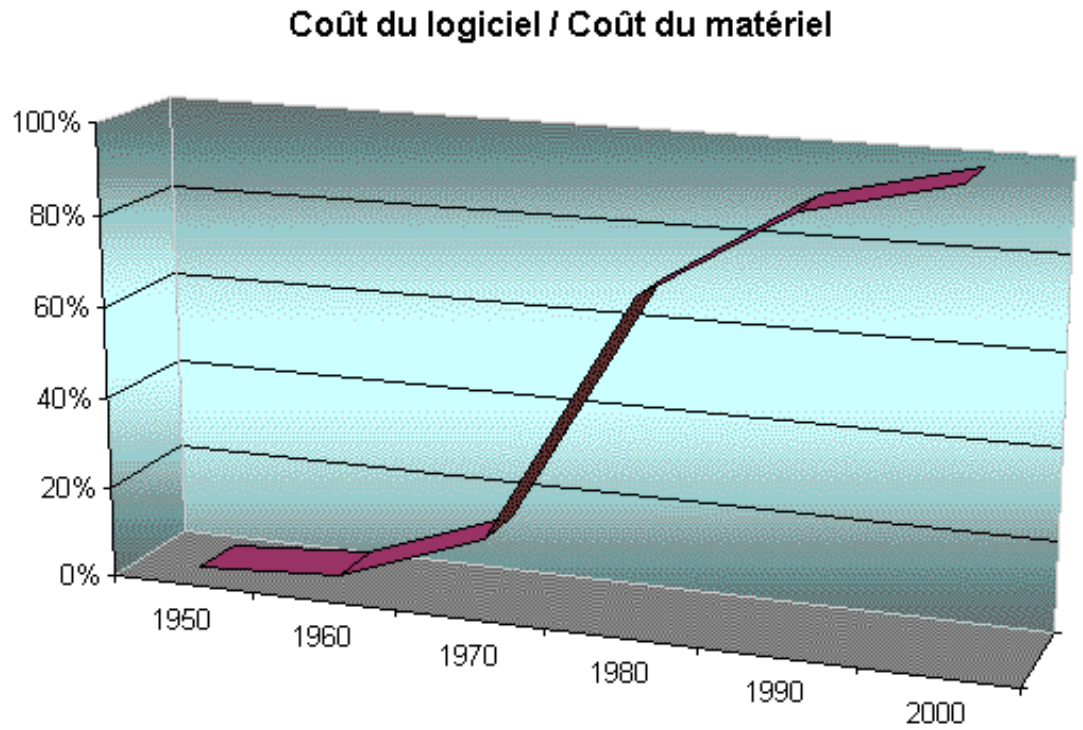
La crise du logiciel

Le logiciel est un objet immatériel (pendant son développement), très malléable au sens de facile à modifier, ses caractéristiques attendues sont difficiles à figer au départ et souvent remises en cause en cours de développement,
Les défaillances et erreurs ne proviennent ni de défauts dans les matériaux ni de phénomènes d'usure dont on connaît les lois mais d'erreurs humaines, inhérentes à l'activité de développement.
Le logiciel ne s'use pas, il devient obsolète (par rapport aux concurrents, par rapport au contexte technique, par rapport aux autres logiciels, ...).

La crise du logiciel

Comparons l'évolution du coût du logiciel et du matériel depuis les années 50 :



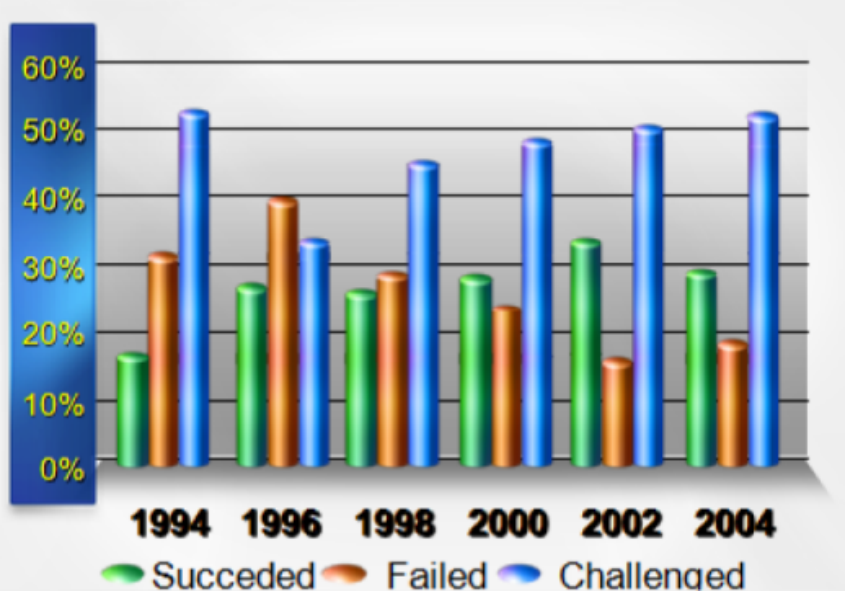
En 2000 :

- L'aspect matériel représente 20% du coût global
- L'aspect logiciel représente 80% du coût global

La crise du logiciel est une baisse significative de la qualité des logiciels qui coïncide avec un coût du logiciel qui dépasse à partir des années 1970 le coût du matériel.

Illustration de la crise du logiciel

- Etude du Standish Group sur plus de 350 entreprises totalisant plus de 8000 projets d'application :



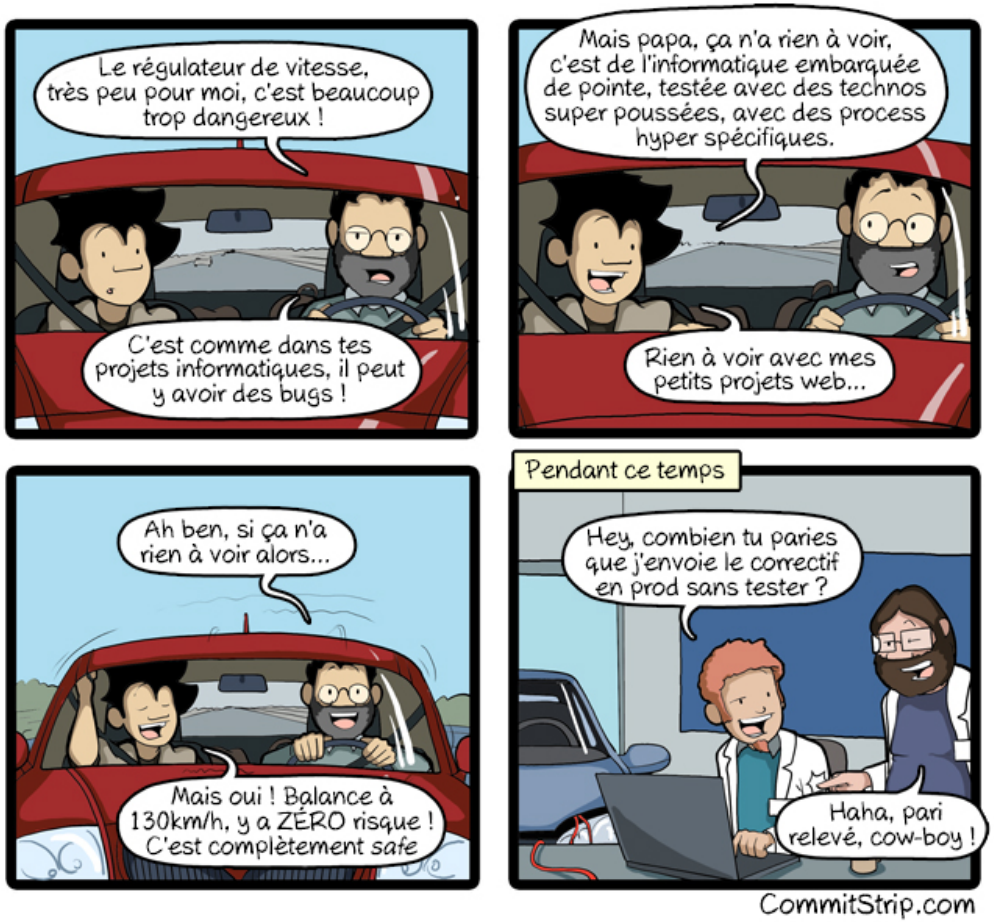
- Succès : livré à temps, sans dépassement de budget et avec toutes les fonctionnalités initialement spécifiées
- Mitigé : livré et opérationnel, mais avec moins de fonctionnalités que prévu et un dépassement de budget et/ou de temps
- Echec : abandonné en cours de route

Exemples d'échecs

Les premières tentatives de création de logiciels de grande ampleur ont vite montré les limites d'un travail informel : les produits réalisés ne sont pas terminés dans les temps, coûtent plus cher que prévu, ne sont pas fiables, sont peu performants et coûtent cher en entretien.

- Exemple d'abandons de projets :
  - Confirm (1992) : projet d'American Airlines de système de réservation commun (avions, voitures, hôtels, etc.).
    Investissement : 125 Millions $ sur 4 ans, plus de 200 ingénieurs.
    Résultat : les différentes parties n'ont pas pu être assemblées en raison de ≪ **la description incomplète des besoins**, un manque d'implication des utilisateurs et une **constante évolution des exigences et des spécifications** ≫ [THE STANDISH GROUP REPORT, 1995].

Exemples d'échecs



- Importance des tests : Ariane V (1996) : explosion de la fusée en vol due à bug logiciel (erreur de débordement lors de la conversion d'un nombre flottant 64bits vers un entier 16bits. Code hérité de Ariane IV, manque de tests). Coût : 500 Millions de $.

Exemples d'échecs

- Importance des tests :
  - Perte de NASA Mars Climate Orbiter, une des deux sondes spatiales du programme Mars Surveyor 98 (1999) : détruite à cause d'une erreur de navigation pendant sa mise en orbite autour de Mars (certains paramètres avaient été calculés en unités de mesure anglo-saxonnes et transmis tels quels à l'équipe de navigation, qui attendait ces données en unités du système métrique)

Principales causes de la crise du logiciel

- Fuite en avant de la complexité (technologie en perpétuelle évolution, croissance taille et complexité des systèmes)
- Faible correspondance avec les besoins des utilisateurs & Coût élevé du changement des fonctionnalités
- Délais de réalisation généralement dépassés
- Faiblesse des tests
- L'importance de la maintenance est souvent sous-estimée

On maîtrise mal le développement des logiciels ; des études se penchent alors sur la recherche de méthodes de travail adaptées à la complexité inhérente aux logiciels contemporains et ont donné naissance au génie logiciel (1968).

Définition du génie logiciel

Génie logiciel (software engineering)

Ensemble de moyens (techniques et méthodes) mis en oeuvre pour la construction de systèmes informatiques et de logiciels.
Objectif : maitrise de l'activité de fabrication des logiciels (temps de développement, critères de qualité).

Nécessité de méthodes/processus de développement logiciel.

Un processus de développement / cycle de vie / méthode définit un ensemble d'**activités** et leurs enchaînements pour la réalisation d'un logiciel

- analyse des besoins / spécification (cahier des charges, maquettes, ...)
- conception, implémentation, intégration, tests
- déploiement, maintenance

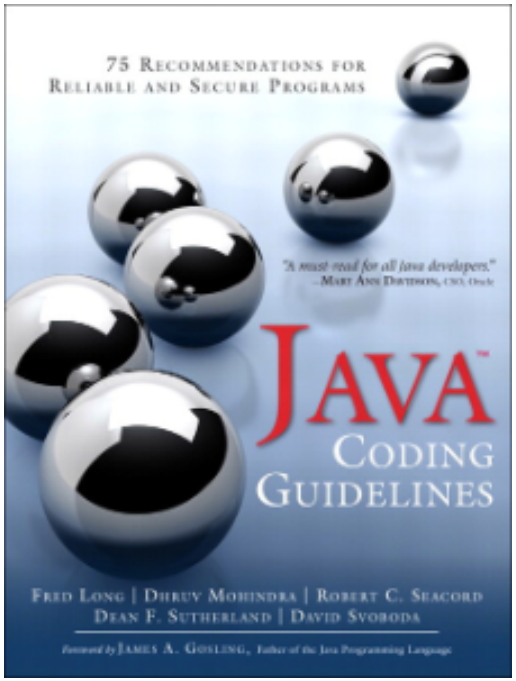
Conclusion sur processus de développement logiciel

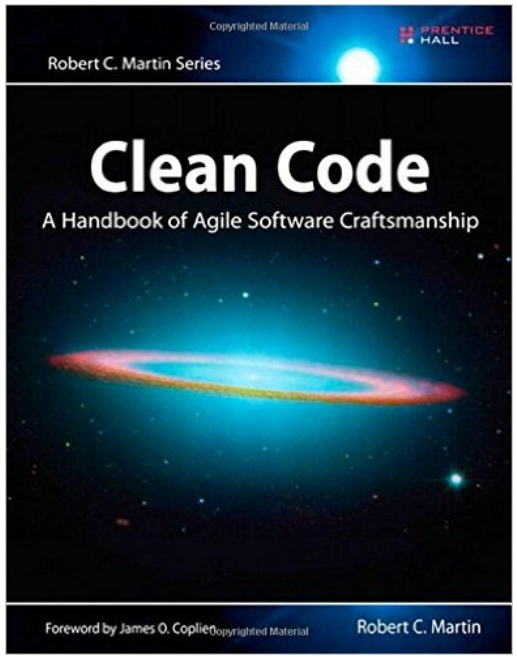
- Pas de modèle idéal : la meilleure méthode est celle adaptée au contexte
  - Type de logiciel
  - Ampleur du projet
  - Equipe de développement
- Importance des tests et des outils autour du développement logiciel.

Outils autour du développement logiciel

Au-delà du code :

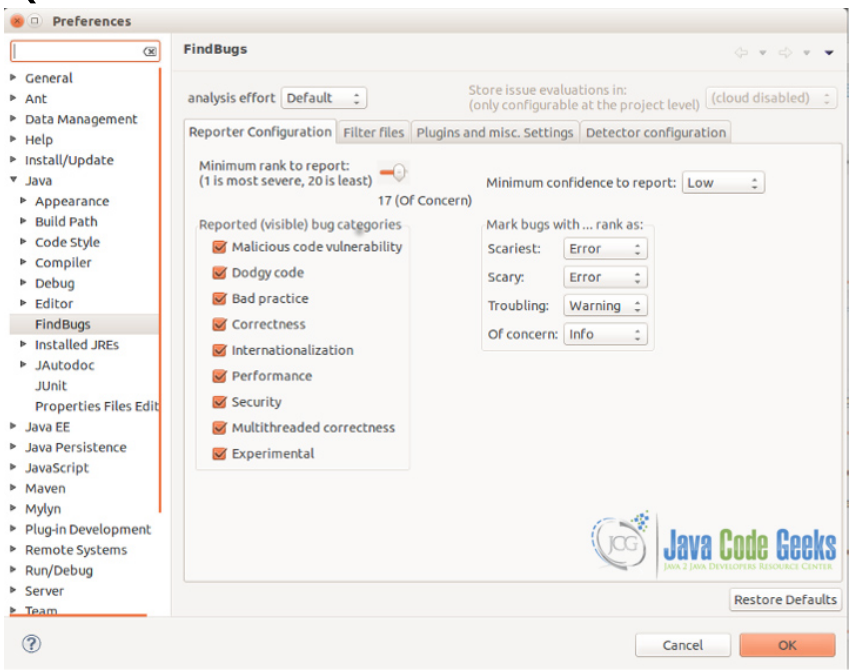
- Documentation : Générateurs de documentation
- Partage des sources : Gestionnaire de versions
- Suivi de bugs / évolution : Gestionnaire de tickets
- Qualité du code : Outils d'audit de code
  - livres du CERT Centre gouvernemental de veille, d'alerte et de réponse aux attaques informatiques
    `https://wiki.sei.cmu.edu/confluence/display/java`





Outils autour du développement logiciel

- Documentation : Générateurs de documentation
- Partage des sources : Gestionnaire de versions
- Suivi de bugs / évolution : Gestionnaire de tickets
- Qualité du code : Outils d'audit de code



- Analyseur statique de code
- `http://findbugs.sourceforge.net/`
- 4 niveaux de bugs (Scariest,Scary,Troubling,Of concern), confidence, catégories de bugs