Génie Logiciel

- Intervenants
  - Laurent TICHIT (617) tichit@lif.univ-mrs.fr
  - *(Noël NOVELLI (613) novelli@lif.univ-mrs.fr)*
  - Liva RAILAVOLA (620) liva@lif.univ-mrs.fr
- Organisation
  - 20h de cours, 20h de TD et 20h de TP
  - Plusieurs projets
- Supports : http://www.dil.univ-mrs.fr/~tichit/GL
- Notation : examen (1/2), projets (1/2)

Ce support a été rédigé en utilisant les supports d'**Isabelle VALEMBOIS** (Thales-services) et de **Jean-Yves ANTOINE** http://www.sir.blois.univ-tours.fr/~antoine/ (Pr. Université de Tours)

Notes de l’an passé-k

Moyenne	Exam	Projets
17	17	17
15,9	15,8	17
15,8	15,2	17
15,4	14,7	17
14,7	14,6	17
14,7	14	17
14,3	14	17
14,3	14	17
14,3	13,9	16
14,3	13,4	16
14,2	13,4	16
14	13,4	16
13,9	13,3	16
13,7	13	16
13,6	13	16
13,6	13	16
13,4	12,8	16
13,3	12,2	16
13,3	12,1	16
13,1	12	16
13	11,9	16
12,9	11,8	15
12,9	11,8	15
12,9	11,6	15
12,9	11,6	15
12,7	11,3	15
12,7	11	15
12,7	10,6	15
12,2	10,3	15
11,9	10,2	15
11,8	10,1	15
11,5	9,8	15
11,4	9,8	15
11,4	9,7	15
11,2	9,2	15
11,1	9,2	15
10,6	8,4	15
10,6	7,9	15
10,5	7,2	14
9,5	6,8	14
8,8	5,7	14
8,2	4,3	14
6,8	4,3	14
6,1	3,1	14
5,9	2,7	14
4,7	2,1	14
4,7	0,3	14
4,6		5
3,7		5

Moyennes	11,76938776	10,58510638	15
Min	3,7	0,3	5
Max	17	17	17

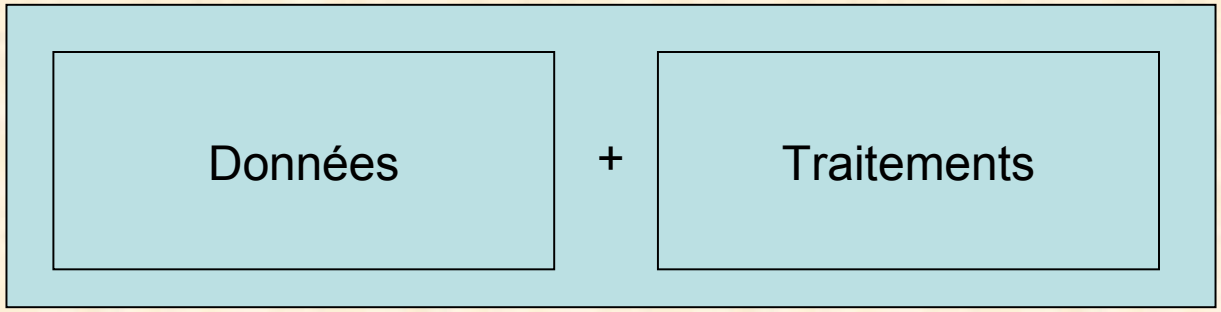
Planning Projets

- Les rendus de projet se vont par courrier électronique à votre enseignant de TD

- Projet 1
  - Sujet : lundi 3 octobre 2011
  - Rendu : vendredi 14 octobre 2011 avant 23h59
- Projet 2
  - Sujet : lundi 17 octobre 2011
  - Rendu : vendredi 28 octobre 2011 avant 23h59
- Projet 3
  - Sujet : lundi 7 novembre 2011
  - Rendu : vendredi 9 décembre 2011 avant 23h59

Evolution des approches (1/4)

- Système informatique

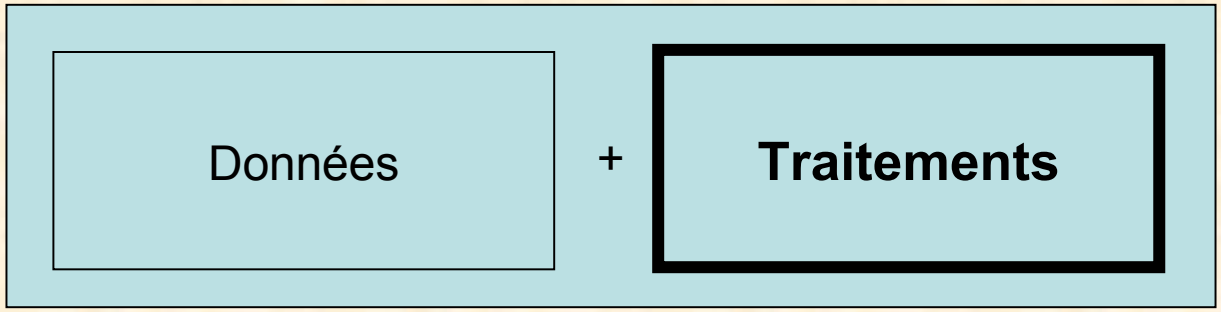


Propriétés statiques

Propriétés dynamiques

Evolution des approches (2/4)

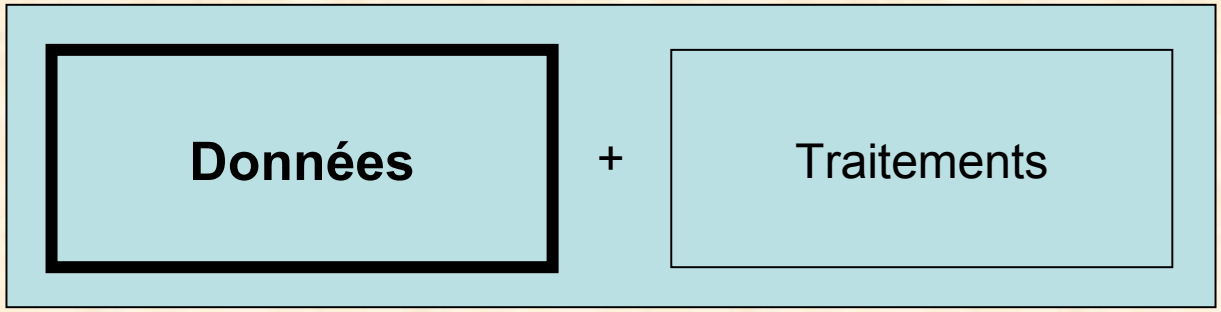
- Logiciels classiques



Les traitements prennent de plus en plus d'importance

Evolution des approches (3/4)

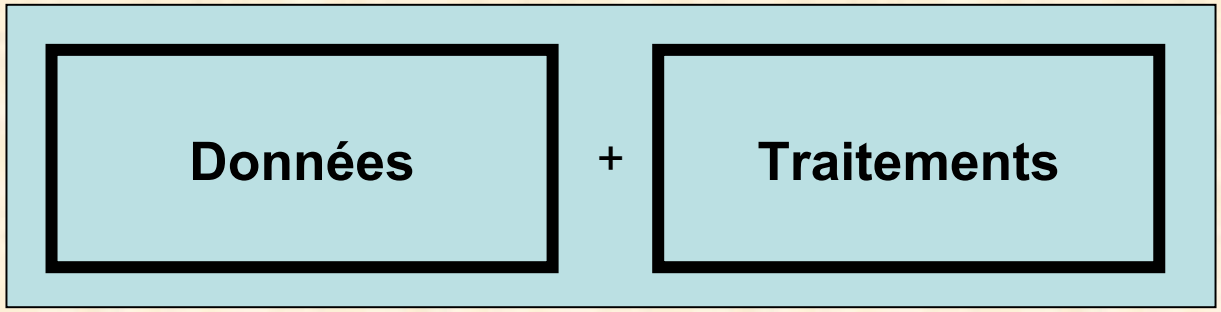
- Systèmes d'Information (BD)



Les données sont au centre du système et persistantes

Evolution des approches (4/4)

- Systèmes d'Information d'Aide à la Décision



Les données sont au centre du système : orientation sujet (entrepôt)
Les données sont persistantes
Les systèmes procèdent de plus en plus à des post-traitements : fouille de données et aide à la décision

Objectif de l'enseignement (1/3)

- Méthodologies permettant de mener à bien un projet informatique complexe et en équipe
- Découpage de problèmes difficiles en plusieurs petits problèmes plus faciles à traiter
- La vie d'un logiciel ne se limite pas à la programmation !

Objectif de l'enseignement (2/3)

- Vous faire passer de … à
  - Programmation individuelle sur de petits problèmes
  - Algorithme, langage de programmation, structures de données
  - Un peu de méthodologie : analyse descendante

« programming in the small »

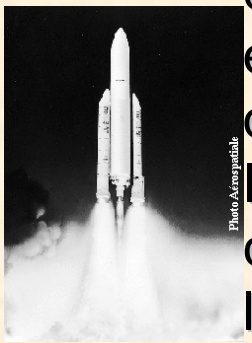
Objectif de l'enseignement (3/3)

- Vous faire passer de … à
  - Travail en équipe sur des projets longs et complexes
  - Spécifications de départ peu précises
  - Dialogue avec le client/utilisateur : parler métier et non informatique
  - Organisation, planification, gestion du risque

« programming in the large »

Bêtisier (1/6)

- Défaillances : les spectaculaires
  - La sonde Mariner qui devait effectuer un passage à 5 000 km de Vénus s'est perdue à 5 000 000 km de ladite planète à cause d'une erreur de programme Fortran : le point avait été remplacé par une virgule (format US des nombres).
  - Le 4 juin 1996, lors du premier lancement de la fusée Ariane V, celle-ci explose en vol. La cause : un logiciel utilisé sous Ariane IV et intégré sans nouvelle validation dans Ariane V. Ariane V ayant des moteurs plus puissants s'incline plus rapidement que Ariane IV, pour récupérer l'accélération due à la rotation de la Terre. Les capteurs ont bien détecté cette inclinaison d'Ariane V, mais le logiciel l'a jugée non conforme à son plan de tir, et a provoqué l'ordre d'autodestruction alors que tout se passait bien.
    Coût : 1/2 milliard de dollars



Bêtisier (2/6)

- Défaillances : les navrantes
  - Le Therac-25, un appareil d'irradiation thérapeutique a provoqué la mort de 2 personnes, et l'irradiation de 4 autres, à cause d'une erreur logicielle.
  - Un navire de guerre anglais a été coulé par un Exocet français, au cours de la guerre des Malouines. Bilan de la catastrophe : plusieurs centaines de morts. Le vaisseau anglais n'avait, en effet, pas activé ses défenses, l'Exocet n'étant pas répertorié comme missile ennemi.
  - En 1983, toute la vallée du Colorado a été inondée. La cause ? Une mauvaise modélisation dans le logiciel du temps d'ouverture du barrage.

Bêtisier (3/6)

- Défaillances : les risibles
  - Une personne s’est retrouvée saisie pour une dette impayée de 0,01F et cela bien qu’elle se soit acquittée de l’intégralité de sa dette. Le coupable ? un arrondi mal maîtrisé dans les calculs.
  - Une centenaire (106 ans) s’est retrouvée convoquée à l'école. La cause ? le codage de l’année de naissance sur deux caractères.
  - Une victime de l’an 2000 s’est vue adresser une amende de 91 500 $ au retour d'une cassette vidéo louée. La raison ? Le retard calculé étant de cent ans car là encore le codage de l’année des emprunts avait été effectué sur deux caractères, pour gagner un peu de place.

Bêtisier (4/6)

- Défaillances : les coûteuses
  - Bug du Pentium en 1994. Coût : 500 millions de dollars
  - Bug de l'an 2000. Coût de la mise à niveau des logiciels à la France : 500 milliards de francs.
  - Le 22 décembre 2001, en pleine période des achats de Noël, les 750 000 terminaux de paiement ne répondent plus chez les commerçants, entraînant de longues files d'attente de clients excédés dont beaucoup finissent par abandonner leurs chariots. La cause : la saturation des serveurs de la société Atos en charge des autorisations de paiement dépassant 600F. Les autorisations de débit qui prennent habituellement quelques dizaines de secondes, nécessitent ce jour-là quasiment la demi-heure. Le coût du préjudice pour le seul groupe Leclerc : 2 millions d'Euros.

Bêtisier (5/6)

- Défaillances : et c’est pas fini…
  - La SNCF a rencontré des difficultés importantes pour la mise en service de son système Socrate (système de réservation des places). Plantages fréquents, mauvaise ergonomie, manque de formation du personnel, ont amené un report important et durable de la clientèle vers d'autres moyens de transport. La cause : le déploiement par la SNCF d'un système de réservation de places destiné aux compagnies aériennes, sans réadaptation véritable du cahier des charges au domaine du transport ferroviaire.
  - …

Bêtisier (6/6)

- Dépassement des délais et des coûts :
  - En mars 1993, la bourse de Londres a renoncé au projet informatique Taurus qui devait assurer complètement le suivi de l'exécution des transactions. Le système avait coûté directement 60 millions de livres et les opérateurs sur le marché avait dépensé 400 millions de livres pour y adapter leurs propres logiciels.
  - Toujours en 1993, en Californie, le DMV a renoncé au projet informatique démarré 6 ans plus tôt qui devait intégrer les systèmes d'information destinés à gérer les immatriculations et les permis. Les coûts étaient de 6,5 fois ceux prévus initialement et la date de livraison probable avait été reportée à 1998.
  - Question pour un champion : Mars 1992, vous êtes nommé(e) Directeur du projet Taurus… quel est l'enjeu du projet ? Quelle est la stratégie à adopter ?

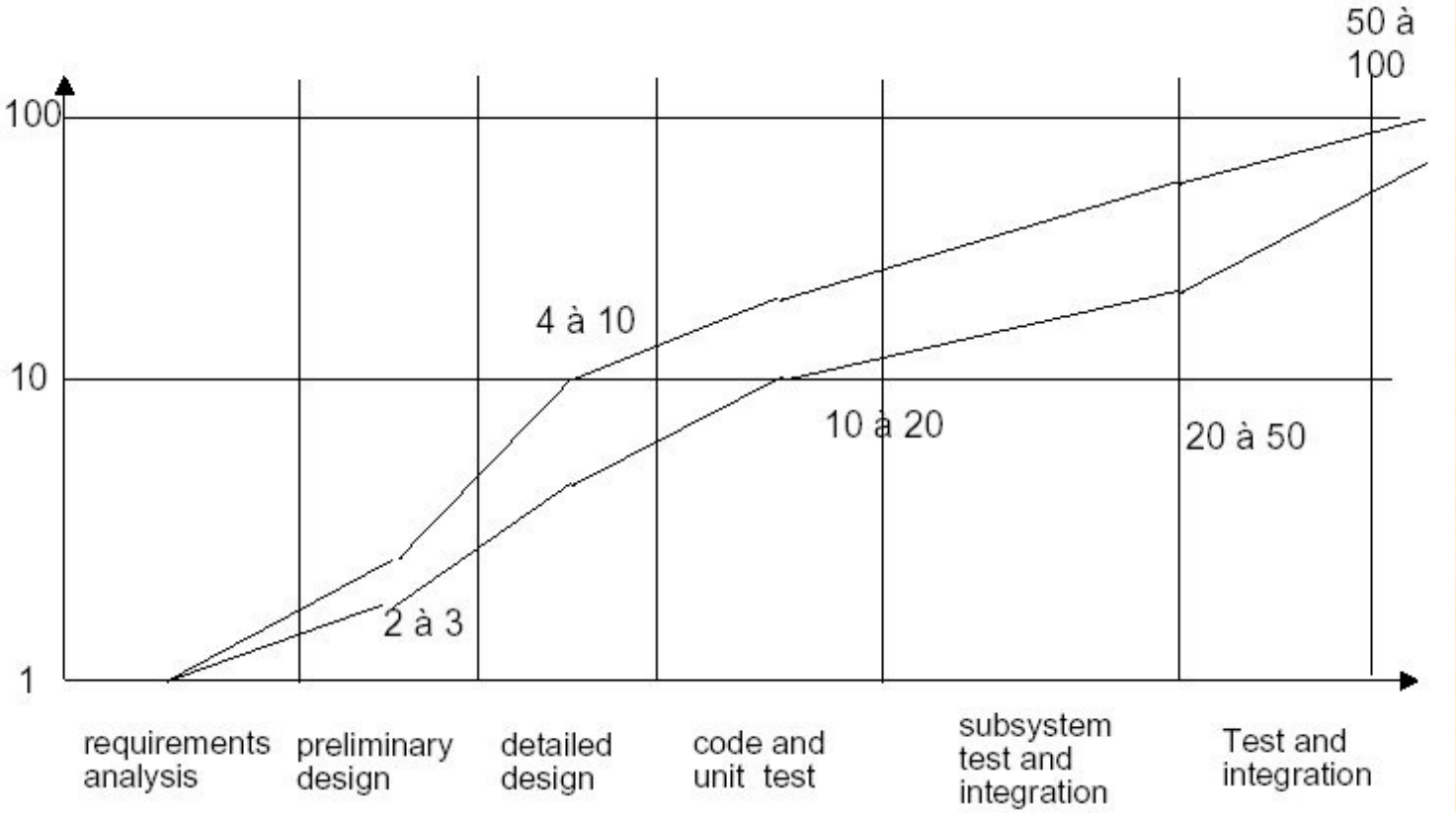
Quelques Études (1/4)

- Respect des engagements :
  - 30% des projets informatiques sont annulés avant la mise en production (Aberdeen).
  - En 1995, aux USA, on estime à 91 milliards de dollars, la somme dépensée pour des projets arrêtés avant la fin.
  - 90% des projets informatiques sortent en retard (Aberdeen).
  - 50% des projets informatiques ne répondent pas au cahier des charges Business (Gartner).
  - 50% des projets informatiques dépassent le budget prévu (Gartner). Ce surplus de coût se répartit de la manière suivante :
    - *maintenance corrective :* *20 %*
    - *maintenance adaptative :* *25 %*
    - *maintenance évolutive et perfective :* *55 %*

Quelques Études (2/4)

- Coût des modifications :

  en fonction du moment où elles interviennent dans le cycle de vie du projet (en % du coût prévu)



Quelques Études (3/4)

- Coûts de l'informatique :
  - Les dépenses en informatique sont extrêmement importantes. A titre indicatif, elles représentent aux USA, en pourcentage du PNB :
    - *1980 : 5 %*
    - *1985 : 8 %*
    - *1990 : 12,5 %*
  - Boehm estimait qu'en 1985, le coût des développements logiciels dans le monde s'élevait à 140 milliards de dollars avec une progression de 12 % par an.

Quelques Études (4/4)

- Productivité :
  - Une étude réalisée par Seckman en 1968 sur la productivité, basée sur le travail de douze programmeurs chevronnés, met en évidence les écarts suivants :
    - *heures de codage :* *de 1 à 25*
    - *heures de mise au point :* *de 1 à 26*
    - *taille du programme :* *de 1 à 5*
    - *temps d'exécution :* *de 1 à 13*
  - Après élimination des extrêmes :
    - *indice de production :* *de 1 à 5*

Les Constats (1/3)

- Tous ces constats font que les années 70 voient naître une crise de l'industrie du logiciel dont les principales raisons sont :
  - le non-respect (augmentation) des coûts
  - le non-respect (augmentation) des délais
  - l’inadéquation avec les besoins des utilisateurs
  - le non-respect des spécifications
  - la non-fiabilité
  - les difficultés d'évolution

Les Constats (2/3)

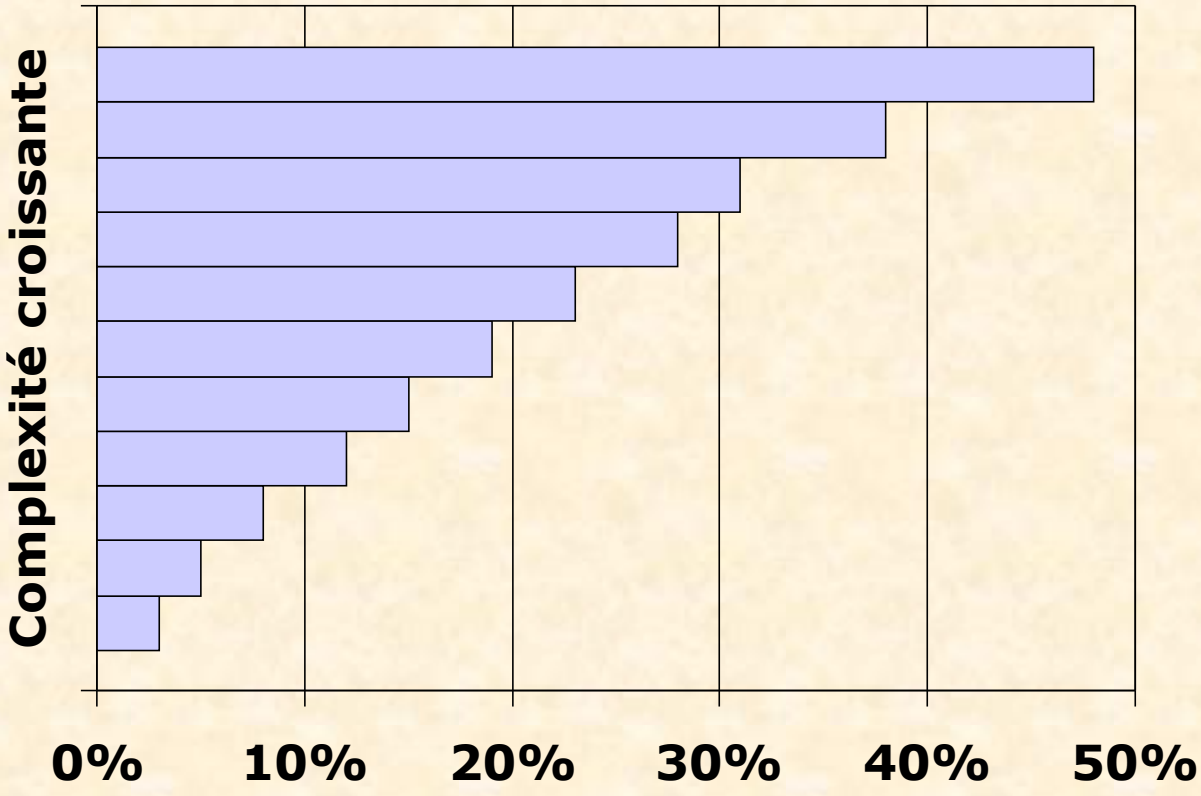
- S'il n'est pas simple d'y remédier, les causes de ces dysfonctionnements sont cependant faciles à identifier :
  - une communication difficile :
    - *jargon utilisateur versus jargon informatique*
    - *évolution de la manière de voir un problème*
    - *réflexion utilisateur pas assez mature*
    - *marché, législation et besoin en perpétuelle évolution*
    - *addition successive d'imprécision dans la communication entre les individus d'une équipe (la complexité ⇗ avec le nombre d'intervenants)*

Les Constats (3/3)

- S'il n'est pas simple d'y remédier, les causes de ces dysfonctionnements sont cependant faciles à identifier :
  - une conception et un développement difficiles :
    - *complexité croissante des logiciels*
      (la productivité ⇘ lorsque la complexité du logiciel ⇗)
    - *trop d'interrelations entre composants logiciels*
    - *trop de modifications (évolutions)*
    - *tests trop souvent insuffisants*

La crise du logiciel

- Influence de la taille du projet



Complexité croissante du logiciel

- Complexité croissante du logiciel
  - Systèmes offrant de plus en plus de fonctionnalités
    Exemple : système d'information BD + SIAD
  - Systèmes distribués : machines hétérogènes en réseau
  - Mutation technologique rapide : langages et environnements de développement, O.S.
  - Évolutions des besoins du client en cours de projet
- Solutions proposées par le génie logiciel
  - Séparer les aspects fonctionnels et technologiques

Décomposition en sous-systèmes ⇒ approche objet

  - Démarche itérative ⇒ approche objet

Qualité du logiciel

- Privilégier la qualité à l'efficacité
  - La prévention des erreurs coûte des dizaines de fois moins cher que leur correction
  - Démarche qualité : ISO 9126 (www.osil.ch/eval/node15.html)
- Qualité externe vs. Qualité interne
  - Externe : vision client
  - Interne : vision développeur

Lois de Lehman (1/2)

- A propos d'évolutions… en 1985, Lehman a émis un certain nombre d'hypothèses concernant la dynamique d'évolution des logiciels :
  - Loi des modifications perpétuelles : Un logiciel utilisé dans un environnement réel doit nécessairement évoluer sinon il devient de moins en moins utile dans cet environnement
  - Loi de la complexité croissante : Au fur et à mesure qu'un logiciel évolue, sa structure a tendance à se complexifier. Il sera nécessaire de consacrer des ressources supplémentaires si l'on veut inverser sa dégradation

Lois de Lehman (2/2)

- Lois de Lehman (suite) :
  - Loi d'évolution des gros logiciels : L'évolution d'un logiciel est un processus auto-régulé. Les attributs du système comme l'augmentation de sa taille, le temps entre 2 versions ou le nombre de bogues signalés ne varient quasiment pas d'une version à l'autre
  - Loi de la stabilité organisationnelle : Sur la durée de vie d'un logiciel, le taux de modifications est à peu près constant, quel que soit l'effort qu'on lui consacre
  - Loi de conservation de la familiarité : Sur la durée de vie d'un logiciel, les modifications incrémentales de chaque nouvelle version sont à peu près constantes

Génie Logiciel (1/5)

- Naissance du Génie Logiciel…
  - Les pères du Génie Logiciel (Software Engineering) se prénomment Friedrich Bauer et Louis Bolliet.
  - Cette spécialité a en effet vu le jour entre le 7 et le 11 octobre 1968, à Garmisch-Partenkirchen, sous le parrainage de l'OTAN. Elle avait pour objectif de répondre à 2 constations :
    1) d'une part, le logiciel n'était pas fiable,
    2) d'autre part, il était difficile à réaliser dans les délais et budgets prévus et en satisfaisant le cahier des charges.
  - En 1993, la branche informatique (Computer Society) de l'IEEE (Institute of Electrical and Electronics Engineers) et l'ACM (Organisation professionnelle américaine des informaticiens) ont établi un comité conjoint dont la tâche était de définir un ensemble de critères et de normes appropriés pour la pratique professionnelle du Génie Logiciel (SWEBOK).

Génie Logiciel (2/5)

- Définitions du Génie Logiciel :
  - D'après la *Norme IEEE 610.12* : Le Génie Logiciel est l'application d'une approche **systématique, disciplinée et quantifiable** au développement, à l'exploitation et à la maintenance du logiciel. C'est-à-dire, l'application de l'ingénierie au logiciel.
  - D'après *Classical and Object-Oriented Software Engineering with UML and Java* de Schach : Le Génie Logiciel est une discipline qui a pour but la fabrication du logiciel **sans faute**, **livré dans le délai et le budget prévus à l'avance**, et qui **satisfait aux besoins du client.**

Génie Logiciel (3/5)

- Définitions du Génie Logiciel (suite) :
  - D'après MC. Gaudel : Le Génie Logiciel est l'art de spécifier, de concevoir, de réaliser et de faire évoluer, avec des moyens et dans des délais raisonnables, **des programmes, des documentations et des procédures de qualité** en vue d'utiliser un système informatique pour résoudre certains problèmes
  - Ou encore… : Le Génie Logiciel est l'art et la science de concevoir et de construire, avec économie et élégance, **des applications, des objets, des frameworks et d'autres systèmes informatiques**, qui soient **corrects, robustes, extensibles, réutilisables, sûrs, efficaces, faciles à maintenir et à utiliser.**

Génie Logiciel (4/5)

- Pour résumer, nous dirons que :
  - La motivation première du Génie Logiciel est la **réduction des coûts de développement** du logiciel.
  - Le Génie Logiciel est un **ensemble de moyens** (techniques et méthodologiques) permettant la construction de systèmes informatiques **répondant à des critères de qualité** préalablement définis.
  - Sa mise en œuvre implique la prise en compte :
    - *des environnements de développement, avec toute la variété d'outils et d'approches dont on peut disposer,*
    - *des méthodes et des techniques de gestion de processus,*
    - *des aspects humains au sein de l'équipe de développement*
    - *des relations que celle-ci entretient avec les commanditaires et les utilisateurs du produit*

Génie Logiciel (5/5)

- Une définition (une de plus)
  - Méthodologie de construction en **équipe** d'un logiciel **complexe** et à **multiples versions**
- Programmation vs. Génie Logiciel (approximation)
  - Programmation : activité personnelle
  - Génie Logiciel : activité d'équipe
- Côté finance
  - Coût du logiciel supérieur à celui du matériel
  - Coût de maintenance supérieur au coût de conception

Génie Logiciel
résumé

- Le GL doit fournir des méthodes de conception de systèmes complexes permettant
  - Une prise en compte du client
  - Une démarche qualité
  - Une organisation du travail en équipe
- Le GL, c'est aussi des outils associés
  - AGL (atelier de génie logiciel)
  - AGL méthodologiques
    Exemple : Rational Rose (UML)

⇒ améliorer la qualité du logiciel !

Terminologie (1/5)

- Projet :
  - Ensemble d'activités organisées permettant de créer un produit ou un service *unique avec une qualité définie dans le cadre d'un budget fixé*
  - Il s'agit d'un effort temporaire caractérisé notamment par un début et une fin déterminés
  - Par qualité définie, on entend un choix de facteurs de qualité déterminés

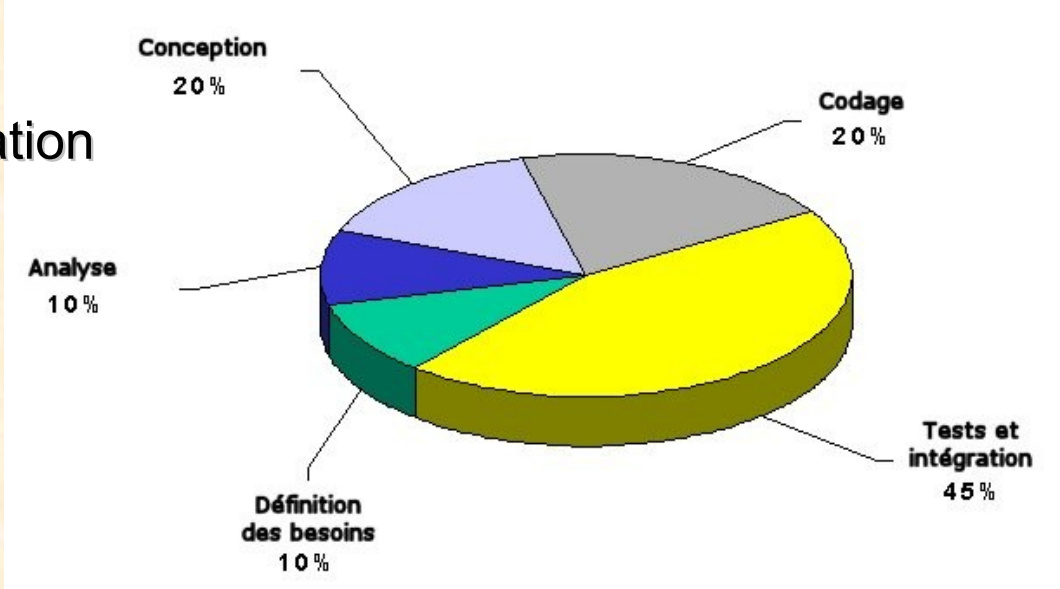
Terminologie (2/5)

- Processus :
  - Un processus est un savoir-faire associé à une discipline (par exemple : le développement de logiciel)
  - Un processus est un ensemble structuré :
    - *d'acteurs (concepteur, chef de projet…)*
    - *d'activités pour chaque acteur (coder, planifier, documenter…)*
    - *d'artefacts pour chaque activité (exécutable, planning…)*

      un artefact est le résultat concret d'une activité
    - *de workflows (un workflow = un ensemble d'activités)*
  - Un processus peut lui-même englober des sous-processus

Terminologie (3/5)

- Phases d'un processus de développement :
  - Définition des besoins ou des requis du système (cahier des charges)
  - Analyse
  - Conception préliminaire
  - Conception détaillée
  - Codage ou implémentation et tests unitaires
  - Tests d'intégration et intégration



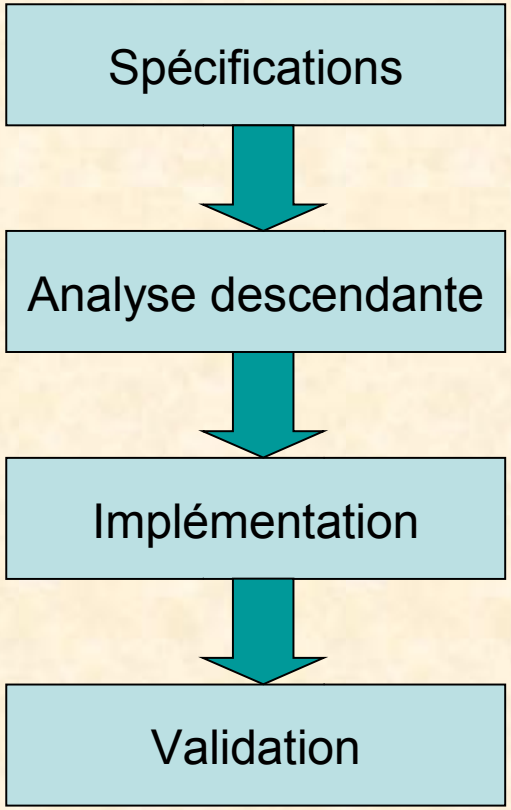
Terminologie (4/5)

- Cycle de vie du logiciel :
  - Le concept de processus logiciel peut être assimilé à celui du cycle de vie du logiciel. Ce second terme introduit une vision temporelle du processus
  - Ensemble séquentiel de phases, dont le nom et le nombre sont déterminés en fonction des besoins du projet, permettant généralement le développement d'un logiciel
- Les différents types de cycle de vie :
  - Cycle en cascade (linéaire)
  - Cycle en V (linéaire)
  - Cycle en spirale (itératif ou incrémental : prototypes)
  - Cycle en Y

Terminologie (5/5)

- Amélioration du processus :
  - La compétition internationale force les organisations à livrer des produits et des **services de qualité** sans cesse croissants, dans des **délais toujours plus courts** et à des prix qui **correspondent aux attentes du marché**
  - Les processus permettent de définir des façons de faire et d'améliorer constamment ces définitions pour qu'elles agissent comme levier pour **accroître la satisfaction** de la clientèle et la compétitivité d'une entreprise
  - **Plus un processus est mature et maîtrisé, plus ces objectifs peuvent être atteints**

Phase des processus (1/2)



données, complètes et précises

décomposition fonctionnelle

un seul langage de programmation

jeux d'essais

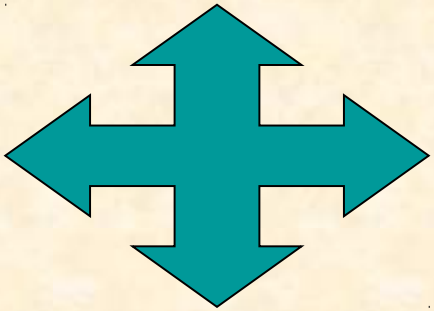
« programming in the small »

Phase des processus (2/2)

Analyse des besoins

Spécifications

Analyse du système



Conception

Implémentation

Validation

Maintenance

« programming in the large »

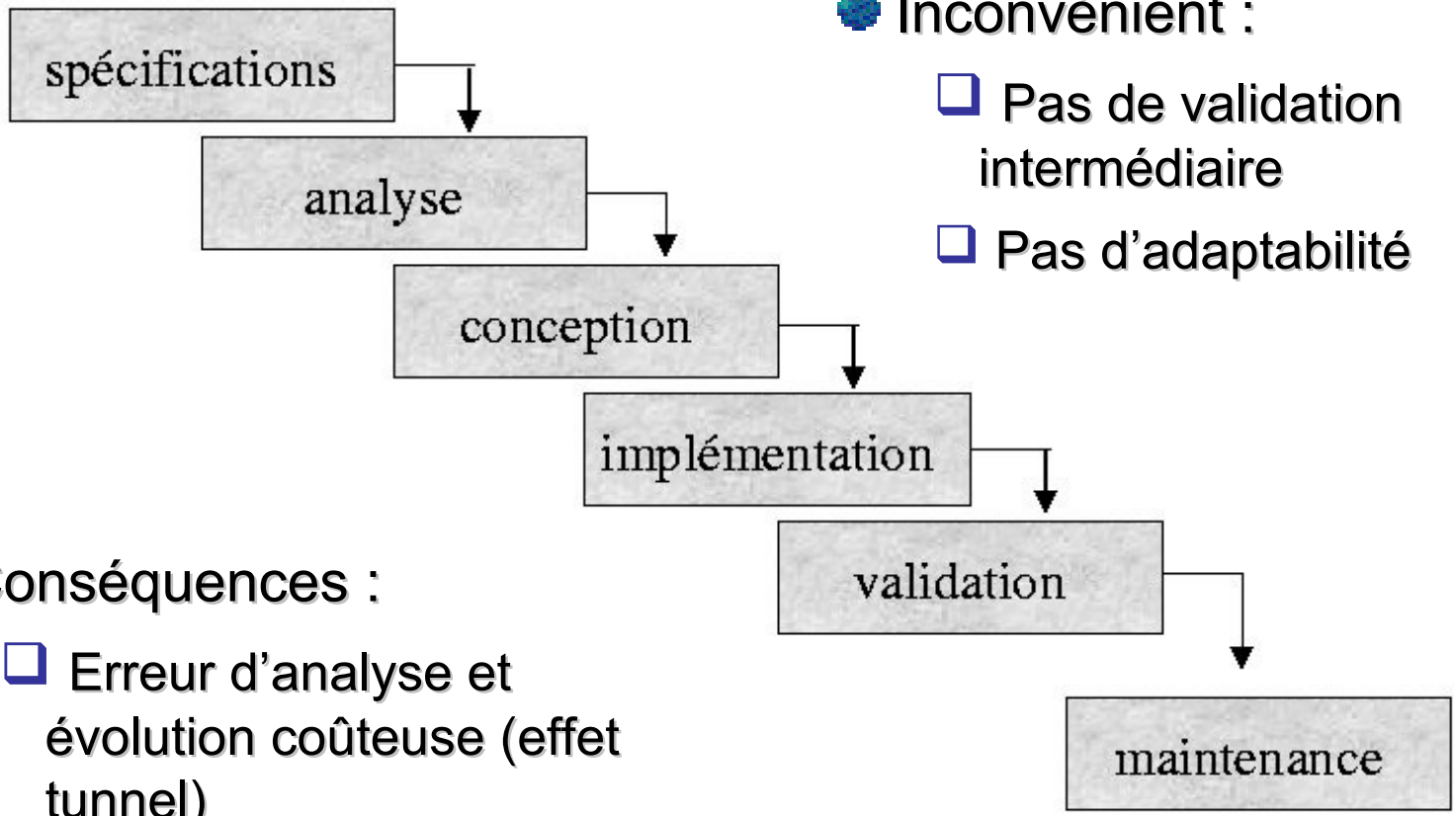
Exemples de coûts

1.

	1	2
Analyse des besoins et spécifications	15 %	6 %
Analyse / conception	25 %	5 %
Implémentation	30 %	7 %
Tests et validation	15 %	15 %
Maintenance	15 %	67 %

Cycle de Vie (1/5)

- Cycle en cascade



- Inconvénient :
  - Pas de validation intermédiaire
  - Pas d'adaptabilité

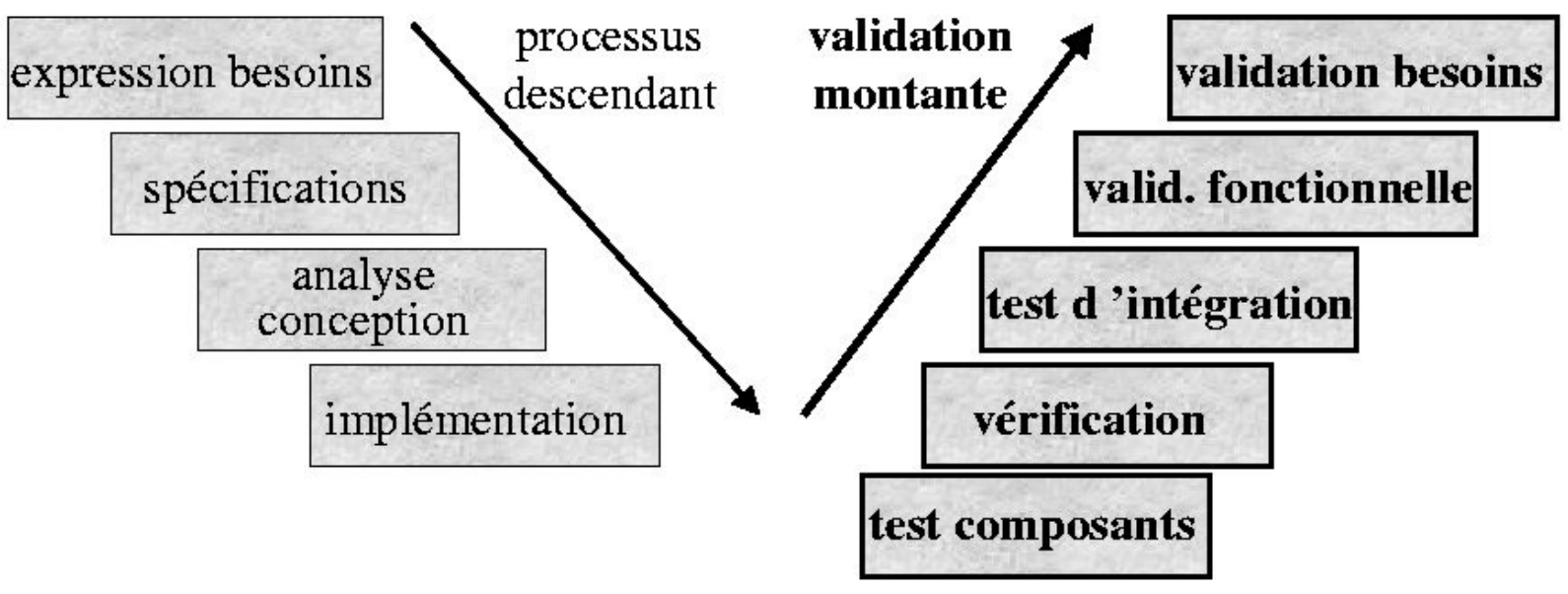
- Conséquences :
  - Erreur d'analyse et évolution coûteuse (effet tunnel)

Cascade : inconvénients

- Aucune validation intermédiaire
  - Impossibilité de suivre le déroulement du projet, donc de planifier un travail en équipe
  - Augmentation des risques car la validation est tardive : erreur d'analyse ou de conception très coûteuse
- Solution limitée aux petits problèmes
  - Risques bien délimités dès le début du projet
  - Projet court avec peu de participants

Cycle de Vie (2/5)

- Cycle en V :



- Avantage (/cascade) :
  - Meilleure validation intermédiaire
- Inconvénient :
  - Toujours peu d'adaptabilité

V : avantages

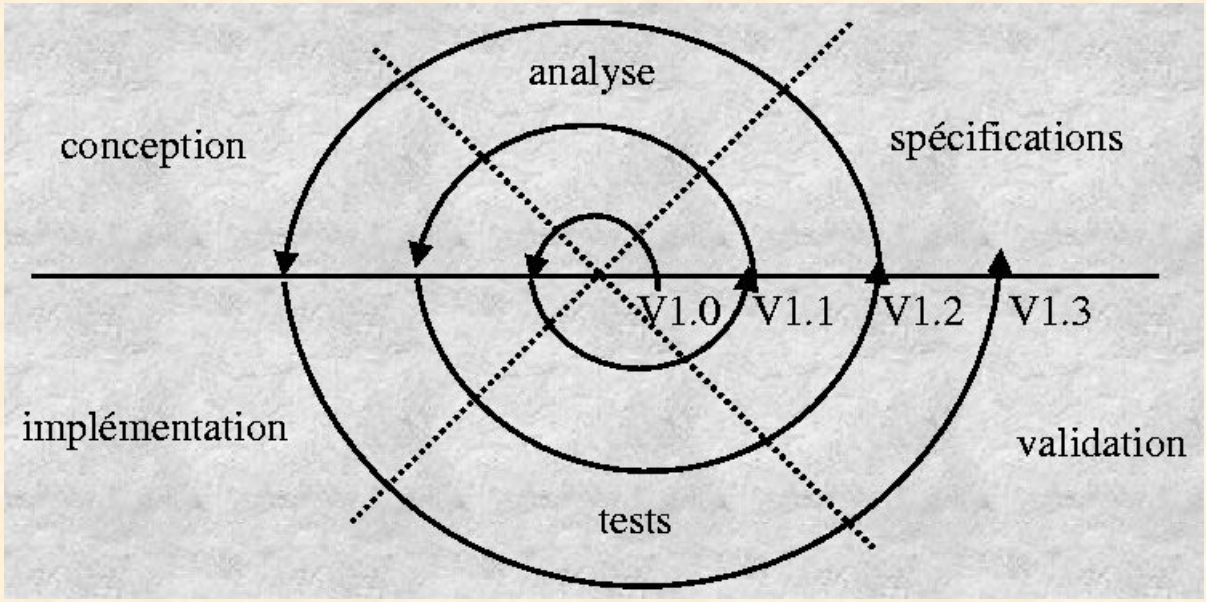
- Validations intermédiaires
  - Bon suivi de projet : points de mesure concrets de l'avancement du travail avec étapes clés
  - Favorise la décomposition fonctionnelle de l'activité
  - Limitations des risques en cascade par validation de chaque étape
  - **Existence d'outils supports !**
- Modèle éprouvé très utilisé pour de grands projets

V : limitations

- Un modèle toujours séquentiel
  - Prédominance de la documentation sur l'intégration : validation tardive du système global
  - Les validations intermédiaires n'empêchent pas la transmission des insuffisances
  - Peu d'adaptabilité
  - Maintenance non intégrée : logiciel jetable
- Adapté aux problèmes bien connu
  - Idéal quand les besoins sont bien connus, quand l'analyse et la conception sont claires

Cycle de Vie (3/5)

- Cycle en spirale :



- Avantages
  - Rectification au plus tôt des erreurs détectées lors des phases précédentes
  - Adaptabilité via une meilleure gestion des exigences et la prise en compte des évolutions

Spirale

- Prototypage
  - **Idée :** fournir le plus rapidement possible un **prototype** exécutable permettant une **validation concrète** et non sur document
  - Progression du projet par incrémentations successives de versions du prototype : **itérations**
  - Certains prototypes peuvent être montrés au client. Par ailleurs, une **maquette** peut être réalisable préalablement au premier prototype
  - La validation par prototype ne justifie pas l'absence de recours à la **documentation** !

Spirale : avantages

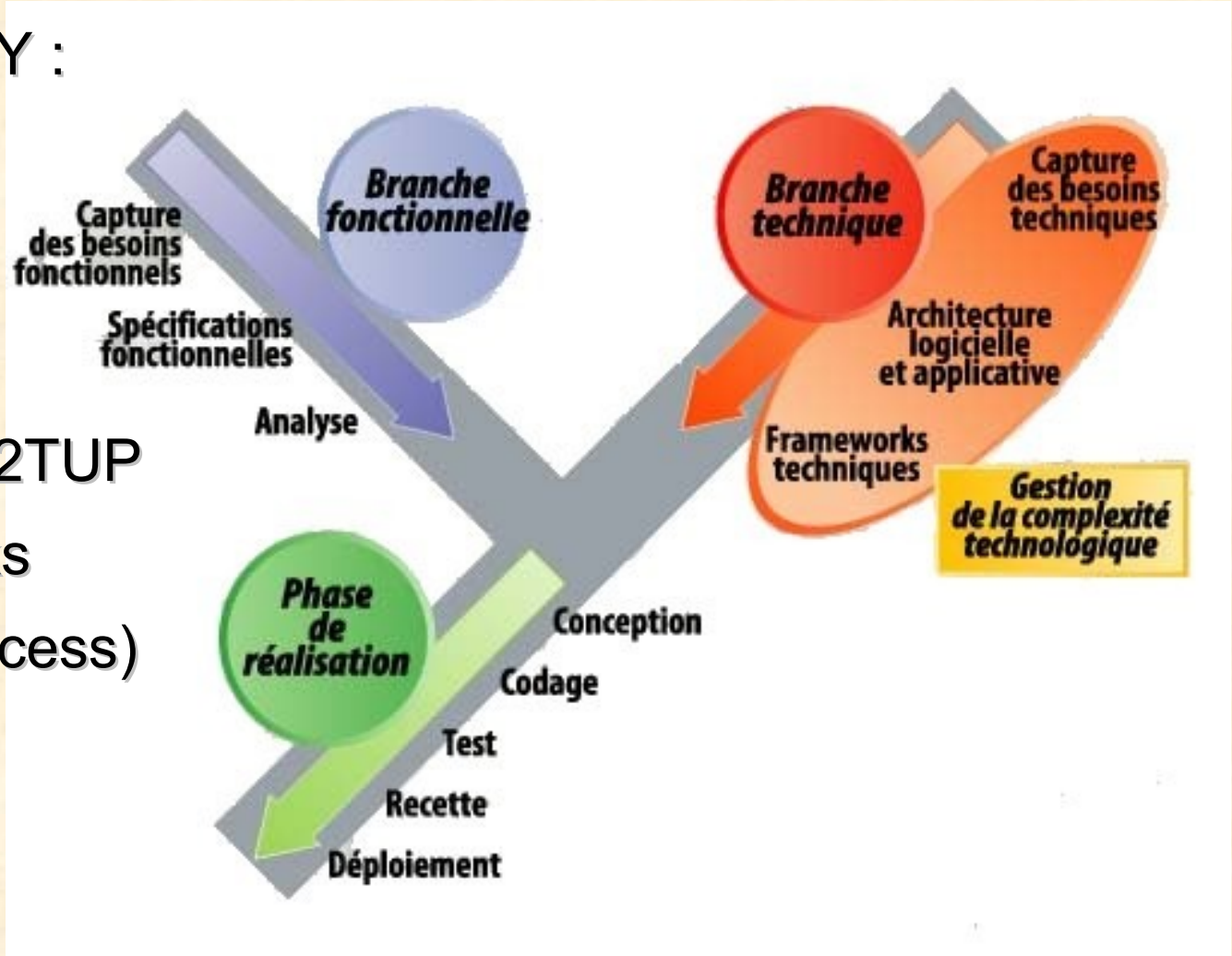
- Validation concrète et non sur documents
- Limitation du risque à chaque itération
- **Client partenaire** : retour rapide sur ses attentes
- Progressions : pas d'explosion des besoins à l'approche de la livraison : pas de *« n'importe quoi pourvu que ça marche »*
- Flexibilité :
  - Modification des spécifications = nouvelle itération
  - Maintenance = forme d'itération
- Planification renforcée

Spirale : limitations

- Pas de processus idéal
  - Cycle itératif : planification très attentive et rigoureuse
  - Cycle en « V » : processus éprouvé le plus répandu surtout pour les systèmes connus
    Cycle itératif : peut dérouter au premier abord
- Processus adapté à la modélisation objet
  - Modèle objet : se prête parfaitement à une démarche incrémentale
  - Le cycle en spirale a cependant une portée générale

Cycle de Vie (4/5)

- Cycle en Y :



- Exemple : 2TUP (Two Tracks Unified Process)

Cycle de Vie (5/5)

- Notions introduites :
  - Itération
    - *L'application est divisée en plusieurs applications plus petites (lotissement), divisant de ce fait le traitement de la complexité de l'application globale*
    - *Le développement itératif permet la validation régulière par les utilisateurs.*
    - *Le contenu d'une itération n+1 comporte non seulement de nouvelles fonctionnalités mais aussi la prise en compte des améliorations de l'itération n*
    - *Il est ainsi possible de rectifier au plus tôt les erreurs et de prendre en compte l'évolution des besoins*
  - Prototypage
    - *Livraison d'un exécutable permettant une validation concrète*

Méthodes (1/2)

- Pour gérer un cycle de vie, il est nécessaire d'utiliser :
  - Des méthodes
  - Des outils
- La méthode : règles et pratiques mis en œuvre par le chef de projet pour conduire son projet
- Longtemps dominant, Merise, avec son approche systémique (par la structure) et ses validations en cascade, est aujourd'hui critiquée pour sa rigidité, son manque d'adaptation et son effet tunnel
- On a vu alors apparaître une tendance à rapprocher les utilisateurs des développeurs, avec des cycles d'itérations plus courts…

Extrait du magazine L'informaticien n° 009

Méthodes (2/2)

- … C'est l'émergence des méthodes agiles :
  - **RAD** (Rapid Application Development)
  - **UP** (Unified Process), **RUP** (Rational Unified Process), **2TUP** (Two Tracks Unified Process)
  - **ASD** (Adaptative Software Development)
  - **DSDM** (Dynamic Software Development Method)
  - **SCRUM** (mêlée au rugby)

  sont des approches par la structure et par les besoins
- Des approches plus radicales telles que :
  - **XP** (eXtreme Programming)
  - **Crystal Clear**

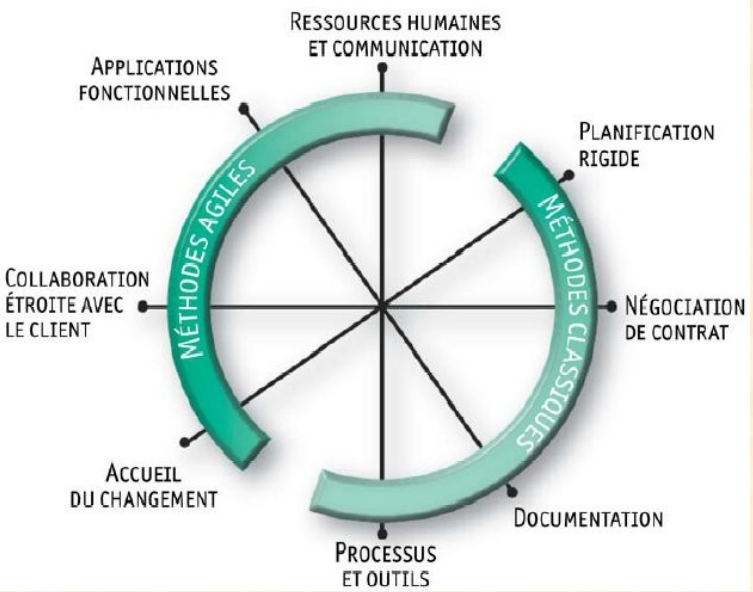
  sont pilotées par les besoins et se veulent totalement incrémentielles et itératives

METHODES AGILES (1/4)

- Objectifs des méthodes agiles :
  - Augmenter le niveau de satisfaction client
  - Faciliter le travail de développement
  - Performance
  - Qualité
- Méthodes adaptatives versus prédictives
- Les pratiques des méthodes agiles :
  - Souvent anciennes, admises et testées. ex. "bon sens"
  - Mais en général oubliées : "pas de mise en pratique"
  - Un exemple significatif : les pratiques relatives aux tests
- En savoir plus sur les méthodes agiles :
  - *Magazines Développeur Référence v2.05 à v2.12 (www.devreference.net)*

METHODES AGILES (2/4)

- En février 2001, les instigateurs des principales méthodes agiles forment l'Agile Alliance. Leur réflexion aboutit au "Manifesto for Agile Software development" qui définit 4 valeurs… :
  - Priorité aux personnes et aux interactions sur les procédures et les outils
  - Priorité aux applications fonctionnelles sur une documentation pléthorique
  - Priorité de la collaboration avec le client sur la négociation de contrat
  - Priorité de l'acceptation du changement sur la planification



METHODES AGILES (3/4)

- … et 12 principes :
  - Livraison ASAP de versions fonctionnelles (⇒ feedback)
  - Le changement comme avantage concurrentiel
  - Livraisons intermédiaires aussi souvent que possible
  - Coopération quotidienne entre utilisateurs et développeurs
  - Construction d'une équipe motivée
  - Favoriser l'échange oral sur l'écrit
  - 1er indicateur d'avancement du projet : le fonct. de l'application
  - Rythme soutenable pour les utilisateurs et les développeurs
  - Attention continue à l'excellence technique et à la conception
  - Toujours favoriser la simplicité
  - Responsabilité confiée à l'équipe entière et volontariat
  - Auto-ajustement de l'équipe pour améliorer son efficacité

METHODES AGILES (4/4)

- La réalité :
  - Pas de grande rupture : les méthodes traditionnelles ne sont pas et ne doivent pas être abandonnées
  - Elles doivent être peu à peu transformées pour prendre en compte le signal fort envoyé par le taux d'échec important des projets informatiques
  - **Toutes pratiques agiles ne sont pas forcément adaptables à tout type de projet**
  - Lors du démarrage d'un projet, il s'agit donc de **définir quelles sont les pratiques que l'on souhaite appliquer en ayant à l'esprit leur plus-value pour le projet considéré**

RAD (1/7)

- RAD : Rapid Application Development
- Objectifs du RAD :
  - Rapidité de l'analyse et du développement
  - Obtenir un résultat opérationnel même limité
  - Respecter une enveloppe budgétaire
- Pour y parvenir :
  - Augmenter la productivité de l'équipe

RAD (2/7)

- RAD – Principe n°1 : Utilisation d'outils performants d'aide au développement
  - AGL de conception
  - AGL de réalisation
  - L4G, langages de haut niveau, générateurs de code
  - Outils de tests
  - Poste individuel de travail
  - Réseau et travail de groupe

RAD (3/7)

- RAD – Principe n°2 : Forte implication des utilisateurs
  - Les utilisateurs ont la responsabilité de la production de certaines tâches
    Exemple : expression des besoins
  - Comportement "positif" des utilisateurs participatifs, disponibles, engagés
  - Utilisateurs centrés sur :
    - *les besoins strictement nécessaires*
    - *la simplicité des règles de gestion*
    - *l'approche "délai"*

RAD (4/7)

- RAD – Principe n°3 : Equipe informatique de gagneurs
  - L'équipe doit être :
    - réduite
    - compétente
    - soudée
    - motivée
    - outillée

RAD (5/7)

- RAD – Principe n°4 : Cycle de vie raccourci
  - Approche de prototypage
  - Parallélisation
    - de la conception
    - de la réalisation
  - Livraison par versions successives

RAD (6/7)

- RAD – Principe n°5 : Pilotage centré sur l'obtention de résultats rapides
  - Arbitrage systématique délais / fonctions à développer
  - Cycle rapide de décisions
    - décideurs impliqués
    - décideurs sachant comprendre les impacts des décisions à court et moyen terme
  - Suivi de projets efficace
    - outils
    - organisation

RAD (7/7)

- Les difficultés du RAD :
  - Pas aisément reproductible
  - Approche plus que méthode
  - Nécessité d’une équipe de "gagneurs"
  - Fort outillage nécessaire
  - Difficulté d’application à des projets d’envergure
  - Peu adapté au développement "au forfait" ?
  - Réservée à une équipe de séniors

UP (1/2)

- Processus unifié :
  - Développé à l'origine par Philippe Kruchten et Ivar Jacobson sous la coupe de la société Rational
  - Le processus unifié est un ensemble structuré de "bonnes pratique" issues de l'expérience
  - Le terme Unifié fait d'ailleurs référence à cet aspect fusion entre les pratiques issues de méthodes antérieures.

Extrait du magazine Programmez n° 57

UP (2/2)

- Principes d'UP :
  1. Proximité avec les utilisateurs et pilotage par les cas d'utilisation
  2. **Pratique de la modélisation graphique des exigences**
  3. Centré sur l'architecture
  4. Fondé sur la production et l'assemblage de composants
  5. Développement itératif et incrémental du logiciel (chaque fin d'itération doit générer un prototype exécutable)
  6. Gestion des besoins et des exigences (**traçabilité**)
  7. Souci permanent de la qualité (recettes fréquentes de versions intermédiaires, automatisation des tests, revus par les pairs)
  8. Gestion des risques permanente
  9. Gestion des demandes de changement

ASD

- Adaptative Software Development, une méthodologie très générique :
  1. Focaliser sur un seul type de mission
  2. Se baser sur des composants
  3. Itérer
  4. Planifier et fixer des dates butoirs
  5. **Gérer le risque**
  6. Tolérer le changement
  7. **Capitaliser les retours d'expérience**

DSDM

- Dynamics Systems Development Method, une méthodologie "canevas" :
  1. Implication active des utilisateurs
  2. Les équipes DSDM sont autorisées à prendre des décisions
  3. Le produit est rendu tangible aussi souvent que possible
  4. L'adéquation aux besoins métiers est le critère essentiel pour l'acceptation des livrables
  5. Développement itératif et incrémental
  6. **Toute modification pendant la réalisation est réversible**
  7. Les besoins sont définis à un niveau de synthèse
  8. **Les tests sont intégrés pendant tout le cycle de vie**
  9. Esprit de coopération entre tous les acteurs

SCRUM

- SCRUM, 3 phases itérables : initiation, sprint, clôture
  1. La phase d'Initiation permet notamment de définir les backlogs : listes de tâches à réaliser, classées par ordre de priorité
  2. La phase de Sprint (30 jours environ), guidée par les backlogs intègre le développement à proprement parler. Il s'agit d'une période durant laquelle l'équipe est **isolée de toute influence extérieure**, (aucun travail supplémentaire, ni aucune modification du backlog ne peuvent être ajoutés à un sprint en cours).
  3. Le Scrum : chaque jour et pendant 30 min, toute l'équipe participe à une **réunion pour partager les connaissances acquises**, faire un point sur l'avancement, fournir au management une visibilité sur la progression du projet.
  4. Contrôle du processus omniprésent via l'évaluation de quatre variables : coût, planning, fonctionnalités et qualité.

XP

- Principes d'eXtreme Programming :
  - un représentant du client est intégré à plein temps dans l'équipe de développement afin d'atteindre une réactivité optimale, aussi bien pour la définition du besoin que pour la validation des livraisons,
  - la programmation se fait en binôme ce qui permet l'appropriation de l'ensemble du code par l'équipe de développement et le partage de l'expertise,
  - l'écriture des tests unitaires se fait avant l'étape de codage, ce qui améliore leur efficacité et donc la qualité du code,
  - l'intégration est continue dès le début du projet, ce qui supprime les risques associés à cette discipline,
  - des livraisons fréquentes permettent d'accélérer la convergence du produit par rapport aux besoins des utilisateurs.

Facteur de qualité

- Qualité externe
  - **Complétude fonctionnelle** : réalise les tâches attendues (ISO)
  - **Ergonomie** : facilité d'utilisation (ISO : usability)
  - **Fiabilité** : fonctionne même dans des cas atypiques (ISO)
  - **Adaptabilité** : adaptation aux modifs (ISO : maintainability)
- Qualité interne
  - **Ré-utilisabilité** : de plus en plus importante aujourd'hui
  - **Traçabilité**
  - **Efficacité** : bonne utilisation des ressources matérielles (ISO)
  - **Portabilité** : adaptation à de nouveaux environnements (ISO)

PHASE DE LANCEMENT (1/2)

- Compléter et valider les informations manquantes pour le PAQ :
  - identifier les différents acteurs et leur niveau d'intervention
  - clarifier les enjeux et les attentes du client
- Exposer et anticiper les risques :
  - sensibiliser les différents acteurs aux risques potentiels
  - énoncer les conditions de succès
  - mobiliser les utilisateurs sur le processus
  - valider l'organisation
- 3 règles valables pour toute réunion :
  - Toujours diffuser l'ordre du jour au moins 24 h à l'avance
  - Toujours diffuser le compte-rendu 24 h maxi. après une réunion
  - Faire relire le compte-rendu par un pair avant diffusion

PHASE DE LANCEMENT (2/2)

- Ordre du jour de la réunion de lancement :
  1) **Acteurs :** tour de table, présentation, coordonnées des différents acteurs de la MOA et de la MOE
  2) **Enjeux :** enjeux généraux, attentes client, indicateurs de qualité/validation à mettre en place
  3) **Pilotage :** comité de pilotage, suivi de projet, protocole de communication, reporting, gestion des risques
  4) **Production :** planning, livrables, cadrage du périmètre
  5) **Budget :** échéancier, gestion des évolutions
  6) **Actions :** plans d'actions, disponibilité des différents intervenants (vacances…), dates des prochaines réunions
  7) **Divers :** tour de table des questions et remarques des différents intervenants

ACTEURS

- Antoine MARTIN
  - Chef de projet de la société M1I
  - Expert fonctionnel
  - Interlocuteur privilégié
  - 04 91 ?? ?? ??
  - 06 ?? ?? ?? ??
  - antoine.martin@société.fr

ENJEUX

- Enjeux :
  - Refonte du Système d'informations
  - Réduire les coûts de …
  - Fidéliser les clients de la société …
  - Homogénéiser les processus de …
- Attentes :
  - Facilité d'utilisation / ergonomie
  - Fiabilité de l'application
  - Performance de l'application …
- Indicateurs de qualité / validation :
  - Respect des délais, du budget
  - Nombre de correctifs …

PILOTAGE (1/2)

- Comité de pilotage, suivi de projet :
  - Périodicité : …
  - Participants : …
  - Sujets traités : …
  - Rédacteur de l’ordre du jour et du compte-rendu : …
- Protocole de communication :
  - Questions et réponses : Courrier / Fax / Mail
  - Procès verbal de recette : Courrier / Fax / Mail
  - Livraison : Courrier / Fax / Mail / CD

PILOTAGE (2/2)

- Reporting :
  - Suivi et avancement de l'activité,
  - Suivi des livraisons, suivi financier,
  - Suivi des anomalies et des évolutions …
- Gestion des risques :
  - Description du processus
  - Comment est fait le reporting (tableau de bord)
  - Dans quel cadre est traité le suivi (comité de pilotage)…

PRODUCTION

- Planning :
  - Planning prévisionnel sous MS-Project par exemple
  - Tableaux récapitulatifs des dates clés
- Livrables :
  - Liste des livrables
  - Pour chaque livrable :
    - *Responsable et date butoir de la livraison*
- Cadrage du périmètre :
  - Ce que l'on fait
  - Ce que l'on ne fait pas
  - Lotissement

BUDGET

- Échéancier de paiement :
  - Descriptif des prestations
  - Pour chaque prestation :
    - *Date de facturation, date de règlement*
  - Exemple : Recette du lot 1, 29/09/2006, 29/12/2006
- Protocole de gestion des évolutions :
  - Émission d'une demande d'évolution ou d'une déclaration d'anomalie
  - Qualification d'une évolution (si déclarée anomalie)
  - Émission d'un devis
  - Validation d'un devis
  - Intégration d'une évolution au planning

ACTIONS

- Plans d'actions :
  - Établir un tableau des actions (tâche, responsable, date butoir, état)
  - Ce tableau devra intégrer toutes les tâches de démarrage du projet (rédaction du PAQ, mise à jour du planning, réunions de travail/pré-cadrage, fournitures de documents ou de livrables, installation de postes de travail, formation…)
- Disponibilité des intervenants
  - Établir un calendrier sur toute la durée du projet avec les dates d'indisponibilité (vacances, formation, contraintes…) des différents interlocuteurs présents
- Date des prochaines réunions
  - Établir un tableau des prochaines réunions sur 2 mois minimum avec le type de la réunion, la date, l'heure et le lieu

DIVERS

- Divers :
  - Cette rubrique a pour objectif essentiel de permettre aux différents intervenants de s’exprimer / écouter
  - Elle peut permettre notamment de relever des inquiétudes de la part du client / le rassurer
  - Elle peut permettre également de lever des risques ou problèmes non identifiés / établir un plan d’actions pour palier le risque ou résoudre le problème

CONCLUSION (1/4)

- 7 idées principales à retenir en terme de gestion de projet :
  - Toujours garder à l'esprit les enjeux du projet
  - Favoriser un comportement positif et participatif :
    - *Sensibiliser, mobiliser, impliquer les utilisateurs*
    - *Motiver, faire participer, faire "grandir" l'équipe projet*
  - Piloter par les risques et les anticiper
  - Évaluer en permanence et de manière rigoureuse le processus :
    - *en terme de coût (suivi financier)*
    - *en terme de délai (suivi de l'avancement)*
    - *en terme d'adéquation (recette utilisateur)*
    - *en terme de qualité via des indicateurs quantifiables*
  - Tracer toutes les décisions et gérer les évolutions
  - Écouter… et agir
  - Utilisation d'outils

CONCLUSION (2/4)

- 6 idées principales à retenir en terme de conception :
  - Utilisation d'outils
  - Conception selon 2 axes : fonctionnel et technique
  - Points de synchronisation entre ces 2 axes
  - Favoriser une architecture de composants
  - Modélisation visuelle (UML)
  - Favoriser les solutions simples

CONCLUSION (3/4)

- 4 idées principales à retenir en terme de réalisation :
  - Mettre en œuvre les best-practices de programmation :
    - *Favoriser les solutions simples*
    - *Mettre en œuvre les principes de développement objet*
    - *Mettre en œuvre les patterns de conception*
    - *Mettre en vigueur les normes de développement*
  - Capitaliser et mutualiser l'expérience :
    - *Travailler en binôme*
    - *Procéder à des réunions de capitalisation (SCRUM)*
  - Documenter suffisamment et régulièrement :
    - *Documenter à mesure*
    - *Documenter les changements et les ajouts*
  - Utiliser les outils appropriés :
    - *Outils de développement, frameworks…*
    - *Outils de gestion de configuration*

CONCLUSION (4/4)

- 2 idées principales à retenir en terme de recette :
  - Favoriser le feedback et éviter les effets tunnel :
    - *Développer de manière itérative*
      Définir au préalable le nombre d'itération
    - *Intégrer une phase de maquettage fonctionnel*
    - *Intégrer une phase de prototypage technique si nécessaire (NTI)*
    - *Effectuer des livraisons intermédiaires*
  - Mettre en place une stratégie de tests structurée et outillée
    - *Conception des scénarios de tests lors de la phase de conception*
    - *Intégration des tests tout au long du cycle de vie*
    - *Les tests d'intégration d'un composant ne doivent pas être effectués par le réalisateur de ce composant*
    - *Utilisation d'outils*
    - *Intégration d'une revue de code systématique*
    - *Intégration d'une revue par un pair systématique*

Analyse des besoins

- Étape préalable si le client n'a qu'une idée peu précise du système à réaliser
  - Étude informelle des fonctionnalités (externes) du système sans considération technique : point de vue métier / utilisateur
  - Dialogue fournisseur / client en termes intelligibles pour ce dernier : l'aider à formaliser le problème à résoudre
  - Produire un document textuel avec schémas…
- Conduit généralement à la définition du cahier des charges

Spécifications

- **Ce que doit faire le système** (côté client)
  - Document précis spécifiant les fonctionnalités attendues
  - Base du contrat commercial avec le client
  - Document facile à comprendre par le client / utilisateur
  - Exemple : *définition de la frontière du système, description des fonctionnalités du système avec scénarii, interactions : enchaînements d'écran / cas d'utilisation de Jacobson*
- **Les spécifications ne sont jamais complètes ni définitives**
  - Évolution du domaine, besoins supplémentaires…

Analyse du système

- **« Quoi faire » : comprendre et modéliser le métier**
  - Réflexion métier hors de toute considération technique
  - Reste un support de discussion avec le client / utilisateur
  - Premier modèle du système (niveau métier) Identifier les éléments intervenants (hors acteurs) et dans le système : fonctionnalités, structures et relations, états par lesquels ils passent suivant certains événements (diagramme de classes, de collaboration)
- **L'analyse n'est jamais complète mais elle doit être juste**

Conception

- Comment faire le système : choix techniques
  - Choix d'une architecture technique (matériel, logiciel) suivant certaines priorités (facteurs qualités : robustesse, efficacité, portabilité…)
  - Expertise informatique : hors compréhension du client
  - **Modèle de l'architecture logicielle du système** Décomposition en sous-systèmes : application (interfaces), domaine (métier) et infrastructure (implémentation)
- Permet la définition des phases d'implémentation, de validation et de maintenance

Implémentation

- Souvent trop de temps consacré au codage au détriment des phases d'analyse et de conception : mauvaise pratique parfois très coûteuse…
- Savoir user de la réutilisation de composants, voire d'outils de génération de code (mise en place automatique du squelette du code à partir du modèle système)

⇒ l'activité de développement sera de plus en plus tournée vers la

réutilisation de composants existants

Validation

- Tests de vérification
  - Vérification de la robustesse et cohérence du système en particulier dans le cas d'exception
  - Testeur ≠ concepteur ou programmeur
  - Logiciels de tests : toute ligne de code doit être testée !
- Recette
  - Validation client : accord avec les besoins
  - Une fois les tests de vérification satisfaits
  - Planification dès les spécifications : cahier de recettes
  - Activité souvent sous-estimée…

Maintenance

- Deux techniques de maintenance
  - Correction des erreurs du système
  - Demande d'évolution (modification de l'environnement technique, nouvelle fonctionnalité)
- Facteurs de qualité essentiels
  - Corrections : robustesse
  - Évolutions : modifiabilité, portabilité
- Étape longue, critique et coûteuse
  - 80 % de l'effort de certaines entreprises (pb de pratiques ?)