivée pour être utilisée directement.
- Exemple: à l'issue de la conception architecturale, le plan d'intégration et les jeux de test d'intégration doivent être complètement décrits

72

Modèle en V

Tâches effectuées en parallèle

■ **horizontalement :** préparation de la vérification

- ▷ Ex. : dès que la spécification fonctionnelle est faite :
 - ▷ plan de tests de qualification
 - ▷ plan d'évaluation des performances
 - ▷ documentation utilisateur

■ **verticalement :** développement des modules

- ▷ Ex. : dès que la conception globale est validée :
 - ▷ conception détaillée des modules
 - ▷ programmation et tests unitaires

73

Modèle en V

■ **Modèles parfois difficiles à appliquer :**

- ▷ difficile de prendre en compte des changements importants dans les spécifications dans une phase avancée du projet
- ▷ durée parfois trop longue pour produits compétitifs

■ **Gestion du risque :**

- ▷ trop de choses reportées à l'étape de programmation (par ex. l'interface utilisateur)
- ▷ pas assez de résultats intermédiaires pour valider la version finale du produit

74

Amélioration

Retours d'expérience avant version finale

■ Dans l'industrie :

- ▷ prototype : premier d'une série
- ▷ maquette : modèle réduit (pas censé être opérationnel)

■ En développement de logiciel :

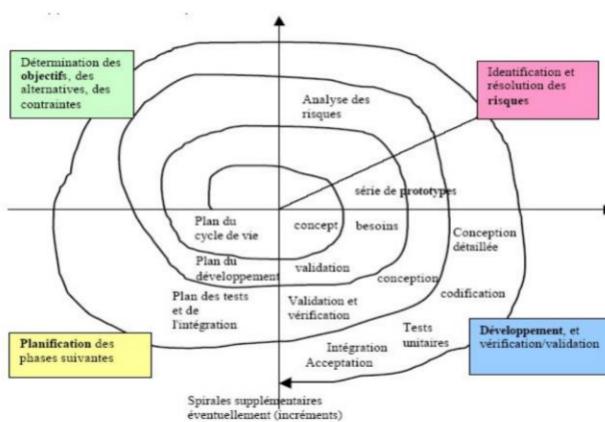
- ▷ prototype rapide (ou « maquette »), opérationnel (mais restreint), expérimental, évolutif

■ Développement incrémental

- ▷ ajouts successifs de nouvelles fonctionnalités sur une base opérationnelle

75

- Cycle de vie – Modèle en Spirale



76

Modèle en spirale

- Selon le schéma ci-dessous, chaque cycle de la spirale se déroule en quatre phases représentées par des quadrants :
- ▷ **1. Détermination des objectifs du cycle, des alternatives pour les atteindre, des contraintes**, à partir des résultats des cycles précédents ou s'il n'y a pas, d'une analyse préliminaire des besoins,
 - ▷ **2. Analyse des risques, évaluation des alternatives**, éventuellement maquettage,
 - ▷ **3. Développement et vérification de la solution retenue** (une portion de développement classique où on peut appliquer un des modèles précédents),
 - ▷ **4. Revue des résultats** et planification du cycle suivant.

77

Modèle en Spirale

- Lors de la phase de définition des besoins, plusieurs variantes du cahier des charges sont élaborées qui correspondent à des solutions satisfaisant différemment les contraintes imposées par les besoins du client.
- Dans la 2ème étape du cycle, chacune des solutions est ensuite expérimentée et analysée à l'aide du prototypage. Ceci permet d'en choisir le cahier des charges le plus adéquat et de le raffiner encore plus dans la troisième étape.
- Dans la 4ème étape, le plan de développement y afférent est élaboré en guise de préparation du cycle prochain portant sur la conception architecturale du système correspondant au cahier des charges retenu.

78

Modèle en Spirale

Principe :

- Identifier les risques, leur affecter une priorité
- Développer des prototypes pour réduire les risques, en commençant par le plus grand risque
- Utiliser un modèle en V ou en cascade pour implémenter chaque cycle de développement

Contrôler :

- si un cycle concernant un risque est achevé avec succès, évaluer le résultat du cycle, planifier le cycle suivant
- si le risque est non résolu, interrompre le projet

79

Modèle en Spirale

- Ce modèle s'applique aux systèmes novateurs et très complexes car:
 - ▷ son coût est très élevé,
 - ▷ la mise en œuvre de modèle demande des compétences et un effort important.
 - ▷ En plus, il s'agit d'un modèle moins expérimenté que

80

Modèle par incrément :

- Dans ce type de modèle, un seul sous ensemble des composants d'un système est développé à la fois :
 - ▷ un noyau est tout d'abord développé,
 - ▷ puis des incrément sont successivement développés et intégrés.
- Le développement de chaque incrément est un des processus classiques.

81

Avantages de modèle par incréments :

- Chaque développement est moins complexe,
- Les intégrations sont progressives,
- Il peut y avoir des livraisons et des mises en services après chaque intégration d'incrément.
- Il permet de bien lisser (répartir) dans le temps l'effort de développement et les effectifs par rapport à ce qu'on rencontre dans les modèles en cascade et en V

82

Risques liés au modèle par incrément :

■ Risques liés au modèle :

- ▷ Mauvais choix du noyau,
 - ▷ Mauvaise décomposition du système en incrément,
 - ▷ Mauvaise définition des interfaces entre incrément,
- Ces risques peuvent causer des erreurs lors de l'intégration des incréments.

■ Solutions :

- ▷ La spécification du noyau, des incrément et de leurs interactions doit être faite globalement au début du projet.
- ▷ Les incrément doivent être aussi indépendants que possible aussi bien fonctionnellement qu'au niveau des calendriers de

83

Modèles de Processus logiciel

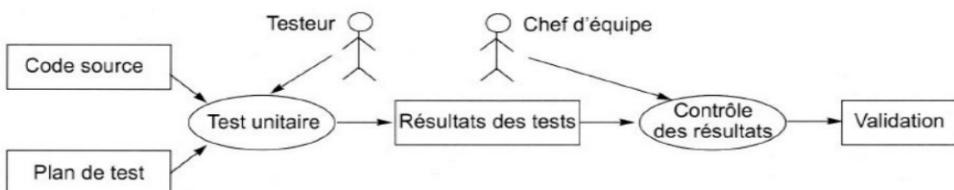
Modèles de Processus logiciel

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

85

Modèles de Processus logiciel

- Il doit y avoir un trajet depuis chaque tache jusqu'à la tache de fin Exemple d'un processus logiciel :



86

Modèles de Processus logiciel

Diagrammes de flots de données (data flow diagrams)

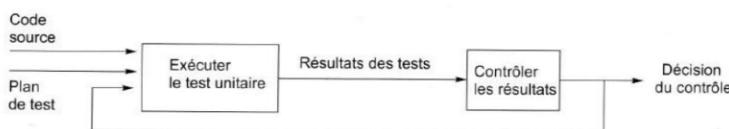
- utilisés pour la modélisation des traitements
- permettent de montrer comment chaque processus transforme ses entrées (flots de données entrants) en sorties correspondantes (flots de données sortants)
- intégrés dans diverses méthodes
- peut être complétée par les diagrammes de contextes (permettant de représenter le flux de donner avec les acteurs externes).
- Indique la circulation des données à travers un ensemble de composants qui peuvent être des tâches, des composants logiciels...

87

Diagrammes de flots de données (data flow diagrams)

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.



88

Références

- Pierrick Gérard. 2007, « Genie Logiciel Principes et Techniques », IUT de Villetteaneuse - Université de Paris 13.
- Renaud Marlet. 2007, « Cycle de vie », 2007, LaBRI / INRIA, <http://www.labri.fr/~marlet>