

Cycle de vie du logiciel et bonnes pratiques de développement



Sylvie Vignes

Objectifs de la présentation



- Présenter les cadres de développement du logiciel en milieu industriel
- En dégager 7 bonnes pratiques
- Illustrer ces bonnes pratiques dans le contexte d'un projet ENST

I - Définitions de base

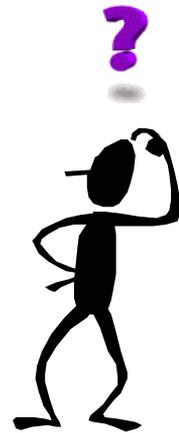
Définitions de base

Qu'est-ce-qu'un projet ?

- **Entreprise temporaire décidée pour obtenir un produit ou un service**
- Ensemble d'activités organisées permettant de créer un produit ou un service *unique avec une qualité définie dans le cadre d'un budget fixé*
- Effort temporaire ayant un début et une fin déterminés
- Un choix de facteurs de qualité

Notre projet...

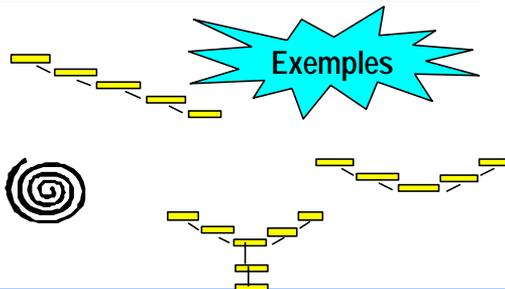
- La DFI de l'ENST souhaite mettre en place un site intranet permettant :
 - l'inscription des étudiants aux briques de leur choix
 - La consultation par les professeurs des élèves inscrits à leurs cours
 - La consultation de l'emploi du temps
- Budget : 12 personnes x mois
- Durée : 3 mois



Qu'est-ce qu'un cycle de vie ?

- Ensemble séquentiel de phases, dont le nom et le nombre sont déterminés en fonction des besoins du projet, permettant généralement le développement d'un service ou d'un produit

- Cycle en Cascade
- Cycle en V
- Cycle en Spirale
- Cycle en Y



Le cycle de vie en cascade

- Cycle de vie linéaire, séquentiel, dit «en cascade»
- Celui-ci a été défini dans les années 70
- Ce cycle de vie est basé sur la production d'éléments livrables
- Le cycle de vie «en V» est une alternative au cycle en cascade



Le cycle de vie en cascade

1 semaine



Recueillir les exigences

1 semaine

Analyser



3 semaines

Concevoir



Que se passe-t-il lorsqu'on découvre à ce stade des changements par rapport aux exigences ?

Et à ce stade ?

1 mois

Coder



Et à celui-ci ?

3 semaines



Intégrer, tester & effectuer le contrôle qualité

1. Interviewer les professeurs et les élèves
2. Rédiger des spécifications du site
3. Analyser le besoin
4. Identifier et formaliser une architecture
5. Coder
6. Intégrer, tester
7. Vérifier la qualité du produit
8. Livrer

Difficultés liées au cycle de vie en cascade (1)

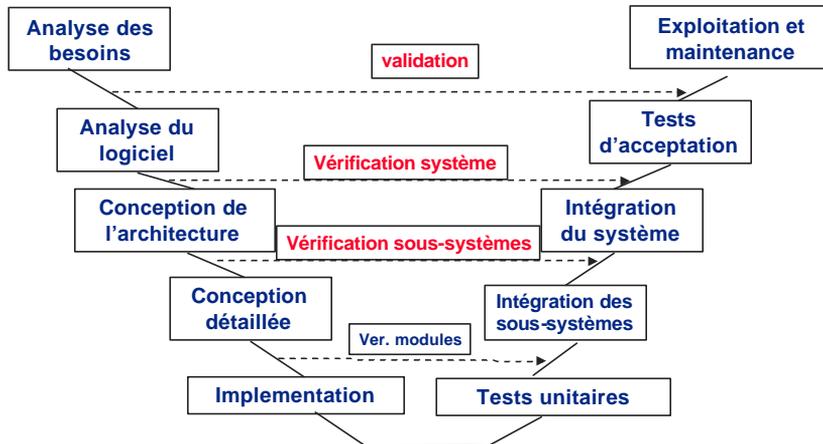
- **Suppose que l'on connaisse précisément les besoins (exigences), ou au moins la plupart, dès le début**
- **Refuse tout changement pour «tout bien faire dès le début»**
 - La formalisation exacte des exigences (spécification) doit précéder la conception, qui doit elle-même être finalisée avant de passer à l'implémentation.
- **Exige d'accorder une attention très importante aux documents**
 - Ex. : livrer un document, attendre 15 jours les retours, intégrer ces commentaires (10 jours) , livrer une nouvelle version,...



Difficultés liées au cycle de vie en cascade (2)

- **Retarde la résolution des facteurs de risque**
 - Par exemple, intégration tardive dans le cycle de vie
- **Entraîne une identification tardive de la conception, et un démarrage tardif du codage**
- **Entraîne des relations conflictuelles avec les parties prenantes en raison :**
 - Du manque de clarté de la définition des exigences
 - D'engagements importants dans un contexte de profonde incertitude
 - D'un désir inévitable de procéder à des changements

Le cycle de développement en V



V&V

Définitions (Boehm 76):

- Validation: Am I building the right system?
- Verification: Am I building the system right?

ISO 9000-3

Processus d'évaluation du logiciel pour s'assurer qu'il satisfait aux exigences spécifiées

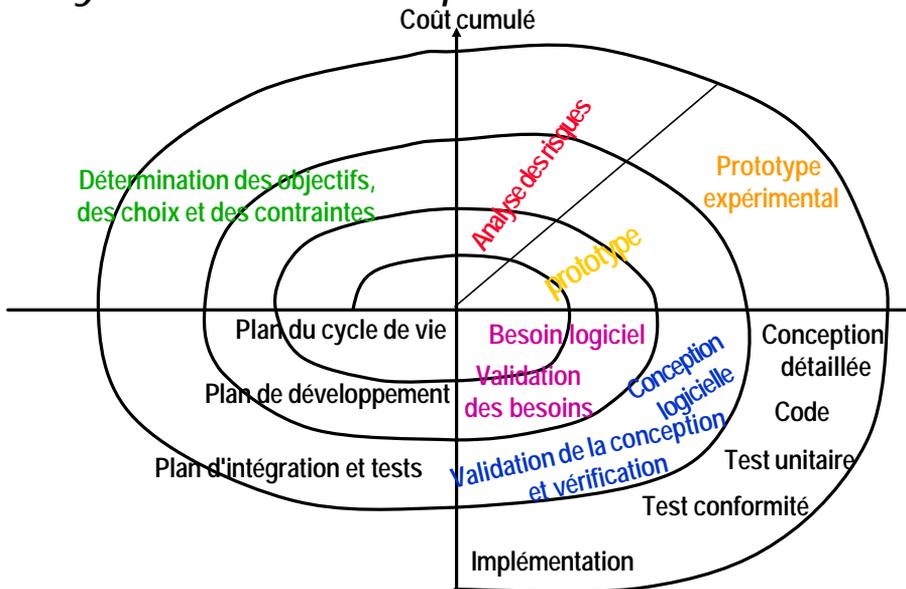
Cycle de vie en V : inconvénients

- hypothèses peu fondées : séquentialité des phases non conforme à la réalité
- incapacité en prendre en compte des évolutions du CdC pendant la construction du système
- absence de V&V à la fin de chaque étape
- absence d'une continuité des outils
- pas adapté aux systèmes non fonctionnels
- trop d'informel
- peu ou pas de possibilité de maquettage et/ou de prototypage.

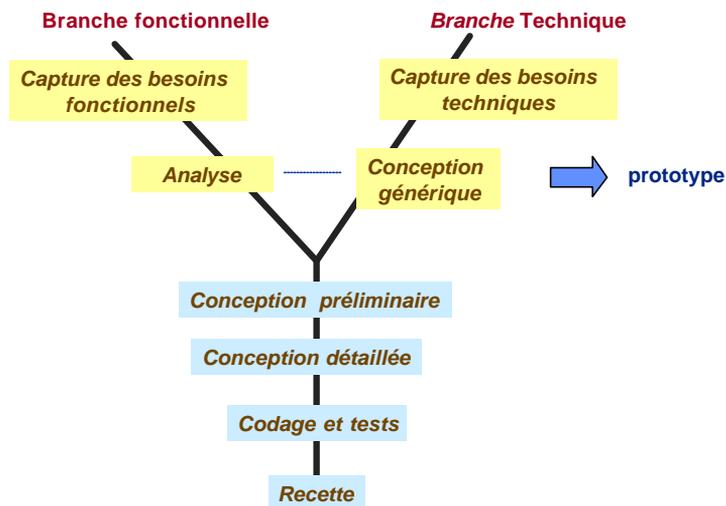
Evaluation du cycle de Vie en V : avantages

- modèle éprouvé car calqué sur la production industrielle classique
 - permet l'organisation du travail et des équipes
 - => prédiction (Cocomo) et contrôle des coûts facilités
 - favorise la décomposition hiérarchique fonctionnelle
 - propose des étapes clés (documentation, revues)
 - => bon suivi du projet
 - permet de garantir une certaine qualité (plan assurance qualité)
 - existence de standards:
MIL-STD-498, GAM-T17(V2), Do 178 B, STAN-CS 055, ESA, ...
 - adapté à de grands projets
- beaucoup d'outils support.

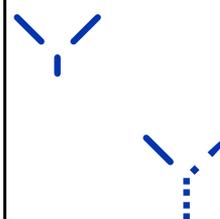
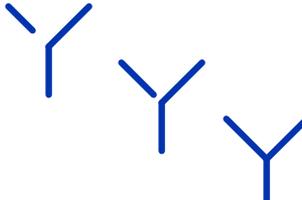
Cycle de vie en spirale



Cycle en Y



Un processus incrémental pour le cycle en Y

Préétude	Elaboration	Construction
		
-Validation du principe -outils de dvlp	-Focalisé sur l'architecture -Réalisation fcts prioritaires	Avancement jusqu'au système complet
Incrément 1	Inc.2 Inc. 3	Inc.4 Inc.5 Inc.6

temps

V1.0

17

Principes de base

Le processus « idéal » pour le développement de logiciel

■ doit permettre de :

- Bien comprendre les demandes des utilisateurs finals
- Tenir compte des changements du cahier des charges
- Empêcher la découverte tardive de défauts sérieux dans le projet
- Traiter au plus tôt tous les points critiques du projet
- Bien communiquer avec le client
- Bien maîtriser la complexité
- Favoriser la réutilisation
- Définir une architecture robuste
- Faciliter le travail en équipe
- ...

V1.0

18

Définition de la fiabilité

- De façon opérationnelle, on parle de **Sûreté de fonctionnement**

Propriété d'un système informatique permettant à ses utilisateurs de placer une confiance justifiée dans le service qu'il délivre

Déclinaisons de la notion de fiabilité

Terme Français	Terme Anglais	Description
Disponibilité	Availability	Capacité à être prêt à délivrer le service
Fiabilité	Reliability	Capacité à maintenir la continuité de service
Maintenabilité	Maintenability	Aptitude aux réparations et évolutions
Sécurité-innocuité	Safety	Absence de défaillances catastrophiques
Confidentialité	Confidentiality	Absence de divulgation non autorisée
Intégrité	Integrity	Pas de détérioration (matériel ou logiciel)
Sécurité-confidentialité	Security	Confidentialité + sécurité + disponibilité

II - Maturité et normes de développement

Des méthodes d'évaluation et d'évolution des organisations ... (Des cadres de « management » normalisés)

■ CMM (Capability Maturity Model)

- Mis au point par Software Engineering Institute
- Standard; version française sur <http://www.CRIM.ca>

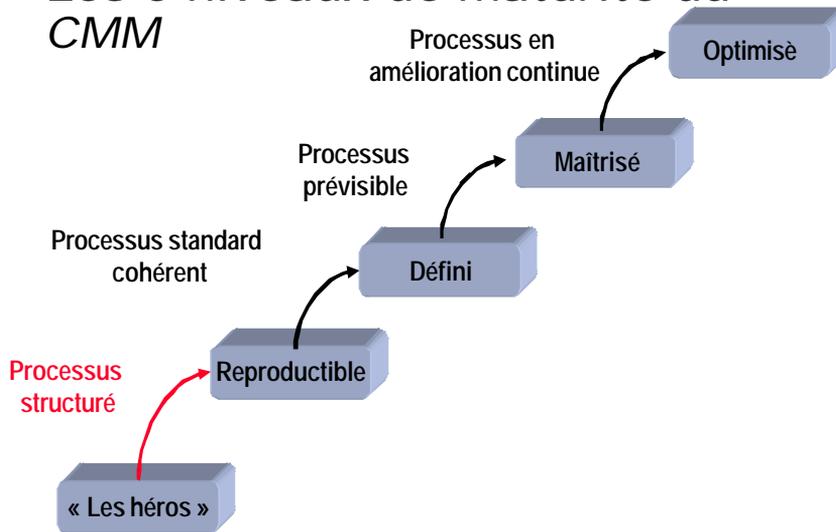
Standard

■ SPICE

- (Software Process Improvement Capability dEtermination)
- Norme Internationale qui évolue parallèlement à la norme ISO 9000

ISO 15504

Les 5 niveaux de maturité du CMM



Niveau « 2 » reproductible : secteurs clés

- Gestion des exigences
- Planification de projet
- Suivi et supervision de projet
- Gestion de la sous-traitance
- Assurance Qualité
- Gestion de la configuration

Consensus dans l'entreprise sur la manière de faire mais pas de formalisation

Gestion rigoureuse des coûts et des délais mais repose sur des compétences individuelles

Niveau « 3 » Défini : secteurs clés

- Focalisation **organisationnelle**
- Définition du Processus
- Programme de formation
- Coordination intergroupes
- Revue par des pairs
- ...

*Processus de développement formalisé et documenté
Service de définition et de suivi des méthodes de l'entreprise*

Niveau « 4 » Maîtrisé : secteurs clés

- Gestion de la qualité logicielle
- Gestion quantitative de processus

*Processus formel de collecte d'informations pour mesurer
le processus d'élaboration de systèmes ainsi que les produits résultants*

Niveau « 5 » Optimisé : secteurs clés

- Gestion des changements technologiques,
- Prévention des défauts
- ...

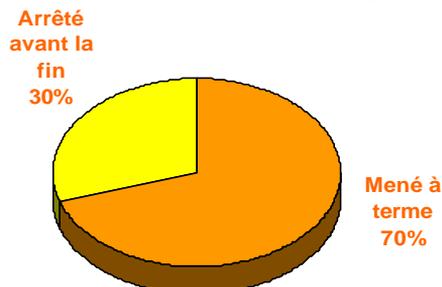
Utilisation des résultats de la métrologie pour améliorer les méthodes

III - Les 7 bonnes pratiques du développement logiciel

Les bonnes pratiques du développement logiciel

Le Logiciel—Un Métier à Risque

- 53% des projets coûtent au moins 200% des estimations initiales.
- On estime à 81 billion de dollars la somme dépensée en 1995 au U.S.A. sur des projets arrêtés avant la fin.



Source: Rapport Standish, 1995

Symptômes Courants d'Echec des Projets (1)

**Symptômes les plus fréquents des
projets ayant échoués :**

- 👉 Incapacité de gérer les modifications des exigences.
- 👉 Mauvaise compréhension des besoins des utilisateurs finals.
- 👉 Les modules qui ne fonctionnent pas ensemble.
- 👉 Du logiciel difficile à maintenir ou à faire évoluer.
- 👉 Du logiciel de mauvaise qualité (beaucoup d'anomalies).

Symptômes Courants d'Echec des Projets (2)

- 👉 Construction d'un processus non fiable.
- 👉 Membres de l'équipe isolés, incapables de déterminer qui a changé quoi, quand, où et pourquoi.
- 👉 Procédures de test coûteuses.
- 👉 Découverte tardive de problèmes.

7 bonnes pratiques (1)

- Meilleures pratiques reconnues
- L'utilisation de ces pratiques maximise les chances de réussite du projet
- Le non respect de ces pratiques introduit un maillon faible



7 bonnes pratiques (2)

- 1 Développement de manière itérative 
- 2 Développement à base de composants centré sur l'architecture 
- 3 Pilotage par les risques 
- 4 Gestion des exigences 
- 5 Maîtrise des modifications 
- 6 Evaluation continue de la qualité 
- 7 Modélisation visuelle 

Ce sont celles préconisées dans le Processus Unifié

IV - Le Processus Unifié

Origines du Processus Unifié

1998 : Apparition de UP ...



- ... Pas de nouvelles idées mais un ensemble de bonnes pratiques largement répandues dans les processus modernes
- Adoption rapide comme standard de fait en Europe et en Amérique du nord
 - IBM, Chase-Manhattan, Alcatel, MCI, British Aerospace, Volvo, Intel, Merrill, E&Y, Deloitte, Ericsson, Cartier International, Valtech . . .

Qu'est-ce que le Unified Process ?

- C'est un processus de développement de logiciel
- Il permet de définir et d'affecter des tâches et des responsabilités au sein d'une organisation de développement
- Son but est d'assurer la production d'un logiciel de grande qualité, satisfaisant les demandes des utilisateurs finals dans les délais et avec un budget prévisibles

IV.1 -Le Processus Unifié

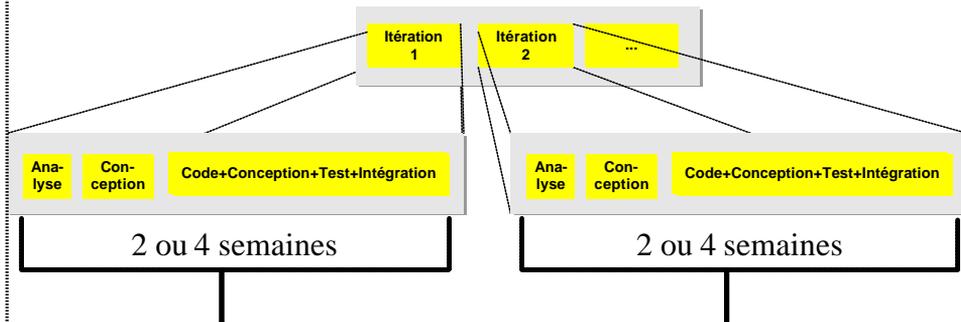
*Bonne pratique :
Développement de manière
itérative*



Qu'est-ce que le Développement Itératif ?



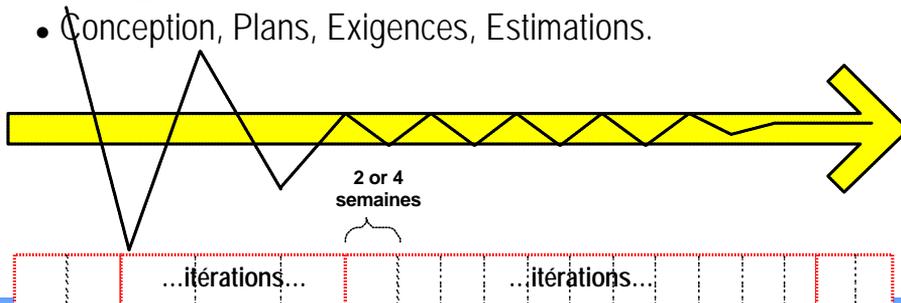
- Basé sur de petites étapes, le feedback et l'adaptation.
- Aussi appelé évolutif, en spirale, ...



Feedback et Adaptation



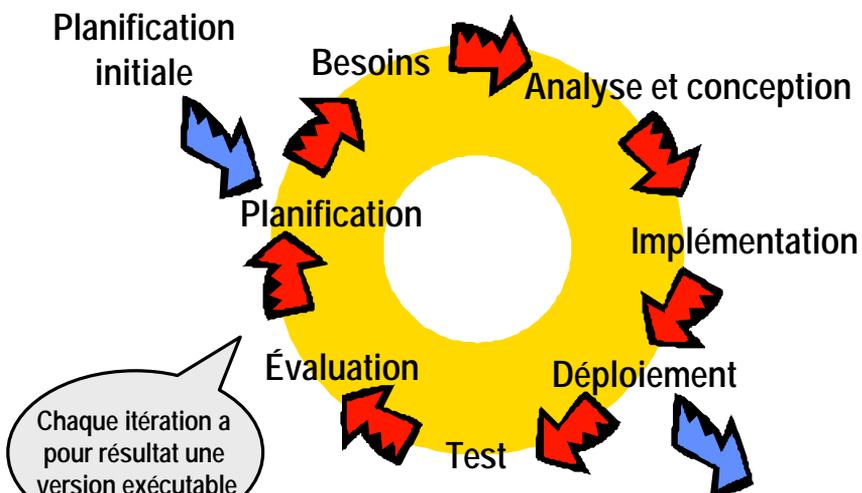
- Le *feedback* continu est un élément clé du succès.
 - De part le retour des utilisateurs, des tests, ...
- A chaque itération, on s'*adapte* en se basant sur le feedback et les leçons de la dernière itération, et on converge doucement vers de meilleurs :
 - Conception, Plans, Exigences, Estimations.



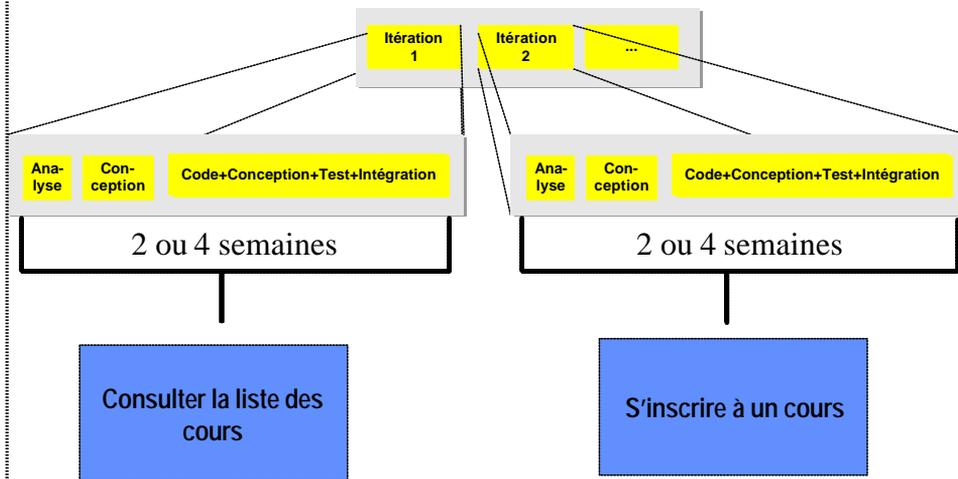


■ **Une itération est une séquence définie d'activités**

- qui se déroule selon un plan établi et se termine par une livraison (interne ou externe), et pour laquelle on a défini des critères d'évaluation



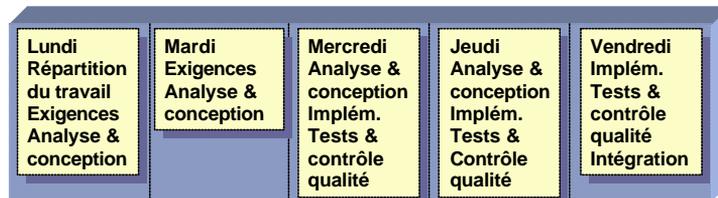
Exemple pour le site de l'ENST



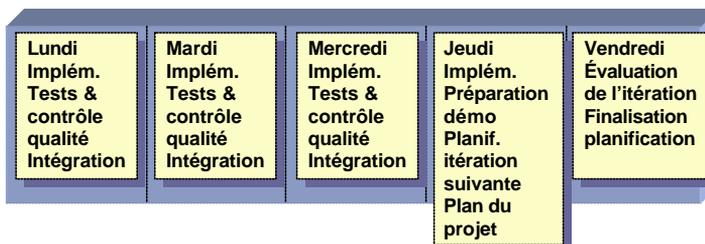
Exemple d'une itération de deux semaines pour le site de l'ENST



Semaine 1



Semaine 2

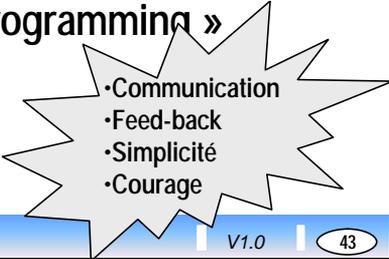


Pratique inspirée les méthodes « agiles »

- Basées sur des itérations très courtes, des incréments petits
- Beaucoup de tests
<http://www.xprogramming.com/>
(des environnements de tests à télécharger)
- Requièrent l'utilisateur quasiment en permanence
- La plus connue « Extreme Programming »

<http://www.xp-france.org/>

<http://agilealliance.org/>



- Communication
- Feed-back
- Simplicité
- Courage

IV-2 Le Processus Unifié

*Bonne pratique :
Développement à base de
composants centré sur
l'architecture*





- **Au cours des premières itérations, on construit et on valide une architecture logicielle**
 - A partir de composants existants, standards du marché
 - Éviter les développements spécifiques
- **Construire l'architecture et la tester tôt même si la solution n'est pas parfaite et incomplète**

Des Composants?



- **Des unités logicielles "boîte-noire" avec une API**
- **Fonctionnent généralement dans une (locale ou distante) architecture à base de composants:**
 - EJB, COM, .NET, JavaBeans, Servlet
- **Le UP encourage la bonne pratique suivante:**
 - Acquérir des composants existants afin d'accélérer le développement
 - Etablir une culture d'acquisition ou d'achat de composants, plutôt que de développer toutes les parties du logiciel.

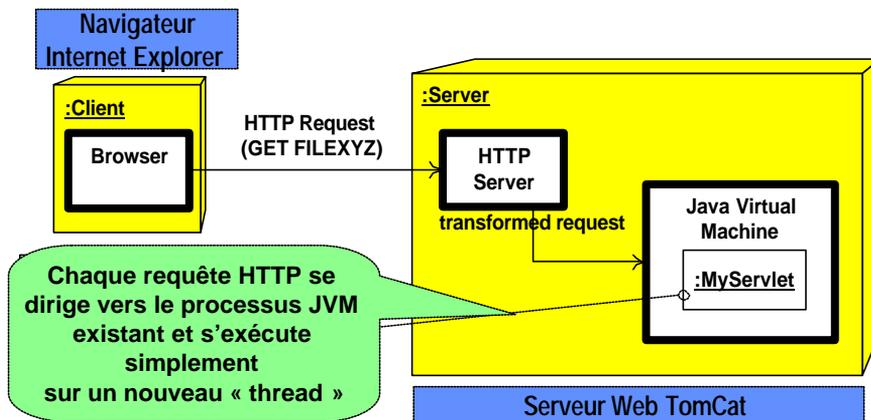
Avantage des Architectures à Base de Composants



- Les composants favorisent les architectures résistantes.
- La modularité permet d'avoir une séparation claire entre les éléments d'un système.
- La réutilisation est facilitée par l'utilisation de composants commerciaux.



Exemple d'une architecture pour le site de l'ENST



IV-3 Le Processus Unifié

*Bonne pratique :
Pilotage par les risques*



Pilotage par les risques

Qu'est ce qu'un risque ?



■ **Un risque est un événement redouté dont l'occurrence est plus ou moins prévisible et provoquant, lorsqu'il se produit, des dommages sur le projet.**

■ **Il ne faut pas confondre risque et problème.**

- Un problème est un risque qui s'est révélé.





- **Le pilotage par les risques, c'est :**
 - Analyser les risques potentiels le plus tôt possible – dès les premières itérations
 - Les hiérarchiser
 - Commencer par travailler sur les éléments les plus exposés
 - Par exemple :
 - *L'intégration, l'architecture*
 - *La gestion des ressources humaines nécessite une attention particulière : Besoins ? Profils ? Organisation de l'équipe projet ? Communication et management de l'équipe*



- **Penser aux risques techniques, mais aussi :**
 - aux risques liés au client
 - aux risques liés au domaine applicatif
 - aux risques liés à l'organisation du projet

Exemple de risques pour le site de l'ENST



- **Risques liés au client :**
 - Convergence du besoin
 - *Cause : de nombreux utilisateurs : direction, professeurs, étudiants*
- **Risques techniques :**
 - Maîtrise du serveur web
 - *Cause : première utilisation du serveur tomcat*
 - Pic d'accès au site au mois de septembre
 - *Cause : rentrée des étudiants simultanée*
- **Risques liés à l'organisation du projet :**
 - Disponibilité de l'équipe
 - *Cause : le projet se déroule durant la période des examens*

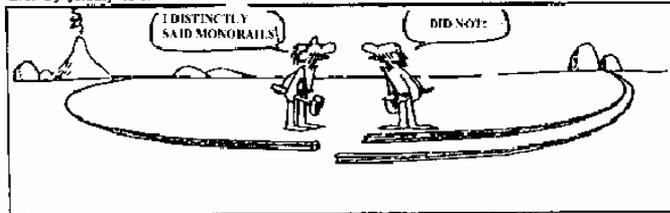
Le Processus Unifié

***IV-4 Bonne pratique :
La gestion des exigences***



Why Do We Need Requirements Management?

B.C. By Johnny Hart



Facteurs de dépassement et d'abandon



- 1) Manque de participation des utilisateurs
- 2) Identification incomplète des Besoins
- 3) Besoins qui changent au cours du projet

→ **Gérer les exigences**

Standish Group, '97

V1.0

55



Identification des exigences

■ Qu'est ce qu'une exigence ?

- Condition à laquelle le système doit satisfaire ou une capacité dont il doit faire preuve.

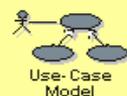
[RUP]

■ On distingue :

- Les exigences fonctionnelles
 - *Qui formulent ce que le système est chargé de faire*
- Les exigences non fonctionnelles
 - *Décrivent la qualité des services attendus du système (performance, sécurité de fonctionnement, IHM)...*

■ Le UP recommande de se servir des Cas d'Utilisation

UML pour identifier les exigences fonctionnelles



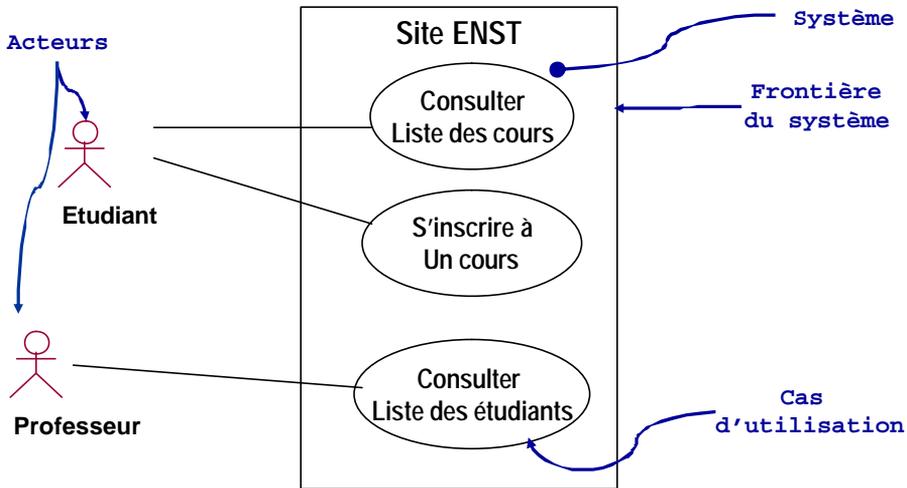
Use-Case Model

V1.0

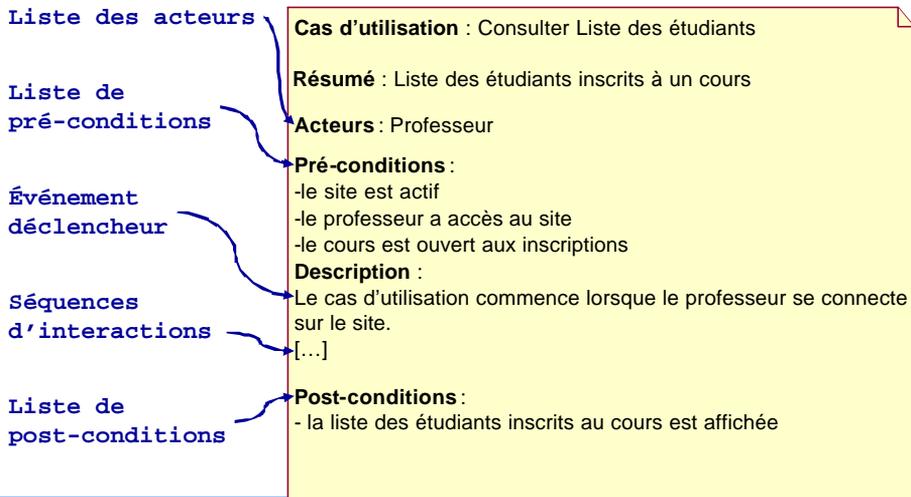
56



Exemple d'exigences pour le site de l'ENST (1)



Exemple d'exigences pour le site de l'ENST (2)



Exemple d'exigences pour le site de l'ENST (3)



■ Un cas d'utilisation spécifique :

- Un enchaînement "nominal"
- Des enchaînements alternatifs
- Des enchaînements d'erreur et d'exception

Enchaînement nominal :

Lorsqu'un enchaînement nominal ou alternatif est exécuté, les post-conditions sont atteintes.

1. Enchaînement nominal :

- a. Le cas d'utilisation commence lorsque le professeur se connecte sur le site
- b. Le site demande un identifiant et un mot de passe
- c. Le professeur saisit son identifiant et son mot de passe
- d. Le site vérifie l'identifiant et le mot de passe
- e. Le site demande le code du cours à consulter
- f. Le professeur saisit le code
- g. Le site vérifie que le code du cours existe et qu'il est ouvert aux inscriptions
- h. La liste des étudiants inscrits au cours est affichée à l'écran

Exemple d'exigences pour le site de l'ENST (4)



Enchaînement d'erreur :

Enchaînement d'exception :

Lorsqu'un enchaînement d'exception est exécuté, les post-conditions ne sont pas atteintes.

Lorsqu'un enchaînement d'erreur est exécuté, les post-conditions sont atteintes.

2. Enchaînement d'erreur : identification incorrect

L'enchaînement démarre au point 1-d. de l'enchaînement nominal
a. Le site indique au professeur que l'identifiant et/ou le mot de passe sont erronés
L'enchaînement nominal reprend au point 1-b.

3. Enchaînement d'exception : le professeur a saisi 3 fois un identifiant ou un mot de passe erroné :

- a. La connexion au site est coupée

4. Enchaînement d'erreur : le cours n'existe pas

L'enchaînement démarre au point 1-g. de l'enchaînement nominal
a. Le site indique que le cours correspondant au code n'existe pas
L'enchaînement nominal reprend au point 1-e.

Comment Perdre un Client ...



1. Votre équipe a l'habitude de toujours essayer de satisfaire le client (cela semble logique).
2. Pendant le développement, un professeur vient voir un développeur et lui dit, "Paul, j'aime ce que je vois mais je souhaiterais également connaître la liste des étudiants inscrits à plusieurs de mes cours. Pensez-vous que vous pouvez le rajouter ?"
3. Paul: "Bien sûr! Pas de problème! Je m'en rappellerai."

Comment Perdre un Client ...



Mauvaise réponse!

Pourquoi?



La Gestion des Exigences



- **Cela ne signifie pas:**
 - Avoir des exigences correctes dès le démarrage du projet.

- **C'est une pensée irréaliste du cycle "en cascade".**

- **Cela signifie:**
 - Ne pas être négligent.
 - Les recueillir efficacement.
 - Enregistrer, tracer, organiser (sûrement avec un outil).
 - Et cela se rapporte au fait de considérer les changements de manière formelle (maîtriser les changements).

Le Processus Unifié

*IV-5 Bonne pratique :
Maîtrise des modifications*





- **Multiplication du nombre de versions**
 - Liée au nombre d'itérations
 - Liée à l'évolution des besoins dans le temps
- **Nécessité de paralléliser les développements**
 - Pour ne pas retarder une équipe
 - Pour répondre à un besoin ponctuel du client
- **Négocier les évolutions et les tracer ; enregistrer, valider, gérer la configuration et les versions**
- **Documenter le projet**
- **Communiquer les changements**
- **Prévenir les conflits**

→ **Gérer les modifications**



Qu'est-ce qu'une demande de changement ?

- **Demande de changement (Change Request ou CR)**
 - Requête pour modifier un artefact ou un processus. Dans la documentation d'une demande de changement figurent des informations sur l'origine et l'impact du problème considéré, sur la solution proposée et sur son coût.
- **Deux types**
 - Demande d'Evolution
 - *Spécifie une nouvelle caractéristique du système ou un changement par rapport au comportement établi.*
 - Rapport d'Anomalie
 - *Erreur ou défaut*

Support type

■ Utilisation de fiches de demande de changement

- Type
 - *Demande d'évolution*
 Connaître la liste des étudiants inscrits à plusieurs cours d'un même professeur
 - *Rapport d'anomalie*
 Le nombre de tentatives pour se connecter au site est égal à 3
- Status
- Informations complémentaires

Change Request	
Identification	
Id	MATP_ --
Build	v0.1
Headline	
Defect type	[Backlog, Correctio]
Severity	[Critical, Major, Minor]
Priority	[High, Standard, Low]
Submit date	
Submitter (> Contact)	
Change request description	
Description :	
Change Control Board	
Action	[Open, Assign, Close, Reopen, Reassign, Validate, Postpone]
State	[Submitted, Assigned, Opened, Resolved, Closed, Rejected, Postponed]
Author	
Resolution and Validation	
Estimation (job of days)	
Owner	
Fixed In build	
Resolution	[Slide modified, Lab modified, Slide deleted, Slide added, Lab added, Agent modified]
Slide Ref. :	
Lab Ref. :	
Responsible for validation	
Validation date	



Le Processus Unifié

*IV-6 Bonne pratique :
 Evaluation continue de la qualité*





Faire les Tests & AQ à la fin ?

■ Il est démontré que :

- Corriger une anomalie plus tard coûte 10-100 fois plus que de la corriger à son origine.
Software Engineering Economics, Boehm, 1981.
- Les produits avec le moins d'anomalies ont les délais les plus courts.
 - *Applied Software Measurement, 1st edition, Jones 1991.*
- La mauvaise qualité est la raison la plus courante de dépassement des délais.
 - *Assessment and Control of Software Risks, Jones 1994, 4000 project study.*
- La correction des anomalies consomme 40-50% du coût total.
 - *IEEE Computer, Boehm, Sept 1987.*
- 60% des anomalies existent au moment de la conception.
 - *Principles of Software Engineering Management, Gilb, 1988.*



Evaluation continue de la Qualité

■ Solution construite – contrôle produit

- Vérification de l'adéquation de la solution aux besoins
- Tests systématiques et périodiques
- Actions qualité

■ Process (UP) – évaluation interne

- Respect des bonnes pratiques du Processus Unifié

■ Avantages :

- Identification précoce des dysfonctionnements
- Réactivité aux déviations constatées
- Maîtrise des risques de dérapage



- Vous avez passé 3 mois à construire le site de l'ENST. Maintenant que vous êtes prêts à sortir le produit...
- Alors, vous demandez à vos amis :
 - "A 11 heures demain matin, tout le monde se connecte et essaye de s'inscrire à un cours ainsi on pourra vérifier si le système résiste à la charge prévue."
- Est-ce le moment de vérifier cette propriété significative de l'architecture?

Le Processus Unifié

*IV- 7 Bonne pratique :
La modélisation visuelle*





Pourquoi Utiliser la Modélisation Visuelle?

- On doit enregistrer nos pensées et communiquer en utilisant des langages visuels et schématiques (par ex., UML).

- Parce que :
 - On estime qu'au moins 50% de notre cerveau est impliqué dans le processus visuel.

 - Les langages visuels sont naturels et faciles pour notre cerveau.



La Bonne Quantité de Modélisation Visuelle

- Se situe entre trop et, évidemment, pas assez de modélisation visuelle.

- Des schémas, spécialement sur un tableau blanc, sont plus rapides que d'écrire ou de changer du code.

- Ils permettent de rapides recherches et la modification des grandes lignes du système en ignorant les détails que la programmation nous obligerait à prendre en compte.

Où UML est-il utilisé dans le UP?

- Par exemple. . .

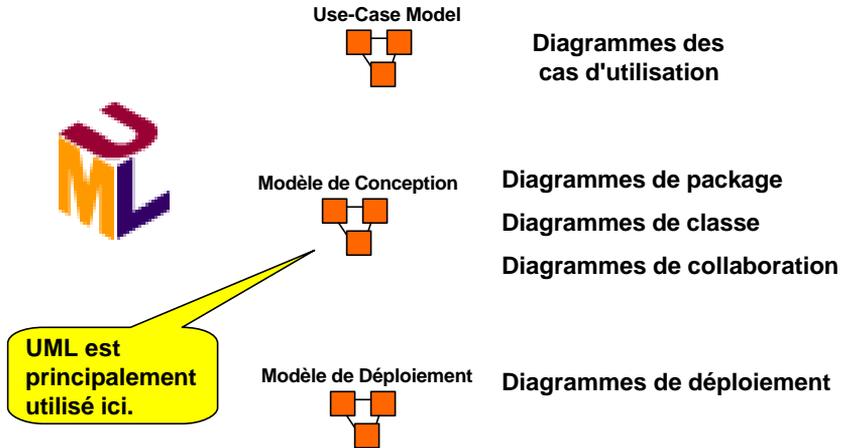
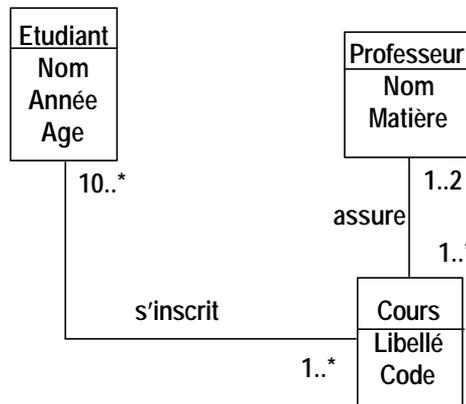
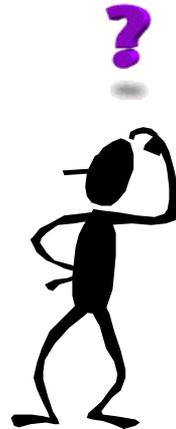


Diagramme de classe pour le site ENST



Les 7 bonnes pratiques du UP ?

- Sans regarder vos notes
- Retrouvez les 7 bonnes pratiques de UP



Conclusion

Conclusion

■ 7 bonnes pratiques pour le développement de logiciel :

- ✓ Développement à base de composants centré sur l'architecture
- ✓ Pilotage par les risques
- ✓ Gestion des exigences
- ✓ Maîtrise des modifications
- ✓ Evaluation continue de la qualité
- ✓ Modélisation visuelle

...mais c'est surtout...

- Décomposer le développement en courtes itérations



Pour surmonter les difficultés : itérer !!

Pour ...

- Bien comprendre les demandes des utilisateurs finals
- Tenir compte des changements du cahier des charges
- Empêcher la découverte tardive de défauts sérieux dans le projet
- Traiter au plus tôt tous les points critiques du projet
- Bien communiquer avec le client

- Bien maîtriser la complexité
- Favoriser la réutilisation
- Définir une architecture robuste

- Faciliter le travail en équipe



Bibliographie

- Livres
 - *Extreme Programming Explained – Embrace Change* – Kent Beck – Addison Wesley, 2000
 - *Software Project Management - A Unified Framework*, Walker Royce, Addison-Wesley, 1998
 - *Rational Unified Process - An Introduction*, Philippe Kruchten, Addison-Wesley, 1999
 - *Object Solutions - Managing the Object-Oriented Project*, Grady Booch, Addison-Wesley, 1996

