Gestion de projets

analyse des besoins et génie logiciel

Philippe Collet

Licence 3 MIAGE

2012-2013

http://miageprojet2.unice.fr/User:PhilippeCollet/Gestion_de_projet_2012-2013

Objectif

- Appréhender et appliquer les concepts de l'analyse des besoins et de la gestion des projets informatiques à grande échelle.

- Pré-requis :
  - Aucun

Evaluation

- Projet réalisé lors des TD : Cahier des charges d'un très grand projet, par équipe de 4 à 5
  - Evaluation intermédiaire : 20 %
  - Evaluation finale sur le rendu du projet : 40 %

- Interrogation de 2h (40 %) à la fin du cours / support de cours autorisé

TD et projet : fonctionnement

- Lundi 22/10 : publication des sujets
- Vendredi 26/10 : date limite de retour par mail (formation des équipes de 5 et liste ordonnée de 2 sujets choisis)
- Mardi 30/10 : validation des équipes et premier TD
- Mercredi 09/01/2013 : date limite de rendu du projet (cahier des charges envoyé par mail)
- Après le mercredi 09/01 : examen terminal

Plan

- Introduction : mythes et réalités
- Génie logiciel et gestion de projet
- Analyse des besoins, cahier des charges
- Cycle de vie du logiciel
- Gestion de projets
- Vérification et Validation

Université Nice SOPHIA ANTIPOLIS

Introduction

- Pourquoi ?
- Génie logiciel, projet : définition(s)
- Pourquoi c'est difficile ?

Pourquoi le Génie logiciel ?

- pour passer du développement logiciel *ad hoc* et *imprévisible*

à

- un développement logiciel systématique et réfléchi

Génie logiciel : historique

- Histoire drôle : la facture à 0 euro
- Réponse à la crise du logiciel, il y a 40 ans
- Conférence OTAN 1968

La crise du logiciel

- Grosses erreurs :
  - Les sondes perdues (Vénus dans les années 60, Mars en 99)
  - La fausse attaque de missiles (1979)
  - Les missiles Patriotes (1991)
  - 1er vol d'Ariane 5 (1996)
  - L'aéroport de Denver (1994-96)
  - L'an 2000...
- Les projets logiciels
  - ne livrent pas le produit dans les temps
  - coûtent beaucoup plus chers que prévu.
  - délivrent un produit de qualité très faible
  - échouent dans la majorité des cas !!!
  - Étude américaine de 1995 : 81 milliard $ / an en échec

Pourquoi ne construit-on pas les logiciels comme on construit des ponts ?

- Génie civil
  - Échecs moins nombreux
  - L 'écroulement est grave et met en danger l 'utilisateur
  - On ne répare pas un pont « buggé », on reconstruit un pont qui s'écroule.
  - On inspecte tous les ponts construits sur le même modèle
  - Les ponts résistent à toutes les conditions (à 99 %...)
- Génie logiciel
  - Échecs très nombreux
  - *Crash* système pas considéré comme inhabituel
  - Cause du bug pas directement identifiable
  - Dommages mineurs
  - A part dans les systèmes critiques, on considère que le logiciel ne peut anticiper **TOUTES** les situations

☞ ***Différence d 'approche face à l 'échec, face aux pannes***

Pourquoi ne construit-on pas les logiciels comme on construit des ponts ?

- Génie civil
  - Plusieurs milliers d 'années d 'expérience dans la construction des ponts
  - Les ponts sont des systèmes continus et analogiques
  - On repeint un pont, on change l 'enrobée de la route…
  - On ne reconstruit pas la moitié d 'un pont
- Génie logiciel
  - Les systèmes informatiques se complexifient trop vite
  - Les logiciels passent par des états discrets, dont certains ne sont pas prévus
  - Ajouts, changements de fonctionnalités, de plate-formes...

☞ ***Différence dans la complexité et dans la maintenance***

Génie logiciel : définition (ou presque)

- Discipline (= méthodes, techniques et outils)
  - basée sur le savoir (théorique)
  - le savoir-faire (pragmatique)
  - et le faire savoir (communication)
  - pour produire (développement)
  - de façon industrielle (taille, diffusion)
  - des logiciels (= les produits)
  - *de qualité au meilleur prix...*

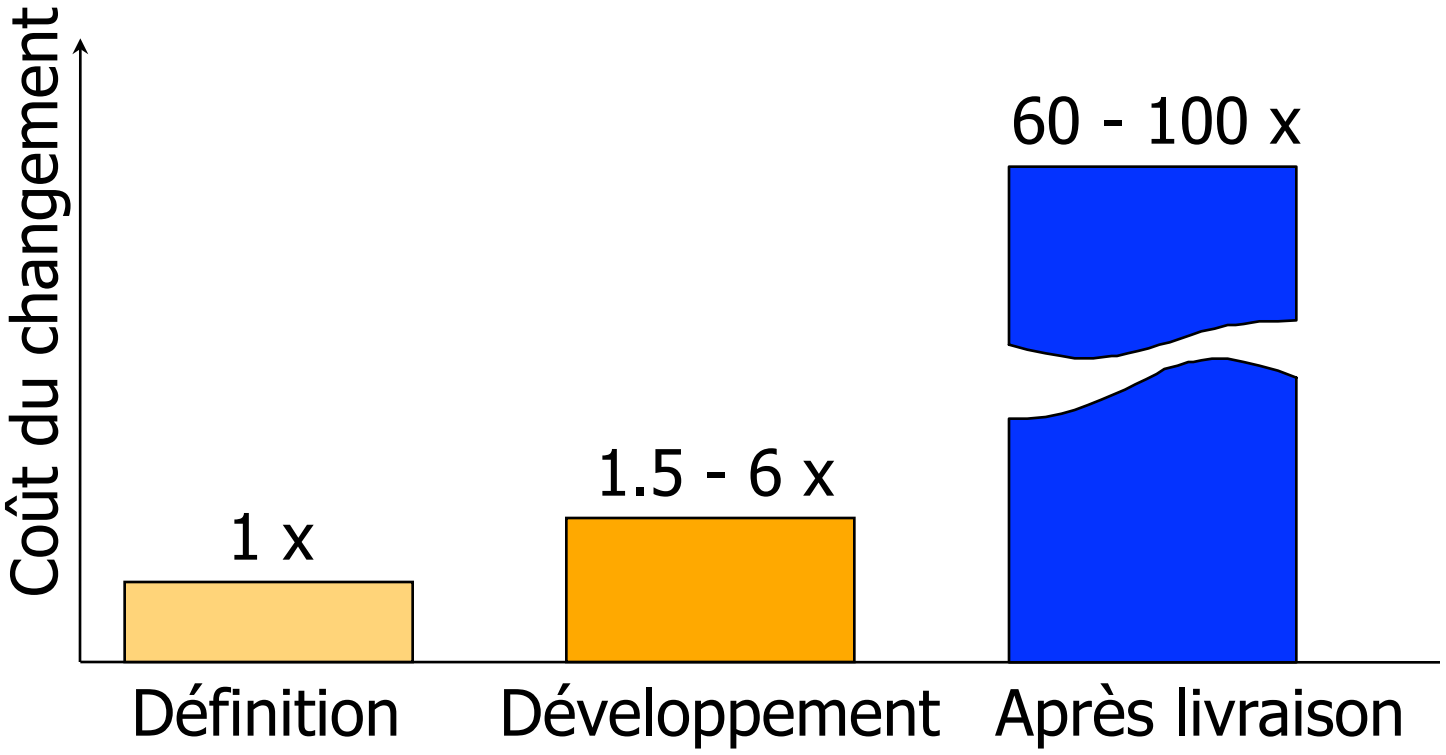
Les mythes de gestion de projet

- Les outils actuels sont la solution
  - un *nul* avec un outil est toujours un *nul*
- Si on est en retard, on ajoutera du personnel

Les mythes du client

- Une idée générale des objectifs est suffisante pour commencer le codage – on ajoutera les détails plus tard
  - Une forte communication entre clients et développeurs est toujours nécessaire
- Les changements peuvent être facilement répercutés parce que le logiciel est flexible
  - Les changements ne peuvent être évités, c'est la vie...
  - Les changements tardifs coûtent très chers

L'impact des changements



Les mythes des développeurs

- Une fois que le programme est écrit et qu'il *tourne*, le travail est terminé
  - 50-70% de l'effort est réalisé après la livraison
- Jusqu'à ce que le programme *tourne*, il n'y a aucun moyen d'évaluer sa qualité
  - Inspections & revues
- La seule chose à livrer pour un projet réussi est un programme qui marche
  - Documentation (utilisateur, maintenance)

Pourquoi c'est difficile ?

- Invisibilité du logiciel
- Facilité apparente d'écriture et de modification
- Le produit fini est différent du programme :
  - produit logiciel : généralisation, tests, documentation, maintenance *** 3**
  - programme intégré dans un système (interfaces) *** 3**

☞ produit logiciel intégré dans un système : *** 9**

☞ *The mythical man-month* de Frédéric Brooks (1975)

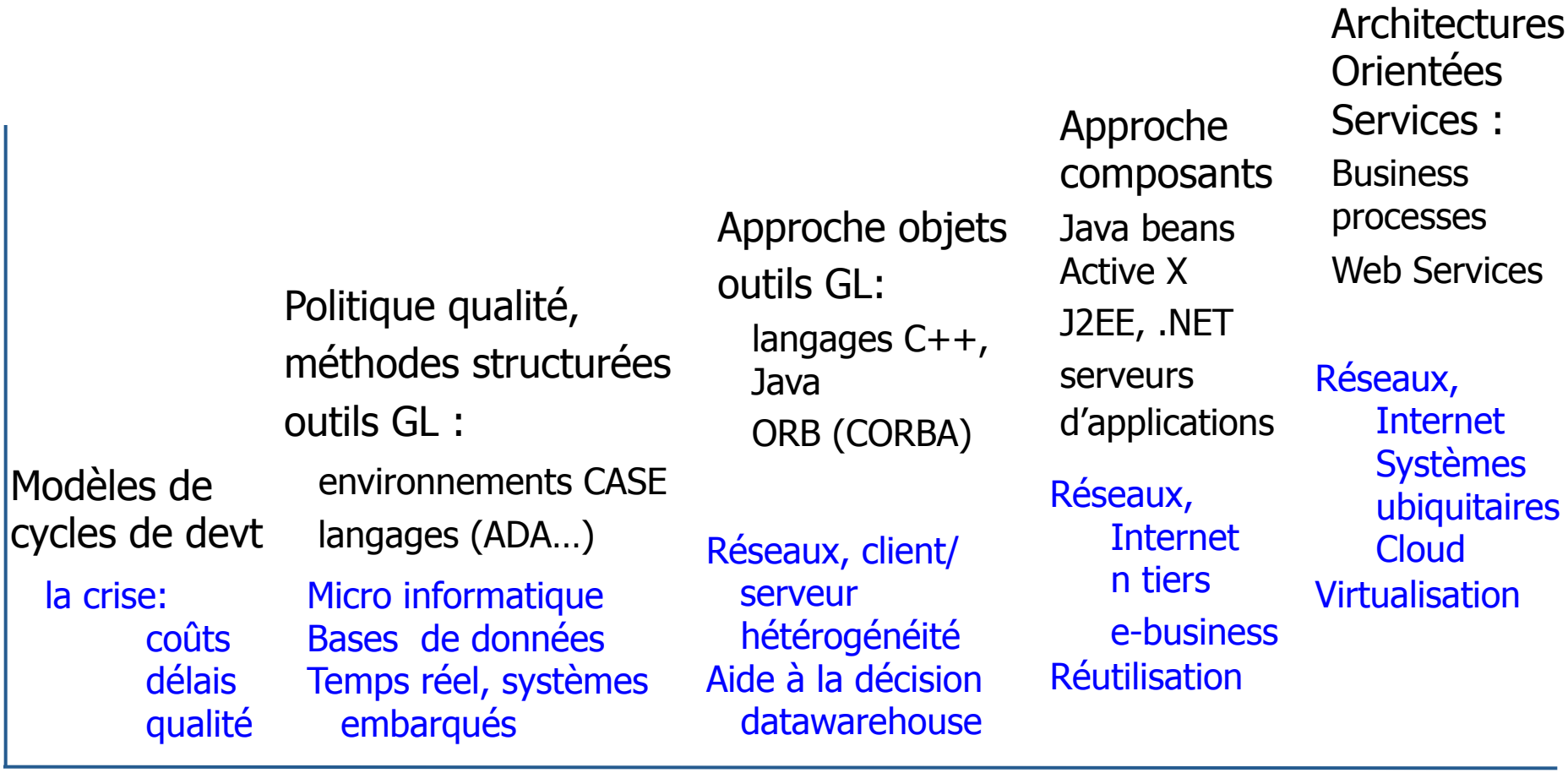
Pourquoi c'est difficile ? (suite)

- La spécification :
  - Le logiciel modifie son environnement
- La maintenance (67 % du coût total)
  - corrective (50 %) : 60 % des défauts proviennent d 'erreurs de spécification ou de conception.
  - évolutive : se méfier de l'effet 2ème version...
- L'optimisme !
  - *Ajouter du personnel à un projet en retard ne fait que le retarder plus* Loi de Brooks

Les réponses à la crise

- Recherche du concept de qualité
  - Maîtrise du processus de développement
  - Méthodes et outils structurés (CASE)
  - Programmes de recherche
- Approche solo
  - Prolog, Lisp, Smalltalk, etc.
- Approche par objets
  - ☞ Réutilisation théorique
- Approche par composants
  - ☞ Réutilisation quasi-effective

40 ans de Génie logiciel



Le logiciel, fin 2012

- Fiabilité meilleure mais…
- **partout**, sous toutes les formes
- gros, **très très** gros, cher, **très très** cher !
- Types :
  - Sur mesure (à partir de composants, de services)
  - Générique (les progiciels)
  - Interconnectés, en constante évolution…
- Acteurs : constructeurs, SSII, utilisateurs

Liste (non-exhaustive) des problèmes

- Productivité
  - Coûts et délais
- Simplicité
  - Uniformité, orthogonalité, unicité, normalisation
- Communication H/M
  - Ergonomie, interactivité, multimédia, simplicité, rapidité, documentation (contextuelle)
- Fonctionnels
  - Étendue et pertinence des services, fiabilité (correction, robustesse)

Liste des problèmes (suite)

- Matériau
  - Logiciel, structure, langage, modularité…
- Organisation
  - Gestion de projet visibilité, protections, contrôles
- Réalisme
  - Adéquation aux besoins, évolutivité
- Économique
  - Réutilisabilité, transportabilité
- Diversité
  - BD, IA, Calcul, Parallélisme, Réseau, Internet, intranet
- Divers
  - Juridique, psychologique

Il faut donc…

- Développer des
  - nouveaux produits
  - nouvelles fonctions
  - nouveaux portages
- A partir
  - d'un cahier des charges
  - d'applications existantes
  - de composants existants
- En
  - interne
  - sous-traitance

☞ ***avec des objectifs de qualité et de productivité***

Génie logiciel : les besoins

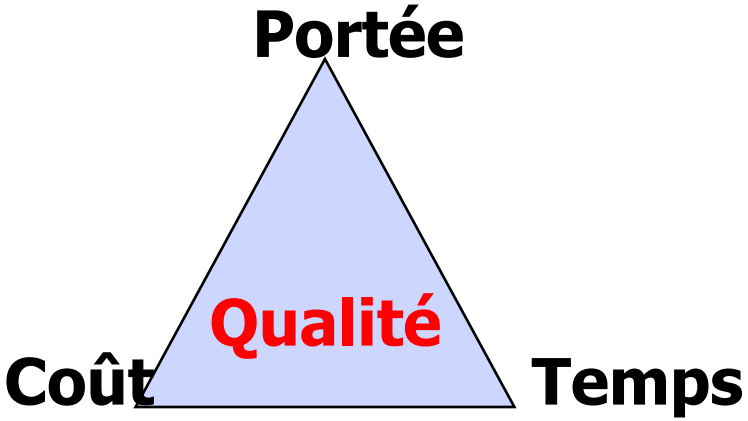
- Langages *pour décrire*
- Outils *pour manipuler*
- Méthodes *pour décider*
- Théories *pour démontrer*
- Professionnels *pour réaliser*
- Logistique *pour supporter*

Qu'est qu'un projet ?

- Définition
  - Un effort temporaire
  - qui est progressivement planifié, contrôlé et exécuté
  - par des personnes travaillant avec des contraintes de ressources
  - pour créer un produit, service ou résultat unique
- Temporaire
  - Début et fin sont définies
  - Pas forcément court, mais fini
- Planifié, contrôlé et exécuté
  - Nécessité d'une planification initiale et d'un suivi
  - Le travail s'organise pour accomplir des objectifs (exécution)
  - Le travail nécessite des vérifications pour être correctement exécuté
  - Et tout cela, progressivement, en étapes, en affinant au fur et à mesure

Qu'est qu'un projet ? (suite)

- Par des personnes
  - La dimension humaine est primordiale
- Avec des contraintes de ressources
  - Contraintes de temps, de coût
  - Tout limitation ou frontière du projet est une contrainte
- *Gérer un projet, c'est essentiellement gérer continuellement ces contraintes, pour atteindre des critères de qualité prédéfinis*



Qu'est qu'un projet ? (suite)

- Pour créer un produit, service ou résultat unique
  - Le projet crée quelque chose de nouveau
  - Quelque chose de tangible (produit) ou non (service, résultat)
    - Exemple : Diminuer le temps d'attente au téléphone de 20 %
- Comment déterminer l'objectif du projet ?
  - L'objectif du projet est quelque chose que l'organisation ne peut obtenir par son fonctionnement normal
  - Exemple de fonctionnement normal : Produire les fiches de paie mensuelles
- Questions
  - Pour un constructeur de maisons, chaque chantier est-il un projet ?

Naissance du projet

- Pourquoi démarre-t-on un projet ?
  - Besoin, demande, idée, inspiration…
- Besoin organisationnel
  - Amélioration dans le processus métier ou création d'un nouveau
- Demande du marché
  - Opportunité pour un produit ou un service
- Demande d'un client
- Avance technologie (ou obsolescence)
- Nécessité légale
- Besoin social

Fin du projet

- Quand
  - Les objectifs sont atteints
  - Il devient clair qu'on ne pourra atteindre les objectifs
  - Le besoin n'existe plus

Projet et production : ne pas confondre

- Production
  - Efforts de l'organisation pour soutenir son métier principal
  - C'est une activité récurrente
- Points communs avec le projet
  - Deadlines (dates limites), personnes, contraintes de temps et de coût
  - Planification, contrôle
- Différences avec le projet
  - Toujours en cours, pas d'objectif fixé, ni de terminaison, organisation stable
  - Incertitude faible, retour sur investissement positif

Caractéristiques du projet

- Livrables
  - La partie la plus importante d'un projet, souvent multiples
  - On parle parfois d'artefact, comme quelque chose qu'il est nécessaire de produire, sans que ce soit un livrable
- Portée du produit
  - Caractéristiques et fonctionnalités du produit
- Portée du projet
  - Comment les objectifs vont être atteints
  - Donc, le travail, et uniquement le travail, pour réaliser les… livrables
  - Donc, directement impacté par le temps et le coût
- Impossible de définir la portée du projet sans la portée du produit

Qualités du logiciel

- Il faut bien distinguer
  - Les qualités utiles à l'utilisateur, donc *a priori* souhaitées par le client
    - Phases d'exploitation
  - Les qualités utiles au développeur
    - Phases de construction et de maintenance

Qualités pour l'utilisateur

- Fiabilité = Validité + Robustesse
  - Validité (Efficacité) ≡ correction, exactitude
    - *Efficacité : qualité d'une chose ou d'une personne qui donne le résultat escompté*

    ☞Assurer exactement les fonctions attendues, définies dans le cahier des charges et la spécification, en supposant son environnement fiable

    ☞Adéquation aux besoins

Qualités pour l'utilisateur (suite)

  - Robustesse: faire tout ce qu'il est utile et possible de faire en cas de défaillance: pannes matérielles, erreurs humaines ou logicielles, malveillances…
- Performance (parfois appelée efficacité)
  - Utiliser de manière optimale les ressources matérielles : temps d'utilisation des processeurs, place en mémoire, précision…
- Convivialité
  - Réaliser tout ce qui est utile à l'utilisateur, de manière **simple, ergonomique, agréable** (documentation, aide contextuelle…

Qualités pour le développeur

- Documentation
  - Tout ce qu'il faut, rien que ce qu'il faut, là où il faut, quand il faut, correcte et adaptée au lecteur : **crucial !**
- Modularité = Fonctionnalité + Interchangeabilité + Évolutivité + Réutilisabilité
  - Fonctionnalité
    - Localiser un phénomène unique, facile à comprendre et à spécifier

Qualités pour le développeur (suite)

- Interchangeabilité
  - Pouvoir substituer une variante d'implémentation sans conséquence fonctionnelle (et souvent non-fonctionnelle) sur les autres parties
- Évolutivité
  - Facilité avec laquelle un logiciel peut être adapté à un changement ou une extension de sa spécification
- Réutilisabilité
  - Aptitude à être réutilisé, en tout ou en partie, tel que ou par adaptation, dans un autre contexte : autre application, machine, système…

Qualités pour l'entité de développement

- Client satisfait *(est-ce possible ?)*
- Coût minimum de développement
  - Nombre de développeurs
  - Formation des développeurs
  - Nombre de jours de réalisation
  - Environnement
  - Réutilisation maximale

Génie logiciel : le défi

- Contradictions apparentes
  - Qualités vs coût du logiciel
  - Qualités pour l'utilisateur vs qualités pour le développeur
  - Contrôler vs produire
- Conséquences
  - ☞ Chercher sans cesse le meilleur compromis
  - ☞ Amortir les coûts
    - Premier exemplaire de composant coûteux à produire ou à acheter, puis amortissement…

Objectifs de qualité

- Réduire le nombre d'erreurs résiduelles
- Maîtriser coût et durée du développement
- sans nuire à la créativité et à l'innovation

- Adéquation aux besoins
- Efficacité temps/espace
- Fiabilité
- Testabilité, Traçabilité
- Adaptabilité
- Maintenabilité
- Convivialité (interface et documentation)

☞ ***doivent rejoindre les objectifs de productivité***

Université Nice SOPHIA ANTIPOLIS

Analyse des besoins et cahier des charges

- Terminologie
- La faisabilité
- L ’analyse des besoins
- Le cahier des charges

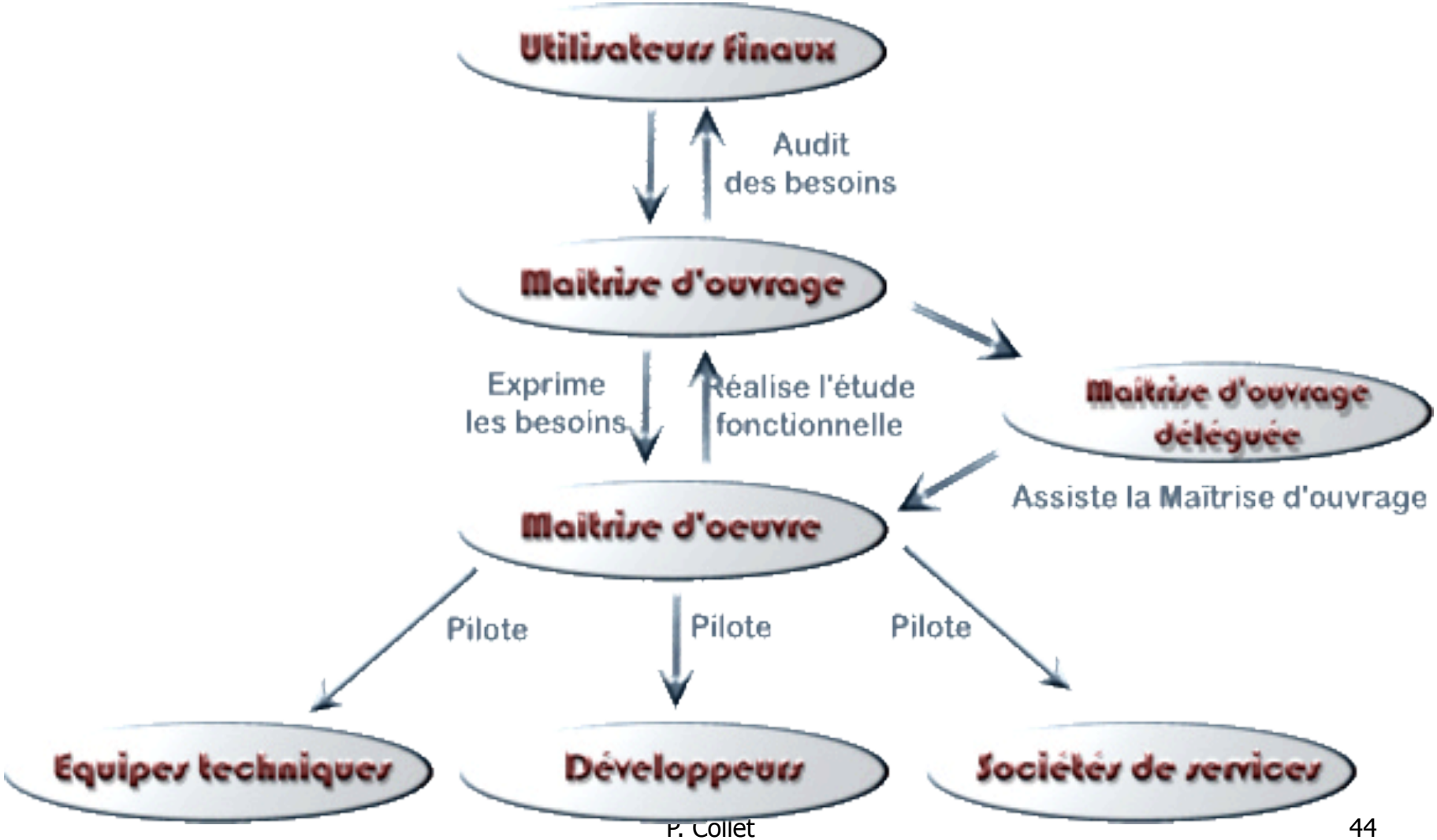
Système informatique

- "Un ensemble d'éléments qui sont organisés pour accomplir un but prédéfini par un traitement de l'information"
- utilise des :
  - Logiciels
  - Matériels (informatiques)
  - Personnes
  - Bases de données (ensemble *organisée* de données)
  - Documentation
  - Procédures (étapes qui définissent comment utiliser les éléments du système)

Développement d'un système

- La maîtrise d'ouvrage
  - Entité responsable de l'expression du besoin
  - Souvent non informaticien
  - *Besoin réel / budget*
  - ☞ Possibilité de **maîtrise d'ouvrage déléguée**
- La maîtrise d'œuvre
  - Entité responsable de la concrétisation de l'idée en outil informatique
  - Pas de connaissance fonctionnelle
  - *Bons choix techniques, adéquation avec les besoins, performances…*

Différence dans les *maîtrises*



Étude de faisabilité

- Tous les projets sont faisables !
  - étant donnés des ressources et un temps infinis
- Mais les ressources sont limitées...

Étude de faisabilité (suite)

- Faisabilité économique
- Faisabilité technique  au plus tôt
  - Risques de développement
  - Disponibilité des ressources
  - Technologie nécessaire
- Faisabilité légale
- Alternatives

Étude de faisabilité : aspects économiques

- Analyse du rapport Coût/Bénéfice :
  - Coût du système
  - Bénéfices mesurables (en €)
  - Bénéfices **non** mesurables
    - meilleure conception
    - meilleures décisions marketing
    - satisfaction accrue du client

☞ L'analyse Coût/Bénéfice est souvent le moyen d'obtenir le feu vert de la direction

Analyse des besoins

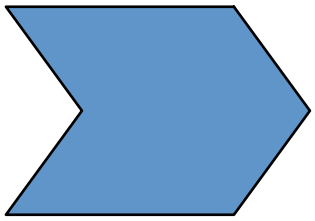
- Définition des besoins à différents niveaux d'abstraction :
  - Besoins de l'utilisateur
  - Besoins des composants
- Définition du système à réaliser avec le point de vue de l'utilisateur et/ou du client
  - ☞ Les utilisateurs doivent être capables de comprendre ce document

☞ Analyse des besoins : *LE QUOI*

☞ Conception : *LE COMMENT*

Le processus d'analyse

- Processus de découverte, de raffinement, de modélisation et de spécification
- Les utilisateurs/clients et les développeurs ont des rôles **actifs**
- Les utilisateurs ne sont pas satisfaits par un système bien conçu et bien implémenté



Les utilisateurs veulent des systèmes qui satisfont leurs besoins

Bases de la communication

- Écouter le client
  - Écoute ≠ Compréhension
- Préparer les réunions
  - Connaissance du client et des contacts
  - Lecture des documents disponibles
  - Penser aux objectifs de la réunion
  - Penser aux problèmes
  - Être à l'heure…

Initier la communication

- La première réunion peut être *bizarre*
  - Pas de connaissance des intervenants
  - Attentes différentes
  - Mais : chacun veut que cela réussisse
- Compréhension minimale du problème :
  - Qui est derrière la demande de cette réalisation ?
  - Qui va utiliser la solution proposée ? Avec quels bénéfices ?
  - Quelle serait une "bonne" solution ?
  - Quel sera l'environnement de la solution ?
  - Y-a-t-il des contraintes ? Des problèmes de performance ?
  - Qui sont les bons interlocuteurs ? => réponses "officielles"
  - Ai-je oublié des questions ?
  - A qui d'autre dois-je m'adresser ?

Une bonne analyse

- Objectif premier : Maximiser la satisfaction des utilisateurs et des clients

- En tenant compte de 3 types de besoin
  - Normaux : besoins explicitement établis
  - Attendus : implicites, pas exprimés mais nécessaires
  - *Excitants* : allant au delà des espérances des clients

Indications à suivre...

- Comprendre le problème avant de commencer à créer la spécification des besoins
  - Ne pas résoudre le *mauvais* problème
- Développer des prototypes des interfaces utilisateurs (IHM)
  - Les interfaces utilisateurs déterminent souvent la qualité…
- Noter et tracer l'origine et les raisons d'un besoin
- Utiliser des vues multiples sur les besoins
  - Réduit les risques de rater quelque chose
- Classer les besoins par priorité
- Travailler pour éliminer les ambiguïtés

Le cahier des charges

- Première étape de l'expression du besoin
- Description globale des fonctions d'un nouveau produit ou des extensions à un produit existant
  - Énoncé du problème à résoudre
  - Liste des fonctions de base
  - Caractéristiques techniques
  - Priorités de réalisation
  - Facteurs de qualité
- Il doit être validé par le client et/ou l'utilisateur
- Il est la base du contrat entre clients et développeurs

Difficultés à établir le cahier

- Expression de la faisabilité
  - utiliser une maquette pour simuler
- Précision et non ambiguïté
  - utiliser un formalisme différent du langage naturel ?
- Le cahier des charges est un document technique, sans considération économique
  - sauf si on lui adjoint un plan de projet
- Recherche de *précision, cohérence, complétude, testabilité, traçabilité, maintenabilité, flexibilité...*

Contrer les problèmes du langage naturel

- Imprécisions et ambiguïtés qui devront être levées lors de la phase d'analyse
  - ☞Scinder le texte en paragraphes pour une meilleure traçabilité
  - ☞Ne pas inclure plusieurs concepts dans un même paragraphe
- ☞Ne pas mélanger :
  - Besoins : ce qui doit être fourni
  - Buts : souhait, vœu pieu, mais impossible à tester
  - Contraintes : qui doivent être décrites séparément

Les besoins non-fonctionnels

- Restrictions ou contraintes sur un service fourni par le système :
  - plate-forme matérielle
  - temps de réponse
  - MTBF : *Mean Time Between Failures*
- Raisons :
  - besoins des utilisateurs
  - contraintes de budget, …

☞Ces besoins doivent être quantifiables !

Cahier des charges ***épuré***

- Couverture
- Introduction
- Spécification des besoins fonctionnels
- Spécification des besoins non fonctionnels
  - Standards à atteindre, plate-forme, taille mémoire
- Glossaire

Couverture :

- Nom du projet / du produit
- Date
- Numéro de version
- Auteur(s)
- Responsabilités de chaque auteur
- Changements **clés** depuis la précédente version

Un plan type norme AFNOR X50-151

1. Présentation générale du problème
   1.1 Projet
     1.1.1 Finalités
     1.1.2 Espérance de retour sur investissement
   1.2 Contexte
     1.2.1 Situation du projet par rapport aux autres projets d e l'entreprise
     1.2.2 Etudes déjà effectuées
     1.2.3 Etudes menées sur des sujets voisins
     1.2.4 Suites prévues
     1.2.5 Nature des prestations demandées
     1.2.6 Parties concernées par le déroulement du projet et ses résultats (demandeurs, utilisateurs)
     1.2.7 Caractère confidentiel si il y a lieu
   1.3 Enoncé du besoin (finalités du produit pour le futur utilisateur tel que prévu par le demandeur)
   1.4 Environnement du produit recherché
     1.4.1 Listes exhaustives des éléments (personnes, équipements, matières…) et contraintes (environnement)
     1.4.2 Caractéristiques pour chaque élément de l 'environnement

Norme AFNOR X50-151 (suite)

2. Expression fonctionnelle du besoin

2.1 Fonctions de service et de contrainte

2.1.1 Fonctions de service principales (qui sont la raison d 'être du produit)

2.1.2 Fonctions de service complémentaires (qui améliorent, facilitent ou complètent le service rendu)

2.1.3 Contraintes (limitations à la liberté du concepteur-réalisateur)

2.2 Critères d 'appréciation (en soulignant ceux qui sont déterminants pour l 'évaluation des réponses)

2.3 Niveaux des critères d 'appréciation et ce qui les caractérise

2.3.1 Niveaux dont l 'obtention est imposée

2.3.2 Niveaux souhaités mais révisables

Norme AFNOR X50-151 (suite)

3. Cadre de réponse
   - 3.1 Pour chaque fonction
     - 3.1.1 Solution proposée
     - 3.1.2 Niveau atteint pour chaque critère d 'appréciation de cette fonction et modalités de contrôle
     - 3.1.3 Part du prix attribué à chaque fonction
   - 3.2 Pour l 'ensemble du produit
     - 3.2.1 Prix de la réalisation de la version de base
     - 3.2.2 Options et variantes proposées non retenues au cahier des charges
     - 3.2.3 Mesures prises pour respecter les contraintes et leurs conséquences économiques
     - 3.2.4 Outils d 'installation, de maintenance … à prévoir
     - 3.2.5 Décomposition en modules, sous-ensembles
     - 3.2.6 Prévisions de fiabilité
     - 3.2.7 Perspectives d'évolution technologique

Cahier des charges / Plan projet : Détails d'une réponse

- 1. Introduction
  - Résumé (ou Objectifs)
    - une demi page pour aller à l 'essentiel avec vue d 'ensemble
  - Fournitures
    - liste de ce qui est livré au client (logiciel, matériel…)
  - Définitions et acronymes
    - explication de tous les termes spécifiques au projet ou techniques au sens informatique
- 2. Organisation du projet
  - Processus
    - décomposition du projet dans le temps, justification du modèle de développement utilisé
  - Organisation structurelle
    - les rôles de chaque acteur du développement

Cahier des charges / Plan projet : Détails...

- Limites et interfaces
  - Tout ce que le système pourrait faire implicitement, mais qu'il ne fera pas
  - Toutes les interactions avec du matériel ou du logiciel extérieur, déjà présent ou apporté par un autre fournisseur

- 3. Gestion
  - Objectifs et priorités
    - Objectifs ? La qualité au meilleur prix et dans les délais !!!
    - Priorités : Si on est en retard ou que cela doit coûter plus cher, explication des propositions
  - Hypothèses, dépendances, contraintes
    - Hypothèses : Tous les décisions prises arbitrairement par rapport à l 'appel d 'offres
    - Dépendances : Identification des liens avec d 'autres systèmes informatiques (Cf. limites et interfaces) ou des actions à entreprendre
    - Contraintes : Identification de certaines contraintes posées par l 'existant ou par les besoins utilisateurs

Cahier des charges / Plan projet : Détails...

- Gestion du risque
  - Solutions pour gérer les risques posés par les hypothèses, les contraintes et les dépendances
- Moyens de contrôle
  - Description des moyens mis en œuvre lors du développement pour assurer la qualité, la satisfaction du client, etc.

- 4. Technique
  - Méthodes et outils employés
    - Notation, outils de conception, développement, de gestion du projet, de gestion des sources, des configurations...
  - Documentation
    - Manière de gérer (et générer) la documentation tout au long du projet

Cahier des charges / Plan projet : Détails...

- 5. Calendrier, Budget
  - Découpage en lots
    - Livraison intermédiaire et paiement intermédiaire
  - Dépendances
    - Identification des dépendances qui peuvent influer sur le calendrier (par exemple : attente d ’un élément spécifique par un fournisseur ou le client lui-même)
  - Ressources
    - Moyens mis en œuvre pour la réalisation (autres que les ressources humaines)
  - Budget
    - Chiffrage complet et… l ’addition SVP !
  - Échéancier
    - Calendrier déplié à partir d’une date précise de début

Cahier des charges / Plan projet : Détails...

- 6. Fonctions du produit
  - Une grande fonctionnalité
    - sous fonctionnalité
      - » opération…
      - » description en quelques lignes de ce que réalise cette opération, pour l 'utilisateur, et éventuellement en interne si cela est pertinent
    - …
  - Une autre grande fonctionnalité
  - …
- 7. Contraintes non fonctionnelles
  - plate-forme matérielle
  - temps de réponse
  - annexes techniques : schémas matériels, architecture logicielle pressentie...

Revue de spécification : questions

- Interfaces importantes décrites ?
- Diagrammes clairs ? Texte supplémentaire nécessaire ?
- Grandes fonctionnalités assurées ?
- Contraintes de conception réalistes ?
- Risques technologiques considérés ?
- Critères clairs de validation établis ?
- Y-a-t-il des incohérences, des omissions, des redondances ?
- Le contact avec l'utilisateur est-il terminé / complet ?

Université Nice SOPHIA ANTIPOLIS

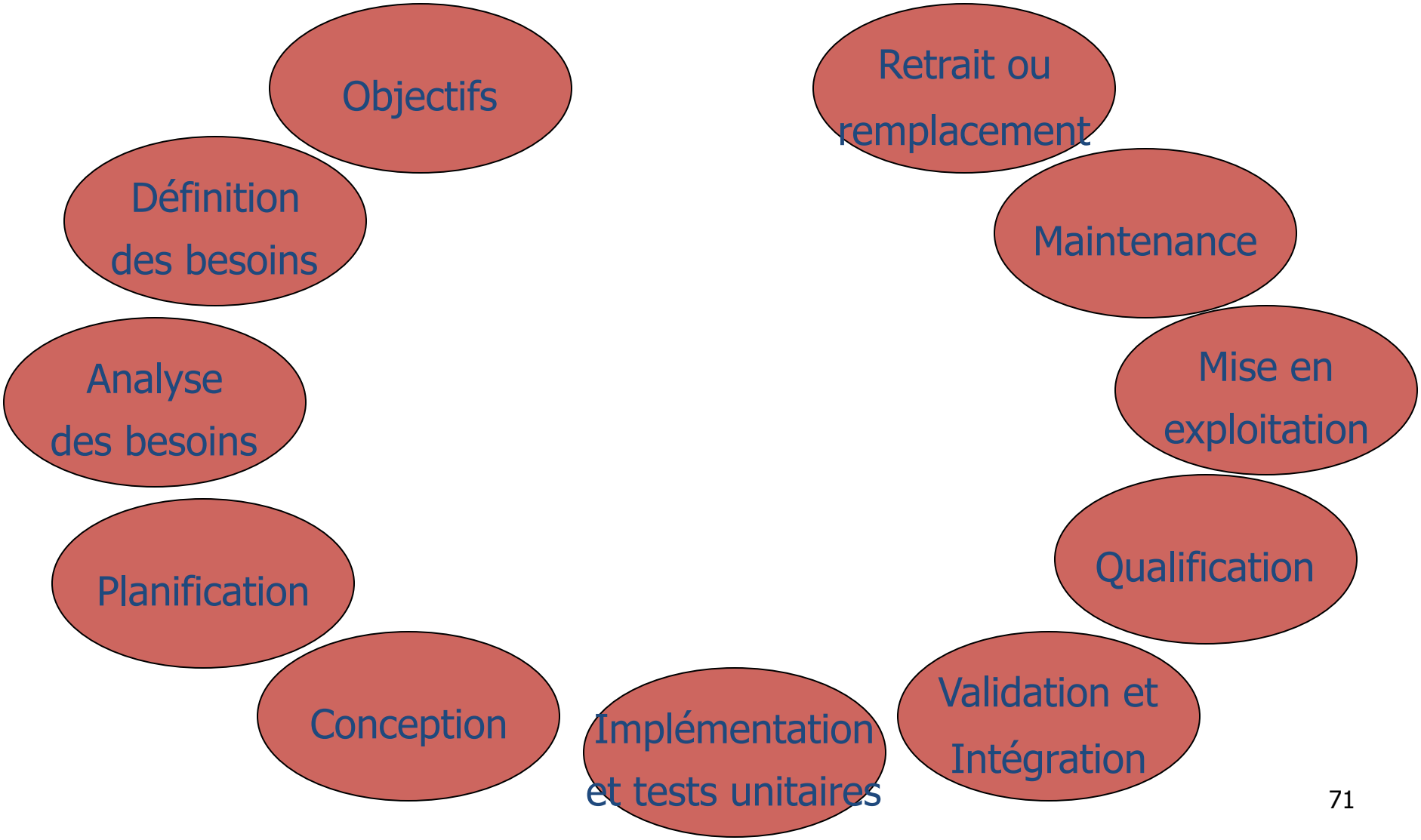
Cycle de vie du logiciel

- Les phases du cycle de vie
- Les modèles de développement

Notion de cycle de vie

- Description d'un processus pour :
  - la création d 'un produit
  - sa distribution sur un marché
  - son retrait
- Cycle de vie et assurance qualité
  - Validation : le bon produit ?
  - Vérification : le produit correct ?

Les phases du cycle de vie



Objectifs

- Fixés par *les donneurs d'ordre*
  - le management
  - ou une (bonne) idée...

- Quelques définitions
  - Clients : ceux qui veulent le produit
  - Utilisateurs : ceux qui vont l 'utiliser
  - Développeurs : ceux qui vont le fabriquer

Définition des besoins

- Un cahier des charges est normalement établi par le **client** en interaction avec utilisateurs et encadrement :
  - description des fonctionnalités attendues
  - contraintes non fonctionnelles (temps de réponse, place mémoire,...)
  - possibilités d'utilisation de *Use Cases*

☞ ***A l 'issue de cette phase : cahier des charges***

Analyse des besoins

- C'est la définition du produit
  - Spécification précise du produit
  - Contraintes de réalisation
- A l 'issue de cette phase :
  - Client et fournisseur sont d 'accord sur le produit à réaliser (IHM comprise)
  - ☞ ***Dossier d'analyse (spécifications fonctionnelles et non fonctionnelles)***
  - ☞ ***Ébauche de manuel utilisateur***
  - ☞ ***Première version du glossaire du projet***

Planification

- Découpage du projet en tâches avec enchaînement
- Affectation à chacune d'une durée et d'un effort
- Définition des normes qualité à appliquer
- Choix de la méthode de conception, de test...
- Dépendances extérieures (matériels, experts...)

☞ ***Plan qualité + Plan projet (pour les développeurs)***

☞ ***Estimation des coûts réels***

☞ ***Devis destiné au client (prix, délais, fournitures)***

Conception

- Définition de l'architecture du logiciel
- Interfaces entre les différents modules
- Rendre les composants du produits indépendants pour faciliter le développement

☞ ***Dossier de conception***
☞ ***Plan d 'intégration***
☞ ***Plans de test***
☞ ***Mise à jour du planning***

Implémentation et tests unitaires

- Codage et test indépendant de chaque module
- Produits intermédiaires :

☞ ***Modules codés et testés***

☞ ***Documentation de chaque module***

☞ ***Résultats des tests unitaires***

☞ ***Planning mis à jour***

Validation et Intégration

- Chaque module est intégré avec les autres en suivant le plan d ’intégration
- L’ensemble est testé conformément au plan de tests

☞ ***Logiciel testé***

☞ ***Tests de non-régression***

☞ ***Manuel d’installation***

☞ ***Version finale du manuel utilisateur***

Qualification

- Tests en vraie grandeur, dans des conditions normales d'utilisation
- Tests non-fonctionnels :
  - Tests de charge
  - Tests de tolérance aux pannes
- Parfois Bêta-test

☞ ***Rapports d'anomalie***

- *Déterminant dans la relation client-fournisseur*

Mise en exploitation

- Livraison finale du produit (packaging)
- Installation chez le client
- Est-ce la fin des problèmes ?

☞**AU CONTRAIRE**

☞**Ce n'est rien en comparaison de la…**

Maintenance

- Rapport d'incident (ou anomalie)
- Demande de modification corrective
- Demande d'évolution (avenant au contrat)
- Code et documentation modifiés...
- Nouvelle série de tests :
  - unitaires
  - d 'intégration
  - de non-régression

Exemples de durée de cycle

- SGBD relationnel
  - 1er proto : 5 à 7 ans

    Investissement > 100H An
  - 1er système commercial : 3 à 4 ans

    Investissement > 150H An
  - Maintenance : > 10 ans

    10 à 15 H par an

    nouvelle livraison tous les 6 mois à 1 an
- Langage ADA (1983)
  - Définition et analyse des besoins : 3 ans
  - Compilateur industriel : 3ans

    Investissement > 50H An
  - Maintenance : > 15 ans

    5 à 10 H par an

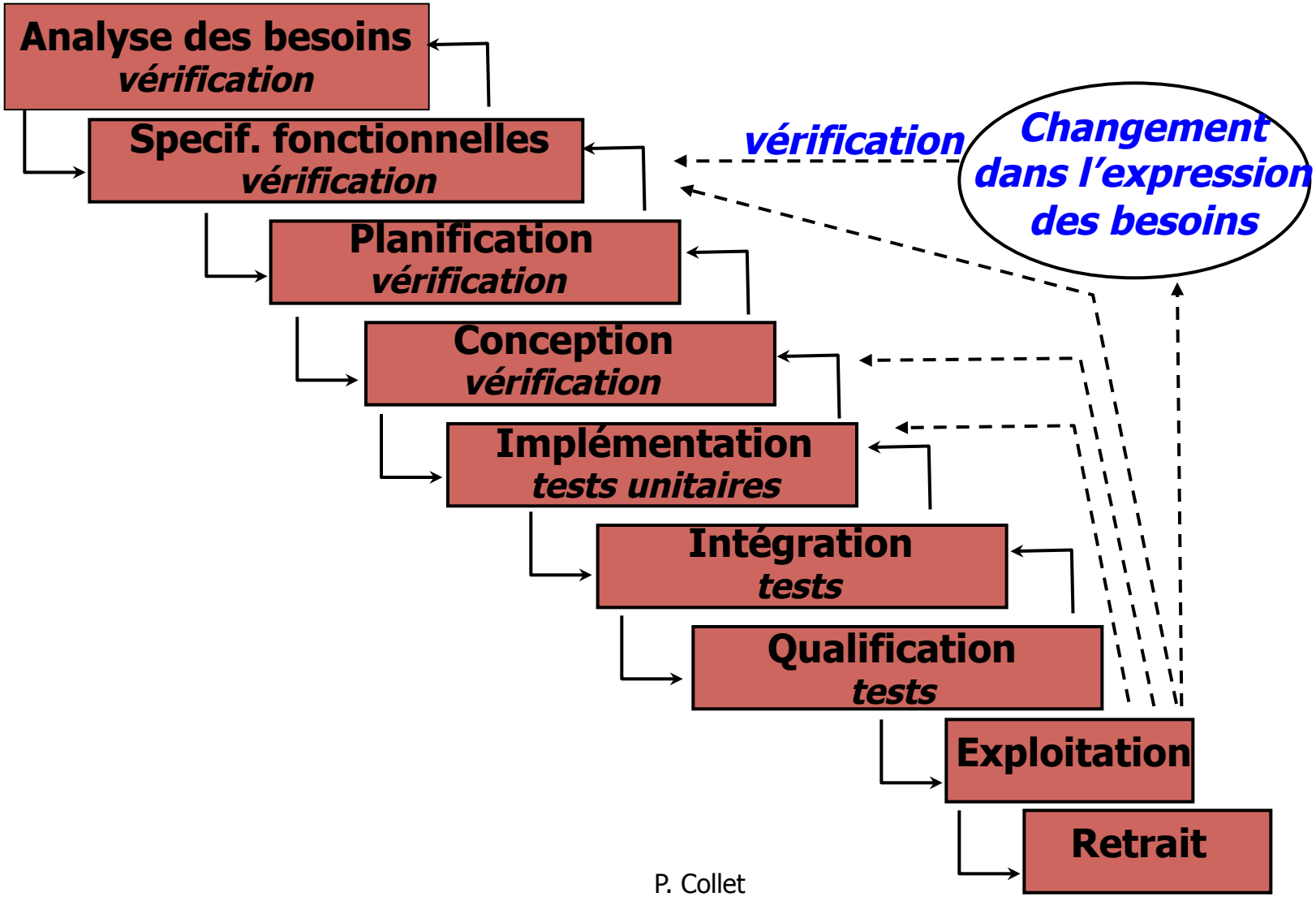
    livraison tous les 1 ou 2 ans

  ☞ Nouvelle version : Ada95

Les approches de développement

- Approche cartésienne, déterministe
  - structurée descendante : *cascade ou V*
- Approche heuristique, par prototypage
  - ascendante : *incrémental ou prototypage*
- Approche objets :
  - aucune organisation **spécifique** n'est vraiment mise en avant

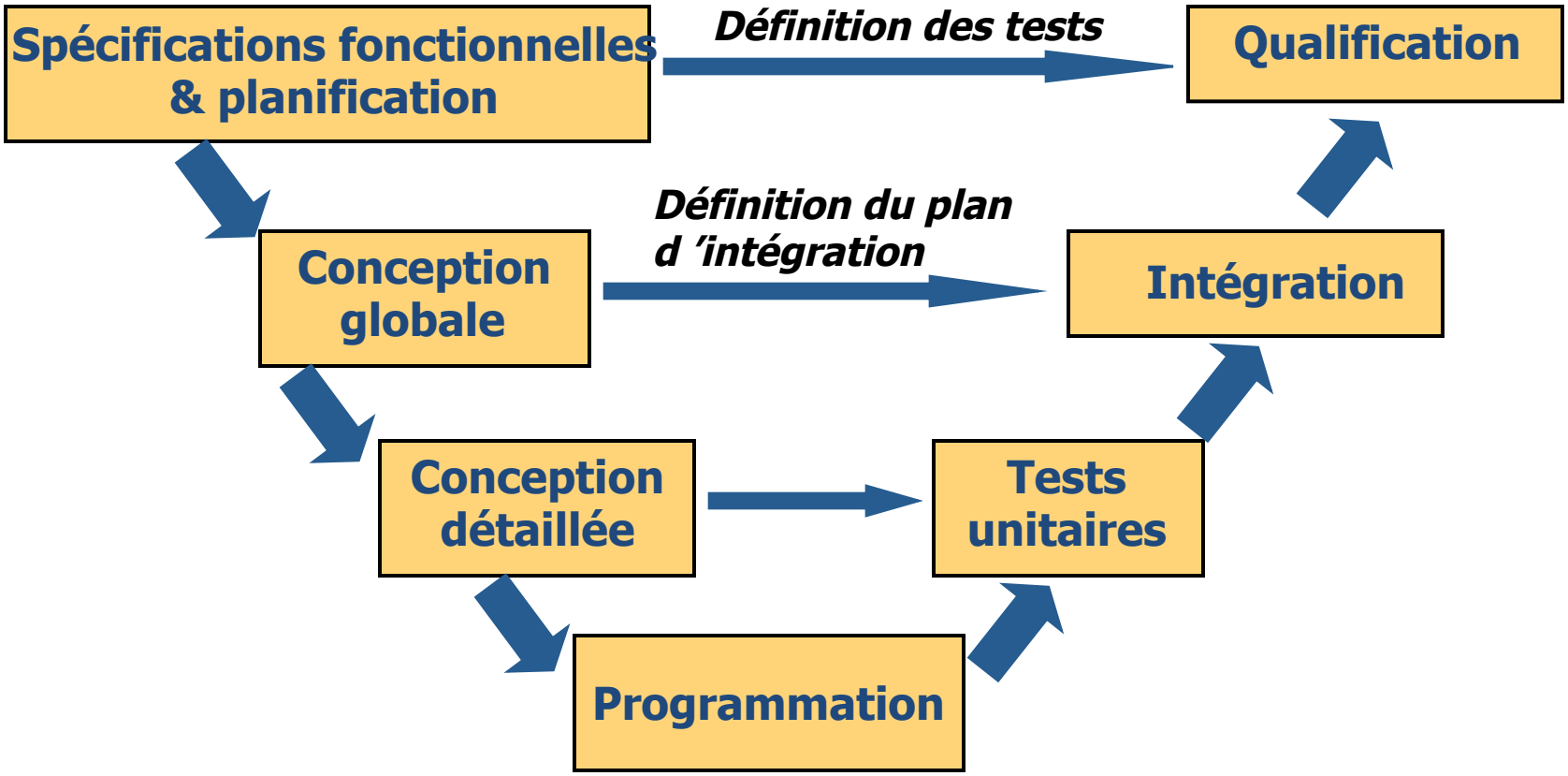
Modèle en cascade (1970)



Problèmes du modèle en cascade

- Les vrais projets suivent rarement un développement séquentiel
- Établir tous les besoins au début d'un projet est difficile
- Le produit apparaît tard
- Seulement applicable pour les projets qui sont bien compris et maîtrisés

Modèle en V

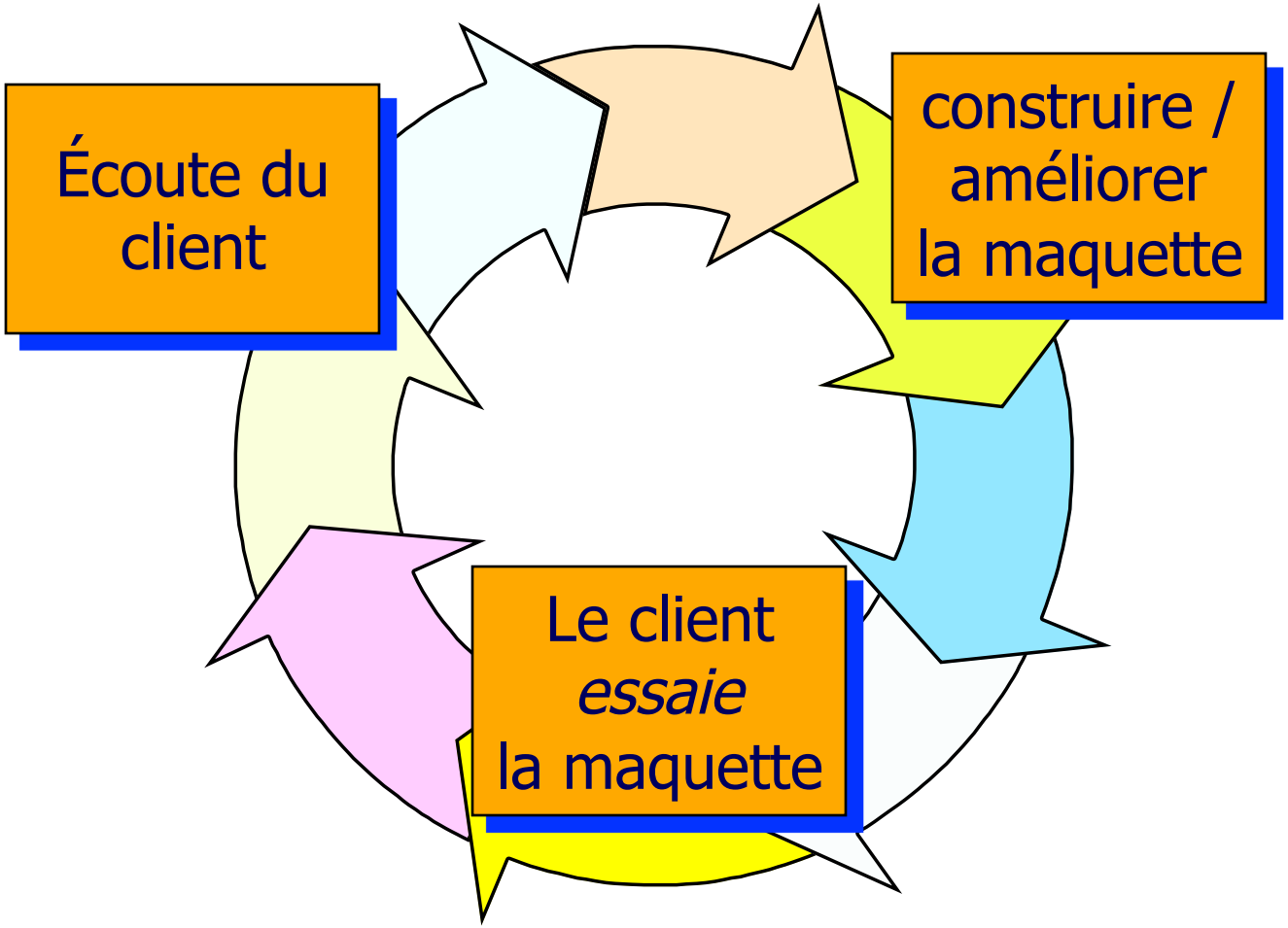


☞ *Gestion des configurations, de projet, plan assurance qualité*

Comparaison

- Le cycle en V
  - permet une meilleure anticipation
  - évite les retours en arrière
- Mais
  - le cadre de développement est rigide
  - la durée est souvent trop longue
  - le produit apparaît très tard

Prototypage



Prototypage, RAD

RAD : *Rapid Application Development*

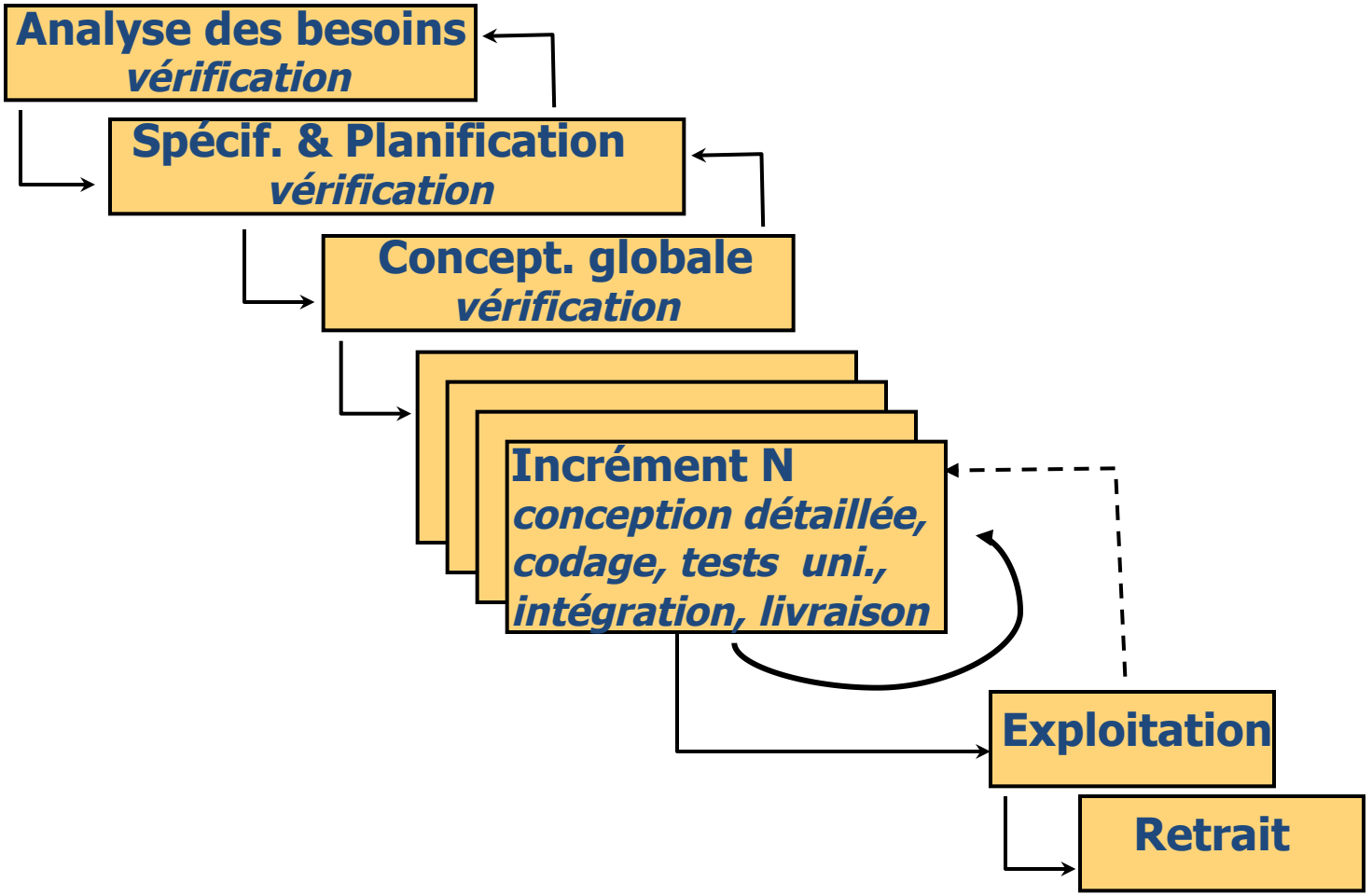
- Discuter et interagir avec l'utilisateur
- Vérifier l 'efficacité réelle d 'un algorithme
- Vérifier des choix spécifiques d 'IHM
- Souvent utilisé pour identifier les besoins
  - Prototype jetable (moins de risque ?)
- Souvent implémenté par des générateurs de code
  - Prototype évolutif

Prototypage, RAD (suite)

- Mais :
  - Les objectifs sont uniquement généraux
  - Prototyper n'est pas spécifier
  - Les décisions rapides sont rarement de bonnes décisions
  - Le prototype évolutif donne-t-il le produit demandé ?
  - Les générateurs de code produisent-ils du code assez efficace ?

☞ *Projets petits ou à courte durée de vie*

Modèle incrémental



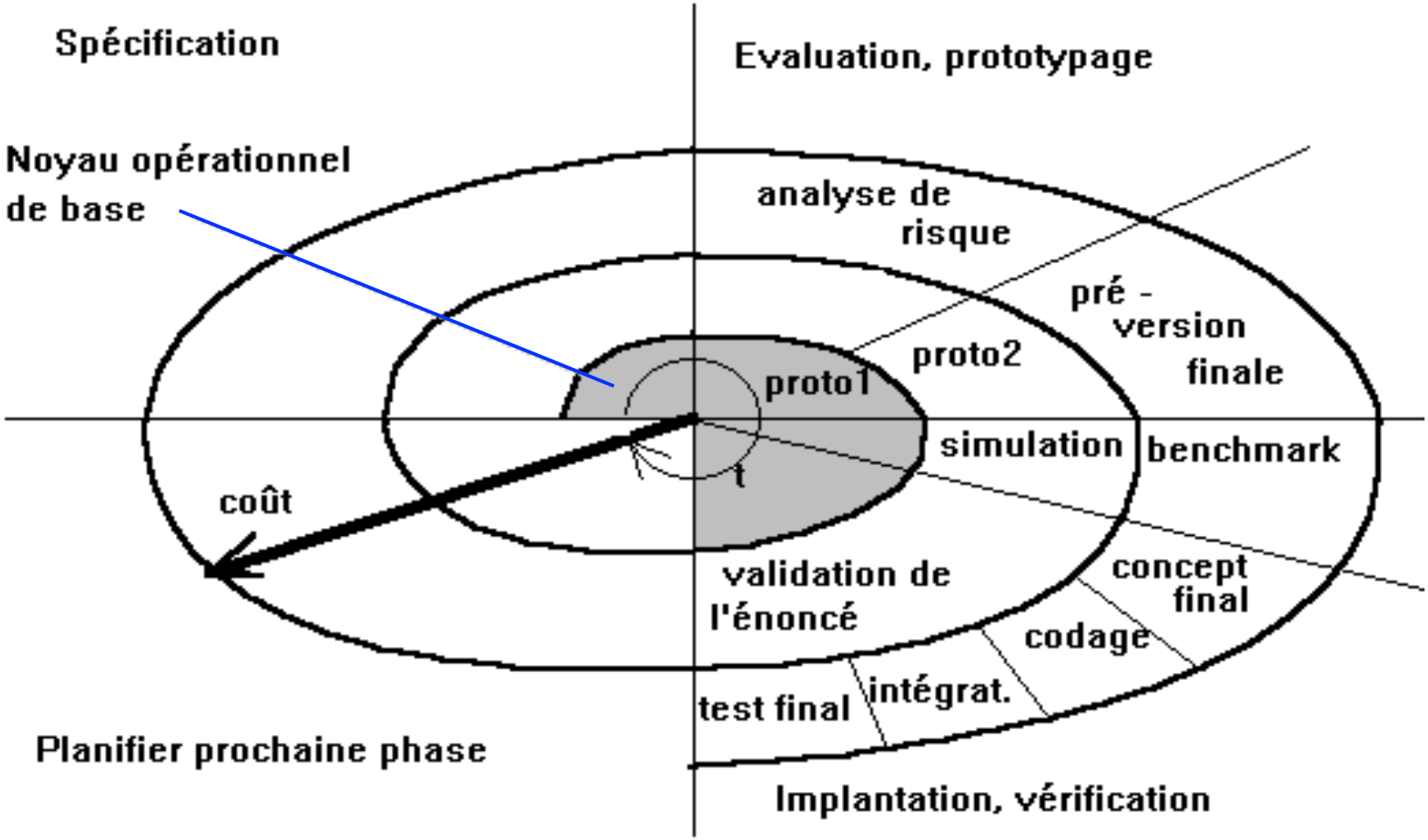
Le développement incrémental

- combine des éléments des modèles linéaires et du prototypage
- produit des incréments *livrables*
- se concentre sur un produit opérationnel (pas de prototype jetable)
- peut être utilisé quand il n'y a pas assez de ressources disponibles pour une livraison à temps

☞ *Le premier incrément est souvent le noyau*

☞ *Les incréments aident à gérer les risques techniques (matériel non disponible)*

Modèle en spirale (Boehm, 1988)



Modèle en spirale (suite)

- Spécification : communiquer avec le client
- Analyse de risque : évaluation des risques techniques et des risques de gestion
- Implémentation et vérification : construire, tester, installer et fournir un support utilisateur
- Validation: obtenir des *retours*
- Planification : définir les ressources, la répartition dans le temps

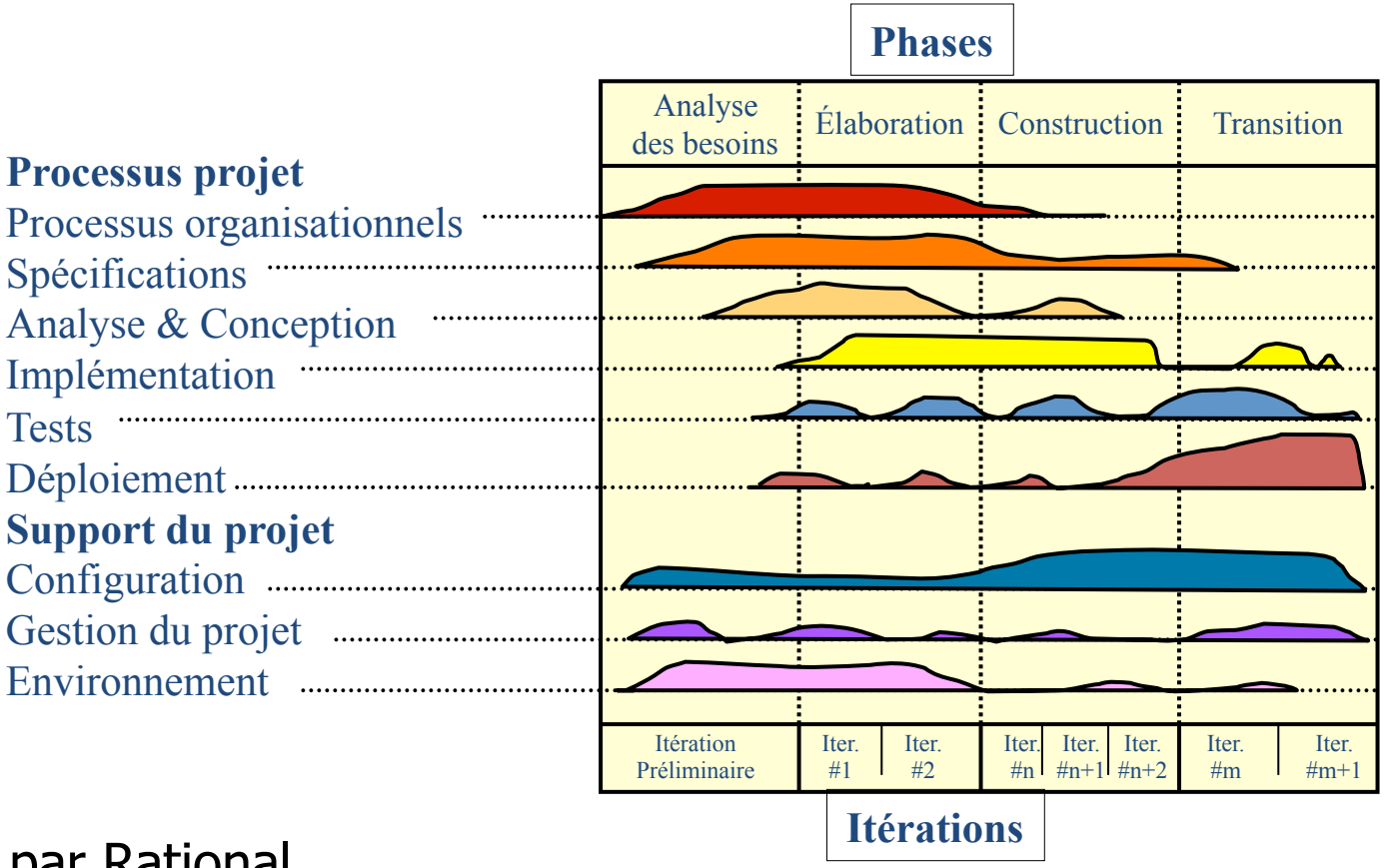
Modèle en spirale (suite)

- Couplage de la nature itérative du prototypage avec les aspects systématiques et contrôlés du modèle en cascade
- Les premières itérations peuvent être des modèles *sur papier* ou des prototypes
- Utilisation possible tout au long de la vie du produit

☞ *Réduit les risques si bien appliqué*

☞ *Les augmentent considérablement si le contrôle faiblit*

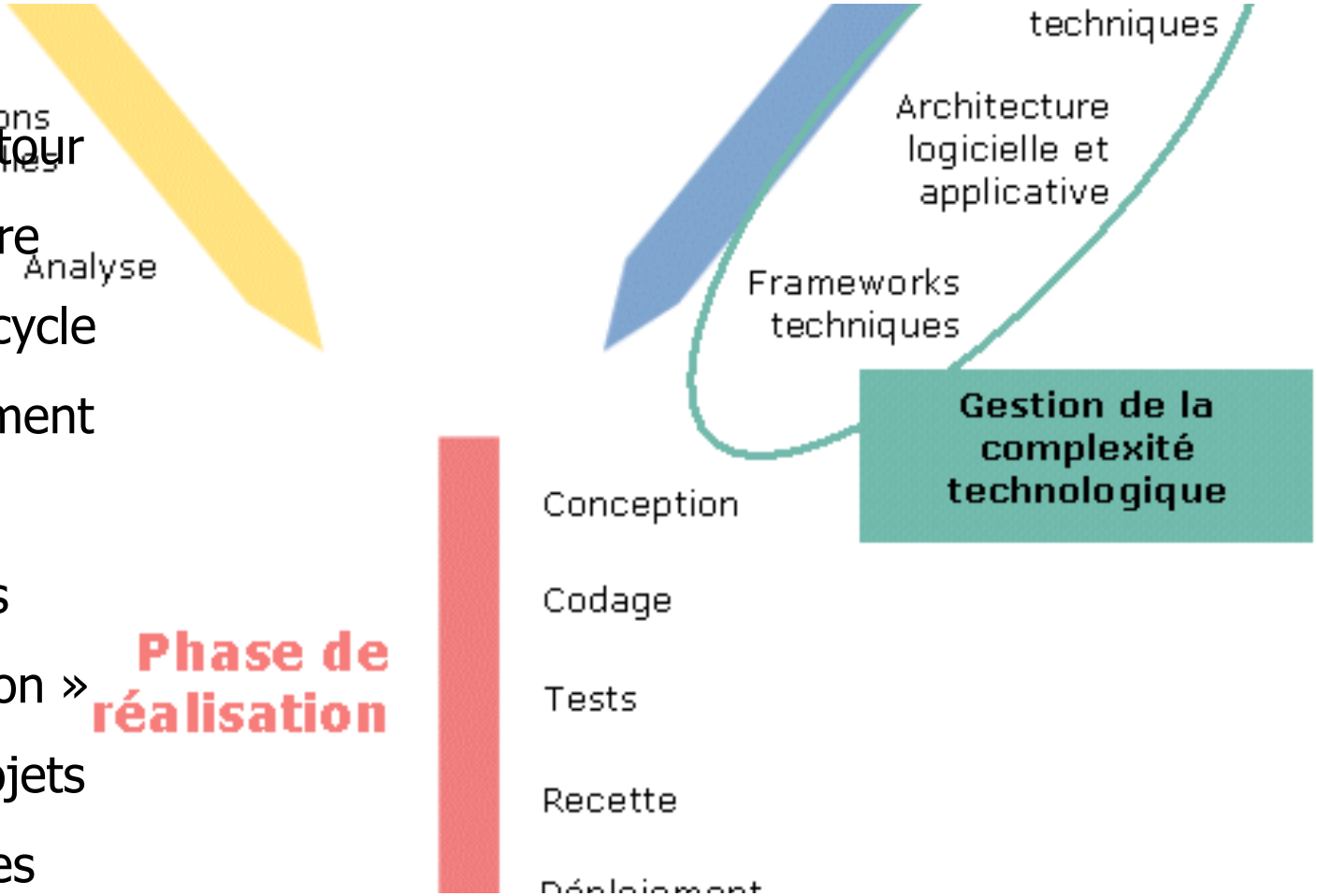
RUP : *Rational Unified Process*



- Promu par Rational
- Le RUP est à la fois une méthodologie et un outil prêt à l'emploi (documents types partagés dans un référentiel Web)
- plutôt pour des projets de plus de 10 personnes

2TUP : *Two Track Unified Process*

- S'articule autour de l'architecture
- Propose un cycle de développement en Y
- Détaillé dans « UML en action »
- pour des projets de toutes tailles

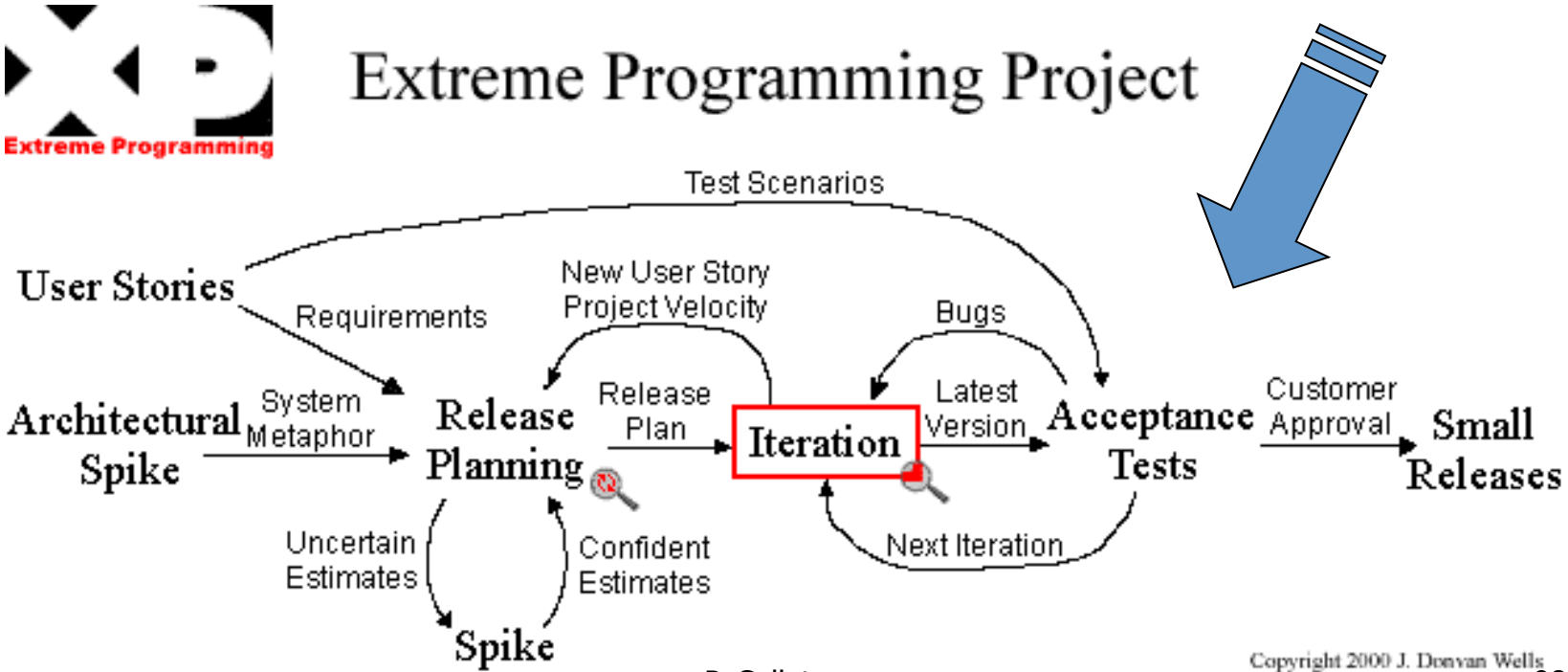


eXtreme Programming (XP…)

- Ensemble de « Bests Practices » de développement
  (travail en équipes, transfert de compétences…)
- plutôt pour des projets de moins de 10 personnes

4 Valeurs

- **Communication**
- **Simplicité**
- **Feedback**
- **Courage**



XP => Développement Agile

- Collaboration étroite entre équipe(s) de programmation et experts métier
  - Communication orale, pas écrite
  - Livraison fréquente de fonctionnalités déployables et utilisables (= qui apportent une valeur ajoutée)
  - Equipe auto-organisée et soudée
- *Test-Driven Development*
  - Ecrire les tests avant le code…

Manifeste Agile : 12 principes

1. Our highest priority is to satisfy the costumer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile process harness change for the customer´s competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face to face conversation.

Manifeste Agile : 12 principes

7. Working software is the primary measure of progress
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
9. Continuous attention to technical excellence and good design enhances agility
10. Simplicity – the art of maximizing the amount of work not done – is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Scrum : principes

- Isolement de l'équipe de développement
  - l'équipe est isolée de toute influence extérieure qui pourrait lui nuire. Seules l'information et les tâches reliées au projet lui parviennent : pas d'évolution des besoins dans chaque sprint.
- Développement progressif
  - afin de forcer l'équipe à progresser, elle doit livrer une solution tous les 30 jours. Durant cette période de développement l'équipe se doit de livrer une série de fonctionnalités qui devront être opérationnelles à la fin des 30 jours.
- Pouvoir à l'équipe
  - l'équipe reçoit les pleins pouvoirs pour réaliser les fonctionnalités. C'est elle qui détient la responsabilité de décider comment atteindre ses objectifs. Sa seule contrainte est de livrer une solution qui convienne au client dans un délai de 30 jours.
- Contrôle du travail
  - le travail est contrôlé quotidiennement pour savoir si tout va bien pour les membres de l'équipe et à la fin des 30 jours de développement pour savoir si la solution répond au besoin du client.

Scrum : rôles et pratiques

- Scrum Master
  - expert de l'application de Scrum
- Product owner
  - responsable officiel du projet
- Scrum Team
  - équipe projet.
- Customer
  - participe aux réunions liées aux fonctionnalités
- Management
  - prend les décisions

- Product Backlog
  - état courant des tâches à accomplir
- Effort Estimation
  - permanente, sur les entrées du backlog
- Sprint
  - itération de 30 jours
- Sprint Planning Meeting
  - réunion de décision des objectifs du prochain sprint et de la manière de les implémenter
- Sprint Backlog
  - Product Backlog limité au sprint en cours
- Daily Scrum meeting
  - ce qui a été fait, ce qui reste à faire, les problèmes
- Sprint Review Meeting
  - présentation des résultats du sprint

Comparaison des 3 processus dans le vent

	Points forts	Points faibles
RUP	▪ Itératif ▪ Spécifie le dialogue entre les différents intervenants du projet : les livrables, les plannings, les prototypes… ▪ Propose des modèles de documents, et des canevas pour des projets types	▪ Coûteux à personnaliser ▪ Très axé processus, au détriment du développement : peu de place pour le code et la technologie
XP	▪ Itératif ▪ Simple à mettre en œuvre ▪ Fait une large place aux aspects techniques : prototypes, règles de développement, tests… ▪ Innovant: programmation en duo…	▪ Ne couvre pas les phases en amont et en aval au développement : capture des besoins, support, maintenance, tests d'intégration… ▪ Élude la phase d'analyse, si bien qu'on peut dépenser son énergie à faire et défaire ▪ Assez flou dans sa mise en œuvre: quels intervenants, quels livrables ?
2TUP	▪ Itératif ▪ Fait une large place à la technologie et à la gestion du risque ▪ Définit les profils des intervenants, les livrables, les plannings, les prototypes	▪ Plutôt superficiel sur les phases situées en amont et en aval du développement : capture des besoins, support, maintenance, gestion du changement… ▪ Ne propose pas de documents types

Les différents types de projet

Durée	Personnes	Budget	Approche
< à 1 an	1	< 100 K€	Documentation a posteriori Validation par le développeur Vie limitée
Env. 1 an	1 à 5	< 300 à 500 K€	Plusieurs phases (dont conception) Planning, réunions d'avancement Contrôle qualité interne et gestion de versions Prototypage
1 à 2 ans	6 à 15	< 5 M€	Etudes préliminaires et cycle en spirale Documents de suivi et d'anomalie, inspections Gestion de configurations Plans de validation et d'intégration
2 ans et plus	16 et plus	> 5 M€	Procédures de communication Recettes intermédiaires Contrôle qualité permanent Gestion des sous-projets et de la sous-traitance Tests de non-régression Effort de synthèse et base historique

Université Nice SOPHIA ANTIPOLIS

Conduite de Projet

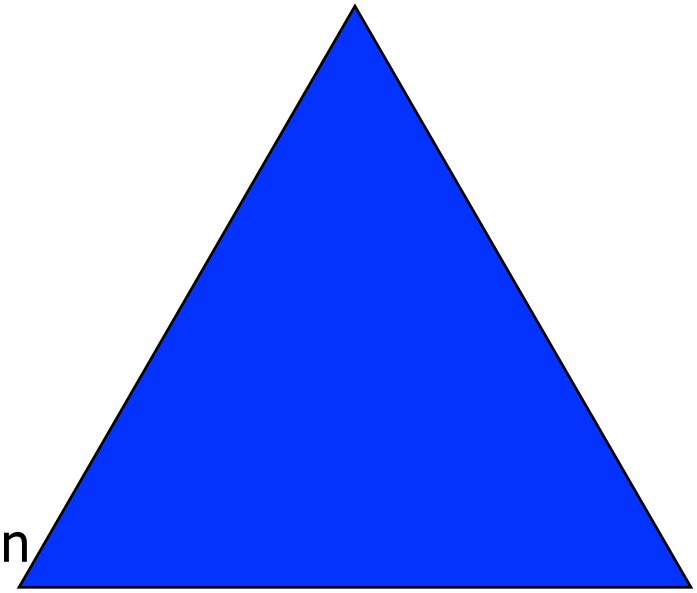
- Gestion d'un projet
- Ce qu'il ne faut pas faire
- Planification : PERT et Gantt
- Estimation des coûts, métriques
- Organisation du travail

Les 3 P

- formation
- compétences
- communication

Personnes

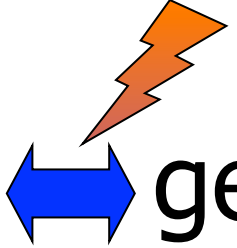
- planification
- coordination
- gestion
- mesures
- analyse
- conception
- implémentation



Processus

Produits

- cahier des charges
- conception
- code source
- exécutable
- documentation utilisateur
- cas de test
- résultats des tests
- demande de changement

Génie logiciel  gestion de projet

- Beaucoup de problèmes de développement logiciel sont des problèmes de gestion (*management*)
  - Estimation des coûts
  - Estimation des durées, des délais
  - Ordonnancement des tâches
  - Gestion des changements
  - Contrôle des versions et gestion des configurations

Objectifs et décomposition

- Gestion de projet =

  planification, organisation, gestion des tâches et des ressources pour accomplir un but défini
- Quoi, qui, quand, combien
- Comment ?
- Les différentes phases de la conduite d'un projet :
  - Planification du projet
  - Évaluation et ordonnancement des tâches
  - Contrôle et analyse de l'avancement
  - Communication des informations relatives au projet

Les tâches de gestion

- Modélisation des tâches
- Ordonnancement
- Gestion des ressources
- Gestion du risque
- Gestion des changements
- Gestion des configurations
- Gestion de la qualité

Planification

Planification des tâches

- Définir les activités constituant le projet
- Détecter les **jalons** (*milestones*) du projet
  - événements significatifs dans le projet
- Évaluer les dépendances entre activités
- Ordonnancer les activités en conséquence
- Évaluer l'effort nécessaire pour chaque activité
  - durée minimum et maximum
- Affecter les ressources nécessaires aux tâches
- S'assurer de la bonne répartition des ressources

Suivi de la planification

- Réaliser des réunions d'avancement du projet de façon périodique
- Évaluer les résultats de toutes les revues
- Déterminer si les jalons du projet ont été atteints
- Comparer les dates de fin réelles et prévues
- Discuter avec les gens (!)

Gestion des risques

- Les risques se planifient comme le reste
- Planification des risques
  - Identifier
  - Catégoriser
  - Résoudre

- Pour catégoriser, on peut faire une Risk Breakdown Structure
- Exemple de décomposition au premier niveau
  - Finance, gestion du projet, technique, humain, politique, naturel, opérationnel, réputation…

Identification des risques

- Le plus tôt est le mieux
  - Influence coût et organisation
  - Certains risques demandent des actions immédiates
- Mais l'identification continue tout au long du projet car :
  - Certains risques n'apparaissent qu'en exécutant le projet
  - Des changements sur un projet fixé entraînent des risques
  - Des changements externes peuvent créer des risques
  - Des actions « plan B » peuvent générer de nouveaux risques
- Comment les identifier ?
  - Dès l'analyse des besoins : hypothèses, dépendances, contraintes, limites et interfaces génèrent toutes des risques
  - Lors de la planification : estimation mal effectuée ou peu précise, absence de marge, coordination mal établie, etc.

Analyser les risques

- Analyse SWOT (Humphrey)
  - Fixer un objectif précis du projet
  - Travailler en groupe varié (brainstorming) pour déterminer les facteurs d'impact dans chaque case du tableau
- Force (strength)
  - Élément positif, interne, qui va aider à atteindre l'objectif
- Faiblesse (weakness)
  - Frein interne au projet
- Opportunité
  - Elément positif externe
- Menace (threat)
  - Elément négatif externe

Enregistrer les risques

- Risque
  - Nom, catégorie, personne en charge
  - Raison profonde, impact, probabilité, symptomes
  - Réponse possible, plan de secours
- Analyse globale des risques
  - Plein de méthodes statistiques possibles (analyse de monte-carlo, diagramme tornado, arbre de décision, etc.)
  - Surtout adaptée pour la planification de production
- Le plan de réponse
  - Prend les risques priorisés
  - Se focalise sur les risques à haute priorité
  - Par analyse SWOT, il tente de minimiser les aspects négatifs, et de faire survenir les aspects positifs

Réponses

- Réponses possibles aux risques négatifs (menaces)
  - Evitement : restructuration de la portée, de la planification
  - Atténuation : réduire la probabilité ou l'impact (choix alternatifs)
  - Transferts : passer par un sous-traitant qui va prendre le risque à sa charge
- Réponses possibles aux risques positifs (opportunités)
  - Exploitation : assurer l'occurrence du *risque*
  - Augmentation : de la probabilité ou de l'impact (choix alternatifs)
  - Partage : avec un sous-traitant ou un tiers intéressé aussi par le *risque*
- Réponses aux deux
  - Acceptation : plan de repli pour impondérable ou coût trop élevé de gestion
  - Plan B : alternative mise en place, avec événements de déclenchement, et d'arrêt du plan, à utiliser en conjonction avec l'atténuation

Pourquoi les projets sont-ils toujours en retard ?

- Dates limites irréalistes, imposées par des éléments externes
- Changements de besoin non répercutés dans la planification
- Sous-estimation de l'effort nécessaire
- Risques mal ou non considérés
- Manque de communication entre les membres de l'équipe
- Les gestionnaires ne se rendent pas compte que le projet est en retard par rapport au planning

Que peut-on faire contre les limites irréalistes ?

- Vous ne pouvez pas les modifier
- Vous ne pouvez pas refuser de faire le travail

☞ Réaliser des estimations détaillées

☞ Essayer d'utiliser des modèles incrémentaux

- ☞ Définir les fonctionnalités critiques
- ☞ Reporter les autres fonctionnalités à des phases ultérieures

☞ Expliquer au client pourquoi vous ne pouvez pas respecter la date limite (en utilisant les estimations basées sur les performances de projets passés)

Les plus mauvaises approches

- Rassemblement de vantards
  - Décisions technologiques influencées par d'éminentes personnes, des magazines, etc.
- Mort par planification intensive
  - Une planification excessive entraîne des plannings complexes qui vont causer des problèmes en aval
  - "On ne peut pas commencer tant qu'on n'a pas un plan d'implémentation complet"

Les plus mauvaises approches

- Paralysie analysatoire
  - La recherche de la perfection et de la complétude dans les phases d'analyse entraîne un ralentissement du projet
  - "On doit refaire cette analyse pour la rendre plus orientée objet, et utiliser beaucoup plus l'héritage pour obtenir beaucoup de réutilisation."
  - Il n'existe pas de méthode évidente pour identifier le niveau de détail exact nécessaire à la conception d'un système informatique

Les plus mauvaises approches

- Conflits permanents
  - Les gens *difficiles* ralentissent et font diverger le processus de développement logiciel
  - "Pourquoi est-il si difficile de travailler avec Maurice ?"
- Violence intellectuelle
  - Utilisation de la connaissance pour intimider d'autres personnes lors des réunions

Les plus mauvaises approches

- Gestion irritante
  - Indécision permanente
  - "Bon, et qu'est qu'on fait maintenant ?"
  - "Il faudrait régler ça avec les gens du management avant de commencer."
- Power to salesmen !
  - L'équipe de direction s'engage sur des délais au delà des capacités de l'organisation

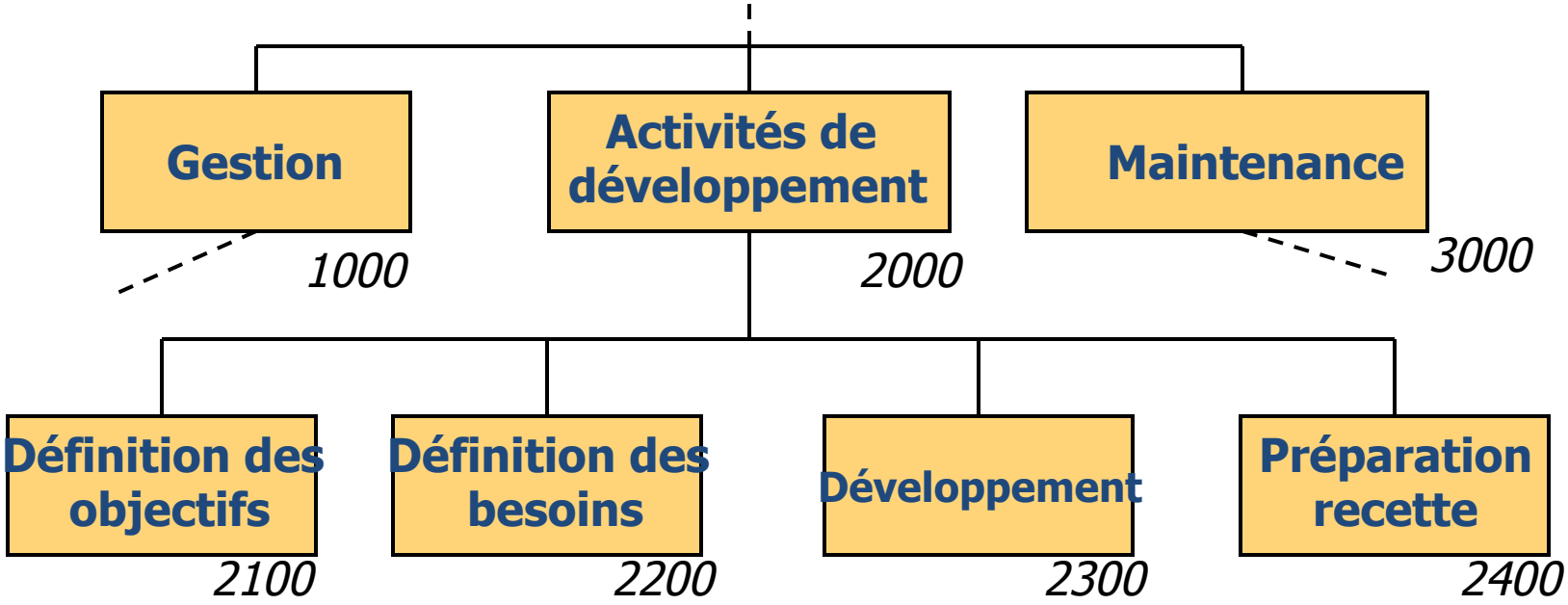
Problèmes de gestion

- Mauvaise gestion
  - Pas de direction à cause d'une minimisation ou d'un oubli des activités clés et des risques
  - "Que s'est-il passé ? Tout allait bien et puis tout d'un coup… BOOM !"
- Un petit peu de Freud
  - Conflits de personnalité au sein de la direction, entre les chefs de projet, etc.
- Les e-mails sont dangereux
  - Ils ne remplacent jamais les réunions

Décomposition structurée des activités

- WBS : *Work Breakdown Structure*
- Décomposition sous forme arborescente
  - purement statique (pas d 'ordonnancement)
- Décomposer jusqu'à obtenir des activités bien définies et faciles à gérer
  - entrées et résultats parfaitement identifiés
  - responsabilité confiée à des personnes précises
- Identification rapide des activités critiques
- Identification des besoins de sous-traitance

Exemple de WBS



- Décomposition optimale lorsque :
  - la durée d'une activité est maîtrisée
  - la connaissance des ressources requises est totale
  - le coût de l'activité est évaluable

Caractéristiques de la WBS

- Elle permet au chef de projet d'établir le graphe PERT et de faire un suivi budgétaire
  - doit être complète, pour élaborer un graphe PERT correct
  - doit être non ambiguë, pour budgéter correctement le projet et contrôler les coûts par la suite
  - les résultats des activités doivent être mesurables pour évaluer l'avancement général
- Certaines activités sont toujours présentes :
  - élaboration des documents, inspections...
  - construction d'outils, apprentissage...

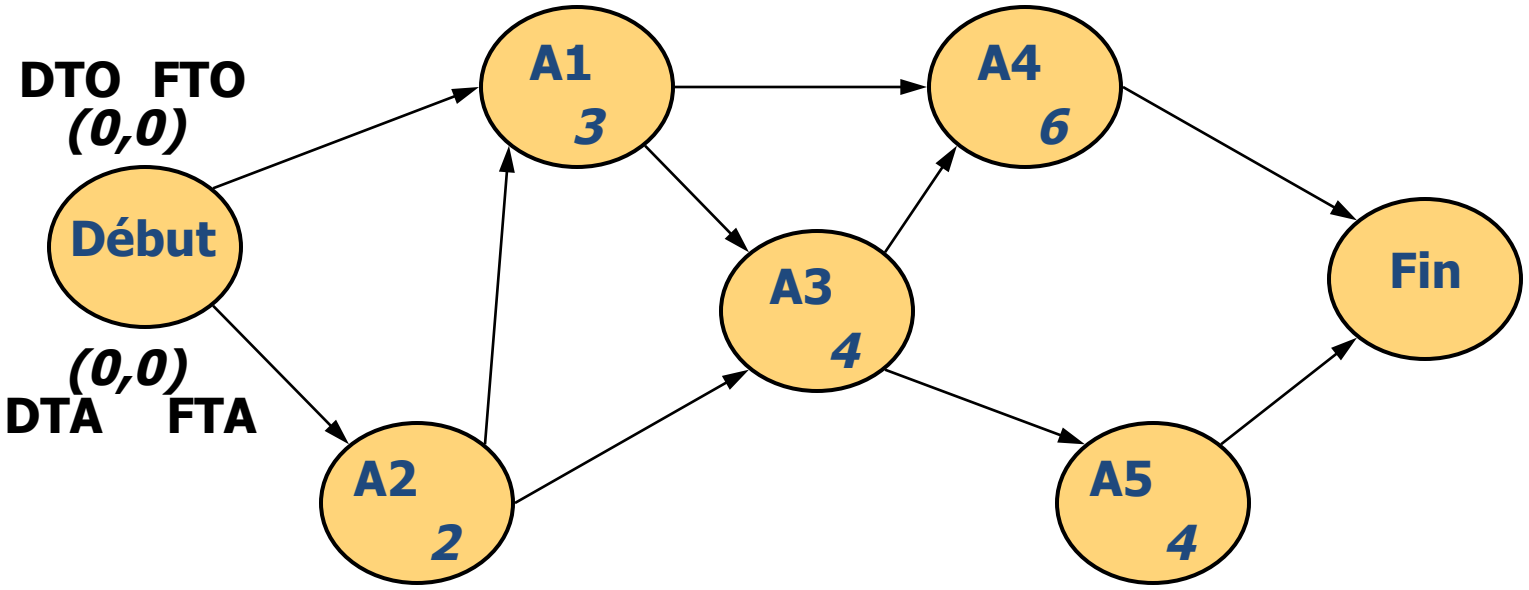
Graphe PERT

- PERT: *Program Evaluation and Review Technique*
- Graphe de dépendances, pour l'ordonnancement
  - Pour chaque tâche, on indique une date de début et de fin, au plus tôt et au plus tard
  - Le diagramme permet de déterminer **le chemin critique** qui conditionne **la durée minimale du projet**

☞ Techniques fortement appliquées en BTP

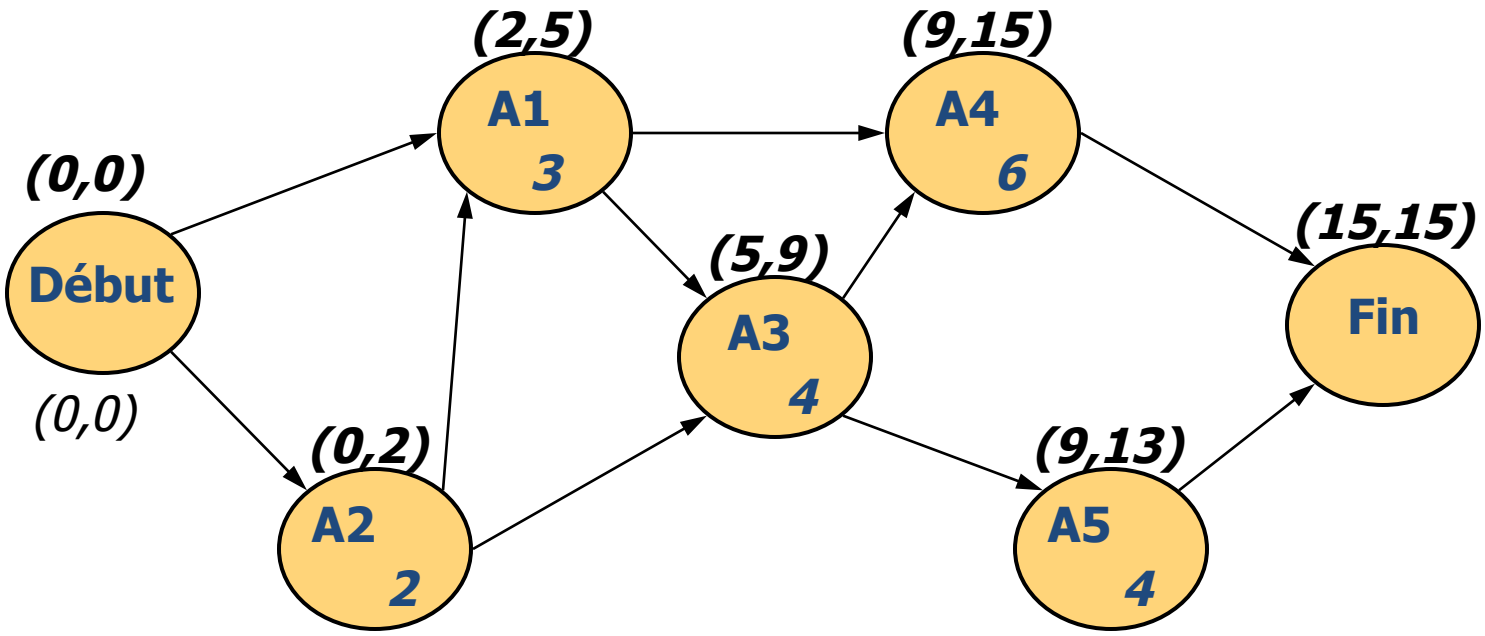
☞ Projets à plusieurs équipes => PERT à plusieurs niveaux

Graphe PERT-flèche : exemple



- Estimation de la durée des tâches : ni optimiste, ni pessimiste
  - ☞ DTO : date de début au plus tôt
  - ☞ FTO : date de fin au plus tôt
  - ☞ DTA : date de début au plus tard
  - ☞ FTA : date de fin au plus tard

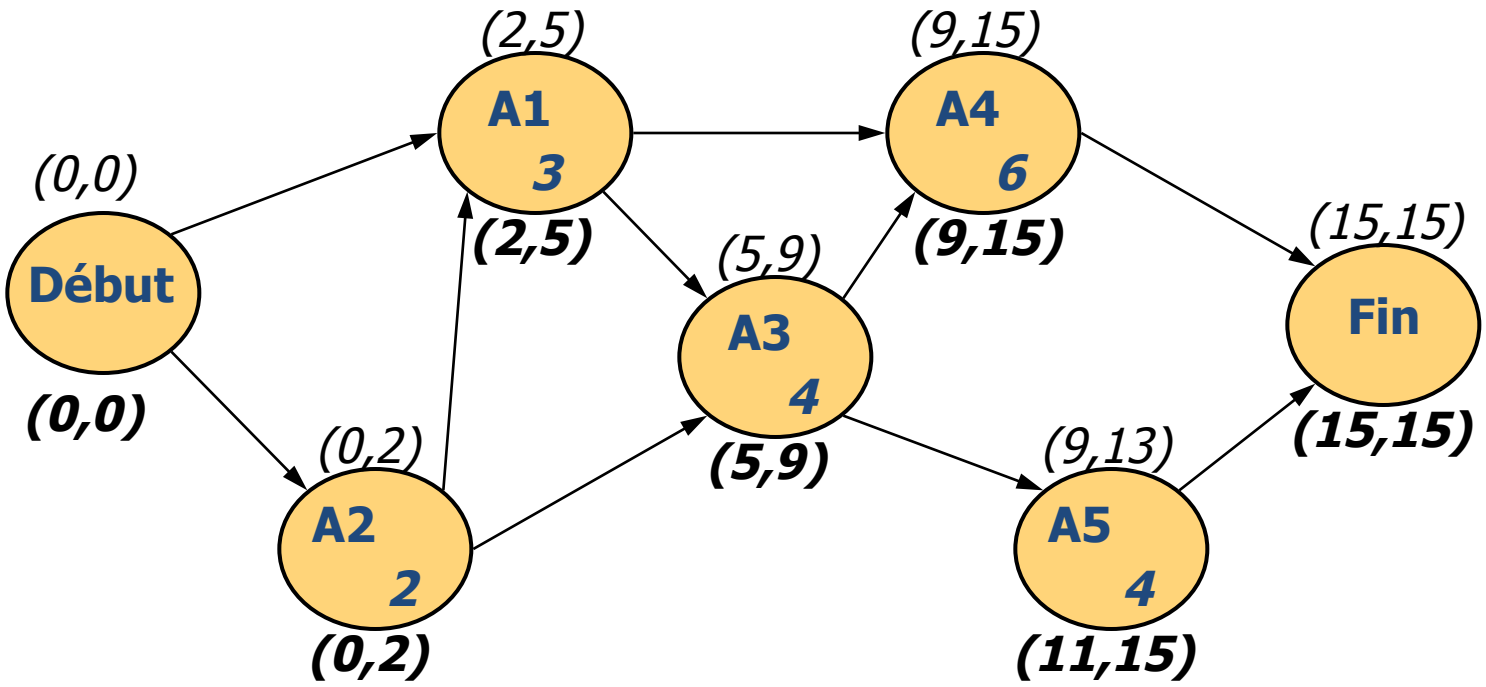
PERT : calcul des dates au plus tôt



- Partant du début, calcul « aller » de la gauche vers la droite :
  - pour une tâche, la durée de début au plus tôt est égale à la plus grande des dates de fin au plus tôt des tâches qui la précèdent
  - FTO = DTO + durée

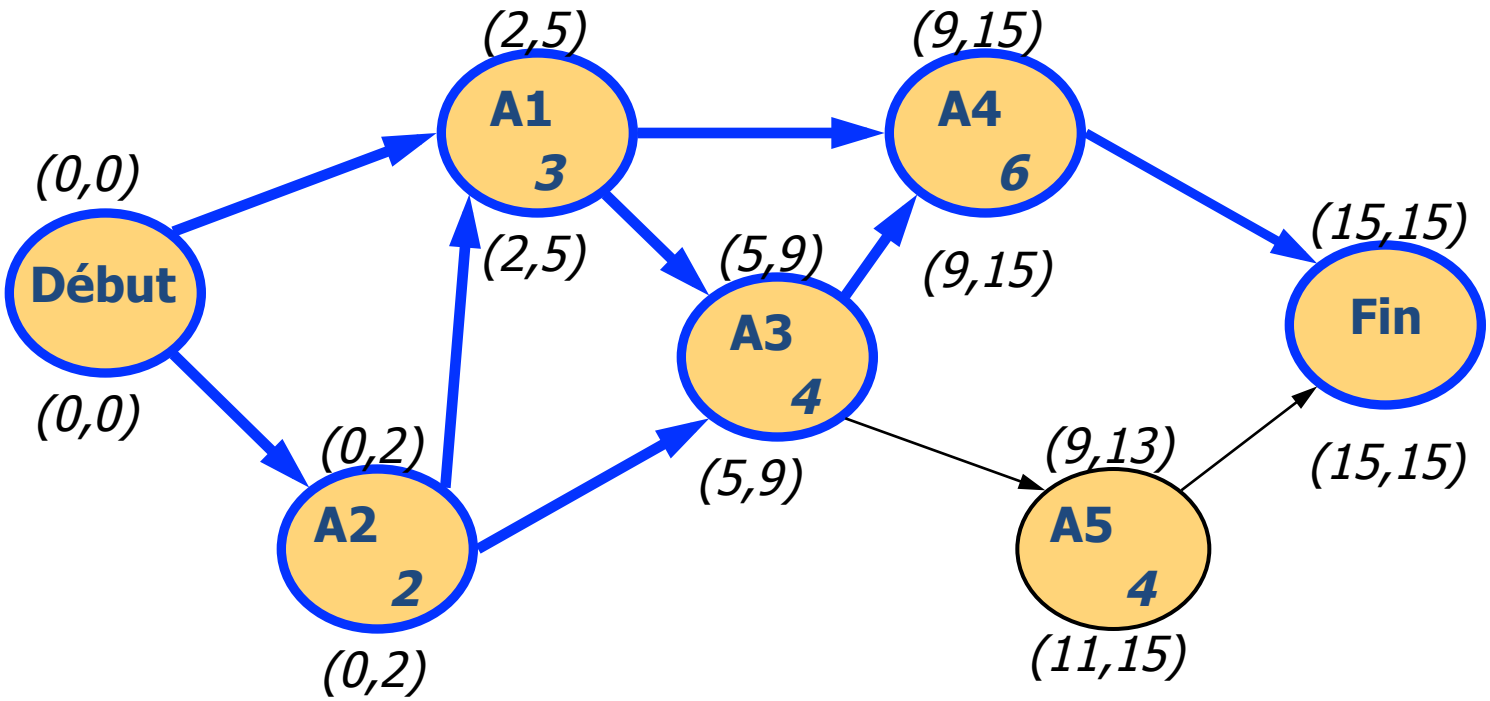
☞ **Délai de réalisation du projet**

PERT : calcul des dates au plus tard



- Partant de la fin début, calcul « retour » en sens inverse :
  - ☞ pour une tâche, la durée de fin au plus tard est égale à la plus petite des dates de début au plus tard des tâches qui lui succèdent
  - ☞ DTA = FTA - durée

PERT : marges et chemin critique



- Durée maximum d'une tâche = FTA -DTO
- Marge totale d'une tâche = FTA - FTO

☞ Une tâche est critique si sa durée est égale à sa durée maximum

☞ Le chemin critique est le plus long, où toutes les tâches sont critiques

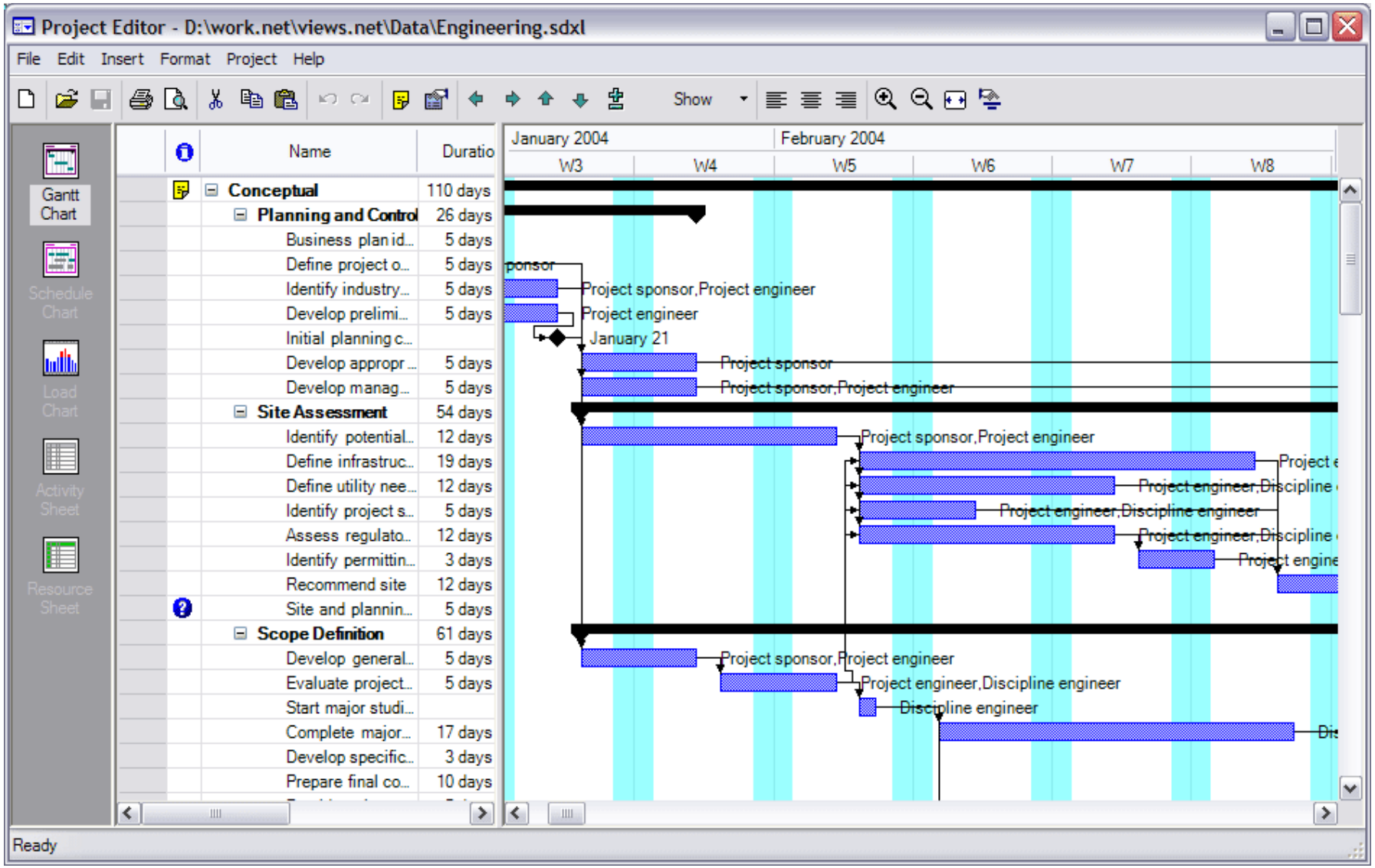
Diagramme de Gantt

- Son but est de faire apparaître
  - la répartition des activités dans le temps,
  - l'affectation des individus.
- Il donne une description détaillée
  - des coûts (en hommes*mois),
  - des dates pour chaque tâche et pour chaque phase.
- A chaque tâche sont attribués
  - un objectif pour repérer la terminaison de l'activité
  - une durée pour atteindre cet objectif
  - des ressources nécessaires à son accomplissement

Diagramme de Gantt (suite)

- Il faut d'abord estimer les durées et les ressources
- Pour harmoniser le diagramme de Gantt, il faut utiliser la même unité de temps
- Les ressources peuvent être humaines ou matérielles
- Après avoir ordonnées les tâches à l'aide d'un PERT :
  - en abscisse, l'échelle des temps
  - en ordonnée, la liste des tâches
  - des rectangles sont tracés proportionnellement à la durée de la tâche, avec l 'affectation des ressources nécessaires

Gantt : exemple



Évaluation des coûts et durées

- Analogie avec des projets déjà achevés
- Expertises et retours sur expérience
- Décomposition du projet pour estimation

☞ Modélisation du processus *(Cf. CMM plus loin)*

☞ Effectuer des mesures  *Métriques du logiciel*

  – utilisées pour *prédire* les besoins du projet (personnel, effort total, …)

Rôles des mesures

- Estimation
  - Déterminer les besoins vraisemblables en ressource
- Prédiction
  - Déterminer les valeurs vraisemblables des mesures
- Évaluation
  - Comparer les mesures aux valeurs prédéterminées
- Comparaison
  - Prendre des décisions pour des compromis
- Investigation
  - Soutenir ou réfuter les hypothèses

Problèmes des LOC (*Lines Of Code*)

- Le code n'est qu'une petite partie du développement
- Que compte-t-on effectivement ?
  - Commentaires, lignes vides, code non livré
- Dépendances fortes vis-à-vis des langages, des applications, des développeurs...
- La complexité du code n'est pas exprimée
- Cela encourage de gros volumes de code
- Cela ne prédit ni la qualité, ni l'avancement
  - Le comptage est forcément effectué a posteriori

Estimation des coûts : COCOMO

- *COnstructive COst MOdel* (Boehm, 1981)
- Coût en nombre d'hommes*mois (MM)
- Temps de développement (TDEV)
- Modèle de régression
  - basé sur un historique de projets logiciels
  - avec analyse des données par régression
  - et relation mathématiques entre les variables
- Fonction de la prévision du nombre de milliers d'instructions sources livrées (KDSI)

COCOMO 81

- COCOMO 81 : projets traditionnels
  - réalisé à partir d ’une étude sur 63 projets de 2000 à 100000 LOC, dans l ’entreprise TWR

- COCOMO dispose de trois niveaux de modèles :
  - Modèle de base (ou simplifié)
  - Modèle intermédiaire
  - Modèle détaillé

COCOMO : modèle de base

- Estimation de l 'effort (MM) en fonction des LOC et d 'un facteur d 'échelle qui dépend du projet
- 3 types de projet sont identifiés :
  - *organique* : innovation minimale, organisation simple et petites équipes expérimentées (ex : petite gestion…)
  - *médian (semi-detached)* : degré d 'innovation raisonnable (ex: banque, compilateurs…)
  - *imbriqué (embedded)* : innovation importante, organisation complexe, couplage fort et nombreuses interactions (ex : gros systèmes, avioniques...)

COCOMO : des formules

Projet	MM	TDEV
Organique	$2.4(KDSI)^{1.05}$	$2.5(MM)^{0.38}$
Médian	$3.0(KDSI)^{1.12}$	$2.5(MM)^{0.35}$
Imbriqué	$3.6(KDSI)^{1.20}$	$2.5(MM)^{0.32}$

Exemples d'application des formules

- Projet organique de 2 KDSI :
  - effort de 5 hommes*mois, sur 4.6 mois
- Projet médian de 32 KDSI :
  - effort de 146 hommes*mois, sur 14 mois
- Projet imbriqué de 512 KDSI :
  - effort de 6420 hommes*mois, sur 41 mois

Formules déduites

- Productivité (KDSI/MM):
  - organique et 2 KDSI donne 400
  - imbriqué et 512 KDSI donne 80
- Nombre moyen de personnes (MM/TDEV):
  - organique et 2 KDSI donne 1.1
  - imbriqué et 512 KDSI donne 157

Les hypothèses

- Le KDSI «livré» exclut en général les environnements de tests, les supports de développement...
- Une instruction source exclut les commentaires, mais inclut le *shell*
- Hommes*mois (MM) correspond à 152 heures (normes américaines !) et tient compte des vacances, arrêts maladie...
  - En fait, c'est même trop avec les 35 heures, mais…
- TDEV correspond au temps entre spécifications fonctionnelles et intégration

La distribution par phases

- Prise en compte de la distribution de l'effort et du temps par phase (en %)
  - RPD: *Requirements and Preliminary Design*
  - DD: *Detailed Design*
  - CUT: *Code and Unit Test*
  - IT: *Integration and Test*

☞ des tableaux, encore des tableaux...

Exemple de distribution

- Projet organique et 2 KDSI
  - Effort: RPD : 16%, DD : 26 %, CUT : 42%, IT : 16%
  - Temps: RPD : 19%, DD et CUT : 63 % ,IT : 18%

- Projet imbriqué et 512 KDSI
  - Effort: RPD : 18%, DD : 24 %, CUT : 24%, IT : 34%
  - Temps: RPD : 38%, DD et CUT : 32 % , IT : 30%

Limites du modèle de base

- Fondé, à la base, sur les statistiques d'une seule entreprise
- Ne prends en compte que le nombre de lignes source
  - alors que plus les programmeurs sont experts, moins ils écrivent de code…
- induit des discontinuités brutales entre chaque phase (nombre de personnes)

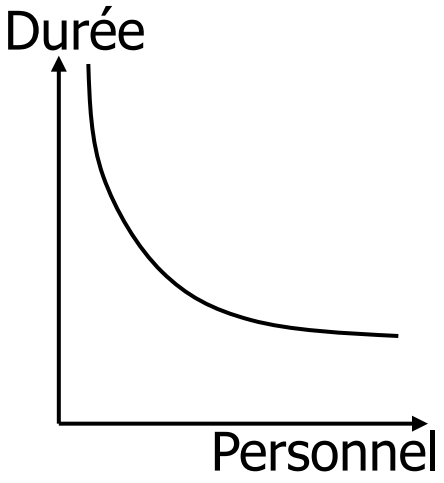
COCOMO 2

- COCOMO 2 (1998)
  - pour les projets basés sur la réutilisation de composants
  - possibilité de recalibrage sur les données de l'entreprise
- 3 modèles :
  - *modèle de composition d 'application* (*GUI builder*)
  - *modèle avant projet* pour obtenir une estimation à base de facteurs de coûts et des LOC
  - *modèle post-architecture*, à utiliser après le développement de l 'architecture générale

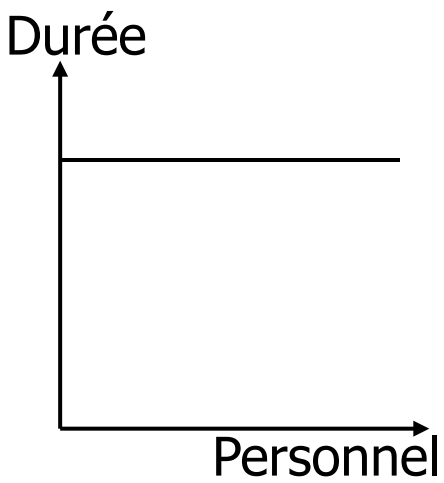
Organisation du travail

- En fonction de la nature des tâches, le nombre de personnes et leur communication influent différemment sur la durée de réalisation

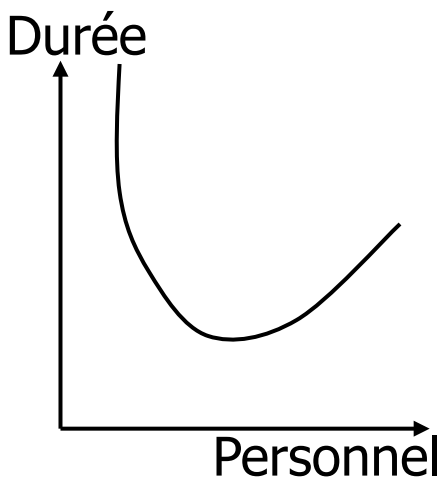
bonne partition



non partitionable



communication complexe



Nécessité de la structuration

- Les équipes doivent être structurées pour diminuer le temps passé à communiquer
- La communication
  - améliore la compréhension du sujet
  - permet une plus grande mobilité dans le projet
- mais
  - fait perdre du temps
  - peut nuire à la documentation, car les besoins de communication externe sont plus faibles

Comment s'organiser

- **Petit groupe de travail sans autorité définie**
  - travail par consensus
  - le travail de chacun est celui de tous
  - Le travail enrichit toute l 'équipe

  ☞ *Le consensus est-il facile à trouver ?*

- **Structuration forte par le chef**
  - un chef de projet dirige de 2 à 5 ingénieurs
  - un adjoint peut le remplacer
  - un contrôleur gère programmes, configurations et documentation

  ☞ *Structure lourde*

Comment s'organiser (suite)

- **1 chef de projet pour plusieurs équipes**
  - Le chef, son adjoint et le contrôleur gèrent plusieurs projets
  - Les équipes travaillent en consensus interne pour les tâches quotidiennes
- **1 comité de direction pour plusieurs projets**
  - le comité est composé de chefs de projets qui gèrent directement plusieurs projets
  - Les chefs de projets travaillent par consensus
  - Ils peuvent se remplacer à tout moment

Attention aux facteurs humains

- Motivations individuelles *vs.* motivation collective
- Relations entre membres de l'équipe
- Relations avec l'extérieur (client, sous-traitant)
- Dynamique du chef de projet
- Formation permanente de l'équipe
- Les problèmes éventuels :
  - sur-spécialisation
  - dé-responsabilisation
  - trop ou pas assez de niveaux dans l'organisation

Université Nice SOPHIA ANTIPOLIS

Vérification et Validation

- Principes
- Approches statiques
- Approches dynamiques
- Intégration

Contrôler la qualité

- Contrôle de la qualité = Ensemble d'inspections, de revues et de tests ➡ pour trouver des erreurs, des défauts…
- Idées préconçues :
  - La qualité ne peut être évaluée que lorsque le code est disponible
  - La qualité ne peut être uniquement améliorée par la suppression d'erreurs dans le code
- Mais les produits intermédiaires sont *contrôlables*
  - Prototypes / maquettes
  - Documents de spécification, de conception
  - Code
  - Jeux de tests

Principes de V&V

- Deux aspects de la notion de qualité :
  - Conformité avec la définition : VALIDATION
    - Réponse à la question : faisons-nous le bon produit ?
    - Contrôle en cours de réalisation, le plus souvent avec le client
    - **Défauts** par rapport aux besoins que le produit doit satisfaire
  - Correction d'une phase ou de l'ensemble : VERIFICATION
    - Réponse à la question : faisons-nous le produit correctement ?
    - Tests
    - **Erreurs** par rapport aux définitions précises établies lors des phases antérieures de développement

Qualité et cycle de vie

- Les spécifications fonctionnelles définissent **les intentions**
  - Valider la conformité aux besoins
  - Définir le plan qualité
- A chaque vérification, on vérifie la conformité aux spécifications fonctionnelles par rapport aux **intentions**
- Lors de la phase de qualification, on valide le produit
  - par rapport aux besoins
  - par rapport aux performances requises

Terminologies

- Norme IEEE (*Software Engineering Terminology*)
  - Erreur : commise par le développeur, entraîne un défaut
  - Défaut : imperfection dans le logiciel, pouvant amener une panne
  - Panne : comportement anormal d'un logiciel
- Classification des faits techniques (qualification) :
  - Non conformité : Erreur par rapport au cahier des charges
  - Défaut : Erreur car le comportement du logiciel est différent d'un comportement normal dans son contexte
  - Évolution : Demande de changement sans prise de garantie

Problèmes

- Plus de 50 % des erreurs sont découvertes en phase d'exploitation
  - Le coût de réparation croit exponentiellement avec l'avancée dans le cycle de vie
  - ☞Contrôles tout au long du cycle de vie + Qualification
- Problèmes lors des contrôles :
  - prééminence du planning sur la qualité
  - sous-estimation des ressources
    - par les développeurs (activité inutile ?)
    - par les dirigeants (budgets séparés pour développement et maintenance !)

V & V : les moyens

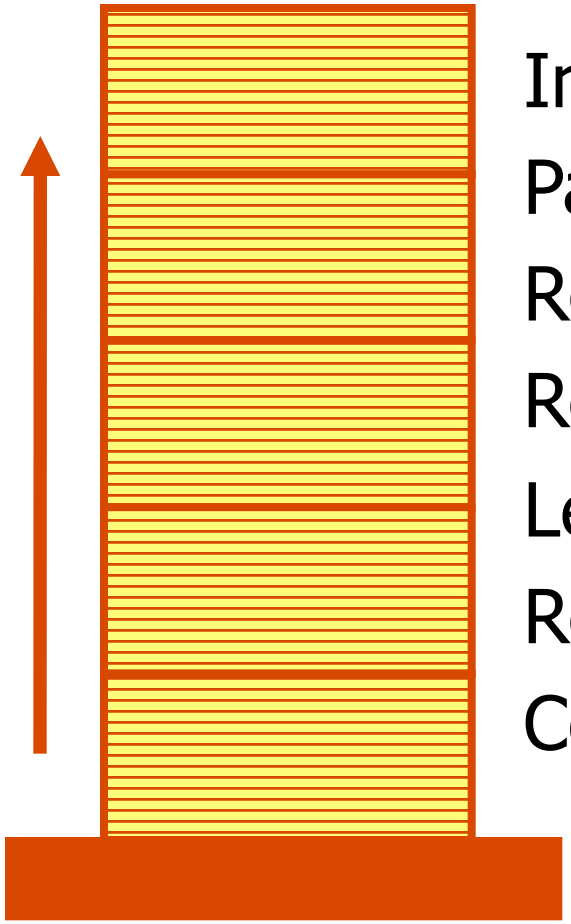
- Statiques :
  - Examen critique des documents : Inspections, revues
  - Analyse statique du code
  - Évaluation symbolique
  - Preuve
- Dynamiques :
  - Exécution du code : Tests
    - Comment les choisir ?
    - Quand arrêter de tester ?

Examen critique de documents

- Minimisation des problèmes d'interprétation
  - Point de vue indépendant du rédacteur
- Vérification
  - Forme : respect des normes, précision, non ambiguïté
  - Fond : cohérence et complétude
  - Testabilité et traçabilité
- Validation
  - Mauvaises interprétations des documents de référence
  - Critères de qualité mal appliqués
  - Hypothèses erronées
- Quelle méthode pour examiner les documents ?
  - Pouvoir de détection
  - Coût
    - 5 à 10p/h Cahier des charges
    - 20 à 50 LOC/h Code

Échelle d'efficacité des méthodes

Plus efficace



Aspects formels

Inspection

Parcours systématique

Revue structurée

Revue en groupe structuré

Lecture croisée

Relecture individuelle

Conversation normale

Relectures et revues

- Relecture individuelle → qualité faible
- Lecture croisée → qualité assez faible
- Revue en groupe structuré
  - Groupe de 10 pers. Max.
  - Lecture puis discussion → qualité moyenne
- Revue structurée
  - Liste séparée de défauts
  - *Check list* des défauts typiques → bonne qualité
- Revue en *Round Robin*
  - lecture préalable
  - attribution de rôles → qualité variable

Parcours et inspection

- Parcours systématique
  - le plus souvent du code
  - audit par des experts (extrêmement coûteux)

- Inspection
  - Préparation : recherche des défauts
  - 🕐 Cycle de réunions
  - 🕐 Suivi : vérification des corrections
  - ☞ Modérateur + secrétaire  meilleure qualité

Ça marche, les inspections ?

- [Fagan 1976] Inspections de la conception et du code
  - 67%-82% de toutes les fautes sont trouvées par des inspections
  - 25% de temps gagné sur les ressources / programmeur (malgré le temps passé dans les inspections)
- [Fagan 1986] nouvelle étude de Fagan
  - 93% de toutes les fautes sont trouvées par inspections
- Réduction de coût pour la détection de fautes (en comparaison avec les tests)
  - [Ackerman, Buchwald, Lewski 1989]: 85%
  - [Fowler 1986]: 90%
  - [Bush 1990]: 25000 $ gagné PAR inspection

Analyse statique du code

- Évaluation du code par des métriques
  - moins cher, mais résultat souvent approximatif
  - qualité des métriques ?
- Recherche d'anomalies dans le code
  - Références aux données (flots, initialisation, utilisation…)
  - Contrôle (graphe de contrôle, code isolé, boucles…)
  - Comparaisons
  - Respect des conventions de style

Méthodes dynamiques : les tests

> “ Testing is the process of executing a program with the intent of finding errors.”
>
> Glen Myers

Tester, c’est exécuter un programme avec l’intention de trouver des erreurs

Tests : définition...

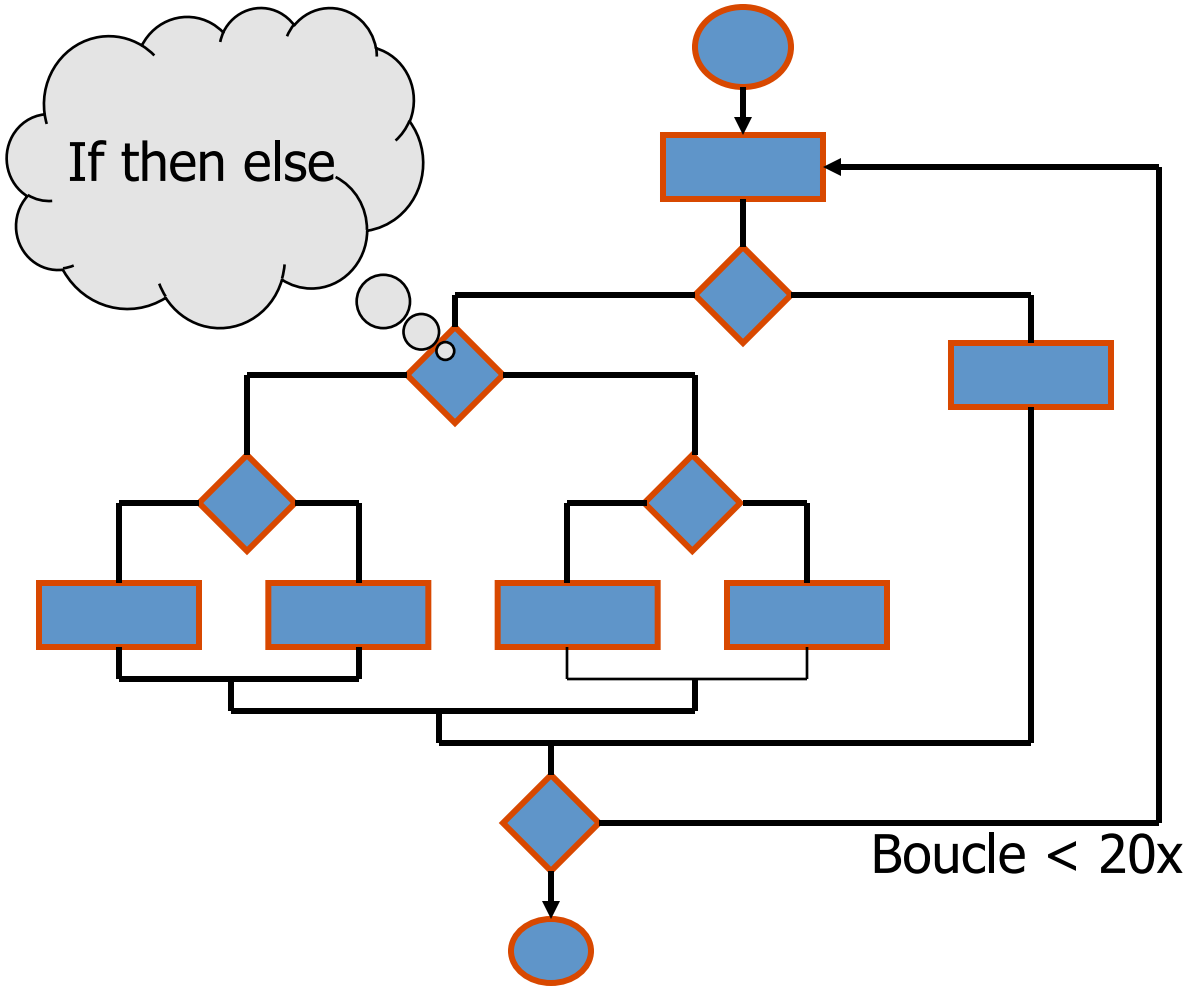
- Une expérience d'exécution, pour mettre en évidence un défaut ou une erreur
  - Diagnostic : quel est le problème
  - Besoin d'un **oracle**, qui indique si le résultat de l'expérience est conforme aux intentions
  - Localisation (si possible) : où est la cause du problème ?

☞ *Les tests doivent mettre en évidence des erreurs !*

☞ *On ne doit pas vouloir démontrer qu'un programme marche à l'aide de tests !*

- Souvent négligé car :
  - les chefs de projet n'investissent pas pour un résultat négatif
  - les développeurs ne considèrent pas les tests comme un processus destructeur

Tests exhaustifs ?



Il y a 5^{20} chemins possibles
En exécutant 1 test par milliseconde, cela prendrait **3024** ans pour tester ce programme !

Constituants d'un test

- Nom, objectif, commentaires, auteur
- Données : jeu de test
- Du code qui appelle des routines : cas de test
- Des oracles (vérifications de propriétés)
- Des traces, des résultats observables
- Un stockage de résultats : étalon
- Un compte-rendu, une synthèse…

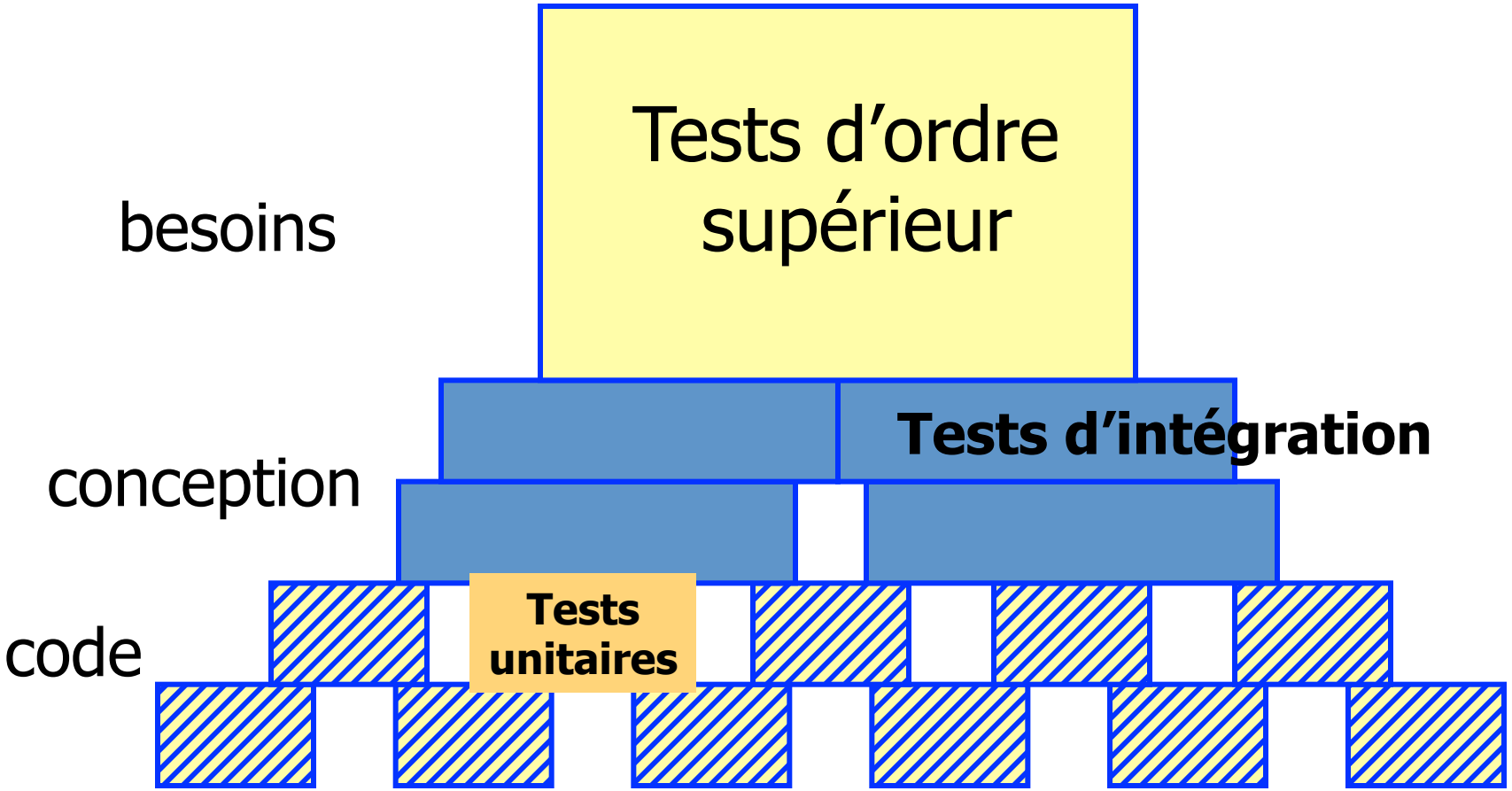
- Coût moyen : autant que le programme

Test vs. Essai vs. Débogage

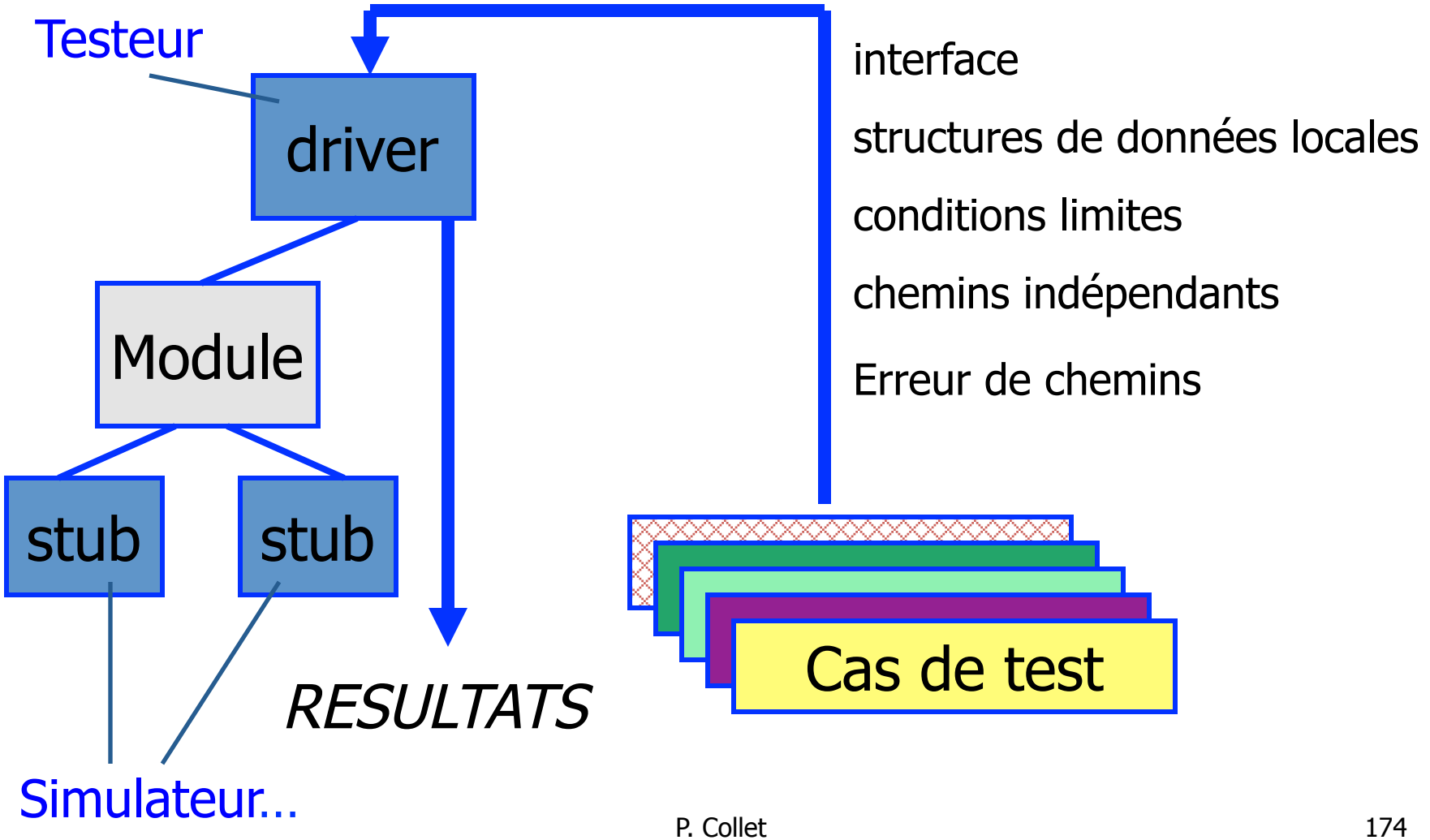
- On converse les données de test
  - Le coût du test est amorti
  - Car un test doit être **reproductible**
- Le test est différent d'un essai de mise au point

- Le débogage est une enquête
  - Difficilement reproductible
  - Qui cherche à expliquer un problème

Les stratégies de test



Test unitaire



Tests d'intégration

Si tous les modules marchent bien séparément, pourquoi douter qu'ils ne marcheraient pas ensemble ?

Réunir les modules : **Interfacer**

Big Bang

Stratégie de construction incrémentale

Intégration partielle

Test de non-régression

Tests de charge et de performance

- Charge :
  - Tests de vérification des contraintes de performance en **pleine charge** : avec les contraintes maximales
    - Mesure des temps d'exécution depuis l'extérieur
    - Vision de l'utilisateur face au système *chargé*
- Performance :
  - Analyse des performances du logiciel en charge **normale**
  - Profilage d'utilisation des ressources et du temps passé par instruction, bloc d'instructions ou appel de fonction
    - Instrumentation des programmes par des outils pour effectuer les comptages

Tests de validation et qualification

- Rédigés à partir des spécifications fonctionnelles et des contraintes non fonctionnelles
- Composition :
  - Préconditions du test
  - Mode opératoire
  - Résultat attendu
  - Structuration en ***Acceptation, refus et panne***
  - Résultat des passages
  - Fiche(s) d'anomalie liée(s)

Organiser l'activité de tests

- Qui teste le logiciel ?
  - Développeur : comprend bien le système mais, testera « gentiment » et est motivé par la *livraison*
  - Testeur indépendant : doit apprendre le système mais, essaiera de le casser et est motivé par la *qualité*
- Mettre en place les différents types de tests :
  - tests unitaires
  - tests d'intégration
  - tests de validation
  - tests de qualification
  - tests de suivi d'exploitation

Organiser l'activité de tests (suite)

- Les jeux de test sont des produits :
  - Spécification et développement des tests
  - Contraintes de **reproduction** des tests
  - Taille et coût minimum pour une probabilité de détection maximum
- Les tâches associées aux tests
  - Planification
  - Spécification et réalisation des jeux de tests
  - Passage des tests et évaluation des résultats

☞ *Commencer le plus tôt possible*

Jeux de test (= $\sum$ cas de test)

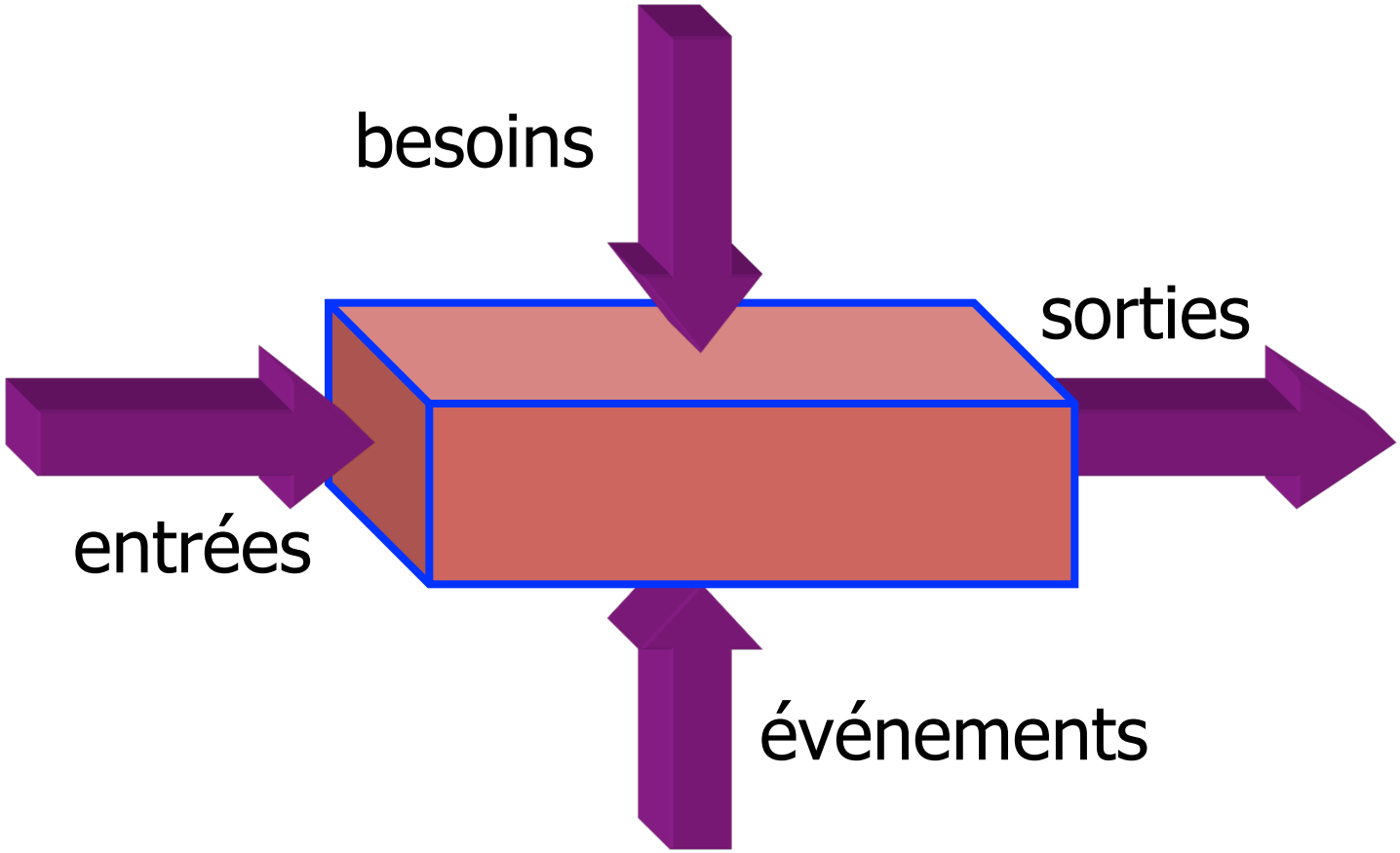
- Décrivent comment tester un système/module
- La description doit faire apparaître :
  - L'état du système avant l'exécution du test
  - La fonction à tester
  - La valeur des paramètres pour le test
  - Les résultats et sorties attendus pour le test
- Objectif ⇒ *Découvrir des erreurs*
- Critère ⇒ *de manière complète*
- Contrainte ⇒ *avec un minimum d'effort et dans un minimum de temps*

Cas de test : exemples

- État du système avant exécution du test
  - *ResourcePool* est non vide
- Fonction à tester
  - *removeEngineer(anEngineer)*
- Valeurs des paramètres pour le test
  - *anEngineer* est dans *ResourcePool*
- Résultat attendu du test
  - ResourcePool = ResourcePool \ anEngineer

- État du système avant exécution du test
  - *ResourcePool* est non vide
- Fonction à tester
  - *removeEngineer(anEngineer)*
- Valeurs des paramètres pour le test
  - *anEngineer* **N** 'est **PAS** dans *ResourcePool*
- Résultat attendu du test
  - EngineerNotFoundException est levée

Test en boîte noire



Tests fonctionnels en boîte noire

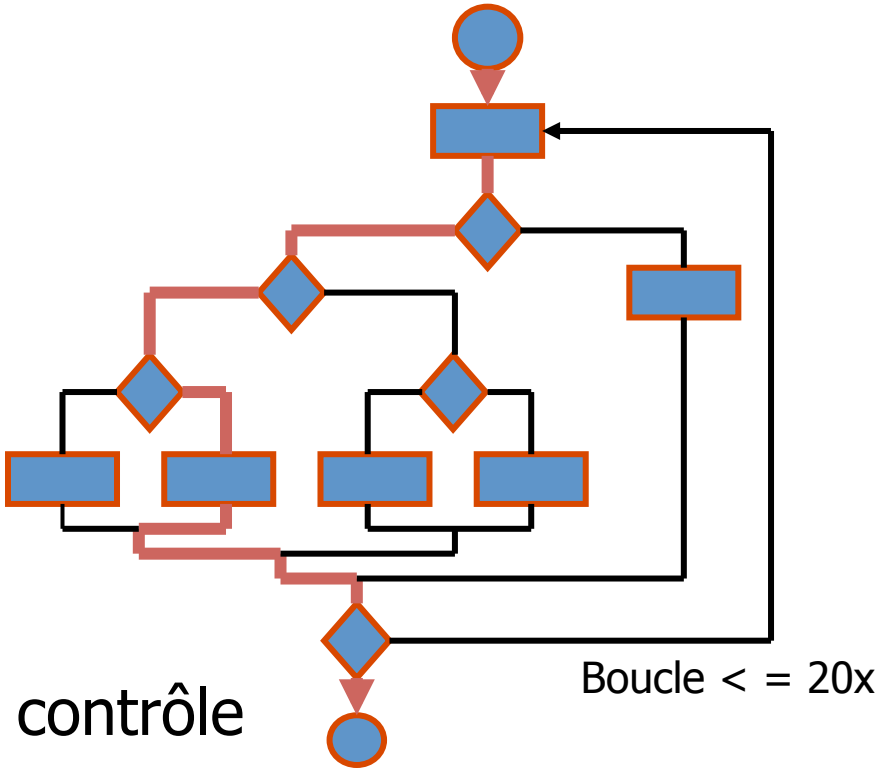
- Principes
  - S'appuient sur des spécifications externes
  - Partitionnent les données à tester par **classes d'équivalence**
    - Une valeur attendue dans 1..10 donne [1..10], < 1 et > 10
  - Ajoutent des valeurs « pertinentes », liées à l'expérience du testeur
    - Tests aux bornes : sur les bornes pour l'acceptation, juste au delà des bornes pour des refus

Pourquoi faire des tests en boîte blanche ?

- Tests en boîte noire:
  - Les besoins sont satisfaits
  - Les interfaces sont appropriées et fonctionnent
- Pourquoi s'occuper de ce qui se passe à l'intérieur ?
  - Les erreurs de logique et les suppositions incorrectes sont inversement proportionnelles à la probabilité d'exécution du chemin !
  - On croît souvent qu'un chemin ne va pas être exécuté ; en fait, la réalité va souvent à l'encontre des intuitions
  - Les erreurs de saisie sont aléatoires; il est vraisemblable que certains chemins non testés en contiennent

Test en boîte blanche

- Les données de test sont produites à partir d'une analyse du code source
- Critères de test
  - Tous les chemins
  - Toutes les branches
  - Toutes les instructions

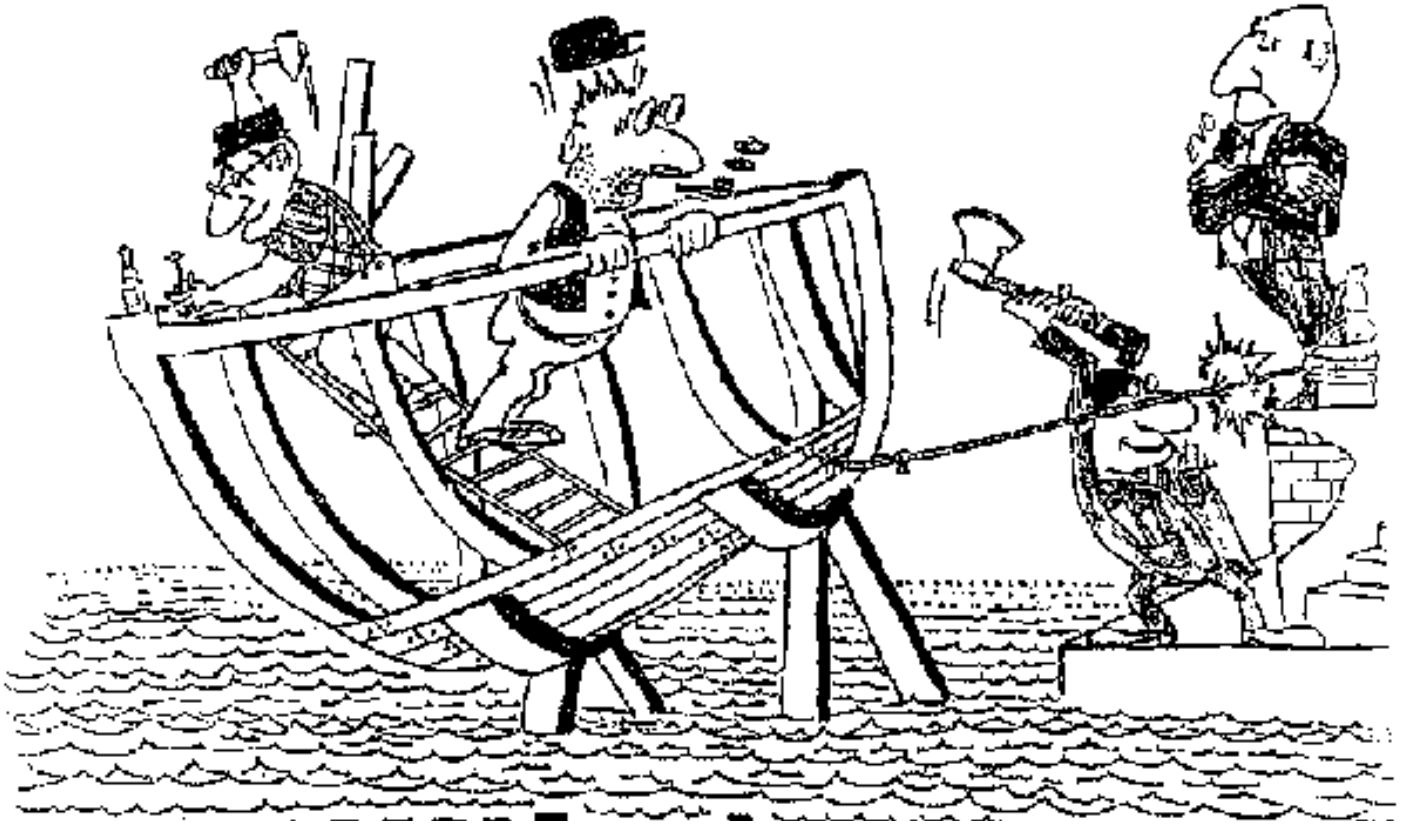


☞ Analyse du graphe de flot de contrôle

☞ Analyse du flux de données

Conclusion

Ne jamais être trop ambitieux…



La date limite, c'est la date limite !