INTRODUCTION AU LOGICIEL

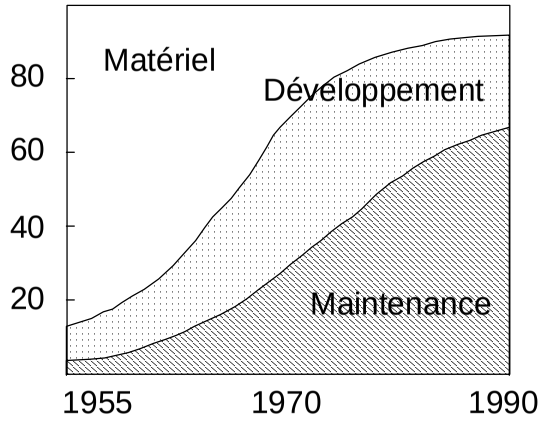
CONTENU

- la "crise du logiciel"
- le génie logiciel et ses objectifs
- les principes du génie logiciel
- le cycle de vie du logiciel
- l'analyse, la conception et leur frontière
- les spécifications
- les méthodes d'analyse et de conception

CRISE DU LOGICIEL

- ne correspondent pas aux besoins
- contiennent trop d'erreurs
- coûts imprévisibles et prohibitifs
- maintenance difficile et coûteuse
- délais en général dépassés
- non portables
- non modifiables ou adaptables
- peu efficaces

TENDANCE DES COUTS



QUELQUES CAUSES

- L'absence d'échelle
- L'absence de droit à l'erreur
- Le manque de modélisation
- L'influence des mythes

QUELQUES MYTHES

- une idée grossière du logiciel est suffisante pour commencer
- quand le programme fonctionne, c'est terminé
- les spécifications peuvent changer car le logiciel est facilement modifiable
- en cas de retard, il suffit de rajouter des programmeurs

PREMIERES CONCLUSIONS

DEVELOPPER DU LOGICIEL EST UN METIER

UNE DEMARCHE DE MODELISATION ET DE SIMPLIFICATION EST INDISPENSABLE

UNE BONNE EQUIPE DE DEVELOPPEMENT DE LOGICIEL COMPREND AUSSI DES SPECIALISTES DU DOMAINE A AUTOMATISER

LE CREDO

IL EST NECESSAIRE ET POSSIBLE DE SORTI DE LA CRISE DU LOGICIEL

LE GENIE LOGICIEL ET SES OBJECTIFS

Le génie logiciel a été créé en réaction à la crise, avec pour objectif de créer des logiciels:

- ✔ aisément modifiables
- ✔ efficaces
- ✔ sûrs
- ✔ compréhensibles
- ✔ qui répondent aux spécifications

en respectant les délais

LES OBJECTIFS: modifiables

- ✔ on doit pouvoir connaître les effets d'un changement
- ✔ il faut assurer l'invariance logique vis à vis des modifications matérielles
- ✔ la conception doit transparaître dans la solution
- ✔ il faut séparer le quoi du comment

LES OBJECTIFS: efficacité

L'efficacité résulte d'un compromis entre le temps d'exécution et l'encombrement:

- ✔ efficacité microscopique (généralement considérée trop tôt)
- ✔ efficacité macroscopique (vue unifiée du problème)
- ✔ règle des 80 - 20

LES OBJECTIFS: sûreté et lisibilité

- ❑ SURETE
  - ✔ prévention des pannes
  - ✔ récupération en cas de pannes
  - ✔ doit être considérée très tôt dans le processus de développement
- ❑ LISIBILITE
  - ✔ directement liée à la gestion de la complexité
  - ✔ on écrit le code une fois, mais on le lit à de nombreuses occasions

LES PRINCIPES DU GENIE LOGICIEL

Pour atteindre ses objectifs, le génie logiciel fait appel à 7 principes:

- ✔ l'abstraction
- ✔ la dissimulation de l'information
- ✔ la modularité
- ✔ la structuration
- ✔ l'uniformité
- ✔ la complétude
- ✔ la "confirmabilité"

LES PRINCIPES DU GENIE LOGICIEL L'ABSTRACTION

- ✔ extraire les éléments essentiels
- ✔ omettre temporairement les autres
- ✔ chaque niveau de décomposition doit représenter une abstraction
- ✔ principale technique de gestion de la complexité
- ✔ on peut se baser sur plusieurs types d'abstraction: fonctionnelle, de donnée, d'objet

LES PRINCIPES DU GENIE LOGICIEL DISSIMULATION DE L'INFORMATION

C'est rendre inaccessibles les détails de l'implantation

- ✔ renforcée par des interfaces bien définies
- ✔ renforce l'abstraction en supprimant les détails
- ✔ évite de base des décisions à haut niveau sur des caractéristiques de bas niveau

LES PRINCIPES DU GENIE LOGICIEL LA MODULARITE ET LA STRUCTURATION

- ✔ permet de gérer la complexité
- ✔ nécessite des interfaces bien définies
- ✔ 2 approches:
  top-down (descendante)
  bottom-up (ascendante)
- ✔ moyens de vérification (couplage et cohésion)

CLASSIFICATION DES SYSTEMES INFORMATIQUES

- systèmes transformationnels
- systèmes réactifs
  - ✔ systèmes interactifs
  - ✔ systèmes temps-réel

peu de systèmes tombent dans une seule catégorie

SYSTEMES TRANSFORMATIONNELS

- ont une durée de fonctionnement limitée
- prennent ou demandent des données en entrée et produisent un résultat
- le temps d'exécution n'est pas critique
- exemples typiques
  - ✔ programmes de calcul
  - ✔ gestion de la paie
  - ✔ ...

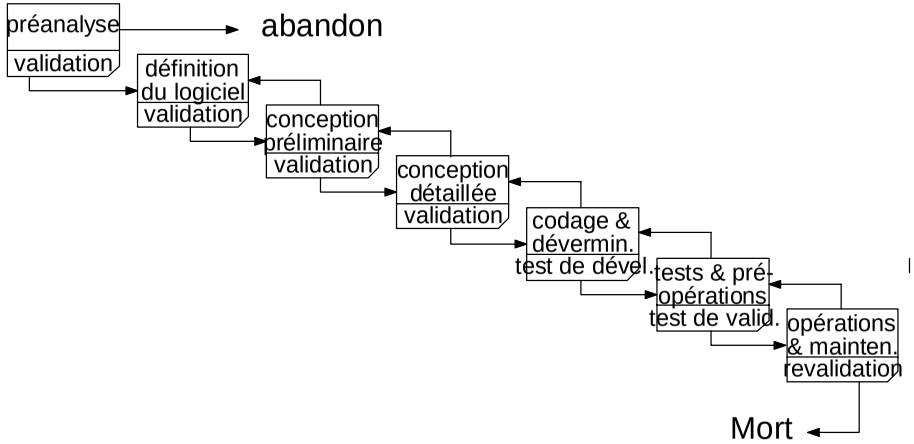
SYSTEMES INTERACTIFS

- en dialogue avec un ou des utilisateurs via des capteurs et des actionneurs
- ne font rien sans demande préalable de l'utilisateur
- plusieurs stimuli peuvent se produire et doivent être traités en parallèle
- le temps de réponse n'est pas très critique
- exemples typiques
  - ✓ programmes de CAO
  - ✓ Minitel / Videotex
  - ✓ traitement de texte / éditeurs / tableurs

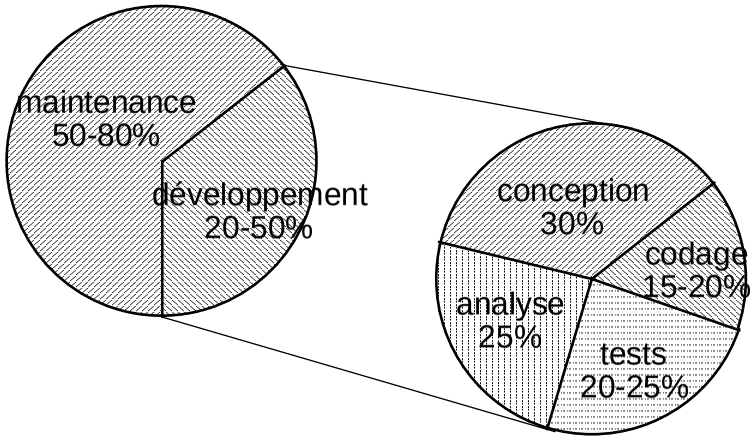
SYSTEMES TEMPS-REEL

- sont connectés à un système externe (procédé)
- doivent réagir dans un temps borné aux sollicitations du système externe
- fonctionnent en général indéfiniment
- ne doivent pas mettre en danger le système externe
- exemples typiques
  - ✓ commandes de machines, de robots ou de procédés industriels
  - ✓ commutation téléphonique
  - ✓ applications aéronautiques, spatiales et militaires
  - ✓ ...

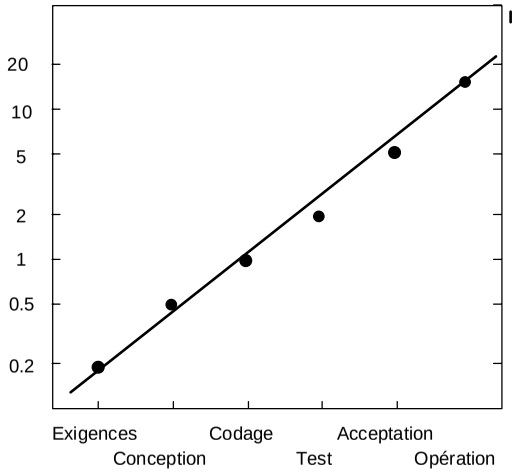
CYCLE DE VIE (modèle de la chute d'eau)



CYCLE DE VIE (proportions)



COUT DES ERREURS



L'ANALYSE

C'est le processus qui partant des exigences du client vise à obtenir un cahier des charges, c'est à dire une description précise de ce que doit faire le logiciel. Entrent dans ce processus:

- les exigences du client
- les sources d'information sur le domaine
- les résultats des analyses de faisabilité

LE PROCESSUS D'ANALYSE

- c'est un processus difficile qui fait appel aux capacités de l'analyste à
  - ✓ manipuler l'abstraction
  - ✓ modéliser
- les erreurs à ce niveau ne sont souvent découvertes que très tard
- il est bon de faire précéder l'analyse d'une étude du domaine
- c'est à ce niveau que se pratiquent les études de faisabilité

ANALYSE et CONCEPTION: les difficultés

- problèmes avec les exigences
- l'analyse dit le "QUOI" non le "COMMENT"
- meta-exigences
- frontière entre analyse et conception
- les spécifications

PROBLEMES AVEC LES EXIGENCES

- les exigences du client sont rarement "propres"
- il y d'ordinaire un ou plusieurs des problèmes suivants
  - ✔ omissions
  - ✔ contradictions
  - ✔ ambiguïtés
  - ✔ duplicata
  - ✔ imprécisions
  - ✔ trop de conception
  - ✔ pas de priorités
  - ✔ informations sans objet ou inutiles

EXEMPLES DE "QUOI"

- tous les calculs doivent se faire avec 10 chiffres
- si un noeud du réseau tombe en panne, le réseau doit se reconfigurer dynamiquement
- le système doit être capable d'acquérir 1000 points par seconde
- à la mise sous tension, le système doit s'auto-tester
- en cas d'arrêt d'urgence, les sorties doivent être mise en état de sécurité

EXEMPLES DE "COMMENT"

- les fonctions sin et cos seront évaluées avec des approximations en séries
- pour remplacer une fiche d'outil, il faut d'abord détruire l'ancienne puis ajouter la nouvelle
- l'état des fins de course devra être lu dans un registre de 8 bits
- la fin du fichier sera le caractère VT

META-EXIGENCES

- Une meta-exigence est l'indication de la manière d'accomplir un comportement particulier du système qui est fourni et exigé par le client. Il peut, par exemple, spécifier l'algorithme de cryptage
- C'est une décision de conception effectuée par le client.
- Il faut minimiser le nombre des meta exigences

FRONTIERE ENTRE ANALYSE ET CONCEPTION

dire que l'analyse ne concerne que le quoi n'est pas suffisant, il faut que sorte de l'analyse

- ✓ un examen du concept, système ou phénomène avec l'intention de le comprendre et de le décrire de façon précise
- ✓ la description de l'interaction du concept, système ou phénomène avec l'environnement existant ou proposé
- ✓ la proposition de 2 ou 3 solutions possibles avec analyse complète et précise de ces solutions
- ✓ un description précise et complète de la solution qui sera fournie au client, ainsi qu'un moyen de juger les alternatives de conception

SPECIFICATIONS ou CAHIER DES CHARGES

= description <u>précise</u>

pour être certain qu'un programme fasse exactement ce que l'on veut, il faut d'abord dire exactement ce que l'on veut qu'il fasse

CONTENU DU CAHIER DES CHARGES

- ✓ une description précise et concise du probléme
- ✓ l'ordinateur cible (matériel, système d'exploitation)
- ✓ les fonctions du logiciel
- ✓ les caractéristiques d'utilisation (performances, contraintes, qualité)
- ✓ ce qu'il ne faut pas faire
- ✓ l'interface utilisateur
- ✓ les extensions envisagées
- ✓ un glossaire des termes utilisé
- ✓ coûts et délais

QUALITES DES SPECIFICATIONS

- ✔ conformes aux besoins
- ✔ adaptées aux lecteurs
- ✔ complètes et cohérentes
- ✔ précises et vérifiables
- ✔ structurées et homogènes
- ✔ modifiables
- ✔ si en langage naturel, au présent

mais d'abord qu'elles existent

FORMES DES SPECIFICATIONS

- ❑ n'existent pas
- ❑ spécifications informelles
- ❑ spécifications standardisées
- ❑ spécifications semi-formelles
- ❑ spécifications formelles

METHODES ET LANGAGES SPECIFICATIONS

- ❑ méthodes
  - ✓ SADT (fonctions)
  - ✓ OORA (objets)
  - ✓ Statemate (temps réel)
  - ✓ Buhr (tâches)
  - ✓ formelles (VDM, ...)
  - ✓ Pamela, ER (données)
  - ✓ ...
- ❑ langages
  - ✓ réseaux de Petri
  - ✓ GRAFCET
  - ✓ français
  - ✓ State charts
  - ✓ VDM
  - ✓ Z
  - ✓ ...

METHODES DITES STRUCTURES

- Analyse
  - ✓ Structured Analysis, SADT, PSL, ...
- Conception
  - ✓ Structured Design
- Extension au temps réel
  - ✓ SART

METHODES DITES STRUCTUREES

- ❑ Avantages
  - ✓ faciles à comprendre
  - ✓ supportées par des outils (CASE)
- ❑ Inconvénients
  - ✓ ne font pas usage de tous les principes du génie logiciel
  - ✓ nécessitent beaucoup de flair et d'expérience
  - ✓ pas de support pour la réutilisation
  - ✓ passage de l'analyse à la conception difficile

METHODES ORIENTEES SUR LES OBJETS

- Analyse
  - ✓ Object Oriented Requirement Analysis (OORA)
- Conception
  - ✓ Object Oriented Design (OOD)

METHODES ORIENTEES SUR LES OBJETS

- ❑ Avantages
  - ✓ moins de couplage
  - ✓ meilleure structuration
  - ✓ supportent la réutilisation de logiciels
  - ✓ passage facile de l'analyse à la conception
  - ✓ supportées par les systèmes d'exploitation
- ❑ Inconvénients
  - ✓ nécessitent de l'expérience
  - ✓ peu d'outils les supportent
  - ✓ moins connues

METHODES FORMELLES

Une telle méthode se caractérise par

- ✓ une base mathématique stricte
- ✓ des directives d'utilisation

METHODES FORMELLES

- ❑ Avantages
  - ✓ permettent de prouver la conformité du logiciel aux spécifications
  - ✓ permettent de révéler contradictions, ambiguïtés et manques
  - ✓ permettent parfois de se passer de la conception et du codage
- ❑ Inconvénients
  - ✓ plus difficiles à comprendre
  - ✓ pratiquement pas de support d'outils

7 MYTHES CONCERNANT LES METHODES FORMELLES

1. permettent de garantir que le logiciel est parfait
2. ne concernent que la preuve du logiciel
3. ne sont utiles que pour les logiciels dont la sécurité est critique
4. nécessitent des mathématiciens ultra qualifiés
5. accroissent les coûts de développement
6. sont inacceptables pour les clients
7. ne sont pas utilisées à large échelle pour des logiciels réels

7 VERITES CONCERNANT LES METHODES FORMELLES

1. très utiles pour trouver les erreurs très tôt
2. fonctionnent parce qu'elles obligent à réfléchir attentivement aux spécifications
3. utiles pour n'importe quel type d'application
4. basées sur des spéc. mathématiques bien plus faciles à comprendre que des programmes
5. peuvent réduire les coûts de développement
6. peuvent aider le client à comprendre ce qu'il achète
7. utilisées sur des vrais projets dans l'industrie

LES LANGAGES DE PROGRAMMATION

le choix d'un langage n'est pas sans effet sur le projet. Un langage doit:

- supporter les concepts du génie logiciel
- permettre une gestion de projet efficace
  - ✓ suffisamment expressif
  - ✓ suffisamment rigide pour éviter le style hacker
  - ✓ être bien supporté par des outils de développement (compilateurs, éditeurs syntaxiques, dévermineurs symboliques
- avoir une pérennité suffisante

CONCEPTION DE LOGICIELS ORIENTES OBJETS

2. Identification des objets et classes d'intérêt (OCI)
   - 2.1 développement d'une stratégie informelle
   - 2.2 Identification des objets et classes d'intérêt
   - 2.3 Association des attributs avec les OCI
3. Identif. des opérations requises et subies par les OCI
   - 3.1 identification des opérations d'intérêt (OI)
   - 3.2 association des attributs avec les OI
   - 3.3 traitement des objets composites
     - 3.3.1 décomposition en opérations primitives
     - 3.3.2 découplage des objets et classes

CONCEPTION DE LOGICIELS OO (suite)

4. création des spécifications de classes et objets pour les objets de conception
   - 4.1 mise ensemble des objets, classes et opérations
   - 4.2 examen de la complétude des objets et classes
5. décision sur l'implantation des objets et classes
   - 5.1 objets et classes identifiés durant l'analyse
   - 5.2 objets (classes) identifiés durant la conception
6. représentation graphique pour le développement OO
   - 6.1 représentations statiques
   - 6.2 représentations dynamiques

CONCEPTION DE LOGICIELS OO (fin)

7. Etablissement de l'interface pour chaque élément
   - 7.1 objets et classes
   - 7.2 systèmes d'objets
   - 7.3 sous-systèmes
8. implantation de chaque élément
   - 8.1 raffinement par étape de l'interface des objets et classes interfaces
   - 8.2 raffinement par étape de l'interface des autres objets et classes
   - 8.3 application récursive du processus de développement orienté objets