Génie Logiciel

Une introduction

Nicolas Delanoue

Université d'Angers - Polytech Angers

- 4A : Génie logiciel I - Généralités et terminologie
  - Programmation orientée objet (rappel concepts fondamentaux)
  - Modélisation avec UML,
  - Gestion des versions avec l'outil GIT,
  - tests et tests unitaires,
- 5A : Génie logiciel II, Focalisation sur la conception :
  - Bonnes pratiques,
  - patrons de conception architecturaux,
  - patrons de conception du GoF.

Définition

Le *génie logiciel* est la science qui étudie les méthodes de travail et les bonnes pratiques des ingénieurs qui développent des logiciels.

Remarques

- On parle aussi de l'ingénierie logicielle ou l'ingénierie du logiciel (en anglais : software engineering)
- Une discipline vaste : aspects techniques mais aussi organisationnels, qualité, financiers,...

Pourquoi est-ce si important ?

Aujourd'hui, le logiciel contrôle le monde !

- Processus métiers (administrative etc.)
- Gouvernement
- L'industrie (Usines, chaînes de fabrication)
- Transports
- Défense, finance, santé...
- Édition, médias...
- Les nouvelles technologies du web, commerce électronique...
- Et bien plus ! !

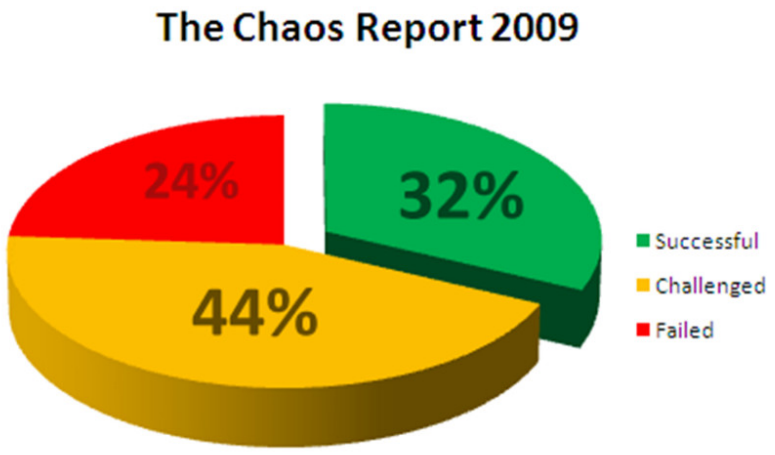
Des enjeux aussi bien économiques que politiques.

Les conséquences en cas de problèmes peuvent être très lourdes !

- Système de Bagages Aéroport Denver, 1995 : 16mois, 3 M$
- Ariane 5 vol 88/501, 1996 : 40s de vol, destruction, 850 M$
- Bug de la division du Pentium Intel, 1994 : image desastreuse, 475 M$,
- Mars Climate Orbiter & Mars Polar Lander, 1999 : destruction.

Talon d'Achille du Génie Logiciel

- le coût,
- le temps,
- la qualité.



- Successful means on-time, on-budget, and with all features and functions as defined in the initial scope ;
- challenged means late, over budget, and/or with less features and functions than defined in the initial scope ;
- failed means cancelled prior to completion, or delivered but never used.

Les difficultés liées à la nature du logiciel

- un logiciel ne s'use pas, sa fiabilité ne dépend que de sa conception
- mais, pour rester utilisé un logiciel doit évoluer
- pas de direction clairement exprimée,
- changements fréquents,
- contradictions des besoins,...

Difficultés liées aux personnes

- ne savent pas toujours ce qu'elles veulent, ou ne savent pas bien l'exprimer
- communication difficile entre personnes de métiers différents (jargons)
- l'informaticien est souvent perçu comme introverti, peu solidaire du groupe (...ça change...)
- beaucoup d'autodidactes qui croient savoir...

Les difficultés technologiques

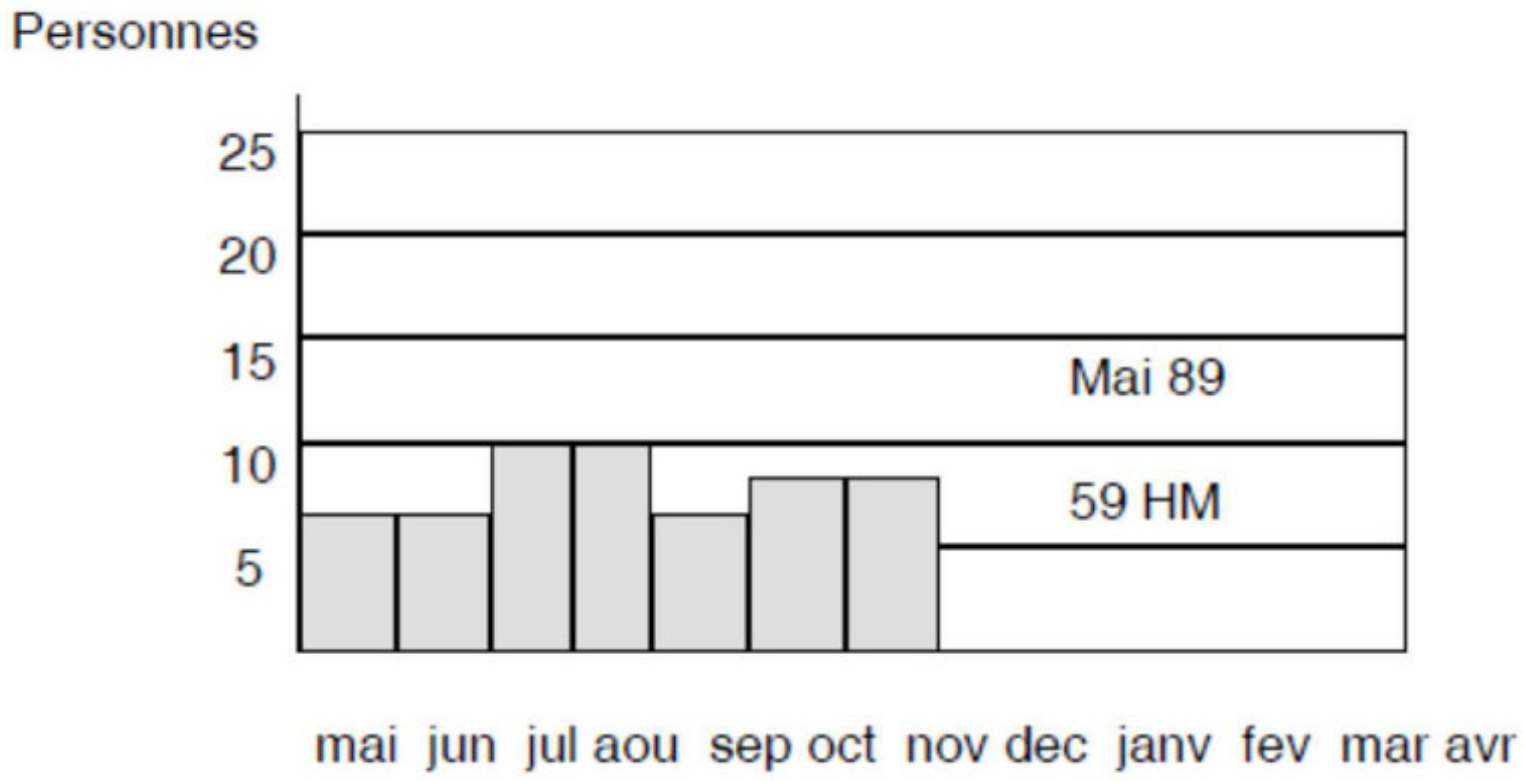
- courte durée de vie du matériel,
- beaucoup de méthodes, de langages
- évolution des outils de développement,...

L'échec d'un projet informatique

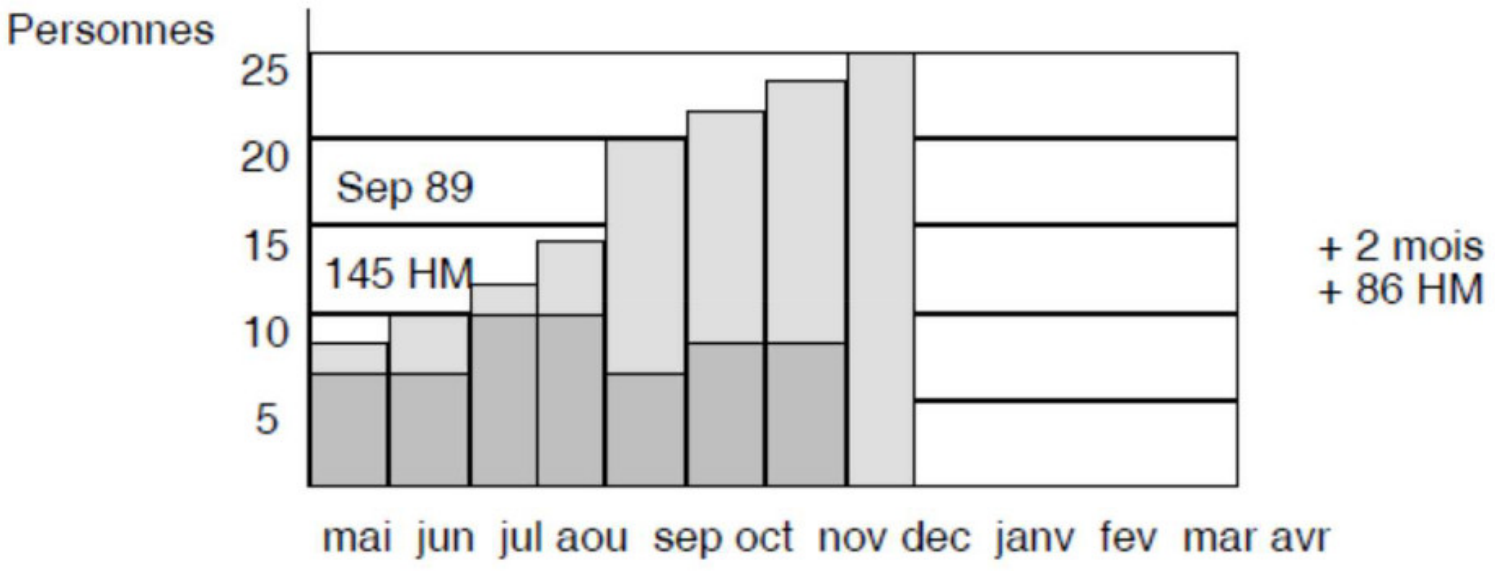
Cinq raisons majeures

- Engagements irréalistes
- Gestion et conduite de projet inadéquates
- Manque de contrôle
  - pas de planification de la part des développeurs
  - connaissances insuffisantes en gestion de projet
- Technologies inappropriées (méthodes, outils, langages)
- Validation et vérification insuffisantes

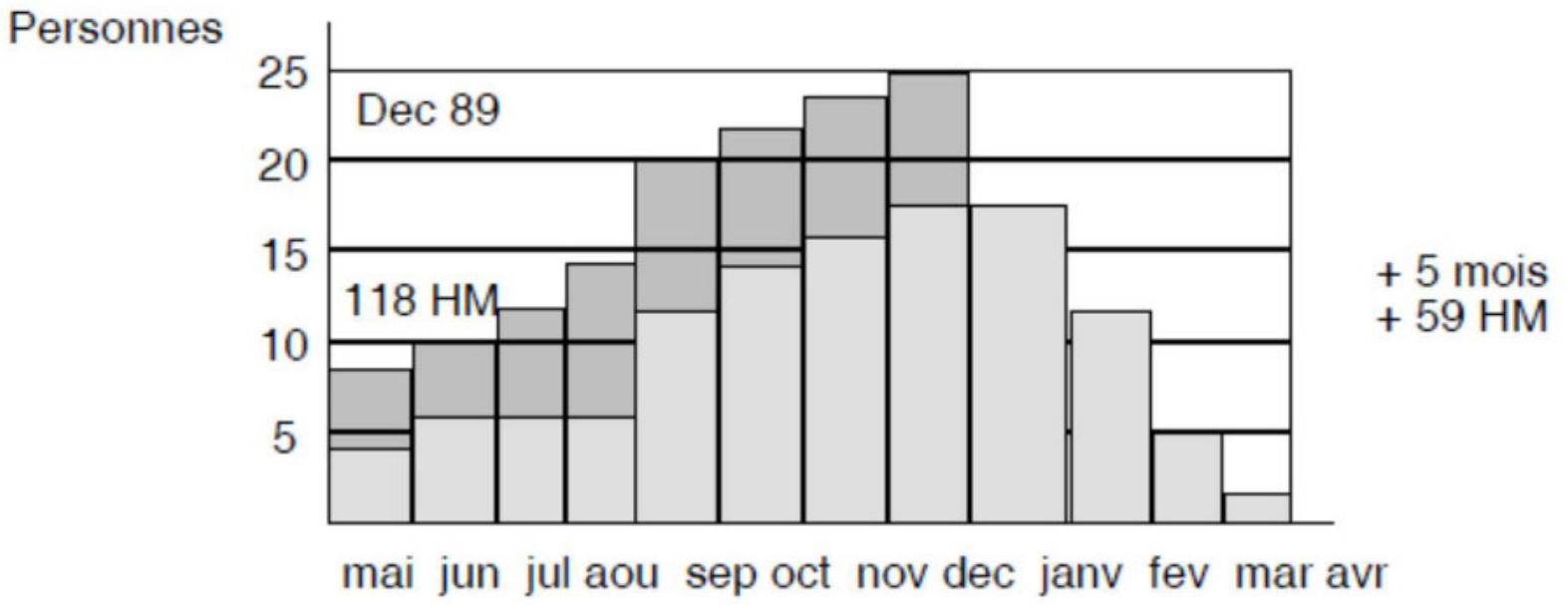
Charge prévisionnelle



Après 3 mois

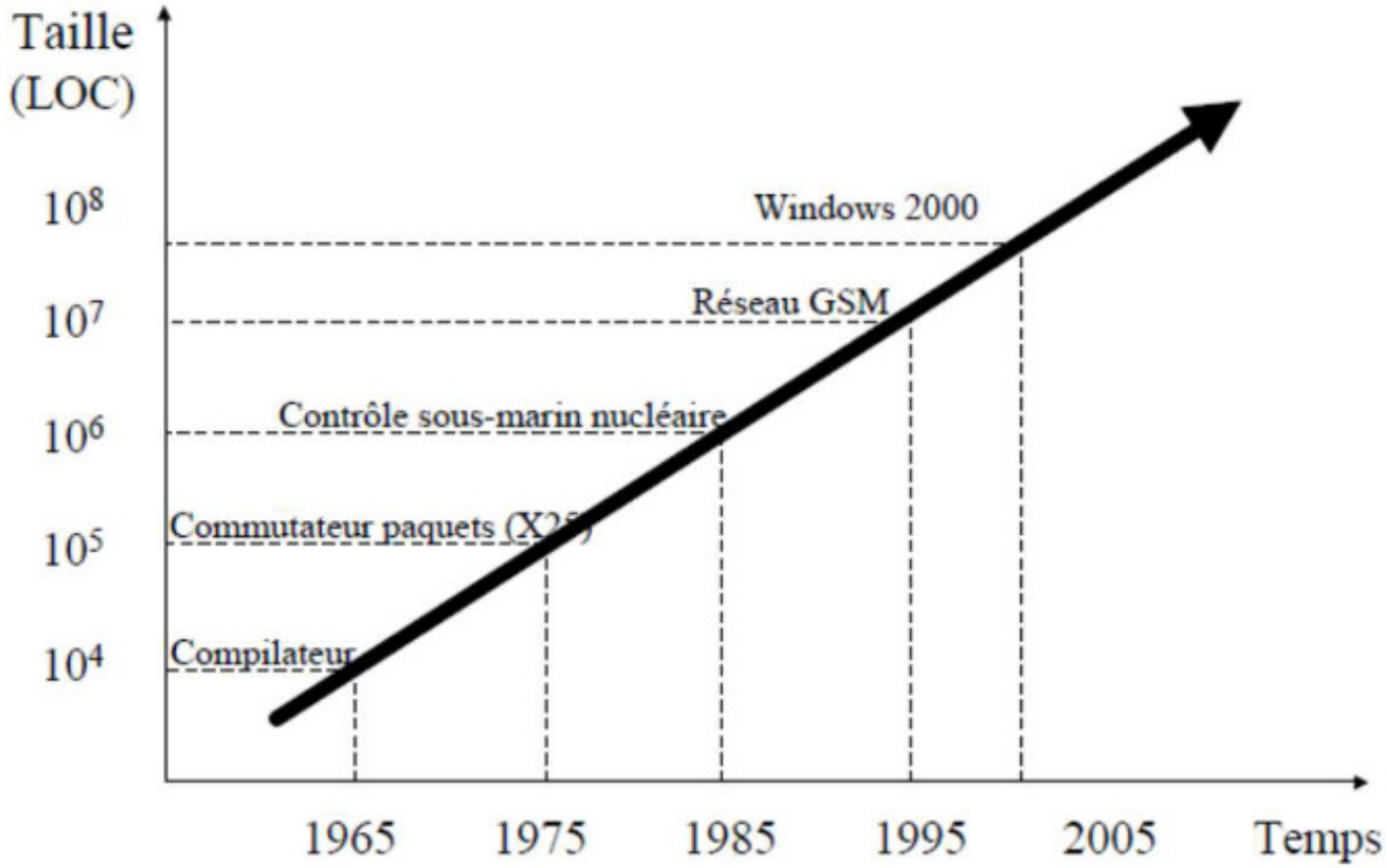

Personnes
25
20
15
10
5
Sep 89
145 HM
+ 2 mois
+ 86 HM
mai jun jul aou sep oct nov dec janv fev mar avr

Encore 3 mois plus tard...



- Aucune information sur ce qui est réalisé . . .
- La confiance diminue . . .
- Audit . . .

Applications de plus en plus larges/distribuées - Le nombre de lignes de code ne cesse d'augmenter

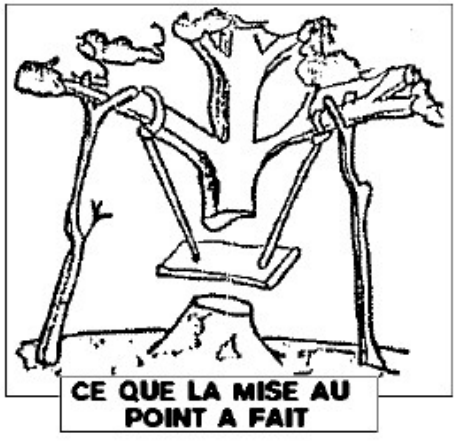


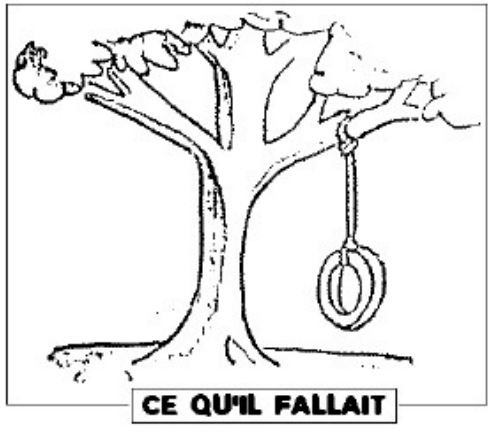

CE QUE DEMANDE LE CLIENT


CE QUE L'ANALYSTE A PRÉVU


CE QUE PRÉVOIT LE CONTRAT


CE QUE LE PROGRAMMEUR A ÉCRIT


CE QUE LA MISE AU POINT A FAIT


CE QU'IL FALLAIT

Quelques solutions que nous allons aborder dans ce cours :

- Formaliser les activités et le rôle des acteurs,
- Gérer la complexité avec la modélisation (UML),
- Améliorer la qualité et la fiabilité avec des tests
  Les patrons de conceptions seront vus en 5A.
- Gestion de projets et gestion de versions avec GIT.

Trois grandes catégories d'acteurs

- *Le client :* entreprise, organisation ou personne qui paie pour le développement du logiciel.
- *L'utilisateur :* personne utilisant le logiciel.
- *Le développeur :* personne participant au développement du logiciel
  - Analyste
  - Architecte
  - Programmeur
  - Testeur (souvent programmeur)
  - Chef de projet (organiser la réussite du projet en terme de qualité du résultat, temps et argent)

Définition

Une *activité* est une tache qui entre dans la réalisation d'un logiciel. De plus,

- elle utilise et produit des documents comme des rapports, modèles, codes ...
- elle fait intervenir des acteurs.

Liste des activités classiques

1. Analyse des besoins ("requirements") : quoi faire,
2. Conception ("design") : comment faire,
3. Programmation ("coding") : faire,
4. Test : vérifier que c'est bien fait,
5. Maintenance : rester à l'écoute du client.
6. Qualité des éléments produits par les activités et de leur déroulement

Activité - Analyse des besoins

Objectifs de cette activité : définir/formaliser ce que le client veut.

- Acteurs impliqués :
  - Le client : finance le développement du logiciel
  - L'utilisateur : utilise le logiciel
  - Le développeur : fabrique le logiciel
- Nature des besoins/contraintes :
  - Besoins fonctionnels
  - Qualité : performance (e.g. temps de calcul), ergonomie, date de livraison, coût...
  - Contraintes en terme de procédure : format des documents intermédiaires, prototypes intermédiaires,...
  - Contraintes de conception : e.g. choix de la plate-forme
- Définition des priorités : Essentiel / Désirable / Optionnel

Les documents produits par l'analyse des besoins

- Description du fonctionnement du système (côté client)
- Conception du logiciel (côté ingénieur)
- Définition des tests d'acceptation / recettes (côté ingénieur) :
  - Validation : scénarii d'exécution démontrant au client le bon fonctionnement du logiciel
  - Éviter les régressions lors d'opérations de maintenance et d'évolution

Activité - conception

Objectifs de cette tache : décrire la manière et les outils.

- Conception architecturale, modélisation haut-niveau : modules,
- Conception détaillée, modélisation bas-niveau : on se rapproche du code,
- Caractéristiques/contraintes particulières : persistance, composants sur l'étagère, langage/paradigme...

Informations produites à destination du client et des développeurs

- Représentation conceptuelle liste des fonctionnalités de la solution proposée. Elle est rédigée en langage simple et validable par le client.
- Représentation technique décrit la structure et le comportement. Elle est rédigée en langage technique et validable par les développeurs.

Activité - Programmation

Objectifs de cette tache : implémenter la solution.

Choix de l'outil technique

- choix du paradigme (procédural, objet, fonctionnel, composant, aspect, rôle,...)
- choix du langage (C, Java, C++, Python,...)
  - Simplicité
  - Performances temps de calcul et mémoire
  - Mécanismes
  - Fonctionnalités des librairies/packages standards associés

Ces choix dépendent du contrat, des besoins, de la conception, des compétences humaines.

Informations produites pour les concepteurs et testeurs

Code source en respectant un certain nombre de bonnes pratiques. Il doit pouvoir être testé, corrigé et amélioré facilement.

Activité - Test

Objectifs de cette tache : valider le fonctionnement,

Activité - Test

Objectifs de cette tache : valider le fonctionnement, ou révéler des dysfonctionnements.

Informations produites dans le cas de tests unitaires

Des lignes de codes supplémentaires qui

- exécutent des fonctions développées avec des entrées choisies,
- et comparent le résultat obtenu avec le résultat attendu.

```
//Code original
static public float addition(float a, float b)
{
 return a + b;
}
```

```
//Code original
static public float addition(float a, float b)
{
 return a + b;
}
```

```
//Test unitaire (JUnit)
public class TestMath extends TestCase
{ public void test1()
  { float entree1 = 1;
    float entree2 = 1;
    float sortieAttendue  = 2;
    float sortieObtenue = addition(entree1, entree2);
    // Verification
    assertEquals(sortieAttendue, sortieObtenue);
  }
}
```

```
//Code avec une erreur
static public float addition(float a, float b)
{
 return a * b;
}
```

```
//Test unitaire (JUnit)
public class TestMath extends TestCase
{ public void test1()
  { float entree1 = 1;
    float entree2 = 1;
    float sortieAttendue  = 2;
    float sortieObtenue = addition(entree1, entree2);
    // Verification
    assertEquals(sortieAttendue, sortieObtenue);
  }
}
```

L'activité de test est omniprésente

- Test unitaires
- Test d'intégration : les modules sont-ils compatibles ?
  - ex : cas de modifications concurrentes
  - ex : à la compilation, à l'exécution (tout n'est pas vérifiable à la compilation)
- Test système : fonctionnalité attendue par le développeur ?
  - éventuellement suivi de test de performances (e.g. temps CPU)
- Test d'acceptation : fonctionnalité attendue par le client ?
  - validation du logiciel avec le client
- Test d'installation (déploiement) : le logiciel fonctionne-t-il chez l'utilisateur ?
  - Alpha test : par des utilisateurs sélectionnés sur site de développement
  - Beta test : par des utilisateurs sélectionnés hors site de développement

Activités - Maintenance

Objectifs de cette activité : assurer les évolution, adaptation.
Surtout si le logiciel livré ne répond pas aux exigences du client !

Définition

Un cycle de vie propose une stratégie qui décrit la manière dont les activités sont organisées.

Les modèles du cycle de vie du logiciel sont des plans de travail qui permettent de planifier le développement.

Plusieurs modèles de cycle de vie :

- le modèle du code-and-fix ;
- le modèle de la transformation automatique ;
- le modèle de la cascade ;
- le modèle en V ;
- le modèle par incrément ;
- le modèle par prototypage ;
- le modèle de la spirale.

Gestion des versions

- Versions
- Intégration continue