Génie Logiciel

Rappels

C. Crochepeyre Génie Logiciel – Rappels 1

Ce cours ne concerne que le **logiciel** : les techniques de conception d'un logiciel, son développement et son suivi tout au long de son exploitation.

Alors que la **conduite d'un projet** informatique, dont l'étude est faite dans une autre partie, considère l'ensemble des moyens humains, financiers, matériels et logiciels mis en œuvre pour la réalisation d'une application informatique.

Commençons par des rappels sur le Génie Logiciel (G.L.): les concepts, les outils et son utilité lors de la conception d'un logiciel.

INTRODUCTION (1)

- **GL: ingénierie appliquée au logiciel informatique**
- Objectif: la qualité
  - diminution du coût du logiciel et fiabilité
- Besoin: complexité des grands systèmes
- Gestion de projet => entre autres la production de logiciel
- Logiciel:
  - Comment le produire ?
  - Comment le contrôler ?
  - Quelle documentation ?

- Une base de connaissances du GL
  - http://fr.wikipedia.org/wiki/SWEBOK (1)

(1) Software Engineering Body of Knowledge est le document de base de l'IEEE-Computer-Society pour la normalisation en ingénierie du logiciel

C. Crochepeyre Génie Logiciel – Rappels 2

1- **Définition du G.L**. selon l'arrêté ministériel du 30/12/83:

"Ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi"

2- Définition selon Patrick Jaulent 'Génie logiciel - Les méthodes'

"Procédures, méthodes, langages, ateliers imposés ou préconisés par les normes adaptées à l'environnement d'utilisation afin de favoriser la production et la maintenance de composants logiciels de qualité"

3- Quelques exemples connus de preuves de l'utilité du G.L. dans de grands systèmes:

. Faux départ de la 1ère navette spatiale: manque de synchronisation entre deux calculateurs

. Efficacité des missiles Exocet lors de la guerre des Malouines: missile non répertorié par la marine anglaise comme missile ennemi

. Passage à 500.000 km au lieu de 5000km du satellite lors de la mission Vénus: remplacement d'une virgule par un point.

et bien d'autres exemples plus récents…

. le site internet voyages-sncf.com fermé plusieurs heures en 2008, suite à l'ouverture d'une nouvelle version du site.

. En 2008, un candidat aux élections cantonales dans les Hauts de Seine obtient officiellement 10% de voix selon un résultat arrondi à 2 chiffres après la virgule. Il a le droit de se présenter au 2sd tour… Or on s'aperçoit qu'en réalité il n'atteint que 9.998% des voix.

Ces erreurs prouvent que la complexité des systèmes informatisés peuvent conduire à des catastrophes et que les logiciels doivent être conçus avec méthode.

INTRODUCTION (2)

- GL: ingénierie appliquée au logiciel informatique
- **Objectif: la qualité**
  - diminution du coût du logiciel et fiabilité
- Besoin: complexité des grands systèmes
- Gestion de projet => entre autres la production de logiciel
- Logiciel:
  - Comment le produire ?
  - Comment le contrôler ?
  - Quelle documentation ?

- Une base de connaissances du GL
  - http://fr.wikipedia.org/wiki/SWEBOK (1)

(1) Software Engineering Body of Knowledge est le document de base de l'IEEE-Computer-Society pour la normalisation en ingénierie du logiciel

C. Crochepeyre Génie Logiciel – Rappels 3

Objectifs du G.L.

Diminuer le **coût** et assurer la **fiabilité** du produit.

1. En 1986, un enquête de EDP Software Maintenance auprès de 50 entreprises a conclu que:

53% du budget total du logiciel est utilisé pour la maintenance: évolution, adaptation, correction, perfectionnement, contrôle qualité, organisation du suivi et divers.

Notons que (selon l'estimation du département de la défense américaine DOD)

. la durée de vie d'un logiciel est de 8 à 10 ans

. la durée de vie du matériel est de 5 ans

En 1995, une étude du *Standish Group* dressait un tableau accablant de la conduite des projets informatiques. Reposant sur un échantillon représentatif de 365 entreprises, totalisant 8 380 applications, cette étude établissait que :

16,2% seulement des projets étaient conformes aux prévisions initiales,

52,7% avaient subi des dépassements en coût et délai d'un facteur 2 à 3 avec diminution du nombre des fonctions offertes,

31,1% ont été purement abandonnés durant leur développement.

En 2000 les résultats sont meilleurs mais restent cependant inquiétants.

2. Lu sur le site GNT Média, juillet 2006 « Le département de la Défense a dépensé à peu près 12 milliards de dollars pour le développement logiciel de ses équipements militaires…..soit 30% de son budget logiciel. Et ces dépenses ne font que croître puisque l'élément logiciel est devenu central dans la mise en œuvre des équipements militaires. L'armée moderne dépend de plus en plus de la fiabilité des logiciels, autant que celle du matériel. »

INTRODUCTION (3)

- GL: ingénierie appliquée au logiciel informatique
- Objectif: la qualité
  - diminution du coût du logiciel et fiabilité
- **Besoin: complexité des grands systèmes**
- Gestion de projet => entre autres la production de logiciel
- Logiciel:
  - Comment le produire ?
  - Comment le contrôler ?
  - Quelle documentation ?

- Une base de connaissances du GL
  - http://fr.wikipedia.org/wiki/SWEBOK (1)

(1) Software Engineering Body of Knowledge est le document de base de l'IEEE-Computer-Society pour la normalisation en ingénierie du logiciel

C. Crochepeyre Génie Logiciel – Rappels 4

Besoin du G.L.

Pourquoi le G.L.?

1. **Complexité** de plus en plus grande des logiciels

. recours aux techniques du G.L.

2. Cette discipline suppose:

. des équipes de développeurs et non quelques spécialistes

. une évolution des méthodes de travail

. apprendre à réutiliser l'existant (Ken Orr affirme que : 80% des développements existent déjà et que 20% seulement sont spécifiques)

GESTION DE PROJETS

La gestion d'un projet inclut la gestion des logiciels.

Comment le produire? Comment le contrôler? Que faut-il documenter?

Le génie logiciel s'intéresse à l'utilisation de techniques d'ingénierie permettant de mieux contrôler:

. l'aptitude de l'ingénieur à communiquer oralement et par écrit, à comprendre les problèmes des utilisateurs, à savoir gérer un projet et sa complexité

. la production des **programmes**

. la production d'une **documentation**: installation, utilisation, développement et maintenance du logiciel.

Des techniques qui suivent le cycle de vie du logiciel lors des différentes étapes de son développement.

Pour faciliter le contrôle du cycle de vie, il existe des modèles pour une meilleure efficacité: rapidité, fiabilité. Des **méthodes** sont proposées aux différentes étapes.

SUIVI DU CYCLE DE VIE

Plusieurs étapes:

•partir de la formulation des besoins

-pour aboutir à un produit logiciel fini

-en passant par des étapes intermédiaires

Mais se posent les questions sur la manière de procéder.

Selon **Swebok**, les domaines liés au génie logiciel :

- Les exigences du logiciel
- La conception du logiciel
- La construction du logiciel
- Les tests logiciels
- La maintenance du logiciel
- La gestion de configuration du logiciel
- L'ingénierie de la gestion logicielle
- L'ingénierie des processus logiciels
- L'ingénierie des outils et méthodes logicielles
- L'assurance qualité du logiciel

SWEBOK : Software Engineering Body of Knowledge

Projet lancé en 1998 par IEEE (en collaboration avec les universités, industries) qui peut servir de référence au génie logiciel en terme de définitions, standards et méthodes.

Le projet SWEBOK énonce 10 domaines qui entrent dans la discipline du génie logiciel qui sont énumérés ici.

1- CYCLE DE VIE DU LOGICIEL

1. **CYCLE DE VIE DU LOGICIEL**
2. EXPRESSION DES BESOINS
3. CONCEPTION DU LOGICIEL
4. LA PROGRAMMATION
5. TESTS ET MISE AU POINT
6. DOCUMENTATION
7. CONCLUSION

C. Crochepeyre Génie Logiciel – Rappels 6

La vie d'un logiciel commence par une étude préalable, se poursuit par son développement, son installation et doit faire l'objet d'un suivi tout au long de son utilisation.

Ce sont ces différentes étapes que nous allons aborder.

1.1. Cycle de vie et de développement

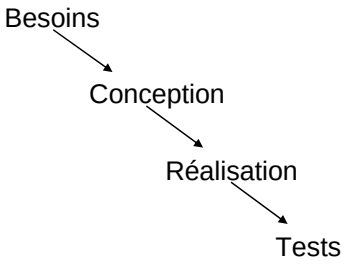
- **Modèle général du cycle de vie**
  - Expression des besoins
  - Conception du système et du logiciel
  - Réalisation et tests unitaires
  - Tests du système
  - Utilisation et maintenance

C. Crochepeyre Génie Logiciel – Rappels 7

La mise en place d'une organisation de production de systèmes permet:
. de maîtriser la qualité des produits
. de maîtriser les coûts
. de maîtriser les délais
Cette organisation repose sur le découpage du processus de développement en plusieurs phases.
Ce découpage est normalisé: norme ISO/CEI 12207. Il est appelé 'Cycle de vie d'un logiciel'.
La norme n'est pas attachée à un type de cycle de développement particulier (en V, en cascade…ou autre) et peut être appliquée à tous les types de projets.

Le modèle général du cycle de vie est le suivant:
1- Analyse et définition des besoins
expression et faisabilité de ces besoins
2- Conception du système et du logiciel
. conception préliminaire du système
. conception détaillée du système
. spécifications fonctionnelles du logiciel
. conception détaillée du logiciel
3- Réalisation et tests unitaires
. codage du logiciel
. tests des unités de programme
4- Tests du système
. intégration des unités, tests d'intégration
. validation du logiciel
. intégration matériel-logiciel
. recette système: validation du système
5- Utilisation et maintenance du système : suivi de la vie du logiciel

- **Expression des besoins**
  - consultation des utilisateurs
  - définitions des fonctionnalités du système
  - rédaction de documents compréhensibles par les utilisateurs et les équipes de développement
- **Conception du système et du logiciel**
  - recensement des diverses fonctions
  - décomposition du système en architectures logiciel et matériel



C. Crochepeyre Génie Logiciel – Rappels 8

L'expression des besoins des utilisateurs est une phase indispensable et qui peut prendre du temps..

Ainsi donc lors de cette phase:

- on consulte les utilisateurs

- on définit avec eux les fonctionnalités et les contraintes du système

- on s'attache à rédiger les documents, compréhensibles par tous, qui doivent servir aux étapes ultérieures et éventuellement qui peuvent être revues et modifiées

Rappelons que nous ne nous intéressons ici qu'au génie logiciel, par conséquent nous ne prenons en considération, lors de l'expression des besoins, que les éléments concernant le logiciel.

A partir de l'analyse des besoins, on décompose les tâches à réaliser sous forme de **fonctions logicielles**. On construit alors un modèle d'architecture informatique de l'application en considérant les deux aspects : matériel et logiciel.
Les différentes fonctions recensées du système vont conduire aux développements d'unités de programmes.

Cette conception comprend:

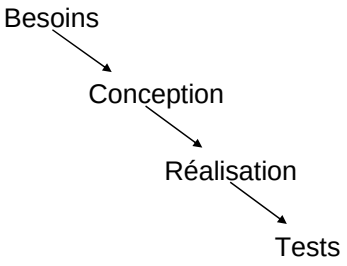
1- une étude préliminaire des fonctions du système

2- une analyse fonctionnelle du logiciel avec ses spécifications : statique ou dynamique selon les besoins

3- une analyse détaillée du logiciel en unités de programme

Ceci avant la phase suivante de réalisation des programmes.

- **Réalisation et tests unitaires**
  - choix d'un langage de programmation
  - production des programmes
  - tests unitaires de ces programmes
- **Tests du système**
  - intégration des unités de programme
  - tests de l'ensemble
  - livraison aux utilisateurs



C. Crochepeyre Génie Logiciel – Rappels 9

On a découpé les fonctions en unités de programme.

On passe à la phase de **programmation** en choisissant les langages de programmation appropriés.

L'analyse détaillée va permettre aux programmeurs de passer à la réalisation de chaque unité de programme.

Les **tests des programmes** sont ensuite effectués. On vérifie que les résultats obtenus sont corrects et correspondent aux spécifications énoncées.

Les unités de programme étant testées, on passe à la **phase d'intégration** de ces unités.

Des jeux de tests globaux vont permettre de valider l'ensemble du système.

On peut à ce niveau, comme au niveau précédent, modifier, remettre en cause, ajouter, supprimer certains éléments du développement.

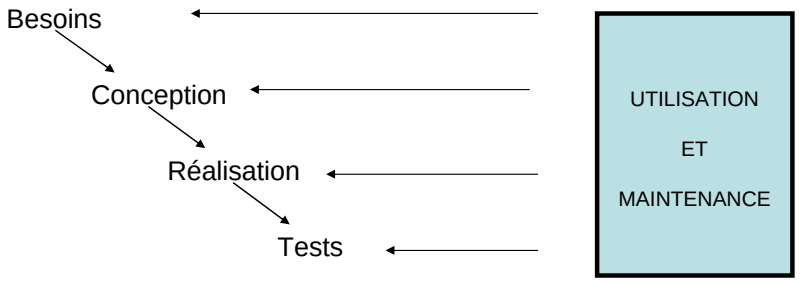
Attention au système livré trop vite!

Lorsqu'on est satisfait du résultat on passe à la phase de livraison aux utilisateurs.

L'acte de recette est officialisée afin d'éviter toute contestation.

Attention! Tous les documents d'installation, de fonctionnement, d'utilisation du système doivent accompagner cette livraison.

- **Utilisation et maintenance (1)**
  - correction des erreurs
  - amélioration des programmes
  - augmentation des fonctionnalités au fur et à mesure des besoins
  - remise en cause des étapes précédentes



C. Crochepeyre Génie Logiciel – Rappels 10

La **mise en exploitation** de l'application doit se faire avec précaution (travail en double) et peut prendre du temps.

Cette mise à l'épreuve du système par les utilisateurs peut conduire

- à des corrections d'erreurs

- à des améliorations des programmes: lenteur, précision des calculs, présentations des résultats....

Attention! Ne pas tomber dans le piège de la refonte du système (améliorations et modifications ou encore ajout de nouvelles fonctionnalités). Les phases antérieures de spécifications doivent être précises et avoir été acceptées par le client.

Il faut distinguer la phase de début d'exploitation (recette) de celle de maintenance qui dure dans le temps.

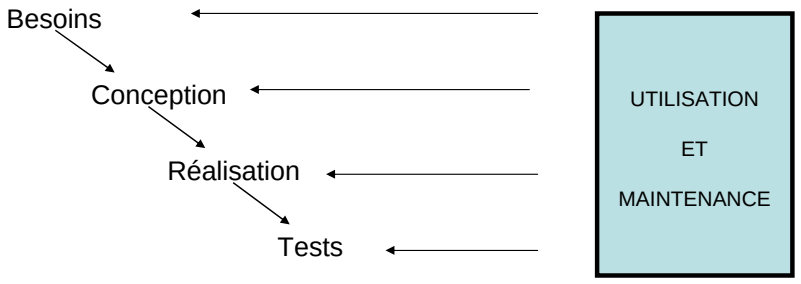
Si des augmentations des fonctionnalités sont nécessaires car les besoins ont évolués, ceci doit, dans la mesure du possible, avoir été prévu dans les phases d'analyse et de spécifications.
Elles ne doivent pas apparaître immédiatement après la livraison

Constat:

Si des retours vers les étapes antérieures ont eu lieu normalement lors de l'analyse et du développement, ces retours en général se font au sein de l'équipe des développeurs.

En revanche à ce niveau, les retours, à la demande des utilisateurs, peuvent avoir des conséquences importantes.

- **Utilisation et maintenance (2)**
  - correction des erreurs
  - amélioration des programmes
  - augmentation des fonctionnalités au fur et à mesure des besoins
  - remise en cause des étapes précédentes



C. Crochepeyre Génie Logiciel – Rappels 11

Le **cycle de vie d'un logiciel** est vivant : on parle de cycle de développement du logiciel qui prend en compte les retours arrière dans ce développement

Le **cycle de développement** consiste à transmettre les informations d'une étape à une autre

- 'en aval' c'est à dire des besoins vers les tests mais aussi

- 'en amont' c'est à dire éventuellement apporter des modifications ou ajouter des informations nécessaires aux étapes précédentes.

En fait, lorsque l'étape d'utilisation est effective, les retours arrière ne se font pas de manière aussi rationnelle. Les changements à apporter peuvent concerner des niveaux différents:

- analyse des besoins
- phase de conception
- phase de réalisation
- affinement de la phase de tests pour des cas précis

Attention! A l'issue des phases finales lors de la livraison du logiciel le plus difficile est de convaincre le client que tous les besoins sont satisfaits et que l'étape de maintenance ne concernera que ce qui a été spécifié....

1.2. Quelques modèles et méthodes

- Le modèle en V
- Le modèle prototypal
- Le modèle incrémental
- Le modèle en spirale
- Les méthodes agiles
  - Méthodes de développement rapide d'applications - RAD
  - Méthode eXtreme Programming - XP
- Le modèle UP, RUP

C. Crochepeyre Génie Logiciel – Rappels 12

Après avoir décrit rapidement le processus d'élaboration d'un logiciel, il s'agit de s'interroger sur **la méthode à suivre** pour aboutir à un résultat satisfaisant.

Des modèles, des méthodes, des recommandations….diverses et variées sont pratiquées.

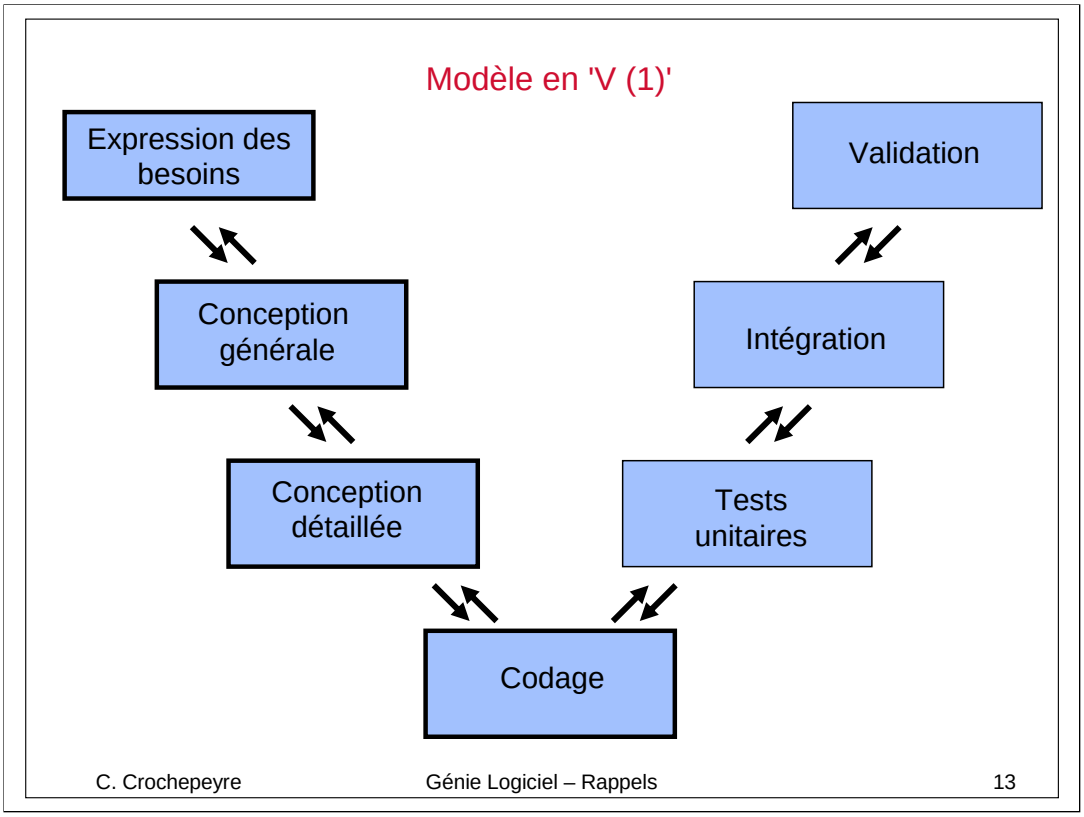
Leur utilisation dépend des applications, des compétences des intervenants, du temps dont on dispose pour analyser et produire le logiciel.

De nombreuses raisons peuvent être à l'origine du choix d'une méthode.

Les modèles les plus anciens, en V et en cascade sont essentiellement basés sur un enchaînement chronologique des étapes : l'étape 2 ne se fait qu'après avoir terminé l'étape1. Le modèle en V ajoute au modèle en cascade, la possibilité de revenir à une étape antérieure.

Des modèles en spirale, incrémental proposent une construction évolutive par itérations successives. Le modèle UP (Unified Process) est plus approprié à l'approche objet et utilise les outils UML.

Des méthodes rapides, dites agiles, sont retenues lorsque l'application n'est pas d'une grande complexité et que l'on souhaite produire au plus vite l'application. Une collaboration étroite avec l'utilisateur permet de procéder de cette manière.



La représentation la plus courante du développement d'un logiciel est une représentation du **cycle de vie en V** comme 'Vie'.

Représentation qui traduit deux niveaux:

1- Niveau cycle de vie du système

A gauche: Expression des besoins et Conception préliminaire

A droite: Validation/ Maintenance et Intégration matériel/ logiciel

2- Niveau cycle de vie du logiciel

A gauche: Conception détaillée

A la base: Codage

A droite: Tests d'Intégration et Tests Unitaires

Les étapes :

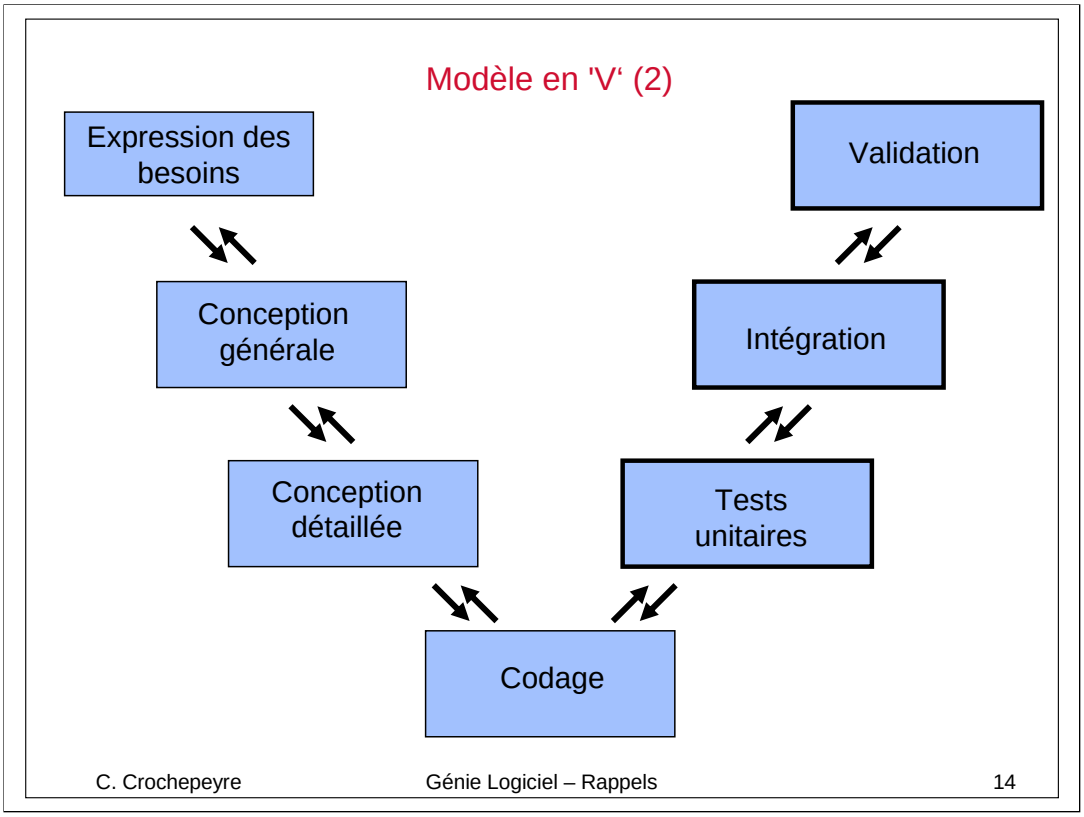
- **Expression des besoins** : l'utilisateur exprime ses besoins en termes de fonctionnalités. Il fournit les informations et les règles de traitement pour chacune des fonctions qu'il souhaite intégrer dans l'application. Il décrit les résultats attendus.

Cette phase est très importante car elle détermine le travail des étapes suivantes. Rappelons que ce modèle en V procède par étapes successives.

- **Conception générale** : c'est à partir de l'expression des besoins que l'on construit un modèle général de l'application informatique. On construit un système d'informations unique qui exige une étude approfondie des données, de leurs dépendances, de leurs volumes, de leurs utilisations….On construit une architecture logicielle en décomposant les fonctions en sous fonctions avec des solutions de communications, des modèles de documents à produire, des saisies de données, des procédures de traitements pour appliquer les règles ,…..

- **Conception détaillée** : Après l'étude de la conception fonctionnelle, l'étape de la conception détaillée permet d'affiner les solutions informatiques de chaque unité de programme à produire.

- **Codage** : la programmation est l'aboutissement de l'étude des besoins. On met en œuvre le produit informatique en choisissant les bonnes compétences et les bons langages de programmation.

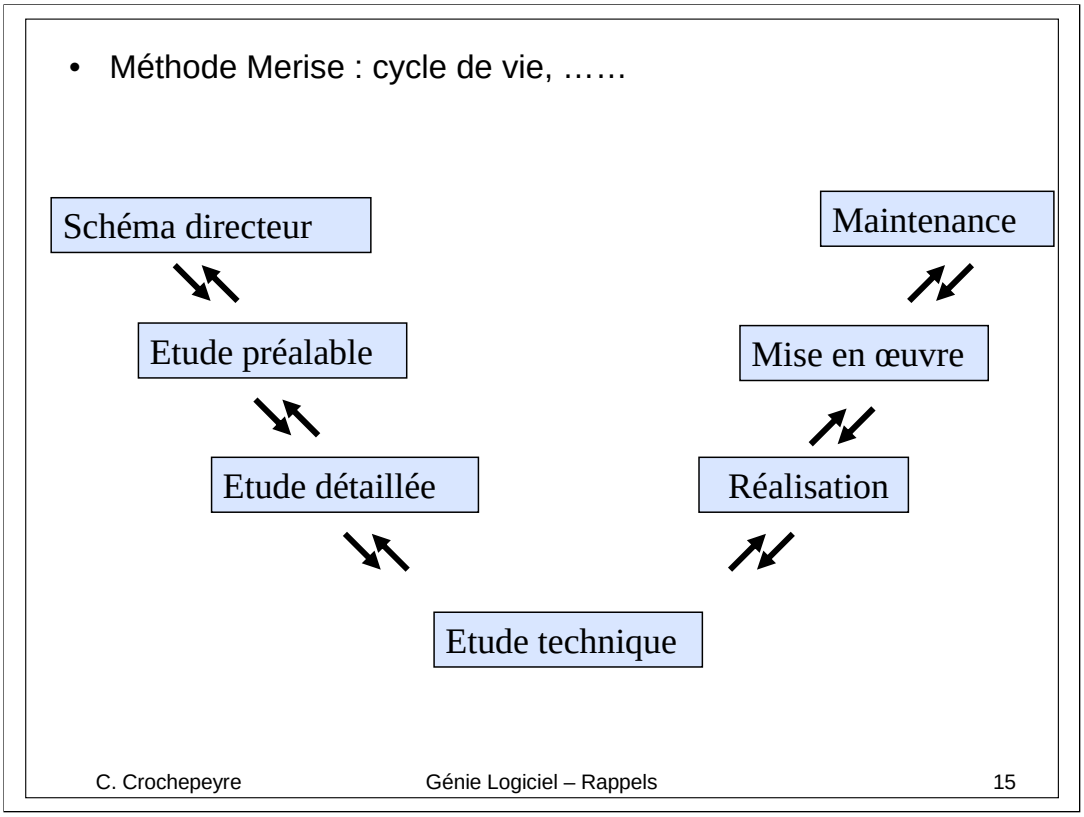


- **Tests unitaires** : chaque unité de programme est vérifiée. Les résultats produits doivent correspondent aux spécifications précédentes

- **Intégration** : les unités de programmes sont alors associées et communiquent entre elles. C'est alors que l'on peut vérifier que la logique de production fonctionne et que les résultats sont satisfaisants.

Lors de ces deux étapes de tests, des jeux de tests rigoureux sont construits.

- **Validation** : l'utilisateur est sollicité car il va mettre à l'épreuve le nouveau logiciel.



Ici un exemple de la première version de la méthode Merise dont les étapes du cycle de vie d'un logiciel sont un peu différentes mais dont la représentation en V est respectée.

Des diagrammes sont associés aux étapes :

MCD : modèle conceptuel des données

MCT : modèle conceptuel des traitements

MOT : modèle organisationnel des traitements

MCC : modèle conceptuel de la communication

MLD : modèle logique des données

MLT : modèle logique des traitements

MPD : Modèle Physique des Données ……

- Méthode Merise : ….cycle de décision, cycle d'abstraction

- Décisions à chaque phase
  - Analyser, décider, justifier et consigner les décisions avant de passer à une autre étape

- L'abstraction du niveau conceptuel au niveau physique
  - Conceptuel: modéliser l'entreprise et ses activités (MCT+MCD)
  - Logique: modéliser la solution informatique (MLT+MLD)
  - Physique: choix techniques informatiques (MPD+MOT)

- Méthode ayant évoluée depuis les années 70 :
  - Merise, Merise/objet, Merise/2

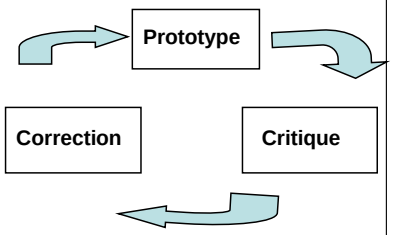
C. Crochepeyre Génie Logiciel – Rappels 16

La méthode Merise est une méthode d'analyse et de conception :
- a chaque étape, il s'agit d'analyser, mesurer, décider, conserver les arguments des choix
- elle est applicable du niveau conceptuel au niveau physique

Merise a évolué depuis 1980:
- d'un modèle classique en V
- en passant par une approche objet Merise/objet
- et enfin la version Merise/2 qui prend en compte l'architecture de communication client/serveur.

Le modèle prototypal (évolutif)

- Partir d'un prototype
  - On recense les 1er besoins, on conçoit, on développe version v1
  - On propose, on critique
  - On améliore, on enrichit
  - On aboutit à une version v2, etc…



- Ce qui suppose**:**
  - Une analyse graduelle des besoins
  - Une évolution de la conception de l'architecture logicielle
  - Un développement qui aboutit au produit final ou seulement une étape préliminaire

C. Crochepeyre Génie Logiciel – Rappels 17

Le modèle prototypal est un modèle évolutif.

Contrairement au modèle en V, on construit un prototype fonctionnel à partir des besoins énoncés.

On le développe et on le propose à l'utilisateur qui apporte ses critiques.

On l'améliore et le complète, ce qui donne une nouvelle version, etc….

En général on procède de manière graduelle en considérant au fur et à mesure les besoins des utilisateurs et en les affinant.
On aboutit ainsi à une version du prototype qui peut être le logiciel final ou bien n'être qu'une étape en vue d'un développement complet du logiciel.

Le prototypage

- approche évolutive du développement
  - Le prototype s'enrichit au fur et à mesure
  - Le prototype est réservé pour certains logiciels (ou parties)
  - Le prototype fonctionne !
- avantages:
  - cas concret de discussion
  - détection des fonctions manquantes
  - amélioration des fonctions complexes
  - démonstration de la faisabilité
  - utilisation comme spécification d'un système
- inconvénients du prototypage:
  - coût de développement
  - incite les changements côté utilisateur

C. Crochepeyre Génie Logiciel – Rappels 18

L'avantage de cette approche est de pouvoir travailler en collaboration étroite avec l'utilisateur qui voit rapidement les premiers résultats de l'application.

Même si toutes les fonctionnalités ne sont pas développées dans le prototype, il fonctionne.

C'est pourquoi, on développe en premier les fonctions dont les spécifications ne sont pas bien définies.

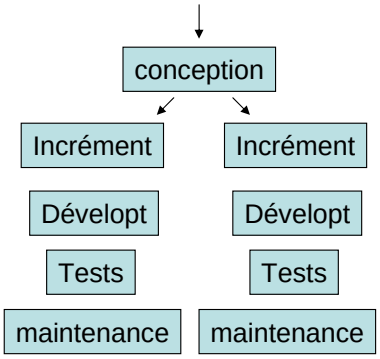
Cette approche peut être lourde donc elle est réservée pour certaine fonction de l'application

Les avantages sont nombreux : comme la détection de certains oublis, la compréhension entre l'utilisateur et le développeur pour des fonctions complexes…Il est plus facile de travailler sur un exemple concret.

En revanche il faut savoir limiter les besoins de l'utilisateur qui peut facilement demander plus qu'il n'était prévu !

Le modèle incrémental (évolutif)

- La base du modèle: le prototype
- Découpage fonctionnel en sous-ensembles
- Développement des sous-ensembles par incrément
- A chaque incrément on repasse par toutes les étapes
- Le développement est essentiellement basé sur l'analyse des résultats pour faire évoluer le produit
- Un incrément = 1 version
- Réutilisation de modules élémentaires



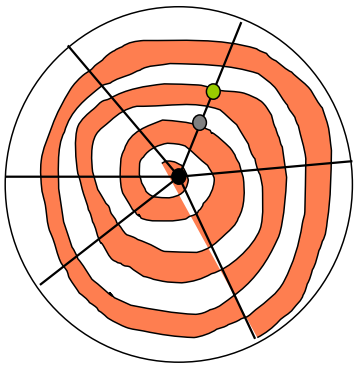
C. Crochepeyre Génie Logiciel – Rappels 19

Le modèle incrémental est une variante du modèle prototypal : un découpage de fonctions et sous fonctions permet de faire des développements par incrément dans chaque sous fonction.

Une version correspond à un cycle : développement – test - maintenance. Ainsi après les résultats obtenus on peut faire évoluer le prototype mais cette manière de procéder permet de réutiliser des modules élémentaires et de faire travailler plusieurs personnes sur la même application.

Le modèle en spirale (évolutif)

- On part de rien
- On part d'une spécification
- On modifie un existant



- Relation contractuelle entre le client et le développeur
- Représenté sous forme de spirale, chaque itération est découpée en phases :
  - Détermination des besoins
  - Analyse des risques,
  - Développement d'un prototype,
  - Tests du prototype, résultats
  - Validation des besoins par le client,
  - Planification du prochain cycle.

C. Crochepeyre Génie Logiciel – Rappels 20

Le modèle en spirale est une autre forme du développement évolutif. Il prend en compte les risques à chaque itération de la spirale.

On part des besoins exprimés, on analyse les risques, on choisit les solutions sur des prototypes ou maquettes puis on passe au développement de la solution retenue. A la vue des résultats, le client valide la version et on peut ensuite repartir sur un cycle de la spirale.

Barry Boehm (en 1988), en proposant ce modèle, préconise ainsi une série de prototypes qui permet de mesurer les risques et de mieux contrôler le développement du logiciel.

Les méthodes agiles

- Réactivité: Implication au maximum du client
- Rapidité
- Dans ce but, elles prônent 4 valeurs fondamentales
  - **L'équipe** « Personnes et interaction plutôt que processus et outils ». La communication est une notion fondamentale.
  - **L'application** « Logiciel fonctionnel plutôt que documentation complète ». Il est vital que l'application fonctionne.
  - **La collaboration** « Collaboration avec le client plutôt que négociation de contrat ». Le client doit être impliqué dans le développement.
  - **L'acceptation du changement** « Réagir au changement plutôt que suivre un plan ». La planification initiale et la structure du logiciel doivent être flexibles (évolution de la demande du client)

C. Crochepeyre Génie Logiciel – Rappels 21

Les méthodes agiles datent des années 90, elles sont issues des approches précédentes mais ont pour particularité d'impliquer au maximum le client afin de réduire le temps de conception. En faisant participer étroitement le client à l'élaboration du produit, on peut :

-Réagir rapidement en fonction des besoins du client : changement

-Le client travaille étroitement avec le développeur : collaboration

-Le logiciel lui-même reflète les besoins du client : la documentation est allégée

-La distinction entre MOA et MOA existe mais les deux travaillent en équipe.

Ainsi cette méthode offre une grande place à la réactivité des acteurs et vise une rapidité de conception pour un résultat tout aussi satisfaisant qu'avec d'autres approches.

Les procédures de documentation, du recueil des besoins, etc… ne sont plus nécessaires et l'on produit un logiciel tout aussi efficace!

Bien entendu cette approche est plus appropriée lorsque le client est prêt à s'investir et que l'application envisagée n'est pas d'une grande complexité.

- **RAD (agile)**
  - Développement rapide d'application avec une implication importante du client
  - Phases MOA:
    - Besoins, contraintes
    - Prise en compte des changements
    - Recette
    - démarrage
  - Phases MOE:
    - Initialisation: organisation
    - Cadrage: objectifs et moyens
    - Design: modèle de la solution
    - Construction: développement par prototypage
    - Finalisation: livraison et contrôle qualité
  - Le GAR Groupe d'Animation et de Rapport se charge de l'animation et de la formalisation des informations

C. Crochepeyre Génie Logiciel – Rappels 22

Ici l'exemple de la méthode RAD (Rapid Application Development)

La maîtrise d'ouvrage **MOA** remplit son rôle en exprimant ses besoins et ses contraintes. Elle participe activement aux changements qui font évoluer les versions iératives vers le produit final. Elle doit être active lors des recettes successives ce qui prend du temps. En final, la MOA peut lancer le démarrage car elle connaît son produit.

La maîtrise d'oeuvre **MOE**

- Définit l'organisation du travail en décomposant les tâches, en choisissant les acteurs et en délimitant le projet.
- Cadre précisemment les objectifs du client en recueillant ses exigences (ses besoins) et en se donnant les moyens
- Concoit et modélise les données, les flux, les traitements … avec l'aide du client
- développe la solution par approches successives de prototypes et demande au client de valider chaque étape
- la solution finale est l'ultime étape du développement lorsque les objectifs sont atteints

Le GAR est essentiel puisqu'il anime le travail de collaboration entre MOA et MOE et se charge de l'élaboration des rapports.

- **Extreme Programming XP (agile)**
  - Principe de base: « pousser à l'extrême ce qui marche » avec une implication importante du client
  - Rapidité de développement par cycle
    - déterminer les scénarii clients
    - transformer les scénarii en fonctions à réaliser et en tests fonctionnels
    - chaque développeur s'attribue avec un binôme des tâches
    - Lorsque les tests sont concluants, l'application est livrée
    - Le cycle se répète tant que le client peut fournir des scénarii.
  - Réactivité aux changements, qualité du code et des tests, travail en équipe

C. Crochepeyre Génie Logiciel – Rappels 23

Autre aspect de cette approche de développement rapide: pousser à l'extrême ce qui marche. Projet qui débute en 1996, XP est une méthodes agile qui énonce 5 règles simples:

- planification: demander aux clients d'écrire les scenari des procédures, créer le planning de développement de chaque scenario avec un nombre de semaines et mettre en commun autour d'une table l'ensemble des scenari. Le principe d'itérations est appliqué et à après chaque discussion, une nouvelle version est adoptée. Il s'agit toutefois de déterminer combien d'itérations sont possibles en fonction de la durée du projet.

- gestion: la communication est si importante que XT préconise un espace commun de travail pour l'équipe. Il est nécessaire dans cette démarche de contrôler et de mesurer le temps passé par scenario, par itération…. Les procédures sont développées en fonction des compétences de l'équipe mais les échanges et les binômes de travail sont recommandés.

- construction : les modules créés doivent être simplifiés au maximum. Ainsi il est préférable de faire deux procédures simples au lieu d'une seule complexe. Ne pas ajouter de nouvelles fonctions trop tot. De nombreuses recommandations existent concernant la construction.

- codage: créer les unités de tests avant d'écrire le code exige de réfléchir sur ce que l'on doit produire. Une autre idée est de programmer par binôme sur un seul ordinateur, un seul écran… Les échanges de compétences sont très efficaces.

- tests: comme toute application , les phases de tests unitaires puis d'intégration sont très importantes.

Le modèle UP (évolutif)

- cycle à 4 phases:
  - étude d'opportunité: faisabilité et risques
  - Élaboration
  - Construction: prototype
  - Transition: final ou nouveau cycle
- modélisation UML : architecture du logiciel (fonctionnelle, logicielle et physique) et cas d'utilisation pour recenser les besoins.
- La méthode RUP suit ce modèle et propose des outils

Le modèle UP est basé sur le principe de développement évolutif, principalement pour des logiciels orientés objet.

UML et UP se complètent en terme d'apport d'outils de conception et en terme de méthode de conception et de développement

Quatre phases caractérisent ce modèle :

- l'étude d'opportunité dont l'objectif est de mesurer la faisabilité et les risques
- l'élaboration qui propose un modèle logique et matériel qui est basé sur des cas d'utilisations
- la construction de prototypes successifs permet cette démarche de développement évolutif
- la transition est l'étape finale d'une version. Cette version peut être la dernière ou être un point de départ vers une nouvelle version.

UP adopte la plupart des concepts des méthodes agiles. UP n'est pas une méthode générique qui offre un ensemble de concepts qui doivent être adaptés en fonction des cas.

Parmi toutes les méthodes issues de UP, citons RUP de RATIONAL;

1.3. Remarques

- **Les questions au bon moment**
  - Vérification
    - réalisation = bonne construction?
  - Validation
    - réalisation = bon produit?
  - Coût du logiciel
    - à quelle étape est-il plus important?
    - comment procéder pour le réduire?

C. Crochepeyre Génie Logiciel – Rappels 25

Le G.L. doit nous aider à répondre aux deux questions:

Est-ce que nous construisons bien le produit?

Démarche de vérification que le produit en cours de réalisation répond à la définition des besoins.

Est-ce que nous construisons le bon produit?

Validation que les fonctionnalités du produit sont bien celles demandées par le client.

Ces deux préoccupations sont essentielles pour **maîtriser les coûts** de développement du produit.

N'oublions pas que l'objectif du G.L. est de réduire le coût du développement du logiciel.

Pour cela il est important de se demander:

- A quelle(s) étape(s) ce coût risque-t-il d'être important?
- Peut on le réduire?

Les statistiques montrent qu'en général ces **coûts** sont plus importants:

- à l'étape de la conception
- aux étapes de vérification et de validation

c'est à dire **en début et en fin de cycle** de développement.

Comment réduire les coûts importants?

En évitant les retours à ces étapes. Pour cela:

- éviter les changements en étant très précis sur l'expression des besoins
- prévoir des évolutions possibles à toutes les étapes.

Ainsi si le coût est plus important en début de cycle, ce n'est pas un hasard:

-> Le client doit exprimer tous ses besoins

Le développeur doit comprendre, faire une synthèse et demander si nécessaire des précisions ou compléments d'informations.

-> Lors des tests de validation, un changement peut conduire à redéfinir les besoins

-> Lors de la maintenance, le problème est similaire. On constate que bien souvent après installation du produit, les opérations de maintenance sont des demandes de changements dans la définition des besoins et beaucoup moins des corrections d'erreurs.

C'est pourquoi des **modèles** de développement tendent de répondre à cette demande de réduction des coûts.

Pour cela deux types de modèles sont proposés:

- **L'évolution des logiciels**
  - demande des utilisateurs
  - changement de l'environnement
  - versions après corrections
- *Les lois d'évolution (Lehman 1976)*
  - le changement est nécessaire
  - la complexité croissante est inévitable
  - l'évolution est constante et régulière
  - conservation de la stabilité organisationnelle
  - conservation du degré d'évolution

C. Crochepeyre Génie Logiciel – Rappels 26

Nous venons de dire que les changements étaient coûteux (plus que les erreurs).
L'évolution des logiciels n'est toutefois pas facile à prévoir, ni à contrôler:

-> une meilleure compréhension de l'utilisateur peut l'amener à demander des **améliorations**

-> un **changement de l'environnement** peut être aussi une autre raison

Dans tous les cas cela mène à des corrections du produit qui se traduisent par des versions différentes du système.

En 1976, Lehmann, après observations, a énoncé cinq lois sur l'évolution des programmes:

1- Changement continu

Un programme doit changer sinon il perd de son utilité dans l'environnement où il est utilisé. Le logiciel modifie l'environnement car les utilisateurs modifient leur comportement donc les besoins changent....

2- Complexité croissante

Lorsqu'un programme change, il devient de plus en plus complexe (sauf si effort important pour éviter ce phénomène!). les besoins changent et augmentent.

3- Evolution du programme

L'évolution du programme est à régulation automatique.
Des mesures telles que taille, nombre d'erreurs, temps entre deux versions...tendent vers des chiffres significatifs et invariants.

4- Conservation de la stabilité organisationnelle

Le taux de développement d'un logiciel, tout au long de la durée de vie, est presque constant et indépendant des ressources allouées.

5- Conservation du degré d'évolution

Pendant la durée de vie du logiciel, le degré d'évolution entre deux versions est quasi constant.

Ces lois sont valables pour la plupart des logiciels.
Lehmann suggère de rechercher la stabilité et d'éviter les changements soudains et importants.
Il préconise des modifications par incréments avec des **versions fréquentes**.

- **La fiabilité du logiciel** dépend de
  - la qualité de conception
  - la qualité de réalisation

  un logiciel est fiable si il:
  - répond aux spécifications
  - ne produit jamais de résultats erronés
  - n'est jamais dans un état incohérent
  - réagit utilement dans une situation inattendue
  - n'est mis en défaut qu'en cas extrême

  => importance de la phase des tests

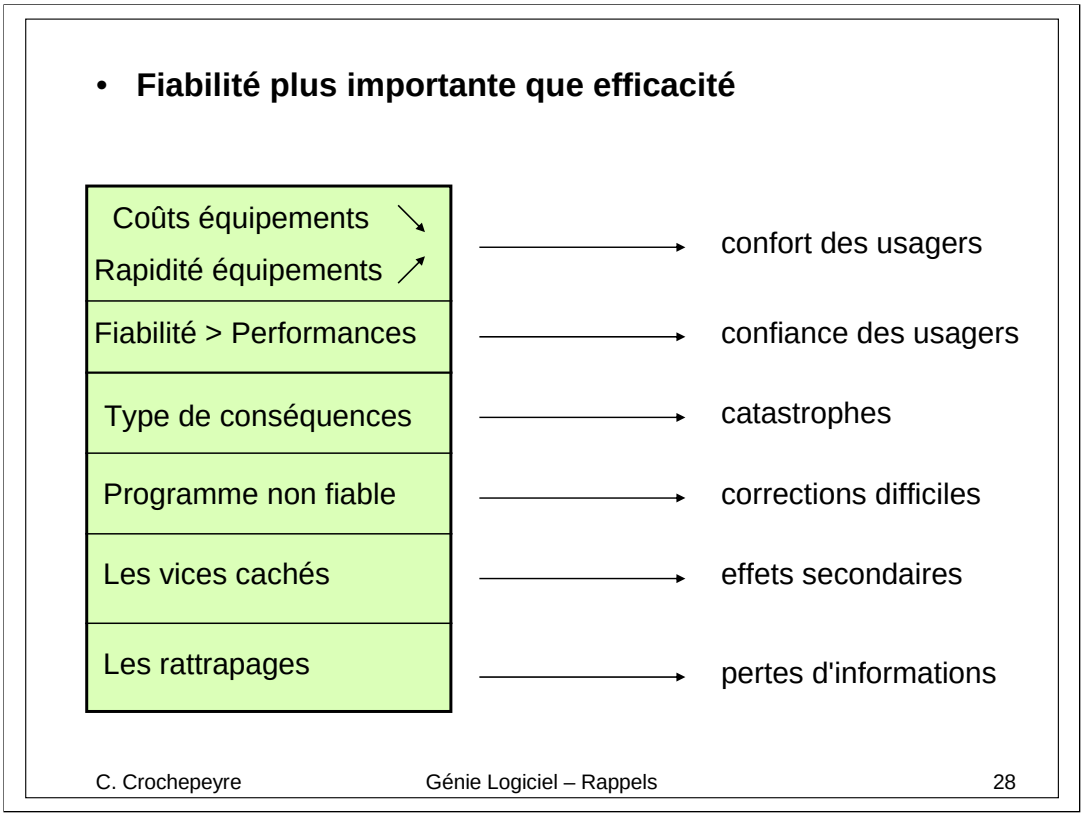
C. Crochepeyre Génie Logiciel – Rappels 27

Si pour une voiture, la fiabilité dépend:
- de la qualité de conception
- de la qualité de fabrication
- de la durée de vie des pièces

Pour les logiciels seuls les deux premiers points entrent en jeu:
-**qualité de la conception**-
-**qualité de la réalisation**

Quels sont les critères prouvant qu'un logiciel est fiable?
•répond aux spécifications
•ne produit jamais de résultats faux
•n'est jamais dans un état incohérent
-réagit utilement dans une situation inattendue
-n'est mis en défaut qu'en cas extrême

Concluons:
Fournir les services attendus par les utilisateurs. En notant que certains services sont extrêmement importants (une seule erreur peut être fatale ex: avionique) et d'autres moins.



Certains critères de qualité du logiciel sont plus importants que d'autres....

- Le **confort**

C'est plus important que l'optimisation des équipements. Ceux ci sont de moins en moins chers, de plus en plus rapides et performants.
Ex: la rapidité d'accès à une information est un confort qui peut être obtenu en dupliquant une information (la capacité d'un disque peut être étendue...)

- Les **performances**

Un usager confronté à des défaillances du logiciel, rejettera le produit alors que des performances contestées l'amèneront à demander une amélioration avant un rejet définitif.

- Les **conséquences** d'un manque de fiabilité

Penser aux conséquences d'un logiciel non fiable. Le coût des dommages peut être supérieur au cours du développement ou à une catastrophe dont les conséquences peuvent être considérables.

- Un programme fiable

Pour aboutir à un tel résultat, il faut **déceler toutes les erreurs**. Ces erreurs ne sont pas toujours visibles et peuvent apparaître après plusieurs utilisations.

- Les **vices cachés**

Une erreur peut en cacher une autre. Les effets secondaires sont parfois importants.

Ex: perte de données

- Les **rattrapages**

Penser aux reprises. Les dégâts tels que les pertes d'informations doivent pouvoir être réparés. On pensera aux duplication des données.

Il est très important de prendre conscience du type d'environnement dans lequel le logiciel est utilisé.

PLAN

- 1. CYCLE DE VIE DU LOGICIEL
- **2. EXPRESSION DES BESOINS**
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- 5. TESTS ET MISE AU POINT
- 6. DOCUMENTATION
- 7. CONCLUSION

Nous venons de voir les problèmes rencontrés lors du développement d'un logiciel et les conséquences qui peuvent s'en suivre si ce développement n'est pas bien contrôlé.

Les différentes étapes de ce développement vont être détaillées une à une.

Tout d'abord: l'expression des besoins.

2. EXPRESSION DES BESOINS

- **Analyse et définition des besoins du système**
  - étude des besoins et de la faisabilité
  - Acteurs: le client, l'analyste-ingénieur informatique
    - définitions des fonctionnalités
    - recensement des données
  - Pas de solution technique approfondie à ce niveau: on ne définit que les besoins en terme de fonctions métiers et on apprécie la faisabilité informatique

C. Crochepeyre Génie Logiciel – Rappels 30

L'analyse et la définition des besoins permet de:
- définir les fonctionnalités du système
- recenser les contraintes auxquelles il est soumis

A travers cette première étude, on confirme la faisabilité de l'informatisation du système
C'est la première étape du cycle de vie du logiciel

L'utilisateur qui est demandeur décrit dans un cahier des charges les besoins du système. Ce document sert de référence
. pour faire un appel d'offres à une société de services
. pour faire une demande au service développement informatique au sein de l'entreprise
L'utilisateur peut aussi faire appel à un informaticien qui l'aidera à formuler sa demande.

Le **développeur** est un informaticien: analyste ou ingénieur informaticien qui prend connaissance de la demande et doit ensuite réaliser l'application.

Cette phase concerne l'étude des fonctionnalités du système:
- recensement des règles de gestion
- recensement des données nécessaires aux traitements demandés
- sans oublier les contraintes liées à ce système.

Important!
On ne parle pas à ce niveau de solution technique.

- **Les éléments de cette étape**:
  - Document: cahier des charges
  - Définition des besoins fonctionnels: modèle conceptuel
  - Recensement des données
  - Définition des besoins non fonctionnels
  - Solution fonctionnelle
  - Validation

C. Crochepeyre Génie Logiciel – Rappels 31

Le document constitué à cette étape est appelé:

- document des définitions des besoins ou

- cahier des charges ou

- étude préalable

Ce document décrit ce que le système doit faire sans spécifier comment le faire. Ce doit être un document complet et cohérent.

- Le **modèle conceptuel**

C'est une vue d'ensemble des fonctionnalités et des relations entre les fonctions.
Plusieurs représentations sont possibles (modèles et méthodes différentes)

- Les **besoins fonctionnels**

La plus grande partie du cahier des charges concerne la définition des besoins fonctionnels, c'est à dire les 'spécifications fonctionnelles'

- Le **recensement des données**

Les données qui sont liées aux besoins fonctionnels sont recensées simultanément. On s'intéressera à la nature de ces informations, leurs liens (associations), leur volume, les règles qui les contrôlent….

Si un système d'informations existe il faudra en tenir compte.

Comment les spécifier dans le cahier des charges?

- Les **besoins non fonctionnels**

Le système est soumis à des restrictions ou contraintes que l'on recense dans le cahier des charges comme des besoins non fonctionnels.
Ex: contrainte de temps de réponse, restriction sur la présentation des données

- La **solution fonctionnelle**

En concertation avec le client, on propose une solution fonctionnelle qui doit répondre aux étapes précédentes. Il se peut que certaines restructurations, en accord avec les responsables, soient prises en compte dans cette solution.

- La **validation des besoins**

La précision et la complétude de l'expression des besoins, comme nous l'avons dit précédemment, sont très importantes.
Cette étape doit être faite avec sérieux avant de passer aux étapes suivantes.

Contenu du document

- Introduction
- Présentation du contexte, de l'existant
- Besoins fonctionnels
- Recensement des données, flux
- Besoins non fonctionnels, contraintes
- Approche de la solution
- Besoins en matériels

C. Crochepeyre Génie Logiciel – Rappels 32

Un document de définition des besoins (cahier des charges) doit satisfaire six critères:

- doit spécifier uniquement le **comportement externe** du système
- doit spécifier les **contraintes de réalisation**
- doit servir d'outil de **référence aux programmeurs**
- doit contenir des indications pour les **étapes suivantes** du développement
- doit spécifier les **réponses acceptables** aux événements non désirables
- doit être **facile à mettre à jour**

il doit être facile de retrouver les informations

=> Table des matières
Index
Glossaire des termes employés

On doit passer du temps à construire un tel document .

- **Introduction générale**

Présentation générale de l'application et du contenu du document

- Présentation du contexte et de l'existant

Avant de commencer toute analyse des besoins, il est prudent de faire un état des lieux de ce qui existe, ce qui est performant, ce qui dysfonctionne…

- **Besoins fonctionnels**

On décrit les services demandés par l'utilisateur, pour cette description on peut utiliser des formes d'expressions différentes:

. langage naturel

. langage semi-formel

. langage formel

- **Besoins en bases de données**

Selon la nature et l'importance des données qui sont traitées, celles ci sont généralement stockées et organisées dans des B.D.

Pour que cette organisation corresponde à celle des données manipulées, il faut une description de l'organisation logique des données et leurs relations

- **Besoins non fonctionnels**

Ces besoins sont des contraintes (ou restrictions) liées au système

Norme IEEE std 830 cahier des charges

I- Intoduction

II- Contexte de la réalisation

1. Objectifs
2. Hypothèses
3. Bases méthodologiques

III- Description générale

1. Environnement du projet
2. Fonctions générales du système
3. Caractéristiques des utilisateurs
4. Configuration du système

C. Crochepeyre Génie Logiciel – Rappels 33

Le contenu du document de définitions des besoins est décrit dans la norme IEEE Std 830.

Cette description n'est pas parfaite car elle propose une description statique des besoins.
On y trouve:
I. Introduction
Présentation succincte de la société et de son domaine d'activité
Présentation générale du projet et de sa situation au sein des activités de l'entreprise.
II. Contexte de la réalisation
Objectifs et organisation du travail
III. Description générale
Environnement et fonctions générales du système

5. Contraintes générales du développement, d'exploitation et de maintenance
   - Contraintes de développement
   - Contraintes d'exploitation
   - Maintenance et évolution du système

IV- Description des interfaces externes du logiciel
1. Interface matériel / logiciel
2. Interface homme / machine
3. Interface logiciel / logiciel

V- Description des objets
1. Définition des objets
   i. Identification de l'objet -i
   i. Contraintes sur l'objet i

C. Crochepeyre Génie Logiciel – Rappels 34

IV. Description des interfaces externes du logiciel

Interfaces et communications entre l'usager, le logiciel et le matériel

V. Description des objets

Pour chaque objet ou entité, sa nature, ses propriétés et ses contraintes.

VI- Description des fonctions
1. Définitions des fonctions
i. Identification de la fonction i
Description de la fonction i
Contraintes opérationnelles sur la fonction i
2. Conditions particulières de fonctionnement
2.1. Performances
2.2. Capacités
2.3. Modes de fonctionnement
2.4. Contrôlabilité
2.5. Sûreté
2.6. Intégrité
2.7. Conformité aux standards
2.8. Facteurs de qualité

VI- Description des fonctions

Comme pour les objets plusieurs rubriques concernent l'identité de la fonction, sa description et ses contraintes

Dans le cas de fonctionnement particulier, les caractéristiques de ces fonctionnement sont décrits avec précision

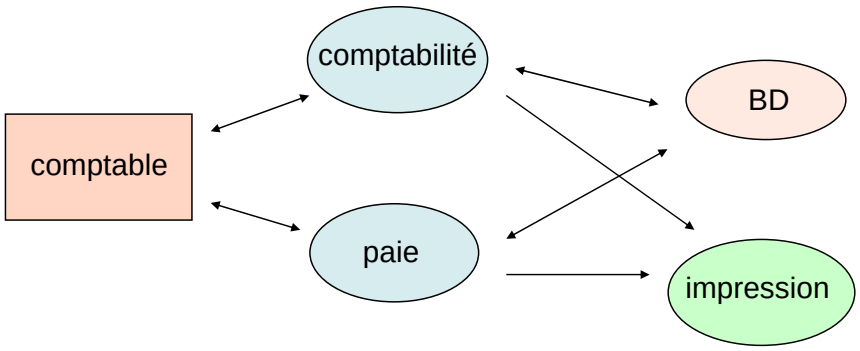
VII- Justification des choix effectués
VIII- Glossaires
IX- Références
1. Annexes
2. Index

Ou Afnor Z67-100-3

En final, le cahier des charges propose des solutions qui doivent être argumentées. La solution retenue est justifiée

Les glossaires et index sont des outils de références fortement utilisés dans ce document de recensement des besoins.

- **Le modèle conceptuel du système: approche fonctionnelle**
  - exemple des automates finis: commande, fonctions et données
  - facilité d'une représentation graphique des fonctions que l'on affine d'étape en étape



C. Crochepeyre Génie Logiciel – Rappels 37

Après avoir présenté le cahier des charges ou document de définitions des besoins, passons à la description du **modèle du système**.

Ce modèle décrit les fonctions principales utilisateur et leurs relations

Pour des applications très simples, le modèle est superflu. En revanche pour des applications plus complexes un modèle mental est insuffisant.

En 1976, Salter utilise des automates finis pour modéliser le système avec les fonctions, la commande et les données où:

- les fonctions transforment l'information

- les données sont les entrées et les sorties des fonctions

- la commande est le mécanisme qui active les fonctions dans l'ordre désiré.

La notion d'état et de transition est existante dans les modèles de Yeh et Zave (1980) et de Heninger (1980)

Le modèle conceptuel est représenté graphiquement.
Dessins et diagrammes sont compris par tous.

Après une présentation du modèle conceptuel général, on définit un modèle conceptuel pour chaque fonction principale.

Le modèle conceptuel peut apporter des éléments complémentaires à la définition des besoins.

Il peut à l'inverse être incomplet par manque d'informations de la part du réalisateur du modèle conceptuel.

Cette approche graphique est loin d'être abandonnée: cf UML

Pour chaque fonction on représente graphiquement un modèle conceptuel.
Ici le modèle conceptuel de la PAIE décrit l'ensemble des sous fonctions qui le composent. Les flèches matérialisent les flux d'informations entre ces fonctions.

Une entité: le comptable et des fonctions concernées par cette entité: la comptabilité, la paie.

Les données associées qui sont des éléments en entrée, dans les calculs ou en sortie des processus. Ces données sont stockées ici dans une base de données.

L'impression permet de visualiser des résultats.

- **Définitions des besoins fonctionnels**
  - définition des services
  - complétude et cohérence
  - modifications lors des étapes ultérieures

  trois méthodes
  - langage naturel
  - langage structuré ou formaté
  - langage de spécification formelle

La ‘spécification fonctionnelle’ du système correspond à la définition des besoins fonctionnels lors du recensement des besoins.

C’est la partie la plus importante de ce cahier des charges.

Besoins fonctionnels = services attendus

L’utilisateur n’a pas besoins de savoir comment réaliser ces services en informatique. Il n’y aura donc pas de présentation de la solution technique dans ce document.

Attention!

La qualité du cahier des charges à ce niveau dépend:

- de la complétude et cohérence des la description des besoins fonctionnels

- de la non contradiction entre les besoins.

Ceci s’obtient au fur et à mesure des corrections, ajouts, modifications des informations.

Il existe 3 façons d’exprimer les besoins fonctionnels:

- par un langage naturel

- per un langage structuré formaté

- par un langage de spécification formelle

Langage naturel

- très usité (compréhensible par tous)
- présentation par paragraphes numérotés

2.1. Paie
Cette fonction comporte trois fonctions:
. la saisie des éléments de paie par employé
. l'édition des bulletins
. la mise à jour de la comptabilité

2.1.1. La saisie
Pour chaque employé, un certain nombre de paramètres sont saisis...

C. Crochepeyre Génie Logiciel – Rappels 39

1ère méthode pour exprimer les besoins fonctionnels: le **langage naturel.**

S'exprimer en français apporte l'avantage de décrire les besoins par un langage:

- expressif (description, précision, exceptions....)
- **connu de tous** (utilisateurs et développeurs)
- facilement modifiable

Beaucoup d'avantages!

Comment s'exprimer?

En principe on classe ses idées en rédigeant des paragraphes numérotés et présentant le texte avec des indentations, des mots soulignés, des chapitres, des sous chapitres....

Bref, **un document textuel mais structuré.**

Cet exemple de la PAIE montre une description textuelle structurée des différentes fonctions.

Contre:

- ambiguïté linguistique
- manque de concision: schéma fonctionnel
- difficulté de distinction entre besoins fonctionnels, non fonctionnels et buts
- difficulté de distinction de complétude et cohérence

Pour:

- compréhensible par l'utilisateur et l'analyste
- facilité de rédaction

C. Crochepeyre Génie Logiciel – Rappels 40

Cette méthode a toutefois de gros défauts:

- **L'ambiguïté** des phrases, des mots ...

L'interprétation différente du lecteur et du rédacteur de certaines expressions ou contenu de paragraphe.

Cette ambiguïté peut passer inaperçue, chacun étant persuadé avoir compris ce que l'autre a dit ou écrit.

- **Le manque de concision**

est évident si on compare une expression écrite à une expression graphique.

Un schéma fonctionnel peut être complémentaire, limiter le texte et éviter ainsi de longs discours.

- **Distinction difficile** entre la nature des besoins et leur but

Dans une même phrase, naturellement on peut être amené à parler de besoins fonctionnels, des besoins non fonctionnels (les contraintes associées) et le but à atteindre (la justification de ces choix).

- **Vérification, complétude et cohérence**

L'écriture naturelle n'incite pas à une rédaction complète des concepts et à la cohérence. En effet on ne peut exprimer en une seule phrase tout ce qui doit être écrit.
N'ayant aucune structure formatée au départ tout doit être écrit, dans le bon ordre et sans contradiction.

Ce n'est pas chose facile! Et encore moins à vérifier: quoi vérifier? où? avec quelle méthode?

N'oublions pas le côté positif : pas d'apprentissage car compréhensible par tous.

Recommandations:

- faire relire et corriger le document
- rédaction séparée par besoin
- 1 paragraphe = 1 idée
- polices de caractères différentes

- utilisation pour la présentation de haut niveau
  - facilité d'expression des besoins fonctionnels
  - difficulté d'expression des besoins logiciels

C. Crochepeyre Génie Logiciel – Rappels 41

La rédaction du cahier des charges en langage naturel est certainement la méthode la plus utilisée.

Il est fortement recommandé de:

- faire relire et corriger le document par les utilisateurs, par des informaticiens non spécialistes du domaine d'activité du système et au contraire par des spécialistes. En quelques mots: par tous et à tous les niveaux.

- lors de la rédaction, penser toujours à une rédaction séparée des différents besoins.

- ne pas mêler des idées différentes dans un même paragraphe.

- ne pas hésiter à utiliser une présentation, des polices de caractères qui mettent en évidence les éléments essentiels du document.

Cette méthode est surtout utilisée pour une présentation de haut niveau (cahier des charges, spécification fonctionnelle)

En revanche, pour les étapes de description des besoins logiciels, on n'utilisera pas le langage naturel.

Langage structuré

- utilisation limitée du langage naturel
- utilisation de notations graphiques

SADT

- Technique d'analyse structurée
- Graphique: structure et relations entre entités
- Symboles spéciaux faciles de compréhension
- Pas de traitement automatique des diagrammes

C. Crochepeyre Génie Logiciel – Rappels 42

Autre méthode pour exprimer les besoins fonctionnels dans le cahier des charges: **le langage structuré**

Si le langage naturel est le plus approprié pour les descriptions de haut niveau, il est moins facile d'exprimer une architecture fonctionnelle, les interactions entre les fonctions, de distinguer des besoins proches mais différents sans ambiguïté.

Le langage structuré est une composition

- du langage naturel limité dans son utilisation
- des notations graphiques plus expressives

Un exemple de langages structuré: SADT de Schoman et Ross (1977)

La technique de SADT est une 'Analyse Structurée' AS ou SA

SADT : Structured Analysis and Design Technique

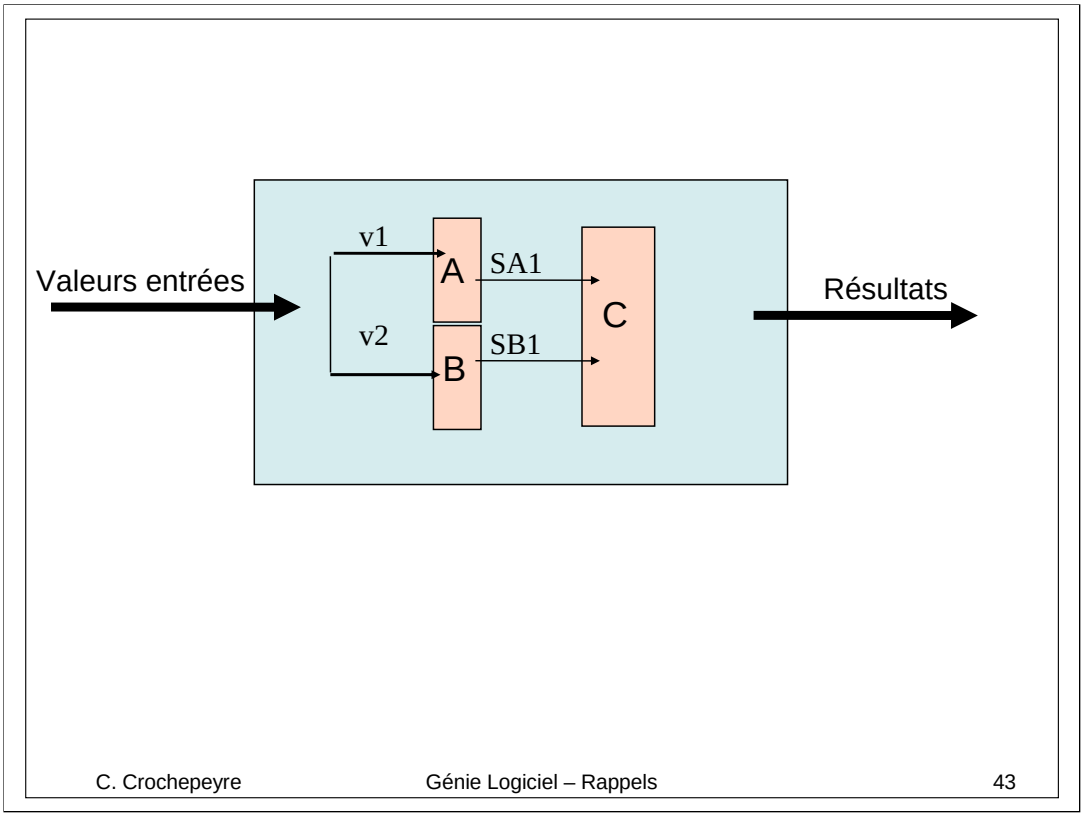
Cette méthode qui a été très utilisée dans les années 80/90.

-La technique d'analyse repose sur la décomposition du problème en partant du général pour aller au particulier : analyse fonctionnelle descendante. La notion de niveaux est à la base de SADT.

-Des diagrammes SADT accompagnent cette analyse. La représentation graphique offre l'avantage de matérialiser les flux des échanges entre les processus par des flèches

-Le langage de représentations SADT est intuitif ce qui permet une compréhension rapide et visuelle du processus. Cependant la notion de temps n'est pas prise en compte.

-Il n'y a pas de langage de traitement automatique des diagrammes pour produire un exécutable.



Le graphisme est essentiel dans SADT, il représente:

- la structure des entités
- les relations entre ces entités

Il y a 40 symboles différents.
Les diagrammes SADT utilisent des symboles clairs et facilement lisibles.
En revanche il n'y a pas de traitement automatique vérifiant la cohérence et la complétude des informations.
(comme dans d'autres méthodes comme RSL et PSL/PSA)

Exemple de diagramme:

Une fonction a un flux de données en entrée (V1 et V2) et produit des résultats.

L'intérieur du diagramme fait apparaître les sous fonctions (A, B et C) avec leurs valeurs en entrée et les résultats après traitements.

Langage spécification formelle

- exemple : langage ADA avec de nouvelles règles
- usage des commentaires

```
package PAIE is
        procedure SAISIE_PAIE
        procedure EDITION_PAIE
        procedure COMPTA_PAIE
end PAIE
```

```
package PAIE_CTES
        PLAFOND_A: constant:= 2600
        PLAFOND_B: constant:= 10400

        ....
```

C. Crochepeyre Génie Logiciel – Rappels 44

Troisième type de langage pour définir les besoins fonctionnels: **le langage de spécification formelle**
Prenons l'exemple d'ADA auquel on ajoute de nouvelles règles et on assouplit celles existantes.

ADA est ainsi transformé en langage de définitions des besoins.
Il faut toutefois faire usage intensif de commentaires pour le rendre expressif!
exemple de restrictions d'ADA
- pas de ; en fin de ligne (CR suffit)
- les clauses 'WITH' et 'USR' peuvent se trouver n'importe où dans une unité de définition.

Attention:
Avec ce type de langage d'analyse ne pas tomber à un niveau trop proche du langage de programmation.....

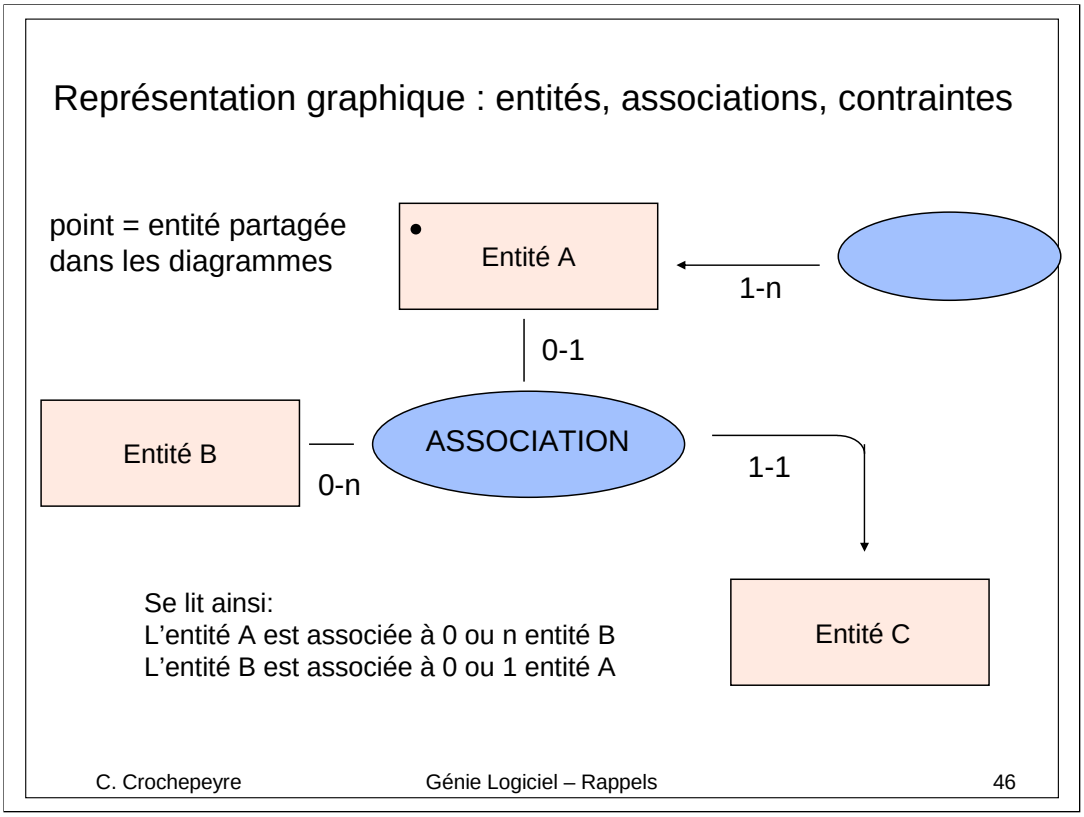
L'exemple proposé permet d'exprimer qu'une fonction « Paie » est composée de 3 sous-fonctions: « la saisie », « l'édition des paies » et « la mise à jour de la comptabilité »
L'autre exemple de package permet de déclarer des constantes comme les plafonds pour le calcul des cotisations

- **Organisation des données**
  - modèle Entité-Association (E-A)
  - aspect conceptuel des données
  - les entités, associations, attributs et identifiants
    - entité: existe d'elle même
    - association: existe entre entités
    - attribut: propriété individuelle d'une entité ou association
    - identifiant: attribut particulier identifiant l'entité
    - cardinalité: nb d'association pour une entité (0,1,n)

C. Crochepeyre Génie Logiciel – Rappels 45

L'aspect conceptuel des données consiste à distinguer:
- les entités
- les associations entre ces entités
Le modèle Entité-Association est un formalisme connu par l'ISO pour décrire cet aspect.

On associe au langage naturel:
- des noms aux entités
- des verbes aux associations
- des adjectifs aux attributs des entités
- des adverbes aux attributs des associations
le modèle le plus connu est celui de Peter CHEN (1976)
La représentation est la suivante:
L'entité
- une boîte nommée
- son nom est un nom singulier « une personne » -> entité « personne »
- c'est un ensemble d'objets ayant des caractéristiques communes
exemple: Paul est un exemplaire de l'entité « personne »
Pour savoir si une chose est une entité, se poser la question: « Peut-on trouver plusieurs exemplaires de cette choses? »
Les sous entités
- sous groupement d'entités ayant certaines particularités
Les associations
- représentation par un ovale nommé
- c'est un type, une classe ou un ensemble
- des pattes relient associations et entités
La cardinalité minimum m et maximum M de l'entité-association
- au dessous ou au dessus de la boîte :
m-M m indique le nombre minimum d'exemplaires de l'entité associées à une autre entité
M indique le nombre maximum d'exemplaires de l'entité associées à une autre entité
La cardinalité peut être : 0, 1 ou n (plusieurs)
Une flèche entre l'entité et l'association rend obligatoire le nombre minimum. Sans flèche l'association est



L'exemple traduit :

- qu' un exemplaire de l'entité A est associée à aucun ou au plus n exemplaires de l'entité B
- qu' un exemplaire de l'entité B est associée aucun ou au plus un exemplaire de l'entité A

De ce modèle simplifié entité-association est naît le modèle entité-relation qui a sert à la définition des bases de données relationnelles. CODD en 1970 a ainsi défini son modèle relationnel.

La structure logique de ce type de bases de données relationnelles est un ensemble de tables ayant des relations avec les objets composant ces tables.

La définition même des relations entre les objets permet ensuite de construire physiquement la base de données.

- **Définitions des besoins non fonctionnels**
  - contraintes du système
  - éventuellement soumis aux évolutions technologiques
  - interaction entre fonctions: conflits -> solutions
  - langage naturel ou structuré

  exemple: matrice besoins/propriétés

propriétés / besoins	vitesse	mémoire
réponse < 3s	D84,D95	A

A: cas analysé
Di: définition Di

C. Crochepeyre Génie Logiciel – Rappels 47

Les besoins non fonctionnels sont des « contraintes »:

- ces contraintes ne sont pas liées directement aux activités des fonctions concernées mais doivent être prises en compte pour le bon fonctionnement du système.

exemple: contrainte sur le temps de réponse

Les contraintes du système évoluent dans le temps car la technologie matérielle change, les objectifs politiques ou commerciaux évoluent, le contexte de l'entreprise peut changer….

Il faut prévoir les évolutions possibles au cours du développement ou tout au moins indiquer lors de la définition des besoins où se situent les besoins non fonctionnels: leur lien avec le matériel, avec le financement, les activités de l'entreprise…. afin qu'ils soient facilement modifiables.

Des contraintes sur certaine fonction peuvent entraîner des effets secondaires ou incompatibilités avec les contraintes d'autres fonctions.
Faire apparaître ces conflits et les compromis possibles.

Il existe plusieurs façons de représenter les besoins non fonctionnels:

On retrouve dans cette matrice les références des définitions (Di) explicitées de chacun des cas. Il s'agit d'exiger des temps de réponses en connaissant les caractéristiques de la vitesse du processeur et d'en déduire la capacité mémoire nécessaire pour approcher cette contrainte.

- **Validation des besoins**
  par les utilisateurs et développeurs
  - cohérence
    - pas de conflit entre les besoins
  - complétude:
    - tous les besoins et contraintes sont recensés
  - réalisme:
    - réalisables avec la technologie matérielle et logicielle
  - validité:
    - réponse aux besoins
    - des fonctionnalités supplémentaires si nécessaire

- **Outils ou méthodes de validation**
  - outils d'analyse et recherche d'anomalies
  - générateur de simulateurs
  - Prototype, Maquette

C. Crochepeyre Génie Logiciel – Rappels 48

Cette phase de validation est importante:

- toutes les étapes suivantes découlent des éléments constituant le cahier des charges

- des oublis, des erreurs ont des conséquences coûteuses

Que doit-on faire?

- La cohérence - Supprimer tout conflit entre besoins

- La complétude - L'utilisateur a une grande responsabilité lorsqu'il recense ses besoins : fonctions et contraintes sur ces fonctions. Il doit ne rien oublier.

- Des besoins réalistes - Les développeurs, analyste, ingénieurs informaticiens doivent informer l'utilisateur des limites matérielles et logicielles. Ils doivent aussi anticiper l'évolution matérielle avec réalisme.

-Des besoins valides - Les services demandés par l'utilisateur se traduisent par des fonctionnalités du système. Ces fonctionnalités peuvent être quelque peu modifiées, améliorées du fait de l'informatisation du système.

Comment effectuer cette validation?

- Par des outils:

. analyseur de mots clés qui repère les paragraphes concernés

. analyseur d'entrées sorties des fonctions. Les mêmes objets en entrée dans deux fonctions différentes doivent être décrits de la même façon.

- Par des hommes

. faire relire les documents par les utilisateurs et les développeurs à la recherche des anomalies, des oublis....

- Par une technique de simulation

. surtout pour les besoins non fonctionnels

. exemple: outils de simulation associés au langage RSL comme un générateur de simulateur qui analyse les définitions en RSL et génère en Pascal un simulateur.

Attention:

Le développement du simulateur peut être aussi coûteux que la réalisation elle même.

Dans des domaines d'activités à hauts risques ces simulateurs sont fortement employés.

Prototype

à ne pas faire:

- développement du prototype avec les outils du produit final
- développement avec le même degré de finesse

mais:

- utiliser des outils qui permettent de
- mettre l'accent sur l'essentiel

Maquette

- n'a aucune fonctionnalité effective
- look extérieur
- donne seulement un aperçu des fonctionnalités

C. Crochepeyre Génie Logiciel – Rappels 49

Le prototypage est une approche évolutive du développement d'un système.
Si le produit est incomplet au départ, il s'enrichit au fur et à mesure que les besoins apparaissent.
L'aboutissement des transformations est le produit final.

Autre approche: le prototype 'jetable'

Sur un 1er prototype on fait apparaître les anomalies d'après les spécifications initiales.
Après mise en évidence des problèmes, on repart sur une version corrigée.

Le prototype est abandonné lorsqu'il est satisfaisant . Le système est alors réalisé.

Importance du prototype lors des phases d'analyse, car:

- explications entre l'utilisateur et le développeur lors des démonstrations
- détection de fonctions oubliées
- mise en évidence de besoins imprécis, incomplets, incohérents
- démonstration de la faisabilité et de l'utilité du système
- peut servir de spécification pour le développeur du système (domaine industriel)

Le prototype a cependant quelques inconvénients:

- l'utilisation d'outils lourds, tels que ceux retenus pour le développement final, sont à éviter car le temps passé à mettre au point le prototype influe sur le coût.
- le prototype incite l'utilisateur à demander de nouvelles fonctionnalités : les idées viennent à la vue d'une maquette.
- le prototype jetable n'est pas le produit final. Il faut donc opter pour cette solution si l'on juge que l'argent dépensé à ce travail sera largement récupéré lors des étapes suivantes.

Parmi les recommandations, citons celles-ci:

- éviter de développer le prototype avec les outils complets du produit final.
- se rappeler l'objectif d'un prototype
  - . montrer les aspects fonctionnels les plus importants
  - . ne pas insister sur les aspects non fonctionnels (contraintes)
- ne pas développer avec le même degré de finesse que le produit final.

Mais:

- utiliser si besoin des langages de haut niveau qui servent à
- mettre en évidence l'essentiel

- **Conclusion: expression des besoins**
  - Définition des services attendus par l'utilisateur
  - Analyse des fonctions, de leurs contraintes et recensement des données
  - Pas encore de solutions techniques précises
  - Collaboration de l'utilisateur importante
  - Utilisation de modèles, méthodes ou de techniques: statiques ou dynamiques
  - Validation correcte avant la phase suivante
  - Définir les limites
  - Envisager les évolutions

C. Crochepeyre Génie Logiciel – Rappels 50

Nous avons terminé l'analyse des besoins.

Nous résumons cette étape en rappelant les diverses phases :

- **Services attendus**

. recensement des règles de gestion, des données, des flux, des volumes…

. compréhension par le développeur

- Analyse des **Besoins fonctionnels** et des **Besoins non fonctionnels**

- Etape de conception qui n'a pas encore de vraies solutions techniques mais plutôt un ensemble de méthodes, d'outils pour aider à la formulation des besoins.

- Ces outils ont deux aspects:

. **statiques**

. **dynamiques** pour les descriptions à contraintes de temps

- La **collaboration de l'utilisateur** est très importante: analyse précise, complète

- La phase de **validation** des besoins doit être faite avec sérieux car il est très couteux ensuite de revenir sur cette première phase d'analyse;

- L'analyse des besoins doit être bien cadrée, les **limites** bien définies afin d'éviter toutes dérives.

- Penser aux solutions futures dans le cadre de **l'évolution** technologique du matériel mais aussi dans le cadre de l'évolution des fonctionnalités du métier.

Changements et évolutions inévitables

- Actuellement meilleure prise en compte des évolutions
  - **CobiT** : référentiel international de gouvernance des SI. Maîtrise et suivi de la gouvernance du SI dans la durée.
  - Modèle de processus qui subdivise l'informatique en 34 processus répartis entre les quatre domaines de responsabilités que sont **planifier, mettre en place, faire fonctionner et surveiller**, donnant ainsi une vision complète de l'activité informatique

Cobit: Contrôle de l'Information et des Technologies Associées

Actuellement on prend conscience des évolutions inévitables des systèmes d'informations.

CoBIT est un référentiel qui concerne tous les utilisateurs: les directions générales, informatiques, fonctionnelles, utilisateurs.

L'objectif de CoBIT est la maîtrise des risques en proposant des contrôles rigoureux et réguliers.

Ce référentiel prend en compte le lien entre les objectifs métiers et les moyens informatiques. Des mesures vont permettre aux dirigeants de savoir si des améliorations sont nécessaires. Des contrôles de diverses natures sont préconisés:

-Tests de maturité

-Objectifs et résultats sont analysés dans des tableaux de bord

-Contrôles de sécurité, etc…

Ainsi CoBIT subdivise l'informatique en 34 processus dans 4 domaines de responsabilités: planifier, mettre en place, faire fonctionner et surveiller. En décomposant ces responsabilités, il est plus facile d'identifier l'activité informatique et de mieux la gérer.

Parmi les 34 processus, on peut prendre comme exemple la sécurité.

Sur ce thème CoBIT recommande

- de gérer des droits d'accès aux applications en référant aux règles d'accès qui ont été définies.

- de contrôler régulièrement ces droits d'accès pour détecter les fraudes et les abus

- de centraliser la gestion des droits d'accès

- **Modèle ITIL** (Information Technology Infrastructure Library): « *ensemble cohérent des meilleures pratiques en matière de gestion de services informatiques basées sur la maîtrise des processus ».*
- Plusieurs chapitres dont la « gestion des changements »

- **Modèle CMMI**: référentiel proposant de bonnes pratiques liées à la gestion, au développement et à la maintenance d'applications et de systèmes.
- Ces bonnes pratiques sont regroupées en 24 processus, eux-mêmes regroupés
  - en 4 types (*Process Management*, *Project Management*, *Engineering* et *Support*)
  - et 5 niveaux de maturité. Niveau 5 : gestion des changements technologiques et des changements de processus.

C. Crochepeyre Génie Logiciel – Rappels 52

ITIL (IT Information Library) ensemble de guides en matière de management des services informatiques. La bibliothèque ITIL a été créée afin d'améliorer le service rendu par les DSI, directions des services informatiques.

Un chapitre particulier concerne les démarches à mettre en œuvre pour la conduite du changement afin d'anticiper les effets de bord.

CMMi (Capability Maturity Model Integration) est une référence des bonnes pratiques dans le développement des logiciels et des systèmes.

C'est un modèle d'évaluation de la capacité à gérer et terminer un projet correctement.

Pour cela 5 niveaux de maturité vont permettre de situer l'organisation dans l'avancement de son projet. Le niveau 1 « initial » est le niveau où l'organisation n'est pas encore prête. Le projet est incertain.

Le niveau 5 « optimisation » est le plus haut. Les processus existent et sont constamment revus et améliorés en fonction des évolutions.

PLAN

1. CYCLE DE VIE DU LOGICIEL
2. EXPRESSION DES BESOINS
3. **CONCEPTION DU LOGICIEL**
4. LA PROGRAMMATION
5. TESTS ET MISE AU POINT
6. DOCUMENTATION
7. CONCLUSION

La phase d'expressions des besoins étant terminée nous pouvons passer à celle de la conception du logiciel: des spécifications et de l'analyse informatique à la conception informatique du logiciel

Nous présenterons quelques méthodes utilisées lors de cette phase du développement du logiciel.

Le système décrit par l'utilisateur est à ce niveau analysé pour construire un modèle informatique.

Après avoir énoncés, dans le cahier des charges, les services attendus et les contraintes à respecter, on définit les unités de logiciel qui réaliseront les fonctions demandées.

Notons qu'une fonction du système peut faire l'objet de plusieurs unités de logiciel..

On va donc s'intéresser:

- à la **spécification** et **à l'analyse fonctionnelles** des logiciels et
- à leur **conception**

3.1. LES SPECIFICATIONS

- **Passage:**
  - De l'expression des besoins à la solution informatique
  - Spécifications informatiques
    - des fonctions
    - des données
    - des interfaces
  - Phase précédant la conception détaillée

- **Document des spécifications**
  - destiné aux développeurs

C. Crochepeyre Génie Logiciel – Rappels 54

Ce travail préliminaire des spécifications est nécessaire pour construire un modèle informatique qui réponde au cahier des charges.

La décomposition des fonctions en unités logicielles, la construction d'un système d'informations unique, la mise en œuvre des règles énoncées dans les traitements… sont autant d'éléments à respecter.

Ceci conduit à :

- mettre en évidence les **relations** entre les différentes unités due logiciel

- analyser en **détail** ces unités afin de passer ensuite à leur programmation

L'analyse préalable décrit le fonctionnement externe du système

L'analyse fonctionnelle propose un modèle informatique pour ce même fonctionnement et précise ce que le logiciel doit faire

La conception ou analyse détaillée précise comment chaque unité de traitement sera réalisée.

La séparation entre les deux derniers niveaux n'est pas toujours nécessaire. Ils sont alors confondus en une « analyse et conception des logiciels ».

On rédige à ce stade un document de spécifications fonctionnelles des logiciels qui se réfère au document précédent (cahier des charges)

Ce document est plus destiné au **développeur** qu'à l'utilisateur. C'est un document technique. Il se compose de **définitions abstraites** du logiciel.

Le document de spécifications des logiciels se présente différemment du cahier des charges car il décrit des unités de logiciel qui selon des enchaînements d'exécution produiront les résultats attendus pour chacune des fonctions.

Comment décrire et présenter ces unités de logiciel?

Par des outils de spécifications.

- **Méthode d'Analyse Structurée SA**
  E. YOURDON (1979)
  - spécification statique du logiciel
  - analyse descendante:
    - affinages successifs des traitements
    - description des flots de données des traitements
    - ensemble de diagrammes ordonnés et hiérarchisés
    - fonctions élémentaires = primitives fonctionnelles
  - outils graphiques et textuels

- Quelques exemples de l'utilisation de cette méthode

C. Crochepeyre Génie Logiciel – Rappels 55

Cette première méthode a été défini par Edward Yourdon et Tom Demorco en 1979.

Cette méthode est une décomposition fonctionnelle descendante. Elle spécifie le logiciel de façon statique.

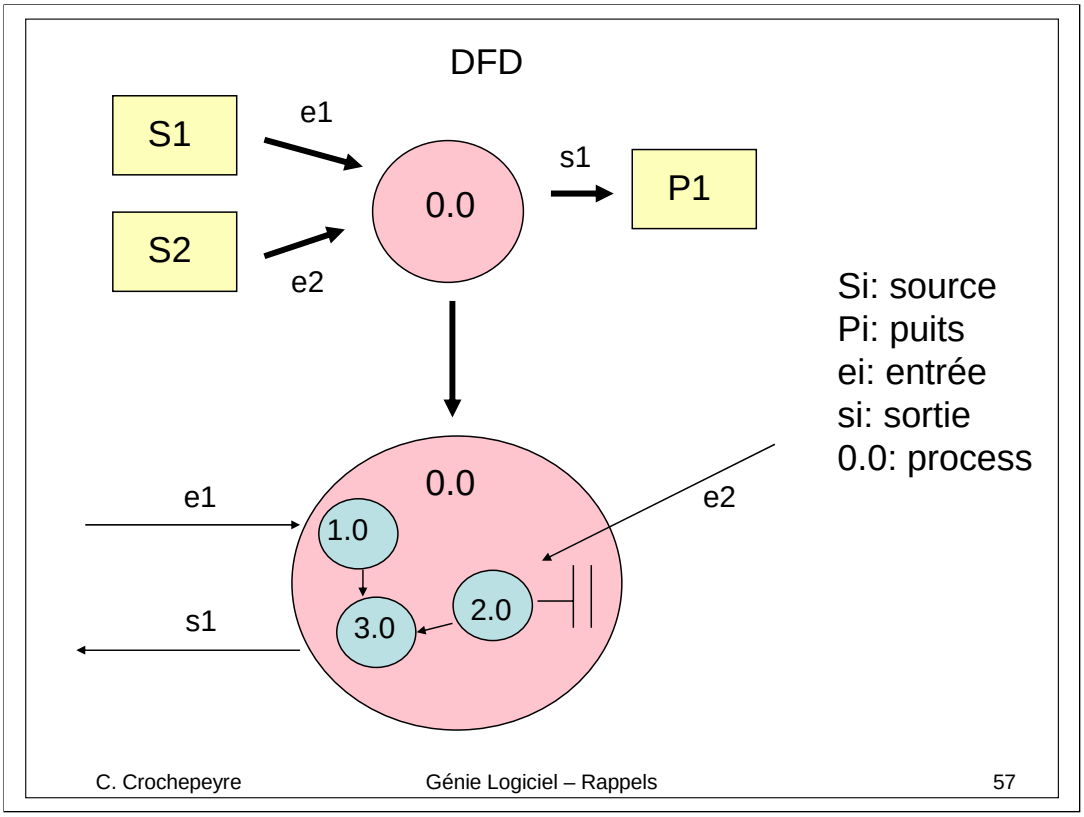
L'analyse descendante consiste à :
- affiner successivement les traitements en terme de flots de données logiques
- décrire des diagrammes ordonnés dans une hiérarchie.

L'affinage fait apparaître des fonctions élémentaires appelées 'primitives fonctionnelles' ou encore 'process primitifs'
cette méthode utlise des outils:
- graphiques
- textuels

- **Data Flow Diagram DFD**
  - diagramme de flots de données = interconnexion de fonctions traversées par une circulation de données
  - 4 éléments graphiques
    - le traitement ou process = cercle
    - le flot de données = trait
    - l'unité de stockage = 2 traits
    - l'entité externe ou terminateur = rectangle

C. Crochepeyre Génie Logiciel – Rappels 56

Les outils purement graphiques sont des diagrammes de transformations de flots de données nommés: DFD (Data Flow Diagram)
Un diagramme de flots de données est:
- une interconnexion de fonctions ou process au travers de laquelle circulent les données.
Le DFD se construit avec 4 éléments:
1. **Le traitement** ou process
Représentation par un cercle
Il transforme les données en entrée en données en sortie
Trois transformations possibles:
- modification de la structure de la donnée
- transformation d'une donnée en une autre
- aiguillage de la donnée et un test de la donnée
2. **Le flot de données**
Représentation par un trait continu avec un nom
Il traduit la circulation des données reliant deux process.
3. **L'unité de stockage**
Représentation par deux traits parallèles //
Représente le stockage des données sous forme de bases de données ou de fichiers
On accède ainsi plusieurs fois à l'information et sans ordre précis puisqu'elle est immobile.
N.B.
On ne mentionne pas encore à ce niveau les clés d'acès, les méthodes d'accès ni les organisations de stockage. Ceci sera fait lors de la conception.
4. **L'entité externe** ou terminateur
Représentation par un rectangle.
Indique la provenance ou la destination des données traitées. On parle aussi de source (provenance) et de puits (destination)



Cet exemple graphique montre:

- les cercles les processus nommés O.O, 1.0, 2.0 et 3.0
- les traits les flots de données nommés en entrées e1, e2 et en sortie s1
- les traits // le stokage de données du processus 2.0
- les terminateurs les sources S1 et S2 et le puits P1

Le processus 0.0 est détaillé dans la partie inférieure

Les process

- sont en général identifiés par un verbe et un n°

  'Editer paie' - n° 2.0

- sont les éléments de transformation des flots de données E en flots de données S

Les flots de données

- sont nommés avec des noms différents que décrivent le mieux possible les données qui circulent.

  'Eléments paie' - 'Paie éditée'

- ont leurs extrémités rattachées:

  . une à un processus

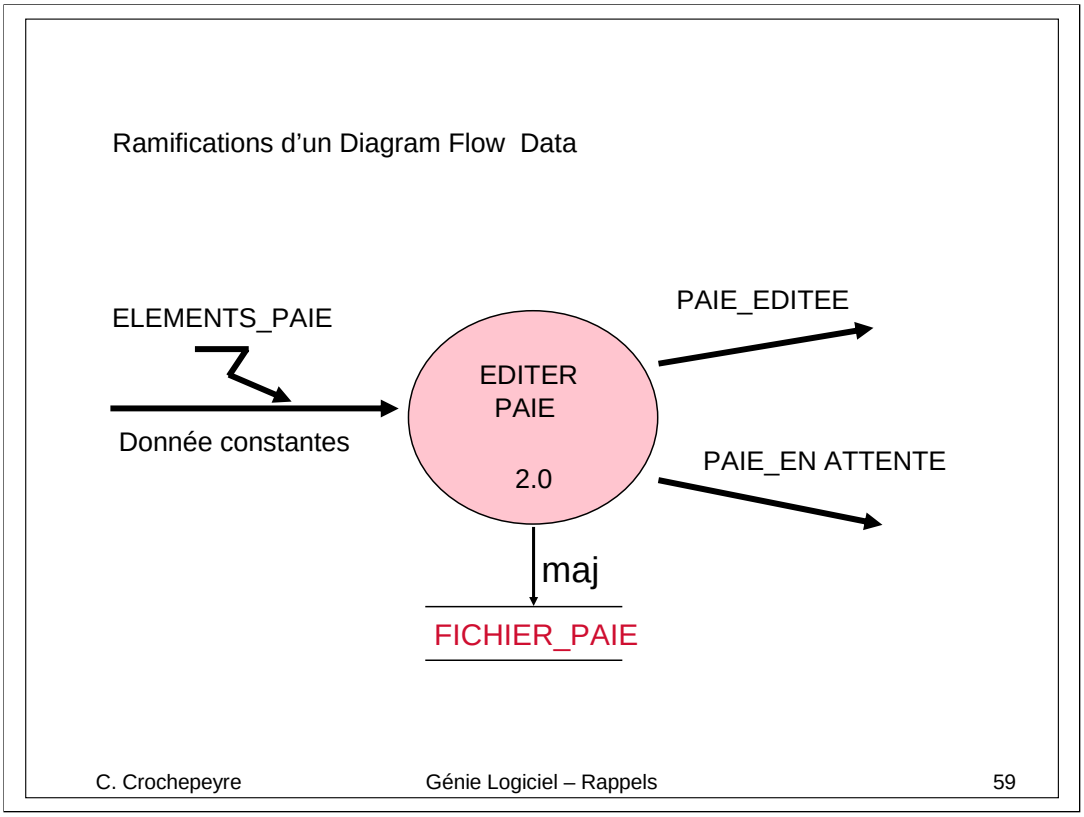
  . l'autre à un processus ou un terminateur ou une unité de stockage

- des ramifications sont possibles.
- Les unités de stockage porte un nom et sur la flèche, désignant le sens de l'échange, un commentaire indique la nature de l'échange

- Les process du DFD
  - sont identifiés par un verbe et un n°
  - transforment les flots de données en entrées en flots de données en sortie
- Les flots de données du DFD
  - portent un nom unique et significatif
  - les extrémités:
    - une à un process
    - l'autre à un process, terminateur, unité de stockage
  - ramifications possibles
  - unités de stockage:
    - Un nom
    - Une flèche avec nature de l'échange

C. Crochepeyre Génie Logiciel – Rappels 58

Les processus et les flots de données sont représentés graphiquement selon des diagrammes personnalisés et des règles de la méthode SA.



Exemple de la représentation graphique d'un processus et ses flots de données.

Les process

- sont en général identifiés par un verbe et un n°

'Editer paie' - n° 2.0

- sont les éléments de transformation des flots de données E en flots de données S

Les flots de données

- sont nommés avec des noms différents que décrivent le mieux possible les données qui circulent.

'Eléments paie' - 'Paie éditée'

- ont leurs extrémités rattachées:

. une à un processus

. l'autre à un processus ou un terminateur ou une unité de stockage

Des ramifications sont possibles.

- **Dictionnaire de données DD**
  - création simultanée des diagrammes flots données
  - contient : sémantique, structure, flots et stockage de chaque donnée
  - opérateurs DD: +, =, max, min, { },( ) …..

exemple:

adresse EST nom
ET adresse rue
ET nom ville
ET code postal
adresse = nom + adresse rue + nom ville + code postal

C. Crochepeyre Génie Logiciel – Rappels 60

Le dictionnaire de données DD est un document écrit sous forme de texte recensant l'ensemble des données du système.
Chaque donnée est décrite avec ses caractéristiques:

- sémantique

- structure

Ce document est créé et interprété en même temps que le diagramme de flots de données DFD

Exemple:

Définition d'une adresse.

On constate que la lecture d'un tel document est aisée car les opérateurs utilisés clarifient le texte et sont explicites.

Ici l'adresse est composée du nom, de la rue, de la ville et du code postal.

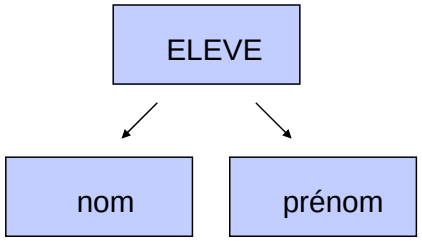
- **Outils graphiques/textuels** :
  - Diagrammes de structures de données DSD
  - Spécifications de process PSPEC
    - algorithmes
    - arbres de décision
    - tables de décision
    - diagrammes

Le **diagramme de structures des données DSD** est un document graphique et textuel.

Les **spécifications des traitements ou process PSPEC** se font aussi à l'aide d'outils graphiques et textuels :

- par des algorithmes abstraits
- par des arbres de décisions
- par des tables de décision
- par des diagrammes de jackson
- par des diagrammes de Schneiderman

- **Diagrammes de structures de données DSD**
  - description des relations entre les données
  - données simples dans le DD
  - données complexes décrites:
    - textuellement (opérateurs DD)
    - graphiquement (diagrammes M. Jackson)



C. Crochepeyre Génie Logiciel – Rappels 62

Le **diagramme de structures des données DSD** permet de décrire graphiquement les relations entre les données.

Comme nous l'avons vu précédemment, la description des **données simples** des unités de stockage se trouve uniquement dans le dictionnaire des données DD.

Les **données complexes** comme des relations entre les fichiers sont décrites

- dans le diagramme de structures des données DSD qui peut contenir des descriptions textuelles utilisant les opérateurs du DD et

- par des descriptions graphiques tels que les diagrammes de Jackson ou autres.

- **Spécifications de process PSPEC**
  - PSPEC Par algorithmes

Séquence ou traitement

traitement_1
traitement_2
traitement_2

Alternatives composées

Si <condition> vraie **Alors**
  traitement_1
Sinon
  traitement_2
Finsi

Décider entre
  Cas_1 vraie **Alors** traitement_1
  Cas_2 vraie **Alors** traitement_2
Autrement
  Erreur
Fin Décider

C. Crochepeyre Génie Logiciel – Rappels 63

Ici un exemple de spécification de process par algorithme.

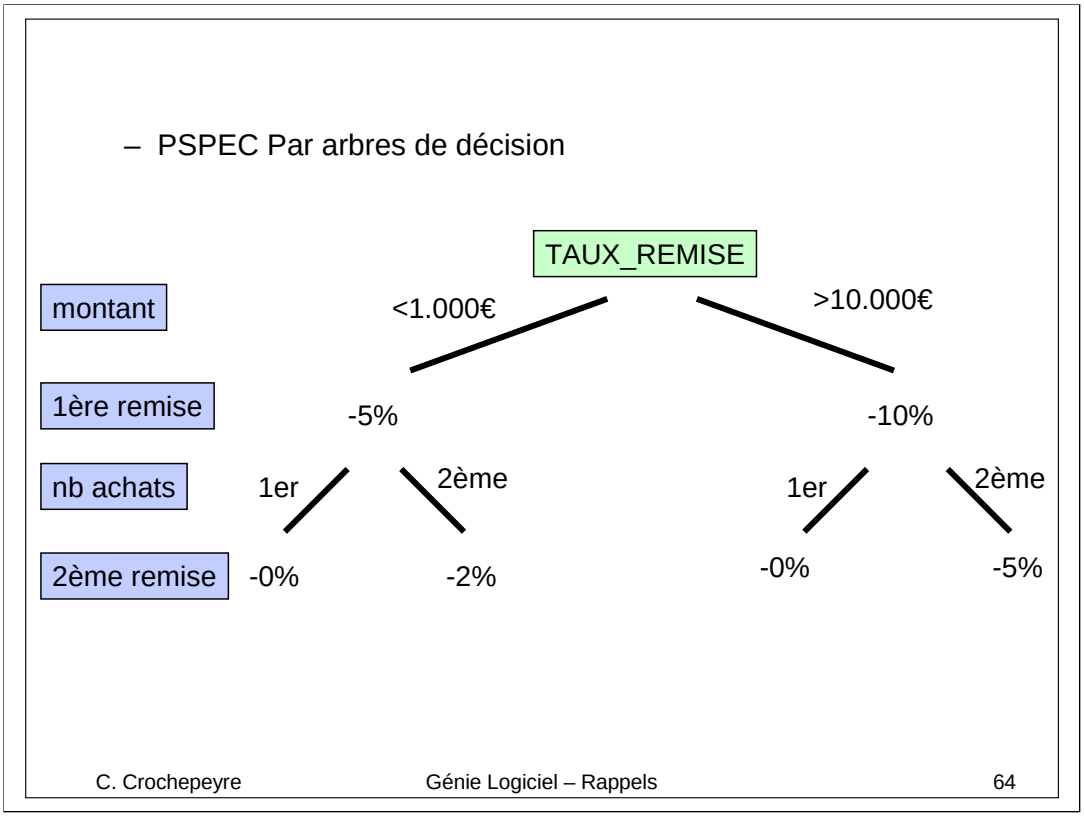
Les transformations complexes d'un processus sont décrites

. par des phrases décrivant les opérations

. des références aux données se trouvant dans le DD

. du pseudo code de programmation pour présenter les structures de contrôle

L'exemple présente les différents traitements et les alternatives écrites en pseudo code dans lesquelles les traitements sont impliqués.



L'arbre de décisions est un autre moyen pour représenter graphiquement et textuellement un enchaînement de conditions déterminant le traitement à appliquer.

Sur cet exemple, un taux remise est fonction du montant des achats:

- si le montant est > 1000€ alors une remise de 5% est automatique et si c'est un second achat s'ajoute une remise supplémentaire de 2%

– PSPEC Par tables de décision

règles

	R1	R2	R3	R4
<1.000	V	V	F	F
2ème achat	F	V	F	V
remise 2%		V		
remise 5%	V	V		V
remise 10%			V	V

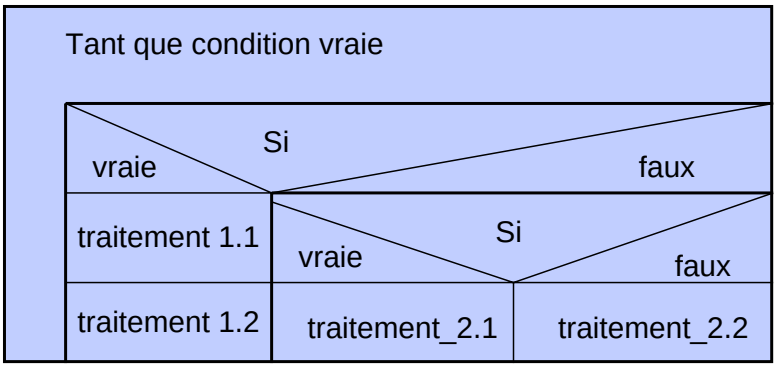
conditions

actions

C. Crochepeyre Génie Logiciel – Rappels 65

Ici une représentation différente des règles de décision sous forme de table de décisions
C'est un outil employé lorsque le nombre de combinaisons est important et qu'un arbre n'est pas suffisant.
La table de décisions recense :
- les conditions
- les règles
et détermine les actions en fonction de ces deux paramètres.
Exemple :
Comme pour l'arbre vu précédemment, on décrit les actions:
- appliquer une remise de 5% si 1er achat < 1.000
- appliquer une remise de 7% si 2ème achat <1.000

– PSPEC Par diagrammes
  • M. Jackson et Warnier
  • N. Schneiderman



C. Crochepeyre Génie Logiciel – Rappels 66

Autre outil pour décrire les spécifications de process: les diagrammes.

Ici un exemple de diagramme de Schneiderman qui permet de décrire un enchaînement de conditions conduisant à un traitement spécifique.

Tant que la condition énoncée est vraie et

- si la 2eme condition est vraie alors traitement_1.1 et traitement_1.2
- si la 2ème condition est fausse alors
    - si la 3eme condition est vraie alors traitement_2.1
    - si la 3eme condition est fausse alors traitement_2.2

- **Méthode SA Temps Réel SA-RT**
  P. WARD et S. MELLOR (1985)
  - extension de SA au temps réel
    - méthode SA: vue statique des processus
    - méthode SA_RT: vue dynamique des processus
  - systèmes temps réels:
    - systèmes combinatoires: E => S
    - systèmes séquentiels: E + états internes => S
  - outils graphiques et textuels

C. Crochepeyre Génie Logiciel – Rappels 67

La méthode d'analyse structurée SA est simple, intuitive puisqu'elle procède par affinement des composants du système (analyse descendante) mais conduit à une description statique des traitements (spécification statique) qui n'est pas toujours suffisante.

La méthode d'analyse structurée temps réel SA_RT est de Ward et Mellor (1984-1985).

Elle permet de traduire l'aspect dynamique de la spécification d'un logiciel temps réel.

Elle est retenue dans de nombreux ateliers de génie logiciel industriels.

C'est une extension de la méthode structurée SA que nous venons de présenter.

- méthode SA : vue statique des process

- méthode SA_RT : vue dynamique des process

On procède à une modélisation **dynamique** et **évènementielle** d'une application temps réel.

Il existe deux types de systèmes temps réel:

- **les systèmes combinatoires**

Les valeurs des signaux en sortie sont entièrement déterminées par les valeurs des signaux en entrée

- les **systèmes séquentiels**

Les valeurs des signaux en sortie sont déterminés par les valeurs des signaux en entrée et par des états internes.

Comme pour la méthode SA, on utilise dans cette méthode des outils graphiques et textuels.

- **Diagramme de flots de données: DFD**
  - le processus de données: un cercle
  - les flots de données: un trait
    - **les flots discrets: une flèche**
    - **les flots continus: une double flèche**
  - l'unité de stockage: deux traits parallèles
  - le terminateur: un rectangle

Cette méthode repend les outils de la méthode SA en y apportant les éléments nécessaires et complémentaires pour décrire l'aspect dynamique du système.

Le diagramme de flots de données DFD de SA_RT distingue

- les flots **discrets** : une flèche

Représentation de la circulation d'une donnée valide à des instants précis

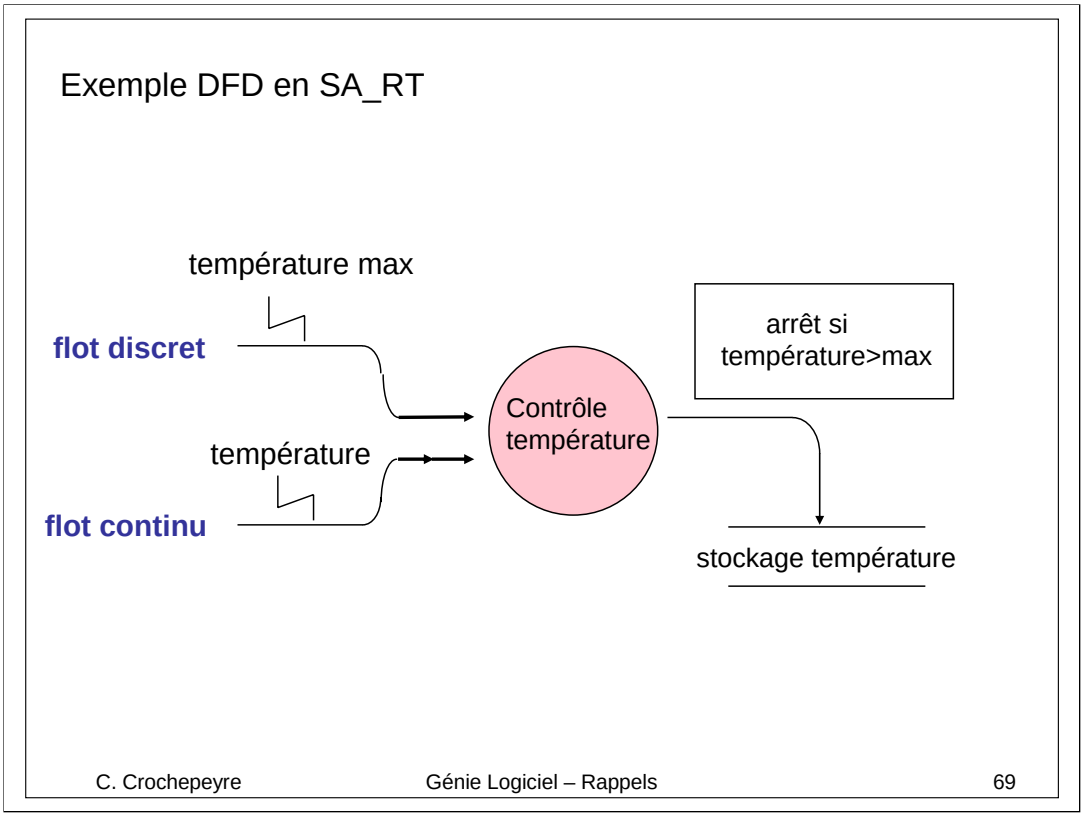
- les flots **continus** : une double flèche

Représentation de la circulation d'une donnée continuellement valide dans le temps

La représentation d'un flot de données de SA est un trait simple.

Les autrs concepts sont représentés de la même manière:

- un process : un cercle
- une unité de stockage: deux traits parallèles
- un terminateur : un rectangle



Sur l'exemple on considère que le processus de contrôle de température se réfère à une valeur précise dans le temps pour s'arrêter : flot discret .

Le flot continu des températures est stocké dans un fichier.

- **Dictionnaire de données: DD**
  - répertoire
    - données
    - flots de données
    - stockages des données
    - **flots de contrôle**
    - **stockage des flots de contrôle ou événements**
  - informations d'après les diagrammes de flots de données DFD
  - informations d'après les flots de contrôle CFD

C. Crochepeyre Génie Logiciel – Rappels 70

Le dictionnaire de données de SA_RT est un outil textuel similaire à SA.

C'est un répertoire:

- des données
- des flots de données (partie textuelle du DFD)
- des description des données stockées dans des unités de stokage

Dans SA_RT c'est en plus une description textuelle

- des flots de controle CFD
- du stokage des flots de controle ou événements.

Notons que les informations concernant les flots de données et les flots de controle viennent compléter les diagrammes respectifs (DFD et CFD)

- **Spécifications de processus: PSPEC**
  - description des traitements élémentaires par
    - algorithmes abstraits
    - arbres de décision
    - tables de décision
    - diagrammes
    - équations
    - fonctions de gestion de bases de données
    - commentaires

Les **diagrammes de structures des données DSD** sont nécessaires pour décrire des données complexes telles que les relations entre fichiers (sinon description dans DD)

La description de ces diagrammes est textuelle et graphique comme pour SA.

Les **spécifications de process PSPEC** utilisent les mêmes moyens graphiques et textuels que SA : algorithmes, arbres et tables de décision, diagrammes.....

- **Diagramme de flots de contrôle: CFD**
  - les flots de contrôle ou événements ou signaux
    - événements: vecteurs pointillés
    - données discrètes: traits pleins
  - les flots de contrôle "prompt"
    - flèche pointillée E: activation
    - flèche pointillée D: désactivation
    - flèche pointillée T: déclenchement (Tigger)
  - les processus de contrôle
    - cercle pointillé
  - les stockages de contrôle ou stockage d'événements
    - deux traits

C. Crochepeyre Génie Logiciel – Rappels 72

Les diagrammes de flots de contrôles SA_RT, qui n'existent pas dans SA, sont couplés avec les diagrammes de flots de données DFD.

Ces diagrammes CFD décrivent graphiquement:

1. Les flots de contrôles ou évènements ou signaux

   - les évènements

   Représentation : vecteur en traits pointillés

   Ils expriment une occurrence dans le temps.
   Ils ne véhiculent pas de contenu.

   - les données discrètes

   Représentation : trait plein

   Elles propagent une valeur uniquement valide à des instants précis qui disparaît après consommation.

   Elles représentent aussi une occurrence dans le temps.

2. Les flots de contrôles « prompt »

   Représentation : flèche en pointillés

   Flèche E : flot de contrôle d'activation

   Flèche D : flot de contrôle de déactivation

   Flèche T: flot de contrôle de déclenchement

   Enable - Disable - Trigger

   Les « prompt » sont déclenchés par les process de contrôle, suite à un signal.

3. Les process de contrôle

   Représentation : cercle pointillé

   Ils sont spécifiés par un ou plusieurs diagrammes Etat-Transition

4. Les stockages de contrôles ou stockages d'évènements

   Représentation : deux traits

- **Spécifications de contrôle: CSPEC**
  - Diagramme État-Transition: STD
    - états ou attributs d'état:
      - rectangle représentant un intervalle de temps pendant lequel l'objet a un comportement déterminé
    - signaux ou transitions:
      - transitions d'entrée
      - transitions de sortie

OU

  - Table État-Transition: STT
    - états: lignes de la table
    - transitions d'entrée: colonnes de la table
    - transitions de sortie, nouvel état: intersection

C. Crochepeyre Génie Logiciel – Rappels 73

C'est la description la plus complète de la logique de fonctionnement des traitements ou process.

On utilise des outils graphiques.

Cette spécification des contrôles comprend:

1. Le diagramme Etat-Transition STD

C'est une psécification minimum d'un process de contrôle

Il est composé de:

- d'états ou attributs d'états

Représentation: rectangle allongé

Il donne l'intervalle de temps pendant lequel l'objet a un comportement déterminé

- de signaux ou transitions

Représentation : flèche orientée de l'état précédent vers l'état suivant.

Il indique le changement de mode de fonctionnement de l'objet.

Deux types de transitions sont possibles:

- la transition d'entrée précisée par un nom au dessus d'une ligne horizontale placée à côté de la flèche oreintée qui indique le signal provoquant le changement de mode.

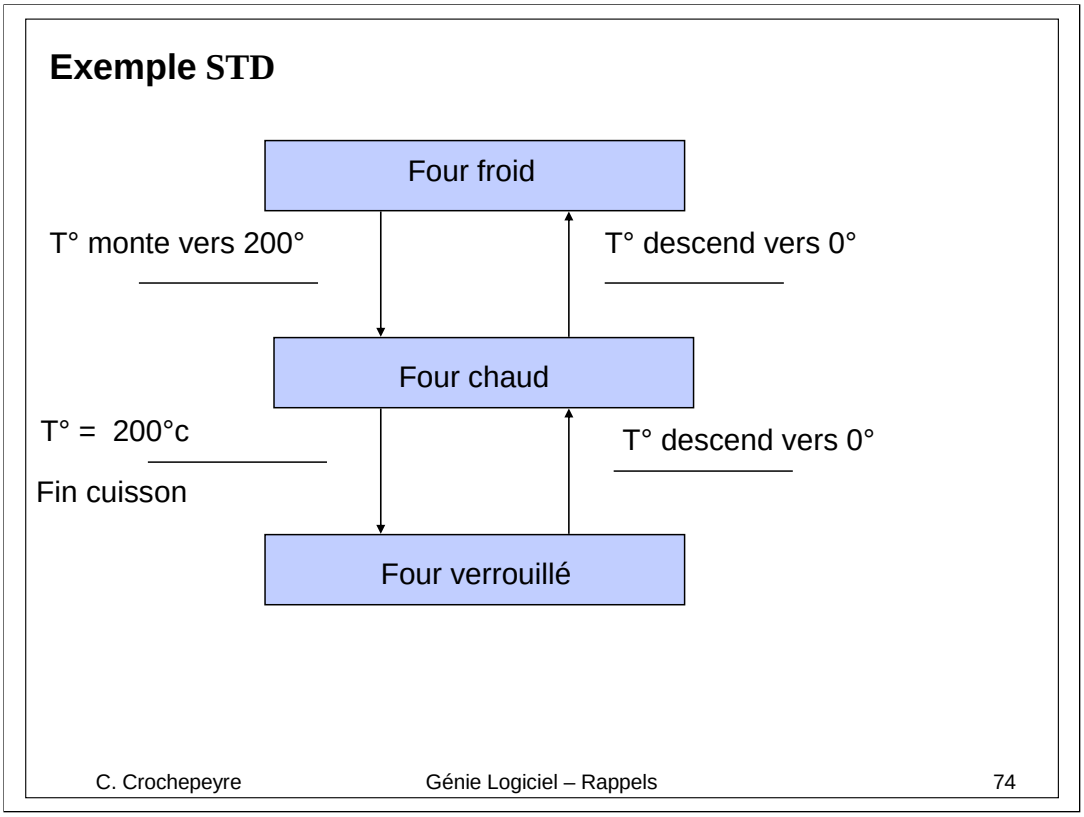
- la transition de sortie précisée par un nom au dessous de la ligne horizontale.

2- La table Etat-transition STT

Elle peut remplacer le diagramme STD.

Les états sont des lignes de la table.
Les transitions des entrées : les colonnes

Les états des sorties : les intersections de la table



Exemple d'un processus de contrôle de température d'un four.

Les contrôles de ce processus sont spécifiés par ce diagramme Etat-Transition STD
Les différents états sont représentés par les rectangles:
- four froid
- four chaud
- four verrouillé

Les transitions vont :
- du four froid au four chaud T1 et vice versa T1B
- du four chaud au four verrouillé T2 et vice et versa T2B

Les tansitions d'entrée:
- T° monte vers 200° - Transition d'entrée de T1
- T° = 200° - Transition d'entrée de T2
- T° descend vers 0° - Transition d'entrée de T2B
- T° descend vers 0° - Transition d'entrée de T1B

La transition de sortie :
- Fin de cuisson - Changement d'état qui permet une sortie de cet état Four verrouillé

Exemple STT

Transitions / Etats	T° monte vers 200°	T° = 200°	T° descend vers 0°
F. froid	ES: F. chaud TS: aucune		
F. chaud	ES: F. verrouillé TS: T° = 200°		ES: F. froid TS: aucune
F. verrouillé		ES: F. chaud TS: Fin cuisson	

C. Crochepeyre Génie Logiciel – Rappels 75

Autre spécification par une table de STD.
Les état sont recensés sur les trois lignes de la table,
les transactions correspondent aux trois colonnes
et les états de sortie sont aux intersections

- **Spécifications de "timing": TSPEC**
  - les tables de temps de réponses
    - fréquence de répétition des signaux
    - temps de réponse entre signal entrée et signal sortie
    - temps d'activation dans un état...

Pour terminer, les spécifications des temps de réponse ‘timing’ sont décrites dans des tables particulières.
On y trouve:
- la fréquence de répétition des signaux
- les temps de réponse entre signal d’entrée et signal de sortie
- les temps d’activation dans un état.....

- **Autre outil de spécification : Réseau de Pétri**
  - Spécifications dynamiques
  - Représentation graphique: réseau
    - Nœuds de condition: « places » - cercle
    - Nœuds d'évènement: « transitions » - rectangle
    - Flèches: sens de circulation
    - Jetons: circulent dans les places

  « un jeton dans une place signifie que la condition est réalisée »

  « la transition a lieu si toutes les places en entrée ont un jeton »

  « les places en entrée perdent leur jeton, les places en sortie sont munies de jetons »

C. Crochepeyre Génie Logiciel – Rappels 77

L'analyse de processus temps réel utilise des outils de spécifications particuliers pour exprimer les contraintes de temps.

La phase de spécification des besoins de tels systèmes se fait pour les composants statiques avec des méthodes telles que SADT

Les spécifications temps réels peuvent être modéliser par un réseau de Petri ou des outils SA_RT que nous venons de voir.

Dans le cas du réseau de Petri, la représentation graphique s'exprime sous forme de réseau ayant

-Des nœuds de condition qui sont nommés « places », représentés par un cercle

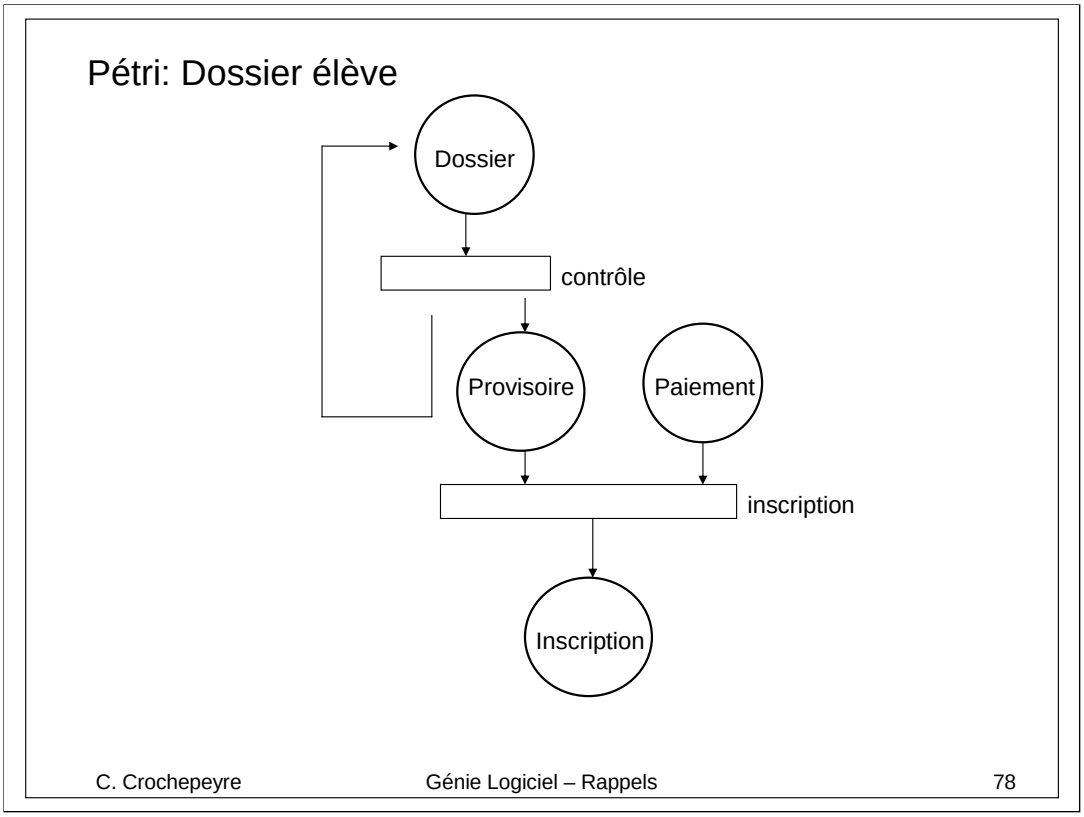
-Des « transitions » qui sont des rectangles et matérialisent des événements

-Des « jetons » qui sont dynamiques car ils circulent dans le réseau en fonction des différentes transitions.

-Des « flèches » qui donnent le sens de la circulation des jetons

Le principe consiste à faire circuler les jetons lors des transitions mais les transitions ne se font que lorsque les places en amont ont toutes un jeton. Alors les places en entrée perdent leur jeton qui sont transmis aux places en sortie.

Ainsi ce réseau exprime les états, les transitions et les règles de transitions: communication et synchronisation sont assurées.



Ici l'exemple du réseau de Petri montre qu'un dossier d'inscription est contrôlé qu'ensuite il constitue une inscription « provisoire ». Cette inscription reste provisoire tant que l'élève n'a pas « payé ». Lors que les deux « jetons » sont présents alors l'inscription officielle permet de passer à un état « d'inscrit »

- **Autre méthode: spécifications par interfaces**
  - 1ère étape: approche extérieure de l'application
    - La source des données en entrée: écrans, fichiers
    - Les documents à produire: description
    - Les données à conserver: fichiers
  - 2ème étape: on analyse, on précise
    - Les entités sources et résultats, leurs associations
    - Les traitements à appliquer
    - Les enchaînements logiques
    - Les conditions, contraintes

La spécification par interfaces consiste à recenser les données « visibles » en entrée tels que les écrans, les saisies…les documents à produire…. les données à conserver sous forme de fichiers…

L'analyse des traitements se fait ensuite. On construit un modèle de données en respectant les associations et dépendances décrites dans le cahier des charges, on prévoit de mettre en application les règles de gestion dans les différentes unités de traitements, on prépare l'enchaînement logique des traitements et leur périodicité. Enfin on contrôle que dans les spécifications du logiciel toutes les conditions et contraintes sont respectées.

- **Autre approche: spécifications objet**
  - Représentation des entités et des relations entre entités
  - Les entités sont des objets
  - Un objet a:
    - Un nom
    - Des attributs
    - Des méthodes
  - Les propriétés des relations
    - L'héritage: relation « est un »
    - L'agrégation : relation « est composé de »
    - L'association: relation « père-fils »

L'approche objet consiste à recenser toutes les entités et leurs associations, on utilise des modèles de représentation tel que le modèle entités-associations

Une entité est un « objet »

Un objet porte un nom unique, il a des attributs qui sont les propriétés qui le caractérisent

Les méthodes sont les traitements auxquels l'objet peut être soumis pour produire des résultats.

Cette approche « objet » lors de la phase de spécification et conception du logiciel a l'avantage d'utiliser des propriétés sur les relations:

-l'héritage, un objet peut hériter des propriétés d'un autre objet

-L'agrégation, un objet peut être un objet construit à partir d'un objet plus complexe

-L'association, un objet ne peut exister que si un autre objet existe…

– on construit les spécifications en opérant par :
  - Généralisation: d'une ou plusieurs classes définies on crée une classe mère plus générale
  - Spécialisation: à partir d'une classe définie on crée une ou plusieurs autres classes plus détaillées
  - Extension: on crée une classe supplémentaire et si besoin une classe mère par généralisation
  - Décomposition : une classe est décomposée en sous classes
  - Composition : une classe est composée à partir d'autres sous classes

Un objet est généré selon un modèle générique appelé « classe » qui conserve la description des propriétés qui sont identiques pour les objets de la classe. On dit qu'un objet est une instanciation de sa classe.

La spécification d'un logiciel par une approche objet peut utiliser les mécanismes suivants

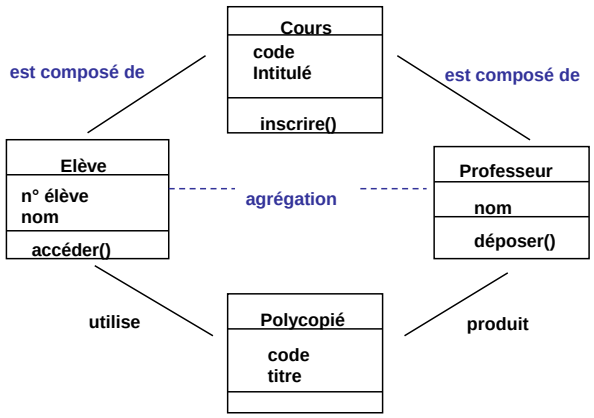
-La généralisation si on compose une nouvelle classe à partir de plusieurs classes existantes

-La spécialisation si on crée une nouvelle classe plus détaillée à partir d'une classe existante

-L'extension si on crée une classe supplémentaire

-La décomposition si on décompose une classe ne sous classes

-La composition si on crée une nouvelle classe à partir d'autres sous classes



Les relations
fonctionnelles: collaborations
structurelles: compositions
Cardinalités: contraintes sémantiques liant deux entités 0, 1 ou n
Schéma similaire au modèle E/R

C. Crochepeyre Génie Logiciel – Rappels 82

Un modèle de classes décrit les entités et associations qui ont été recensées dans le cahier des charges.

la représentation objet utilise:

-les rectangles pour les classes avec leur nom, propriétés et méthodes

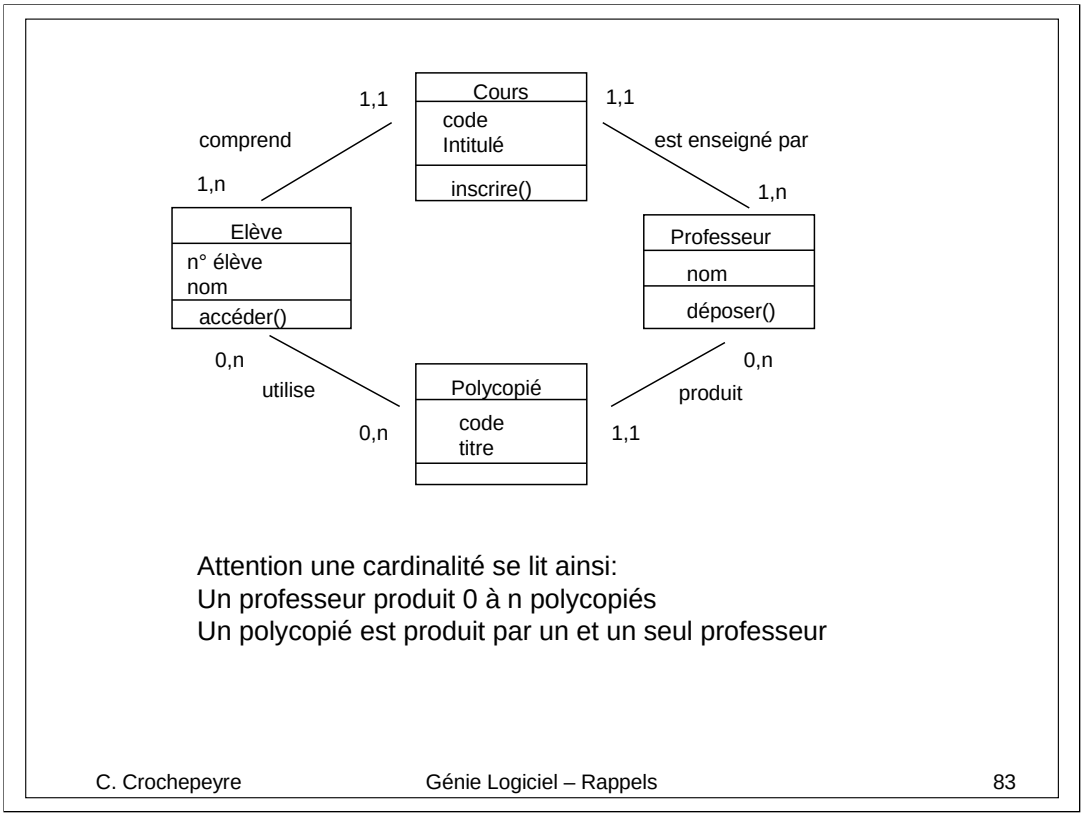
-Les liens représentés par des traits qui associent les classes

-les associations entre les classes avec des verbes pour des relations fonctionnelles ou structurelles

Ici la classe « cours » est composée par la classe « élève » et la classe « professeur ». Un cours ne peut exister sans ces deux entités.

La classe « polycopié » est utilisée par la classe « élève » mais est produite par la classe « professeur »

La classe « cours » a deux attributs: code et intitulé et utilise une méthode (traitement) « inscrire ». Tous les cours sont instanciés à partir de cette classe.



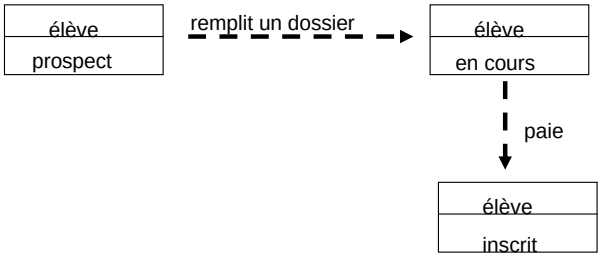
Pour compléter ce diagramme de classes, on peut indiquer la cardinalité des associations que l'on déduit à partir des éléments du cahier des charges. Cette information exprime les règles d'existence des associations

Sur l'exemple, on lit l'information de la manière suivante:

« Un professeur produit aucun ou plusieurs polycopiés »

« Un polycopié est produit par un et un seul professeur »

- Aspect dynamique des objets
  - Les objets changent d'états selon les évènements
  - Diagrammes d'états



C. Crochepeyre Génie Logiciel – Rappels 84

Il est possible d'exprimer les changements d'états en fonction des événements.

Ici on montre qu'un dossier élève est considéré comme « prospect » tant qu'il n'a pas rempli de dossier d'inscription.
Après avoir rendu son dossier celui est « en cours »

Enfin lorsque le paiement est effectué, l'élève est « inscrit »

Les flèches pointillées et commentées montrent la dynamique des objets qui évoluent dans le temps.

- **Utilisation des Patterns**
  - S'appuyer sur des modèles standards
    - Motifs de création
    - Motifs de structuration
    - Motifs de comportement
  - Il s'agit de modèles de conception et non de code
- On utilise les design patterns
  - Pour guider les choix
  - Pour profiter de l'expérience dans le domaine

Les design patterns sont des modèles de cas connus. Les motifs de conception sont des structures de classe qui sont classées en trois familles de motifs:

- motifs de création
- motifs de structuration
- motifs de comportement

Chaque famille comprend plusieurs motifs

Ces motifs sont des modèles de classes que l'on réutilise ou adapte pour éviter de refaire ce qui marche!

Les design patterns concernent des aspects divers: dialogue, données, tableaux, document, tableaux de bord….

- **Conclusion: spécifications du logiciel**
  - Passage de l'expression des besoins à une solution informatique
  - Des méthodes différentes mais:
    - des outils similaires (graphes, textes, tables)
    - des besoins différents (statique ou dynamique TR)
    - des niveaux différents (besoins, logiciels)
    - des apprentissages plus ou moins faciles
  - Puis passage de la spécification fonctionnelle du logiciel à sa conception détaillée

A travers deux méthodes complémentaires SA et SA_RT ou encore avec approche objet, nous avons vu comment spécifier et analyser un système informatique au niveau fonctionnel.

Nous sommes passés de l'analyse des besoins exprimés par l'utilisateur et consignés dans un cahier des charges à une analyse fonctionnelle informatique.

Le découpage du système en unités fonctionnelles nous a obligé à tenir compte pour chacune de ces unités:

- des données en entrée
- des résultats
- des règles de fonctionnement et leurs contraintes
- de l'enchaînement de ces unités

Nous avons vu que l'aspect dynamique du système pouvait être pris en compte par une gestion des contrôles comme dans SA_RT.

Les méthodes et outils employés dans ces méthodes peuvent être différents selon

- la nature des besoins
- les niveaux d'analyse souhaités
- la difficulté d'apprentissage

On remarque que les outils d'aide à la spécification sont similaires : diagrammes, tables, textes..

Mais l'approche peut être différentes: fonctionnelle ou objet

L'analyse étant terminée, passons à la conception du logiciel ou encore analyse détaillée.

3.2. LA CONCEPTION

- **De la spécification à la conception**
  - transformation en unités de programmes des fonctionnalités du logiciel
    - conception fonctionnelle descendante
    - conception orientée objet
  - éléments de conception
    - concepts de structuration
    - méthodes de conception
    - langages de programmation
- **Conception impérative**
  - fonctionnelle descendante
  - modulaire
- **Conception applicative**
  - orientée objets
  - héritage

C. Crochepeyre Génie Logiciel – Rappels 87

La phase de conception est la phase ultime avant la programmation.

On arrive à l'étape de transformation des spécifications logicielles en des unités de programmes.

Il existe deux manières d'aborder cette conception:

- **impérative**

C'est le procédé classique employé depuis des années.

La démarche de conception des fonctionnalités est descendante et modulaire.
Elle conduit à une programmation algorithmique et/ou procédurale.

- **applicative**

Cette nouvelle approche considère que le système agit sur des objets.
Ce type de conception par objet est de plus en plus utilisé de nos jours.

Cette conception conduit à une programmation objet

Quelle que soit la méthode de penser retenue, il s'agit de:

1. Transformer les fonctions en programmes

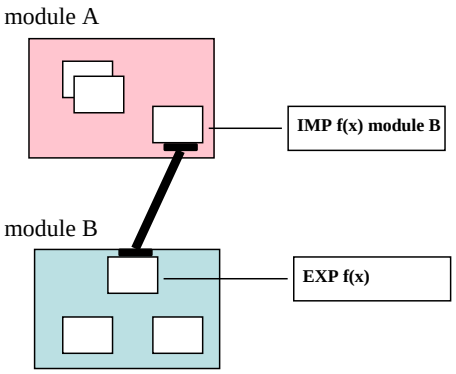
par une conception fonctionnelle descendante

par une conception orientée objet

2. Prendre en compte les éléments suivants:

- les concepts de structuration : modules ou objets, vues internes et externes, respecter les relations
- les méthodes de conception : choix de la méthode et de l'organisation pour la conception
- les langages de programmation: choix des langages en fonction des besoins de conception.

- **Conception fonctionnelle descendante**
  - module
    - unité fonctionnelle reliée aux données
    - langages: Ada, C, Modula 2
  - décomposition d'un module:
    - l'interface: spécifications des composants exportés
    - le corps: composants réalisant les fonctionnalités



C. Crochepeyre Génie Logiciel – Rappels 88

La **conception fonctionnelle descendante** est une analyse **impérative** du logiciel.

C'est la démarche de conception qui conduit à une programmation procédurale.
L'unité de conception ou constituant est le module.

Le module

C'est une unité fonctionnelle du système en liaison avec les données qu'elle manipule.

Citons les langages actuellement utilisés: ADA, C

Les données sont présentées en termes de structures de données.

La décomposition d'un module produit:

- **l'interface 'publique'** partageable et exportable avec d'autres modules

- le corps 'privé' : les informations sont locales au module pour réaliser les fonctionnalités du module.

L'exemple montre deux modules A et B avec l'importation et l'exportation de certaines opérations.

- L'interface
  - spécification d'un module
    - son identification
    - ses liens avec d'autres modules
    - les opérations partageables avec d'autres modules
    - les données partageables avec d'autres modules
    - commentaires
- Le corps
  - description du corps des opérations externes
  - valeurs des données externes
  - liste et description des opérations internes
  - liste des données internes
  - commentaires

C. Crochepeyre Génie Logiciel – Rappels 89

L'**interface** ou spécification d'un module décrit les informations publiques, partageables et exportables vers d'autres modules.
On y trouve :

- L'identification du module : le nom du module, son auteur, les dates de création et modifications, les versions, etc…

- Le lien de référence entre les modules: xxpression des couplages entre les différents interfaces d'autres modules.

- Les opérations partageables qui peuvent être mise à disposition d'autre module: nom, paramètres, contraintes…

- Les données partageables On fournit alors le nom de chaque donnée.

- Des commentaires qui donnent les caractéristiques du module et justifie les choix effectués dans le module.

Le **corps** décrit les informations locales et les informations publiques exportables vers d'autres modules avec :

- La description du corps des opérations externes qui inclut: les déclarations des données locales, les algorithmes en pseudo code,

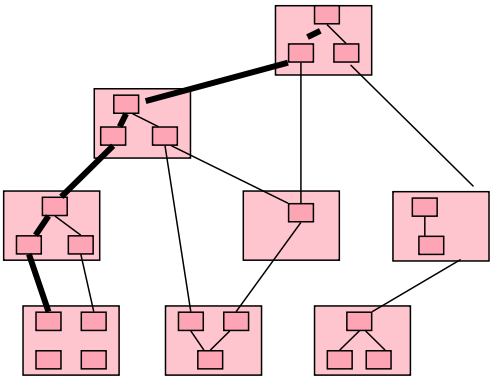
- les valeurs externes: constantes…..

- La liste et la description des opérations internes

- La liste des données internes

- Des commentaires

La hiérarchie des modules



- Propriétés d'un module
  - imbrication de module
  - exportation de module
  - généricité de module
  - couplage de module
    - de données
    - de collection
    - de contrôle
    - de données communes
    - de contenu
  - cohésion
    - fonctionnelle, séquentielle, temporelle..

C. Crochepeyre Génie Logiciel – Rappels 90

La notion de hiérarchie des modules consiste à créer des niveaux ordonnés avec les particularités suivantes :

- chaque niveau est une décision de conception
- chaque niveau représente ce qui est nécessaire pour le niveau suivant
- aucun niveau n'a connaissance des niveaux supérieurs
- des relations d'importations et d'exportations existent entre des niveaux adjacents

Ici un exemple de hiérarchie de modules avec la vue des importations et exportations d'opérations.

- Les types de modules
  - modules de **données**
    - gestion de données statiques
    - gestion de données dynamiques
  - modules de **traitements**
    - entité d'exécution parallèle
    - communications et synchronisation entre processus
  - modules **fonctionnels**
    - imbrications de modules données, traitements et communications
    - composants élémentaires
  - modules de **communications**
    - interface de communications entre modules de traitements
    - communications externes ou internes
  - couches
    - architecture en couches

C. Crochepeyre Génie Logiciel – Rappels 91

On peut décrire cinq types de modules de traitements;

. Le module de données

Il concerne la gestion des données de type abstrait ou un ensemble de données reliées fonctionnellement, indépendamment de leur implantation ou représentation.

Il y a deux sortes de données:

- la donnée **statique** : description d'un état ou environnement non modifiés lors de l'exécution. Initialisée au début du traitement , seule l'accès en lecture est possible

- la donnée **dynamique** : description d'un état modifié en cours d'exécution. L'accès se fait en lecture et en écriture.

. Les modules de traitements

Ils gèrent un ou plusieurs **processus** effectuant le traitement spécifié.
Ce type de module représente une exécution parallèle des processus. D'où le contrôle des communications : communications externes et communications internes.

. Les modules fonctionnels

Ces modules ont des composants de mêmes fonctionnalités associés à des processus.

Ils snt composés:

- **d'imbrications de modules** de données, de traitements et de communications

- et de **composants élémentaires**.

. les modules de communications

Ils ont pour rôle de :

- regrouper des fonctions de synchronisation, de communication, d'échanges de données entre processus.

. rendre la modification des moyens de synchronisation et de communication.
De tels modules sont les chefs d'orchestre des traitements.

On distingue:

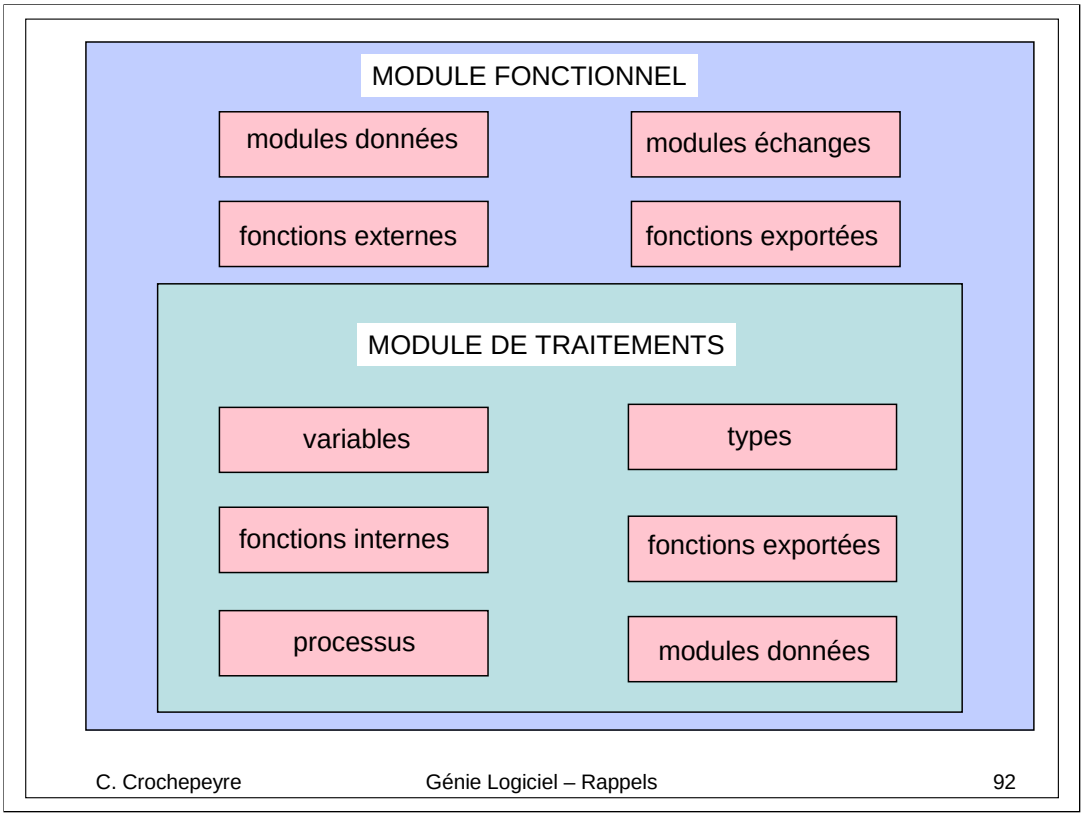
- les communications externes entre processus de modules différents

- les communications internes entre processus du même module.

. Les modules couche

Ce type de module est spécifique aux architectures en couches.

Le modèle de structuration respecte cette architecture.



Ici l'exemple d'un module de « traitements ».
Dans ce module on trouve les éléments concernant :
- les variables
- les procédures et fonctions internes
- les processus
- les types
- les procédures et fonctions exportées
- les modules de données

Ce module de traitement est imbriqué dans un module « fonctionnel ».
Rappelons que dans un module fonctionnel on regroupe les composants d'une même fonctionnalité :
- module de données, de traitements, de communication
- composants élémentaires

- **Méthode SA/SD**

  Structured Analysis Structured Design
  - spécification fonctionnelle
    - analyse structurée SA
    - l'analyse structurée modélise le problème
  - conception préliminaire
    - conception structurée SD (P. Jones 1980)
    - la conception structurée modélise la solution
  - conception détaillée
    - description des modules MD

SA + SD = SA/SD

C. Crochepeyre Génie Logiciel – Rappels 93

Lors de la seconde étape de spécification du logiciel, nous avons présenté la méthode SA qui permet de modéliser les besoins d'un système.
Nous présentons maintenant une méthode qui intègre SA et permet aussi de modéliser la solution : SASD

SA pour analyse structurée

SD pour conception structurée (Structured Design)

La conception structurée SD aboutit à une conception détaillée qui est décrite par des modules MD

Cette méthode s'appuie sur la conception fonctionnelle impérative que nous venons de voir.

Spécification fonctionnelle SA - Conception préliminaire SD - Conception détaillée MD

- **Conception orientée objets**
  - les fonctions changent - les objets restent
  - penser réutilisation
  - unité de conception = classe d'objets
  - langages: Java, C++

"Sur quoi le système agit-il?"

La conception du logiciel orientée objets émane de la pensée applicative.
La conception par objet s'appuie sur le fait que :
- les fonctions changent, évoluent, sont modifiées tout au long de la vie du système.
- alors que les objets sont plus stables
Autre préoccupation:
- la réutilisation des programmes afin d'éviter les problèmes de coûts, délais, faisabilité....
L'unité de conception de la conception O.O est la classe d'objets
Les langages associés à cette pensée applicative sont par exemple : Java, C++
On ne se pose pas la question
« Que doit faire le système? »
mais
« Sur quoi le système agit-il? »

- Un objet, c'est:
  - Un ensemble d'attributs soumis à des actions spécifiques
    - les attributs -> état de l'objet
    - les actions -> les méthodes
- il a un comportement:
  - déterminé par ses actions
  - avec des contraintes sur les actions ou attributs
  - représenté par:
    - réseaux de pétri, automates à états finis, des diagrammes, des matrices, des grafcet....

Un objet est un ensemble d'attributs associé à des actions spécifiques.

- Les attributs d'un objet définissent son état.
Les attributs d'état expriment l'aspect statique de l'objet

- Les actions ou méthodes définissent le comportement de l'objet

Il a un comportement :

- qui est déterminé par ses actions ou méthodes.
Le comportement d'un objet évolue lorsqu'il existe une méthode.
- Les actions peuvent avoir des contraintes

« tel objet a tel comportement existe seulement lorsque..... »

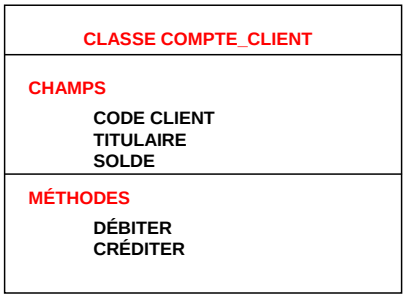
- Les attributs peuvent aussi avoir des contraintes

« la valeur de tel attribut est toujours < 100 »

Les comportements peuvent être représentés par :

- les réseaux de Petri
- diagrammes Etat-Transition (SA_RT)
- matrices Etat-Evènement
- automates à états finis
- Grafcet
- tables de pré et post conditions
- etc....

- Classe d'objets
  - ensemble d'objets
    - mêmes attributs statiques
    - mêmes attributs dynamiques
  - une instance est un objet particulier de la classe
    - crée avec les attributs d'états de la classe
    - se comporte selon les méthodes de la classe



C. Crochepeyre Génie Logiciel – Rappels 96

Toutes les classes d'objets doivent être définies de manière précise lors de la conception du logiciel. Ces classes sont les résultats des informations recueillies lors des étapes précédentes de l'analyse.

Ainsi l'ensemble des objets du système sont regroupés par classe en considérant les objets ayant les mêmes attributs statiques et dynamiques

Les classes sont instanciées pour créer un objet.

Chaque objet possède alors les attributs d'états de la classe et son comportement sera celui décrit dans les méthode de la même classe.

L 'exemple montre une classe « compte-client » dont les attributs sont:

-Le code client

-Le nom du titulaire

-Le solde

Avec ses méthodes qui seront activées à la demande:

-Débiter le compte

-Créditer le compte

- Héritage
  - des sous-classes: organisation hiérarchique
  - héritage des champs et des méthodes
    - extension des champs et méthodes de la classe dans la sous-classe
    - évite la duplication de code
  - héritages:
    - simple: nouvelle classe issue d'une classe origine
      - de toutes les propriétés
      - de certaines propriétés
    - multiple: nouvelle classe issue de plusieurs classes

L'une des propriétés de la conception objet est l'héritage

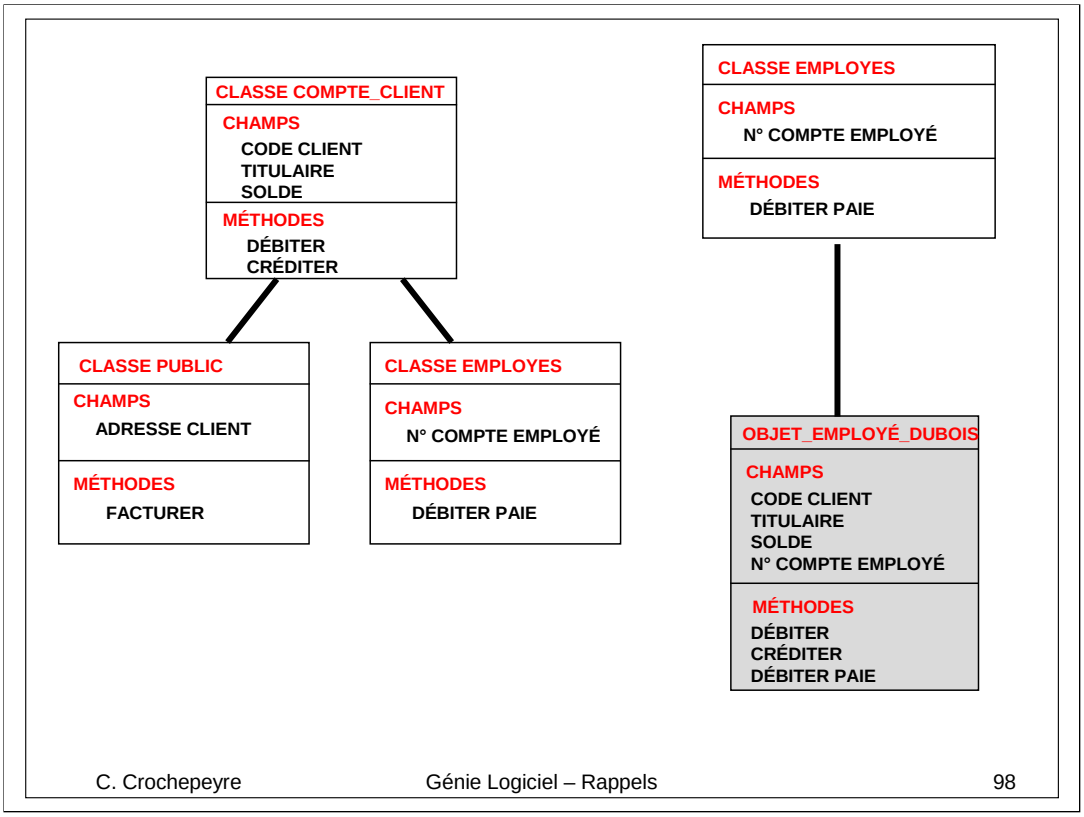
Les classes sont organisées de manière à pouvoir hériter des propriétés de classe déjà existantes et qui ont des propriétés communes.

Ainsi la nouvelle classe créée hérite des champs (attributs) et des méthodes (fonctions) et étend sa classe avec des champs nouveaux et des méthodes qui lui sont propres

Cet héritage évite les redondances et permet de dupliquer du code déjà conçu.

On parle d'héritage simple lorsque la nouvelle classe est issue d'une classe d'origine avec toutes les propriétés ou une partie des propriétés

Et on parle d'héritage multiple lorsque la classe est issue de plusieurs classes.



Les exemples de sous classes de la classe COMPTE_CLIENT sont une classe PUBLIC et une classe EMPLOYE.

En effet un CLIENT peut être un client public dont les achats seront facturer (facture) ou un client employé de la société dont les achats internes seront débités sur sa paie. Dans les deux cas les comptes leur compte seront débiter du montant des achats (COMPTE_CLIENT).

La classe EMPLOYES hérite donc des propriétés de la Classe COMPTE_CLIENT avec

-Le code client

-Le nom client

-Le solde

-Le n° code employé

Et les méthodes:

-Débiter solde

-Créditer solde

-Débiter paie

Dans le cas d'un compte client public, l'envoi de la facture au client déclenchera une procédure de paiement par chèque ou virement et une mise à jour du solde.

- L’encapsulation
  - Opacité du fonctionnement interne d’un objet
  - Accès aux seules propriétés et services nécessaires
- Le polymorphisme
  - Traitement d’adaptant aux diverses versions des objets
- La surcharge
  - Réécriture d’un traitement hérité: ajout ou suppression de fonctionnalités
- L’abstraction
  - Le traitement (abstrait) est défini en fonction des objets inférieurs

Le mécanisme d’encapsulation consiste à mettre dans une structure protégée les propriétés et méthodes d’un objet afin de protéger son usage.

Le fonctionnement de l’objet n’est pas connu et les accès sont contrôlés, comme

public : les attributs dits publics sont accessibles à tous,

protégé : les attributs dits protégés sont accessibles seulement aux classes dérivées,

privé : les attributs privés sont accessibles seulement par l'objet lui-même.

Le polymorphisme concerne les méthodes. De manière générale une méthode peut agir de différentes façons

Comme le polymorphisme ad hoc : la méthode qui a un nom « afficher » aura des comportements différents en fonction de l’objet qui l’appelle comme afficher image, afficher tableau.

Le polymorphisme paramétrique qui applique la bonne méthode en fonction du type de la donnée, si plusieurs méthodes portent le même nom…. Des variantes existent.

La surcharge est une sur-définition des propriétés héritées (attributs, méthodes) : on ajoute ou supprime des fonctionnalités comme un n° de client qui devient un n° de client employé

L’abstraction est une interprétation du particulier au général: le traitement est défini en fonction des objets des classes inférieures

3.3. Quelques méthodes d'analyse et de conception

- SADT
- SART
- SA/SD

- Merise

- OMT de J. Rumbaugh
- OOD de R. Abbott et G. Booch
- OOSE de I. Jacobson
- UML (1) Unification des méthodes de Booch, Rumbaugh et Jacobson
- UP Unified Process utilisant la méthode de notation UML

(1) : chapitre suivant

Voici une liste de méthodes d'analyse et de conception dont certaines sont plus anciennes mais persistent dans leur utilisation et d'autres plus récentes, en particulier celles qui sont liées à l'approche objet

- **Conclusion**
  - conception du logiciel = étude détaillée
  - éclatement des fonctions en unités de programme
  - interfaces entre les modules
  - description des données en entrée
    - origine, format, contraintes
    - écrans
  - description des données en sortie
    - format, présentation
    - impression des résultats
  - stockage
  - description des traitements

La conception du logiciel, que nous venons de décrire, ne signifie pas « programmation » mais conception détaillée informatique faite à partir des spécifications informatiques, elles mêmes étant la transposition du modèle décrit dans le cahier des charges.

Les fonctionnalités du projet sont décomposées en fonctions informatiques (modules, méthodes), les informations (données, objets) sont organisées et conservées sur des supports informatiques. Les interfaces entre les fonctions respectent les règles énoncées précédemment.

On distingue les données en entrées: leur origine, format, contraintes associées comme pour la description d'un écran de saisie ou les données en sortie avec une organisation et une présentation comme pour un état d'impression.

C'est aussi un travail d'organisation des données et du choix du support comme une base de données sur disque.

De manière plus large on dira que la conception du logiciel est une étude détaillée des traitements et des données informatiques qui répond aux exigences du cahier des charges.

PLAN

1. CYCLE DE VIE DU LOGICIEL
2. EXPRESSION DES BESOINS
3. CONCEPTION DU LOGICIEL
4. **LA PROGRAMMATION**
5. TESTS ET MISE AU POINT
6. DOCUMENTATION
7. CONCLUSION

La programmation est l'ultime étape de la production du logiciel.

Les étapes précédentes sont déterminantes car lors de la programmation on ne devrait pas s'interroger sur ce que l'on doit réaliser, ni comment mais simplement utiliser le langage de programmation pour aboutir aux résultats prévus.

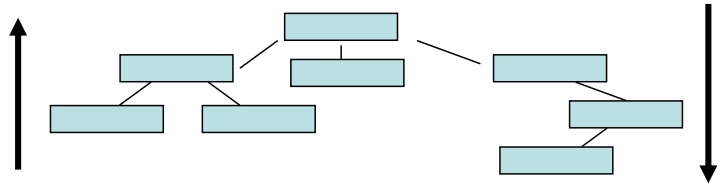
Encore faut-il choisir le bon langage!

4. LA PROGRAMMATION

- La méthodologie
- La lisibilité des programmes
- Les outils
- La portabilité des programmes
- Les langages

Les points abordés dans ce chapitre concernent
-la méthodologie qu'exige la programmation,
-les points importants comme la lisibilité des programmes,
-les outils qui peuvent apporter une aide au développement,
-les préoccupations lors de la production d'un logiciel comme sa portabilité d'un site à un autre et enfin
-les divers langages de programmation, le bon choix.

- **Méthodologies**
  - passage de l'analyse à la programmation
  - méthodologie ascendante
  - méthodologie descendante
- **Masquage des informations**
  - accès aux seules informations nécessaires/programme
  - appliqué dans la conception O.O
  - sécurité + indépendance des données



C. Crochepeyre Génie Logiciel – Rappels 104

Concevoir un programme peut se faire en développant les concepts du général au particulier ou inversement.

Les tâches peuvent être réparties entre plusieurs personnes à condition que le découpage soit fait clairement et avec les bonnes compétences.

Chaque module doit être conçu et préparé de manière à ne considérer que les éléments nécessaires et suffisants pour la programmation du module. Le masquage des informations inutiles apportera clarté et précision au programme.

La sécurité des données, l'indépendance des données (propres au module) ou dépendances (entre des modules) sont considérées à ce niveau avec beaucoup de soin.

- **Lisibilité des programmes**
  - dépend du langage et du style d'écriture
  - le choix des noms
  - la mise en page des programmes
  - de bonnes structures de contrôles (boucles, condition.)
- **Portabilité des programmes**
  - Compilation sur la machine cible
  - dépendance due à l'architecture machine
  - dépendance due au système d'exploitation

C. Crochepeyre Génie Logiciel – Rappels 105

N'oublions pas que certains langages sont complexes, peu lisibles, d'autres plus proches de notre langue naturelle mais qu'il est très facile de rendre un programme peu lisible et ce de fait peu exploitable par d'autres personnes;
Des recommandations sont toujours d'actualité comme:

-faire le bon choix des noms de modules, variables ou autres,

- pendre du temps de mettre en page les lignes de code afin d'en faciliter la lecture

-Choisir de bonnes structures de contrôles même si a l'écriture cela prend un peu plus de temps

Autre aspect important à considérer est qu'un programme doit être portable d'une plate-forme à une autre, d'un site à un autre, … les architectures machines diffèrent, les systèmes d'exploitation, les performances aussi…

Cette remarque pourrait être accompagnées de bien d'autres remarques de ce type mais l'essentiel est de considérer lors de la programmation qu'un logiciel vit et dure dans le temps et que l'environnement humain et matériel évolue.

- **Les outils**
  - outils de préparation des programmes
    - éditeurs
    - outils de traduction: interpréteurs, compilateurs…
  - outils d'analyse
    - références croisées
    - mise en forme du source
    - liste de partie de programme
  - outils de gestion
    - traces du développement
    - suivi de la cohérence des versions
      - SCCS et MAKE (unix)
      - RCS (GNU équivalent SCCS),
      - CVS : Concurrent Versions System

C. Crochepeyre Génie Logiciel – Rappels 106

Lors qu'on parle de programmation on associe au langage l'environnement de programmation : les outils qui l'accompagnent, la facilité de mise en œuvre, le jeu des instructions, la rapidité d'écriture…..

On pense aussi à des environnements particuliers qui doivent aider à la mise en œuvre des programmes: les AGL ou ateliers de génie logiciel. Ces AGL proposent au développeur un ensemble cohérent de logiciels pour l'aider à concevoir, contrôler, tester, archiver…. des programmes.

Parmi les outils, inclus ou non dans un AGL, on trouve:

- les outils de préparation des programmes comme des éditeurs de programmes, des outils de traduction: interpréteurs, compilateurs plus ou moins performants

- les outils d'analyse et contrôles comme ceux qui donnent les références croisées à parti du code écrit, qui aident à la mise en forme des instructions ou proposent différentes forme de visualisation des lignes de codes (écran, impression…)

- les outils de gestion qui permettent au programmeur de suivre la trace d'exécution des instructions de son programme ou plus globalement des outils de gestion des différentes versions du programme

- Les environnements de programmation
  - logiciels de communication entre
    - machine développement et machine cible
  - outils de tests et mise au point
    - simulateurs de machine cible, bancs de test,
    - analyseurs de programmes
  - outils de spécifications fonctionnelles
  - outils graphiques de description
  - outils de gestion de projets: génération de rapports d'avancement du projet

Nous avons cités précédemment les AGLs mais de manière plus générale les environnements de programmation sont très divers, citons:

- les logiciels de communication entre l'ordinateur sur lequel on développe le programme et la ou les ordinateurs cibles

- les outils de tests comme les bancs de tests et simulateurs ou analyseurs de programmes

- Les outils de spécifications ou les logiciels de description graphiques

- les outils de gestion de projet…..

- **Les langages**
  - les langages d'assemblage : processeur
    - Intel – Motorola - …..
  - les langages de réalisation de systèmes
    - C
  - les langages statiques de haut niveau
    - COBOL, Visual Basic
  - les langages de haut niveau à structure de blocs
    - Ada
  - les langages dynamiques de haut niveau
    - Prolog
  - les langages objets
    - C++ - Java
  - les langages spécialisés: Perl, SQL, HTML, XHTML, PHP...

C. Crochepeyre Génie Logiciel – Rappels 108

Les langages sont nombreux et ont des usages qui sont différents:

- les langages d'assemblage sont réservés à des usages très particuliers comme le domaine des systèmes embarqués dont les codes exécutables doivent être le plus efficaces possible avec le moins d'instructions possibles (mémoire limitée, temps d'accès contrôlé…)

Exemple MASM assembleur Microsoft pour Intel famille X86

- les langages de réalisation de systèmes d'exploitation ou langages qui y sont proches

Le langage C est un bon exemple puisqu'il a été créé pour le portage du noyau du système d'exploitation Unix sur des machines d'architectures différentes. Il reste un langage de programmation offrant de nombreuses possibilités de contrôles sur la gestion des processus, la synchronisation, les communications, la messagerie… qui sont des fonctions de base d'Unix.

- les langages statiques de haut niveau

Cobol existe toujours et il est utile pour les applications de gestion. La programmation de ces applications avec des structures de données lourdes (en entrées comme en sorties) ainsi que les traitements qui leur sont propres nécessite de tels langages.

Visual basic a été créé lors de l'apparition des micro-ordinateurs pour lesquels il était utile d'avoir un langage pour programmer de petites applications. Il a évolué ensuite.

- les langages dynamiques de haut niveau

ADA a été conçu pour des applications militaires et industrielles. Il a la particularité d'être structuré avec des concepts comme : les paquetages, les types, les objets.

- les langages dynamiques de haut niveau

Prolog est très particulier il utilise la logique des prédicats. On écrit des clauses dans des paquets qui constituent des prédicats. A l'exécution d'un prédicat la réponse sera positive et produira des résultats ou négative et alors un autre prédicat sera exécuté.

- les langages objets

Utilisation actuelle de ce type de langage dont la particularité et de considérer qu'un objet du monde réel appartient à une classe qui le caractérise. Les fonctions ou méthodes (traitements) qui sont exécutées dépendent de la classe.

C++, Java sont des langages fortement utilisés.

- les langages spécialisés

Ce sont tous des langages en limite de la programmation. Les programmes ou séquences d'exécution d'ordres sont en général dédiés à des actions particulières sur les données.

SQL est un langage de définition et de manipulation de données dans une base de données, Perl est un langage des script très complet, HTML issu du méta langage SGML permet d'introduire dans des textes des balises et le rendre ainsi exploitable par des interfaces différentes d'un système à un autre, etc…

PLAN

- 1. CYCLE DE VIE DU LOGICIEL
- 2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- **5. TESTS ET MISE AU POINT**
- 6. DOCUMENTATION
- 7. CONCLUSION

Les tests qui sont mis en place en fin de production des programmes doivent prouver le bon fonctionnement de chaque module.

5. TESTS ET MISE AU POINT

- Tests
  - détection des erreurs
  - absence d'erreurs?
  - choix judicieux des tests
- Mise au point
  - localisation des erreurs et corrections
  - mode d'écriture facilitant la mise au point

Les tests sont des jeux de données inscrits dans des scenarios pré établis qui doivent prouver:

- qu'il n'y pas d'erreurs: de résultats faux, de valeurs imprécises, pas de résultats…

- qu'en l'absence d'erreur, cela ne cache pas un dysfonctionnement

- que tous les jeux de tests ont été choisi de manière judicieuse: complet, invoquant toutes les possibilités fonctionnelles avec des données réelles…

La mise au point suppose qu'à la détection d'erreurs on retrouve dans le programme l'origine de l'erreur et que l'on fasse le nécessaire pour la corriger.

Bien entendu, la qualité d'écriture des programmes facilite cette mise au point;

- **Les tests**
  - les types de tests
    - tests unitaires
    - tests de modules
    - tests de sous-systèmes
    - tests d'intégration
    - tests d'acceptation
  - méthode descendante ou ascendante
    - des tests sous-systèmes vers les tests unitaires
    - des tests unitaires vers les tests sous-systèmes

C. Crochepeyre Génie Logiciel – Rappels 111

On classe les tests par type:

- les tests unitaires sont faits les premiers car ils consistent à éprouver chaque unité fonctionnelle

- les tests de module intègrent plusieurs unités fonctionnelles

Et ainsi de suite, de manière graduelle on intègre des programmes que l'on a testés précédemment.

L'intégration totale est l'étape finale de test qui prouve que l 'ensemble de l'application fonctionne correctement.

Lors de la recette du produit par le client, celui effectue ses propres tests et accepte après corrections éventuelles le produit qui lui est livré.

Nous avons décrit un processus de test allant du module le plus simple au module composé plus complexe, procédure ascendante. Lors de cette phase, on est bien obligé de procéder aussi de manière descendante lors qu'une erreur est détectée et que l'on doit situer l'erreur.

- la conception des tests
  - ensemble de données réalistes
  - spécifications entrées
  - + spécifications fonctions
  - + spécifications sorties
  - effets des données incorrectes
  - combinaisons de données
  - générateur automatique de données
- les tests de programmes temps réel
  - interactions entre processus
  - événements externes
  - opérations de tests graduelles

C. Crochepeyre Génie Logiciel – Rappels 112

Concevoir une série de tests est long et doit être fait avec rigueur et méthode.

Il ne s'agit pas de créer quelques données imaginées mais de faire une étude à partir des spécifications des besoins. Il faut recenser des cas reflétant la réalité. Il s'agit aussi de tester la montée en charge du système en considérant des volumes de données en rapport avec la réalaité.

Les jeux de tests sont décrits avec des données précises en entrée du système, qui sont soumises aux traitements des fonctions et doivent produire les résultats qui sont calculés auparavant.

Lors des tests on note les effets produits lors de données incorrectes.

On teste les combinaisons de données. Des générateurs de données peuvent dans certains cas être utiles.

Si nous considérons les applications dites temps réel, il faut aussi tenir compte des contraintes qui leur sont particulières.

Il est important de contrôler les effets de bords d'événements synchrones et asynchrones et de procéder de manière très graduelle.
Très souvent ces applications ne tolèrent pas d'erreurs et sont testées avec des garanties (sécurité)

- la vérification des programmes
  - correspondance entre programme et spécification
  - réduction des coûts de tests
  - preuve mathématique des programmes
- les inspections de codes
  - lecture du code et explications devant l'équipe
- les outils de validation
  - générateurs de tests
  - analyseurs de flots continus
  - les comparateurs de fichiers
  - les simulateurs
  - les vérificateurs de programmes

C. Crochepeyre Génie Logiciel – Rappels 113

Comment vérifier qu'un programme fonctionne bien?

- simplement en regardant si la programmation est bien celle qui correspond aux spécifications décrites lors des étapes précédentes

- notons qu'une vérification préalable du code réduira les coûts des tests ensuite

- il existe des procédures plus complexes qui consistent, lorsque cela est possible, de prouver mathématiquement que le code produit est correct.

Certaines méthodes préconisent de travailler en équipe, d'autres en binôme…. En effet en lisant le code et en expliquant ce que l'on a fait à quelqu'un, cela peut être une façon simple de mettre en évidence des incohérences.

Les outils de validation et d'aide aux tests sont divers:

- des générateurs automatiques de données

- des analyseurs de données, de flots de données

- des comparateurs de fichiers: tailles des fichiers qui diffèrent ou analyses des contenus

- des simulateurs qui sont très courants pour les applications industrielles qui sont développées sur des systèmes de développement et qui migrent ensuite sur des sites cibles d'exploitation

-etc….

- **La mise au point**
  - identification la cause des erreurs
  - localiser les erreurs
  - modifier le code
  - liste des résultats de tests
  - trace de l'exécution
  - les outils de mise au point
    - vidage mémoire
    - débogueurs symboliques
    - analyseurs de programmes statiques
    - analyseurs de programmes dynamiques

La mise au point est en général plus longue que prévue!

Identifier la cause d'une erreur peut très rapide ou inversement très long mais après l'identification de la cause il s'agit de localiser dans le code les instructions concernées. Lorsque la correction est faite il faut s'assurer que la nouvelle exécution est bien celle attendue.

Il faut donc à chaque nouvelle exécution de test conserver les données qui ont été utilisées et les résultats, bons ou mauvais, associés.

Encore mieux si on a utilisé un traceur (debogueur) d'exécution, conserver une trace des étapes importantes de validation est parfois très utile.

Il existe des outils d'aide à la mise au point permettant: une visualisation du vidage de la mémoire pour contrôler les données dans des variables, des structures; cette trace de la mémoire peut être faite à la demande ou à des étapes précises.

Des outils d'analyseurs de programme statiques, hors exécution, sont utiles pour détecter des anomalies de codage comme des références non utilisées, des boucles s ns fin… En revanche les analyseurs dynamiques sont utiles pour vérifier le bon fonctionnement du programme lors de son exécution.

PLAN

- 1. CYCLE DE VIE DU LOGICIEL
- 2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- 5. TESTS ET MISE AU POINT
- **6. DOCUMENTATION**
- 7. CONCLUSION

La documentation est la partie visible du logiciel qui est indispensable pour le développement et la maintenance du logiciel.

6. DOCUMENTATION

- Documentation
  - les différents documents
  - la qualité des documents
  - les outils de production
- Maintenance
  - des logiciels et des documents

Le travail de documentation aux différentes étapes du développement du logiciel est un travail important et qui doit être rigoureux.

La documentation est utile :

- lors du développement
- lors de la maintenance

Les documents précisent comment utiliser le programme, comment il a été développé et pourquoi.

La maintenance nécessite une très bonne connaissance du logiciel que l'on acquiert par ces documents.

Nous allons présenter

- les différents documents aux différentes étapes
- la qualité des documents
- les outils de production et de maintenance de ces documents
- la maintenance des logiciels

Nous ne détaillerons pas chaque document, ce point est abordé dans le cours Conduite de projet.

- **Les documents d'un logiciel**
  - doc interne: informaticiens
    - développement du logiciel
    - maintenance du logiciel
  - doc externe: utilisateurs
    - présentations des fonctions
    - pas de détail de réalisation
  - L'ensemble des documents
    - un cahier des charges: besoins
    - un dossier de spécifications
    - un dossier de conception détaillée
    - un dossier de programmation
    - un dossier des procédures de tests
    - un manuel d'installation et de mise en œuvre
    - un manuel d'utilisation

C. Crochepeyre Génie Logiciel – Rappels 117

Il existe deux niveaux de documentation :

- une documentation interne pour les développeurs et la maintenance du logiciel dans laquelle sont consignées les aspects de conception, réalisation et validation.

- une documentation externe destinée aux utilisateurs avec présentation des fonctions sans détail technique

L'ensemble de la documentation comprend :

- un cahier des charges ou encore expression des besoins

- un dossier d'analyse fonctionnelle

- un dossier de conception : analyse détaillée

- un dossier de programmation avec le code des programmes

- un dossier de procédures de tests et validation.

et

- un dossier d'installation du logiciel

- un dossier d'utilisation

Le premier document permet au client d'exprimer **ses besoins** seul ou avec l'aide du développeur.

Dans ce cahier des charges, il transmet alors toutes les informations et les règles de gestion qui doivent être comprises par le développeur pour réaliser le logiciel.

Le document d'analyse fonctionnelle propose **une solution informatique**. Il résume les besoins et explique l'organisation fonctionnelle du logiciel.

On y insère des exemples simples

On n'entre pas dans le détail des modules mais c'est un document qui sert plutôt à donner une vue générale de la solution informatique.

L'analyse détaillée reprend l'architecture générale de la solution informatique retenue et détaille chaque unité

Le dossier de programmation regroupe l'ensemble **des programmes**.

Les procédures de **tests** sont énoncées dans un document.

Les différents cas et jeux de tests correspondants y sont décrits.

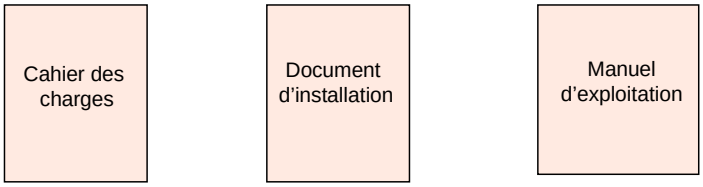
Lorsque les tests sont effectués on complète avec les résultats.

Le document **d'installation** explique comment installer le logiciel en l'adaptant à la configuration matérielle et logicielle.

Plusieurs descriptions sont nécessaires:

- support sur lequel le logiciel est fourni
- configuration matérielle minimale

- La documentation utilisateurs
  - choisir une structure de document adaptée
    - niveaux généraux et niveaux détails
  - des documents en fonction des usages:
    - réponse aux besoins
    - installation
    - démarrage
    - fonctionnement



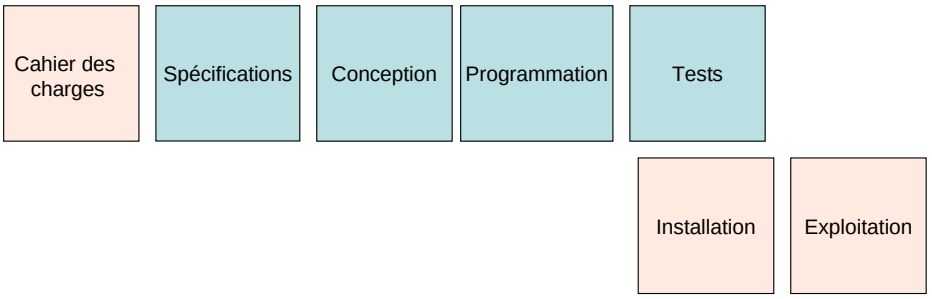
C. Crochepeyre Génie Logiciel – Rappels 118

Les documents doivent fournir les informations à deux niveaux:
- niveau général
- niveau détail
Chaque document doit être rédigé en fonction du profil de l'utilisateur du document.
Le cahier des charges : le client
Le manuel d'installation: un technicien
Le manuel d'utilisation : l'usager du logiciel
Les autres documents : les informaticiens développeurs.

Pour chaque document se poser les questions:
- qui consulte?
- que doit-on y trouver?
Les informations sont souvent redondantes, plus ou moins détaillées avec des présentations différentes.

- La documentation développeurs
  - un cahier des charges: besoins
  - un dossier de spécifications fonctionnelles
  - un dossier de conception détaillée
  - un dossier de programmation
  - un dossier des tests et validation

  - un manuel d'installation et de mise en œuvre
  - un manuel d'utilisation



C. Crochepeyre Génie Logiciel – Rappels 119

La documentation interne est destinée aux développeurs.

Elle est composée de tous les documents qui ont servi à la réalisation :

- définition des besoins avec justification des principes énoncés

- spécifications fonctionnelles.

Comment les besoins sont satisfaits. Les grandes fonctions ou programmes et leurs interactions.

- les unités de programmes.

Pour chaque unité une description de la conception globale de cette unité et son détail si nécessaire.

- les tests unitaires avec un plan de tests pour chaque unité.

- les tests d'intégration avec un plan de tests

- les tests de validation du système complet et une vision future éventuellement.

- le dictionnaire de données

chaque entité du système est détaillée. Pour accéder à un objet il suffit de le nommer afin de le retrouver dans ce dictionnaire.

Très usité et indispensable pour les bases de données mais aussi à d'autres fins.

Il peut contenir des éléments tels que procédures, enregistrements, fichiers.....

Les utilisateurs sont aussi amenés à utiliser le dictionnaire de données. Un mode interactif est vivement conseillé.

- La qualité des documents
  - écriture
  - présentation
  - complétude
  - Actualisation

CHARGE IMPORTANTE

- Quelques conseils
  - construire des phrases simples, une seule idée par phrase, paragraphes courts
  - faire attention à l'orthographe !
  - utiliser des références explicites
  - présentation sous forme de tableaux ou listes
  - répéter les descriptions complexes
  - termes précis avec glossaire

C. Crochepeyre Génie Logiciel – Rappels 120

Les documents sont trop souvent:
- mal écrits
- difficiles à comprendre
- périmés
- incomplets

Le travail de documentation est sous estimé car il est important et coûteux.

La solution est de mettre en place une structure de production de documents commune et adaptée:
- présentation
- notations
- formats des couvertures
- numérotations des pages
- renvois, titres, sous titres.....

Notons que les méthodes vues précédemment proposent des outils graphiques et textuels aux différentes étapes pour élaborer ces documents.

Il n'en reste pas moins que la qualité des documents dépend :
- du style d'écriture
- de la présentation
- de la complétude des informations
- de l'actualisation des documents

C'est une charge importante.

Quelques conseils concernant le style d'écriture pour des documents clairs et concis.
- construire des phrases simples (utiliser le mode présent)
- correction orthographique
- une seule idée par phrase
- utiliser des références explicites : quelques mots plutôt que des n° de renvoi ou trop d'abréviations
- présenter chaque fois que possible par liste ou tableau des informations s'y prêtant: plus clair qu'une longue phrase

- **Les outils de production de documents**
  - production du logiciel et de la documentation sur la même machine
- **Maintenance des documents et logiciels**
  - modification simultanée avec le système
  - numérotation -> remplacement des sections
  - les types de maintenances:
    - amélioration
    - adaptation
    - correction
  - la portabilité
    - la documentation doit accompagner le logiciel

C. Crochepeyre Génie Logiciel – Rappels 121

Il existe des outils de
- production
- maintenance
des documents intégrés dans les AGL ou indépendants.

Une recommandation importante:
- utiliser si possible la même machine pour le développement du logiciel et sa documentation.

En effet cette solution offre les avantages suivants:
- documentation toujours accessible
- facilité de maintenance donc documents à jour
- analyse automatique des documents possible comme la création d'index ou contrôle de l'orthographe des noms.
- simplification de la gestion des documents: outil de gestion des versions, des configurations

La solution présentée précédemment montre le souci de maintenance simultanée du système et de ses documents.

La numérotation des documents permet de remplacer facilement des sections : section 3-1-1 remplacée par document 3 section 1-1

Les causes des modifications sont diverses:
- amélioration : changements demandés par les utilisateurs ou programmeurs
- adaptation: le changement de l'environnement peut conduire à des modifications
- correction : nécessaire en cas d'erreurs

La maintenance est inévitable mais les coûts sont difficiles à évaluer.

Si l'on pense à la portabilité du logiciel, on doit aussi penser à la portabilité de la documentation....

- Les coûts de maintenance
  - Critères généraux
    - nouveauté du domaine de l'application
    - stabilité du personnel de développement
    - durée de vie du logiciel
    - dépendance avec l'environnement
    - stabilité du matériel
  - Critères techniques
    - modularité du logiciel
    - langage de programmation
    - style de programmation
    - qualité des tests et validation
    - qualité de la documentation

C. Crochepeyre Génie Logiciel – Rappels 122

Certains critères peuvent déterminer les coûts de maintenance:

- Le type d'application

Si c'est un nouveau domaine d'application, sans expérience préalable il est fort probable que la maintenance importante

- La stabilité du personnel

L'idéal est de faire assurer la maintenance du logiciel par celui qui en a fait le développement

- La durée de vie du système

Les coûts augmentent avec l'âge des systèmes. Des systèmes de plusieurs dizaines d'années sont encore en service....

- La dépendance avec l'environnement

L'exemple du changement d'un taux de TVA qui doit se répercuter dans plusieurs applications.

- La stabilité du matériel

L'évolution du matériel qui se prévoit selon un plan d'investissement à long, moyen ou court terme nécessite une évolution du logiciel selon le même plan.

Il faut aussi penser aux critères plus techniques tels que:

- La modularité

Concevoir des modules de manière à ne pas remettre en cause une suite d'autres modules en cas de modification

- Le langage de programmation

Facilité de maintenance avec des programmes écrits dans des langages de haut niveau

- Le style de programmation

La maintenance sera d'autant plus facile que les programmes seront clairs: écriture et présentation

- La qualité des tests et validation

Le temps consacré à la période de tests et à l'évaluation influe sur le temps de maintenance par la suite.

Notons que les erreurs de codage sont plus faciles à corriger que les erreurs de conception.

- La qualité de la documentation

Point sur lequel nous avons insisté précédemment.

Il existe des outils de
production

PLAN

- 1. CYCLE DE VIE DU LOGICIEL
- 2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- 5. TESTS ET MISE AU POINT
- 6. DOCUMENTATION
- **7. CONCLUSION**

7. CONCLUSION

- Le développement d'une application exige de:
  - 1. procéder par étapes
    - prendre connaissance des besoins
    - effectuer l'analyse
    - trouver une solution informatique
    - réaliser
    - tester
    - installer
    - assurer le suivi

- 2. Procéder avec méthode
  - du général au détail et au technique
  - fournir une documentation
  - s'aider de méthodes appropriées
- 3. Savoir se remettre en question
  - bonne construction?
  - bon produit?
- 4. Choisir une bonne équipe
  - trouver les compétences
  - définir les missions de chacun
  - coordonner les actions

- 5. Contrôler les coûts et délais
  - aspect économique
  - bonne maîtrise de la conduite du projet
  - investissements au bons moments
- 6. Garantir le succès du logiciel
  - répondre à la demande
  - assurer la qualité du logiciel
- 7. Envisager l'évolution
  - du logiciel
  - du matériel
  - de l'équipe

***Loi de Murphy*:**

« Si quelque chose peut mal tourner,
alors ça tournera mal. » !!!!

BIBLIOGRAPHIE

- Voir la bibliographie de votre cours de génie logiciel