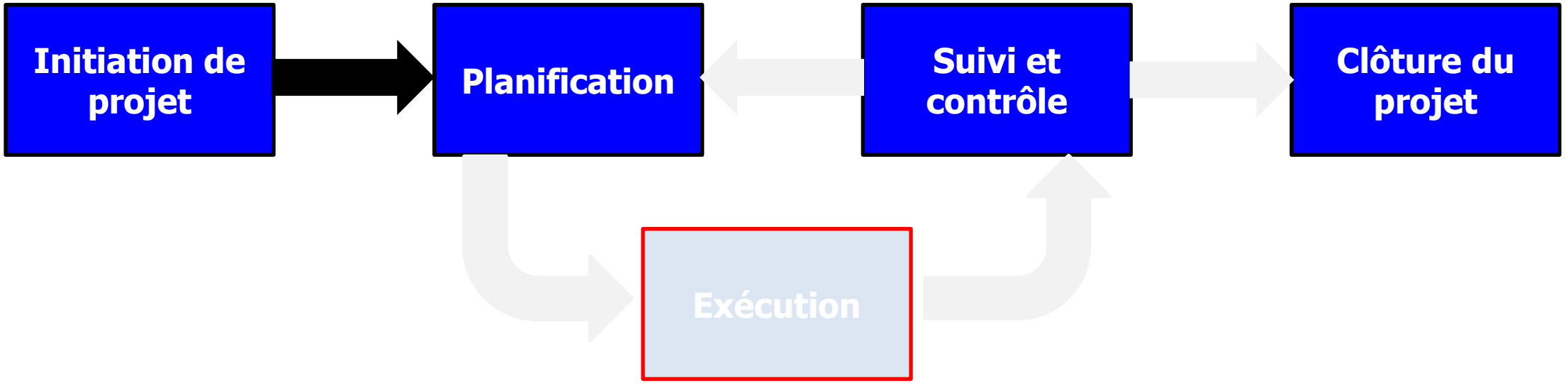
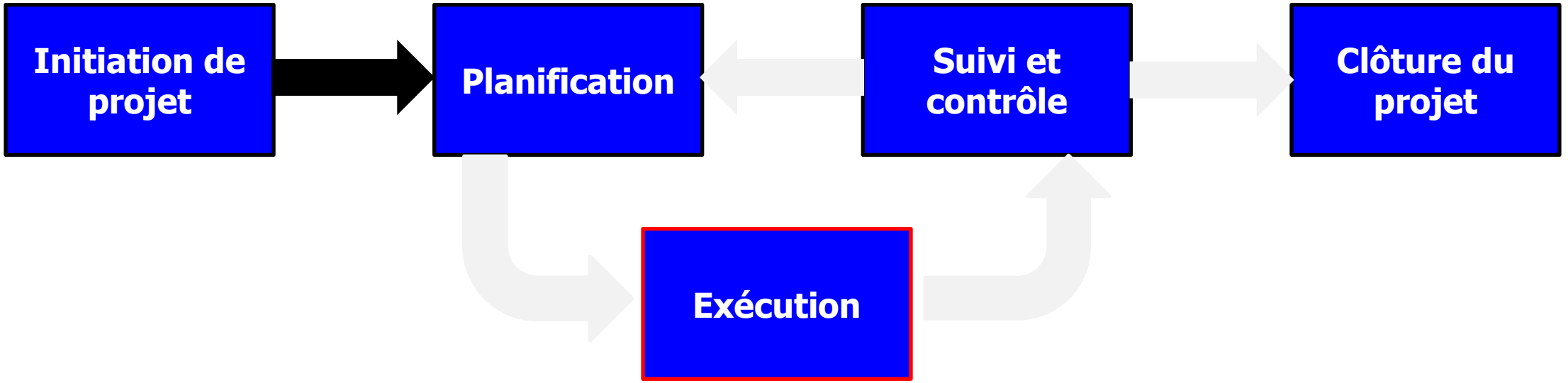
Chapitre 2

Cycle de vie de logiciels

Vue d'ensemble

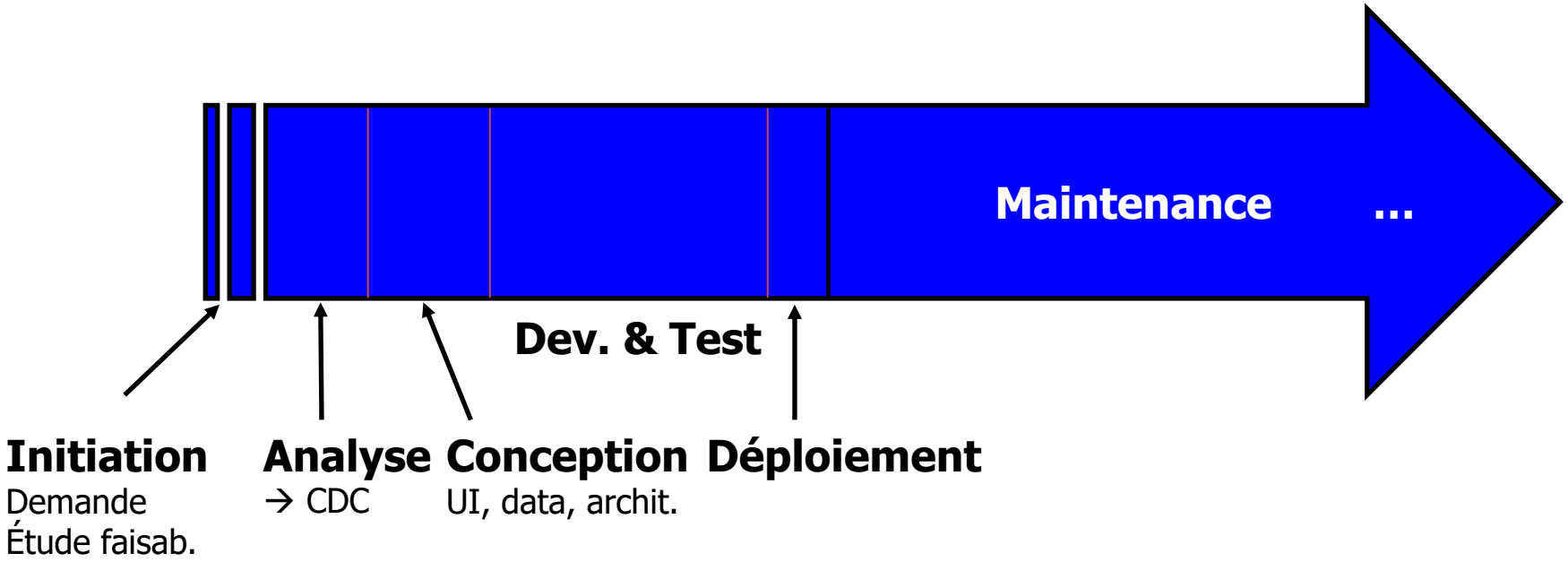


Vue d'ensemble



En résumé

Cycle de vie d'un logiciel ou système :



Les phases de développement

Il n'y a pas de standard (terminologie, nombres, frontières exactes, etc.)

Mais il y a des « classiques »

- Gestion de projet
- Développement
  - Analyse
  - Conception
  - Réalisation
  - Mise en production

Les phases de développement

Gestion de projet : planification, suivi

Analyse : spécification des besoins, modélisation

Conception : architecture, interfaces, gestion de données

Réalisation : codage, intégration, tests

Mise en production : déploiement, formation, transition vers la maintenance

2.1 Analyse des besoins

Définition des besoins

Ce que fait le logiciel et non comment il le fait !

Deux types de besoins / d'exigences

Fonctionnelles : fonctions et services offerts par le logiciel

Non fonctionnelles : contraintes imposées au logiciel, ou au processus de développement.

Techniques pour capturer les besoins

Interviews

Conception collective (Joint Application Design)

Questionnaires

Analyse des documents

Observation

Interviews (1)

Technique la plus utilisée

Poser des questions et discuter avec ceux qui détiennent l'information

Aspects importants

Sélection des participants

Sélection des questions

Préparation des interviews

Conduite des interviews

Documentation et suivi

Conception collective (2)

Technique développée par IBM à la fin des années 1970

Se focaliser sur les exigences

Éviter les spécifications trop vagues ou trop floues

Participants

- Coordinateur ou animateur (facilitator)
- 10 à 20 participants
- 1 ou 2 personnes qui assistent l'animateur (notes, vidéos, etc.)

Les questionnaires (3)

Lorsque le nombre d'utilisateurs est important

Pour atteindre des utilisateurs « externes »

Formats : papier / dématérialisé (mailing, forms, etc.)

Aspects importants

- Sélection des participants
- Conception du questionnaire
- Administration du questionnaire
- Analyse des résultats

Analyse (4)

Analyse des documents

Permet de mieux comprendre l'existante

→ Découvrir les exigences futures

Documents à analyser

Des rapports concernant le système existant

Les documents de travail

Les annotations et ajouts par les utilisateurs

La circulation des documents (workflow)

Observations (5)

Observation

Les utilisateurs ne sont pas toujours capables de décrire leur fonctionnement

Cette technique peut être combinée avec des interviews (sur place)

Nécessite une expertise particulière : didacticiens, sociologues, …

Spécification des besoins

Hiérarchisation des besoins (MoSCoW)

- **Must** (essentiel, incontournable)
- **Should** (souhaitable)
- **Could** (envisageable)
- **Won't** (superflu)

Formalisation des besoins

Cahier des charges

Langages naturels

Simples, ne nécessitent aucun apprentissage particulier

Ambigus, manque de précision, verbeux, etc.

Langages semi-formels

Diagrammes, Tableaux, Graphiques, user stories

Synthétique, moins verbeux, dépend de la standardisation

Formalisation des besoins

Langages formels

Précision, clarté, et absence d'ambiguïté

Difficile à maîtriser, nécessaires pour logiciels critiques (nucléaire, santé,…)

Cas d'utilisation (use case)

Manière d'utiliser le système

Décrire les exigences fonctionnelles

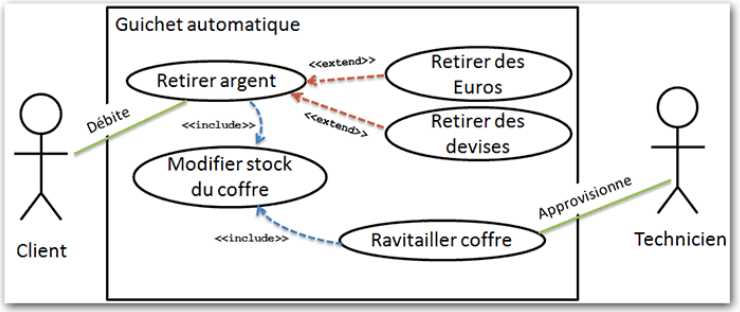
Un cas d'utilisation : événements faisant quelque chose d'utile.

Un système peut être décomposé en cas d'utilisation

Spécification des besoins (exemples)

UseCase	MOVE UP
Actors	Elevator user
Intent	Move from the current floor to another floor above the floor.
Preconditions	THE ELEVATOR IS OPERATING
Flow of events	1) Included use case "call cabin" 2) extending use case "select floor" [Exception]
Exceptio ns	
Rules	Look at included Use Cases
Quality constraints	Look at included Use Cases
Monitored environmental variables	Look at included Use Cases
Controlled environmental variables	Look at included Use Cases
Postconditions	ELEVATOR MOVES TO THE DESTINATION AND OPENS THE DOOR

UseCase	CALL CABIN
Actors	Elevator user
Intent	To make the elevator cabin come to the floor of the user
Preconditions	The elevator is operating
Flow of events	1. Actor indicates his/her intention of moving up or down with the elevator (move up or down) 2. Elevator determines next moving direction based on the floor of request 3. Elevator moves the cabin to the requested floor : extending use case "change location" 4. Cabin opens its door [Exception: technical problem]
Exceptions	
Rules	The elevator stops at every floor where this use case is initiated on the way of moving. Repeated requests are ignored
Quality constraints	Arrival of Cabin and Opening of cabin door
Monitored environmental variables	Floor of request : 1 to N Moving request : Up or Down
Controlled environmental variables	Current position : 1 to N Moving direction : Up, Down, Halt
Postconditions	Current position = floor of request



http://www.jot.fm/issues/issues_2007_09/article4

2.2 Conception

4.2 Tâches de la phase de conception

Passer du **quoi au comment**

Comment mettre en œuvre les modèles définis dans l'analyse

Principales activités

Faire évoluer les modèles d'analyse : découpage - factorisation

Conception des architectures : logicielle - matérielle - réseaux - …

Conception des interfaces utilisateurs

Conception des données persistantes

Conception détaillée des classes ou modules

Les stratégies de conception (1)

Développement interne

concevoir et développer le système avec l'équipe du projet

AVANTAGES

Meilleur contrôle sur le projet

Meilleure connaissance des spécifications

Flexibilité dans la solution des problèmes métiers

Acquisition d'expérience

Plus de maîtrise lors de la maintenance

Les stratégies de conception (1)

Développement interne

concevoir et développer le système avec l'équipe du projet

INCONVÉNIENTS

Nécessite et monopolise des ressources

Difficulté à accumuler différents profils et compétences en interne

Tendance aux débordements, distractions, ….

Les stratégies de conception (2)

Achat de logiciel "packagé"

Acheter / adapter un logiciel (composant, asset, plugin, librairie, IP core) générique qui répond aux specs.

AVANTAGES

- Minimiser les risques
- Coûts souvent plus abordables
- Temps de réponse très court
- Choix entre variantes

Les stratégies de conception (2)

Achat de logiciel "packagé"

Acheter / adapter un logiciel (composant, asset, plugin, librairie, IP core) générique qui répond aux specs.

INCONVÉNIENTS

Accepter l'existant sur le marché

Nécessité de gérer les configurations, paramétrages, …

Problèmes d'intégration

Les stratégies de conception (3)

Sous-traiter le développement (« outsourcing »)

Laisser une société de services développer le système

Nécessite une spécification très claire des besoins et un choix rigoureux du prestataire de service

AVANTAGES

- Profiter de l'expérience du prestataire
- Avoir une vision externe
- Gérer les délais

Les stratégies de conception (3)

Sous-traiter le développement (« outsourcing »)

Laisser une société de services développer le système

Nécessite une spécification très claire des besoins et un choix rigoureux du prestataire de service

INCONVÉNIENTS

- Risque selon le prestataire
- **Travail important pour l'établissement des spécifications**
- Dépendance pour la maintenance par la suite

Les stratégies de conception

Choisir une stratégie de conception :

- Les besoins métiers
- L'expérience de l'équipe
- La spécificité du projet
- Les contraintes de temps

Définition d'une matrice des alternatives

Possibilité de combiner les approches

Conception de l'infrastructure

Les systèmes développés sont intégrés à l'existant :

Infrastructure réseau, Infrastructure matérielle, Infrastructure logicielle, Politique de sécurité

La conception doit

- Situer le système dans cet environnement
- Préciser les structures de communication, la structure matérielle et la structure logicielle du système
- Décrire son intégration dans l'environnement existant.

Autres aspects importants

Aspects liés aux langues : codage, interfaces

Politique de contrôle : centralisation des applications, configurations, distribution, personnalisation

Standards, normes, cultures : standards de fait, normes internationales, pratiques locales, etc.

Support : assurer un support 24/7, backup et sauvegarde

Exemple

Conception des interfaces utilisateurs

Conception des interfaces

Aspects importants d'une GUI :

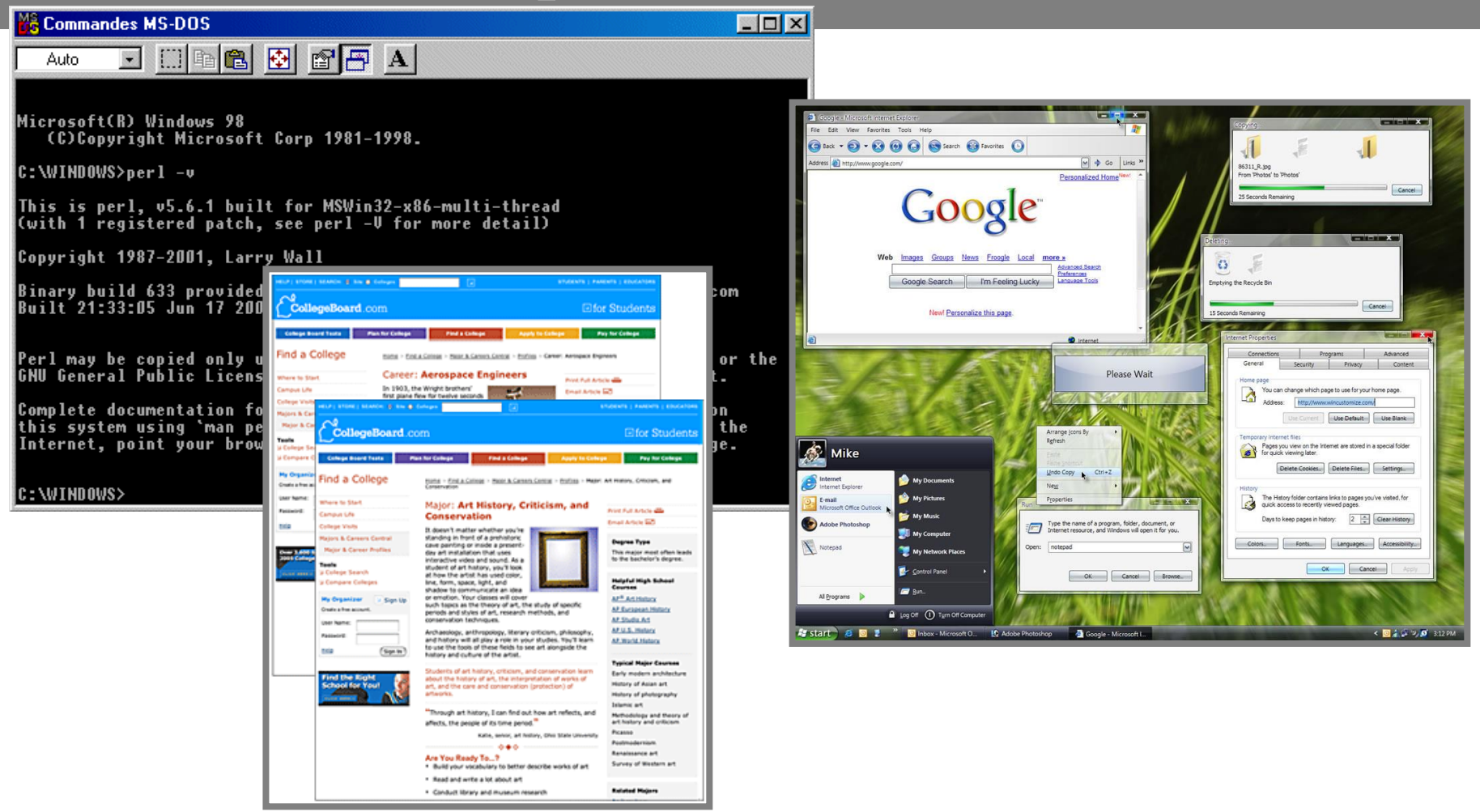
La navigation

Les entrées

Les sorties

L'esthétique

Définir le « Layout »



Conception de la navigation

Les types d'activation de la navigation :

- Langages de commande (CLI): shell, scripts, DOS, SQL, …
- Les menus: fixes, contextuels, ruban, onglets, …
- La manipulation directe: liens, raccourcis, Tab, Enter, …

Les messages : interagir avec l'utilisateur et ses actions (erreurs, confirmation, information, etc.)

La documentation : documenter les choix réalisés dans la définition de la navigation

Définir les interactions

Conception des entrées

Aspects importants : traitements batch ou en ligne, maximiser l'extraction des données à la source, minimiser les entrées clavier

Type d'entrées : texte, nombres, sélections

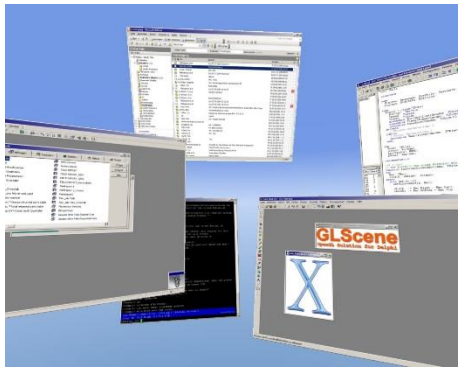
Importance de la validation des entrées : complétude, format, intervalle de valeur, consistance et cohérence

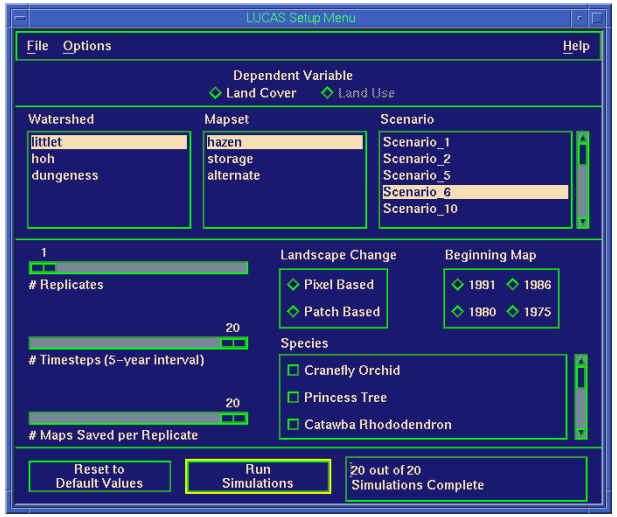
Conception des sorties

Aspects importants : identifier les besoins pour éviter la surcharge cognitive, minimiser les sorties biaisés

Types de sorties : affichage écrans, les rapports, les médias

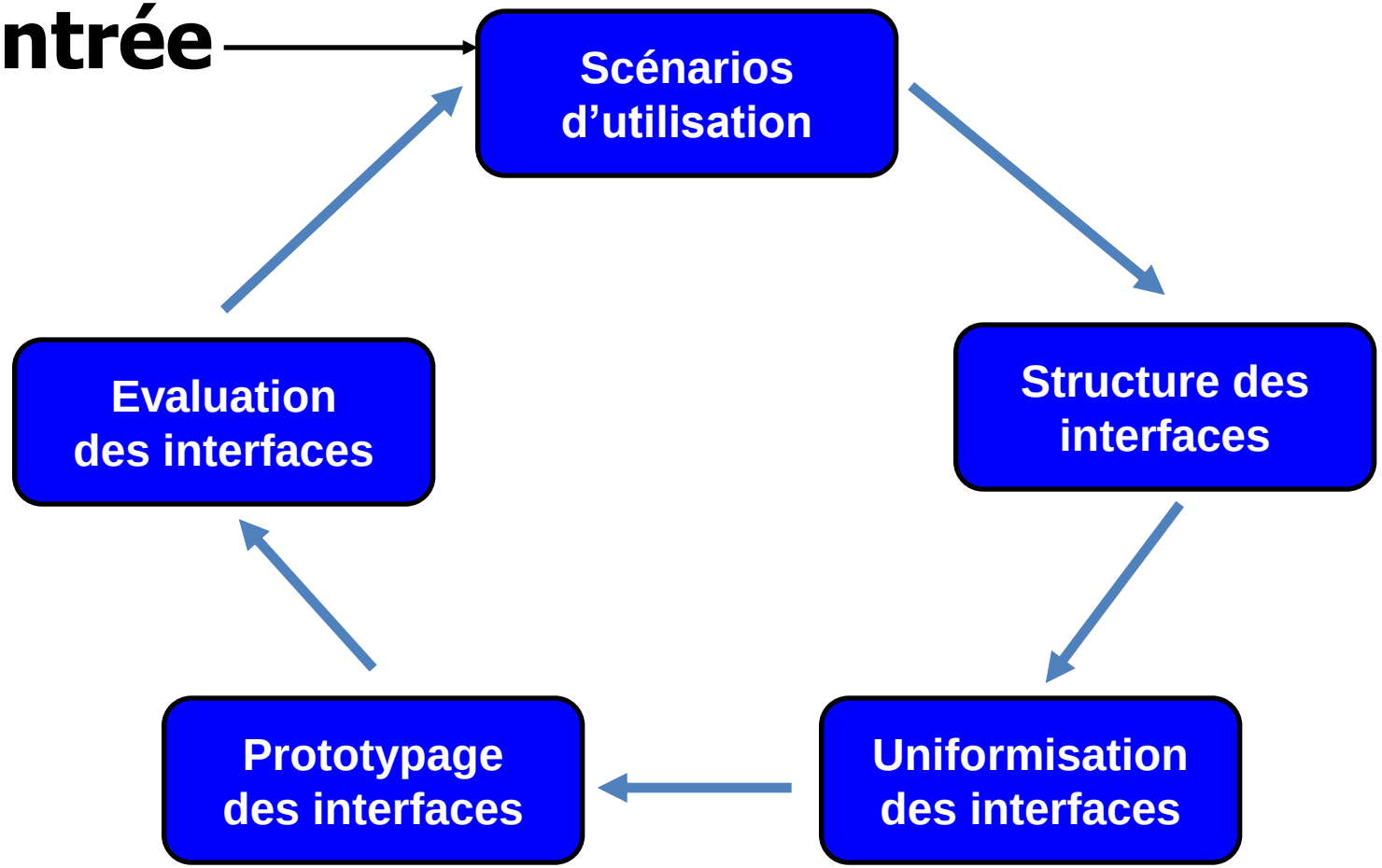
Définir l'esthétique





Conception des interfaces utilisateur

Point d'entrée



Structure de l'interface utilisateur

Listes et nature des composants d'interface utilisés

Liens et séquence des composants

Techniques : description textuelle, diagramme de navigation (Windows Navigation Diagram), diagrammes d'états

Si possible : maquettes, prototype, références

Standards et règles communes

GUI cohérente, compréhensible et uniforme

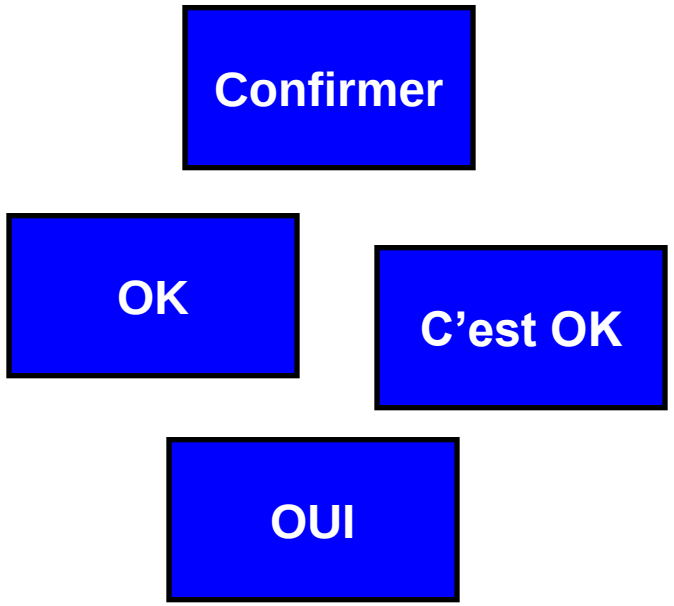
Créé un environnement intuitif pour travailler

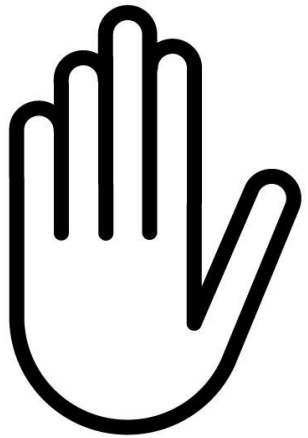
Important pour **uniformiser** ou **standardiser** l'interface :

- **Les métaphores :** cockpit - caddie - journal – dashboard -
- **Terminologie :** noms des objets, nom des actions
- **Les icônes**
- **Les templates :** région, look, entête, couleur, …

Consistance et cohérence

Ceci n'est pas un lien

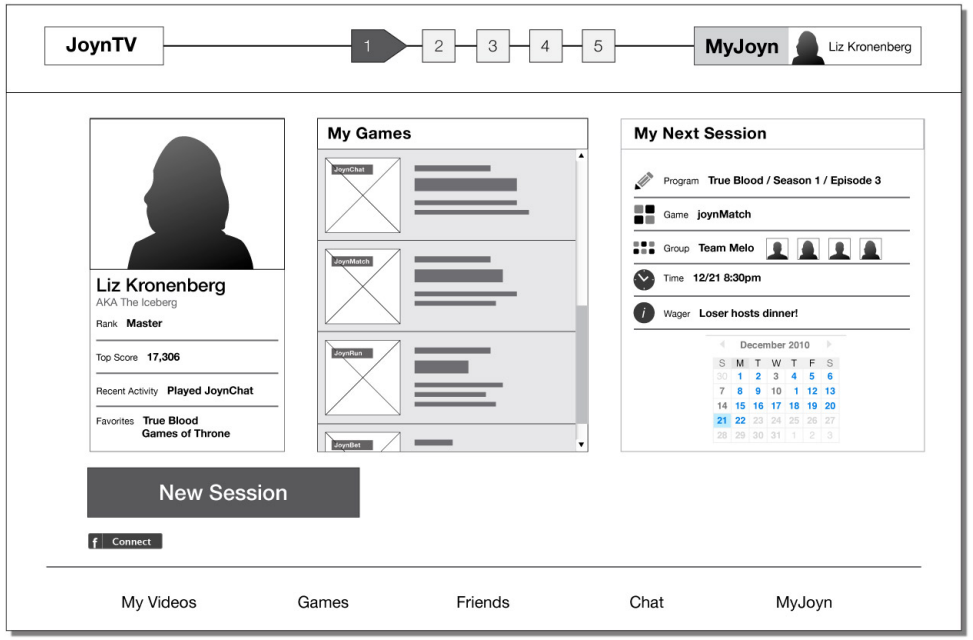




Continuer

Prototypage des GUIs

Concrétiser la structure et les standards définis auparavant



Techniques de prototypage

- Storyboard
- Prototypage HTML
- Langages de prototypage dédiés
- Langages standards avec des outils de développement intégrés

Evaluation des interfaces utilisateurs

Valider les choix réalisés et les améliorer en cas de besoin

L'évaluation doit être réalisée **le plus tôt possible** afin d'éviter des modifications conséquentes par la suite

Evaluation des interfaces utilisateurs

Techniques d'évaluation :

- **évaluation heuristique :** utilisation par l'équipe
- **évaluation par parcours :** démonstration à l'utilisateur
- **évaluation interactive :** laisser l'utilisateur tester
- **Tests formels de « usability »**
- **GUI testing tools :** Selenium, Ranorex, etc.

Persistance des données

Stockage des données du système

- Simple fichier texte
- Fichier texte structuré (XML, JSON)
- Fichier binaire avec un format (version, header, …)
- Base de données légère (fichier local)
- BDD relationnelle traditionnelle
- Dépôt (avec historique, …)

2.3 Implémentation

Codage

Techniques, outils, environnements de programmation : voir autres cours

Aspects importants :

- Attribution des tâches de programmation
  - Tenir compte de la compétence et de l'expérience
  - Éviter les équipes de taille importante
- Coordination des activités
  - Serveurs partagés
  - Meetings
  - Procédure de contrôle et de suivi normalisée
- Gérer le planning
  - Éviter les ajouts en cours de développement
  - Éviter les petits retards successifs

Codage

Bases pour XP (Xtreme Programming)

- Client sur site
- Jeu du Planning (Planning poker)
- Intégration continue
- Petites livraisons
- Rythme soutenable
- Tests fonctionnels
- Tests unitaires
- Conception simple
- Utilisation de métaphores
- Refactoring (ou remaniement du code)
- Appropriation collective du code
- Convention de nommage
- Programmation en binôme

Codage: la revue de code

Examen systématique d'un code source afin d'en **améliorer la qualité** (trouver des bugs, etc.)

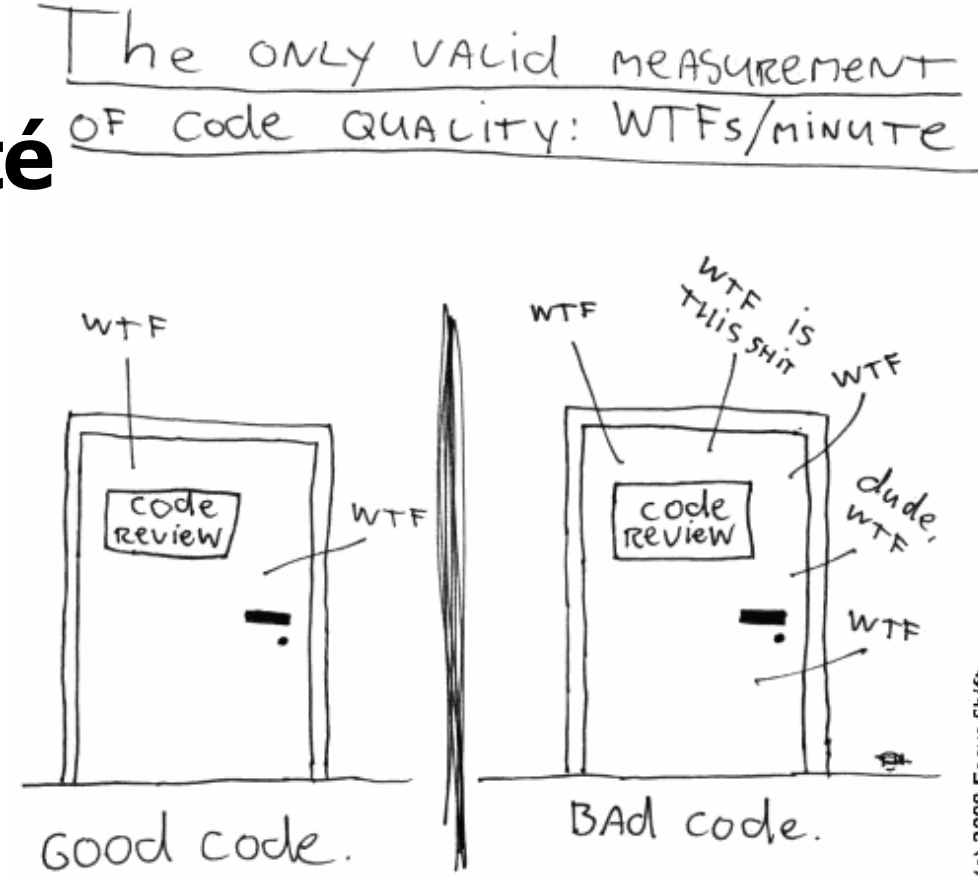
Peer review (collègues)

Pair programming, code review session

Automatisée

"Linter", analyse statique (sans exécution)) du code, tests unitaires,

Calculs de métriques (CCCC)



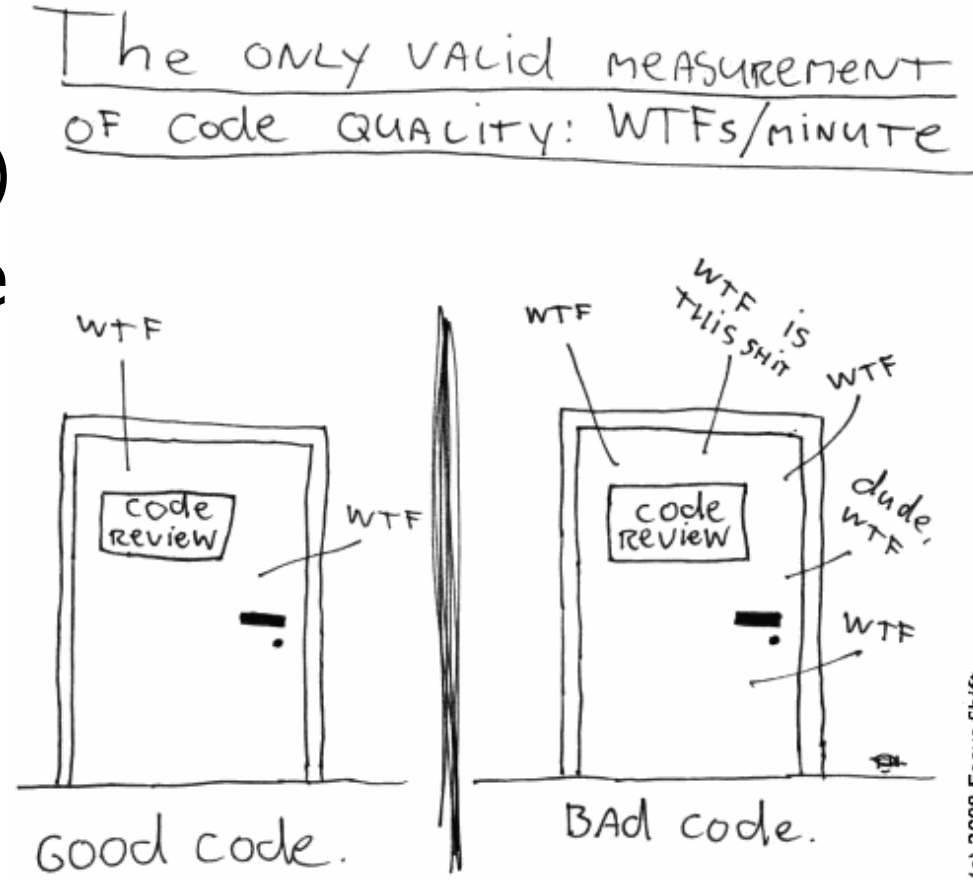
Codage: La revue de code

Pourquoi ?

2-4x plus rapide que le test (exécution)

>50% de défauts détectés en plus que par test

Un reviewer trouve en moyenne 15 défauts par heure (durée max)



2.4 Tests

Tests (définition)

erreur → **défaut** → **anomalie**

© Can Stock Photo

Attitude du testeur

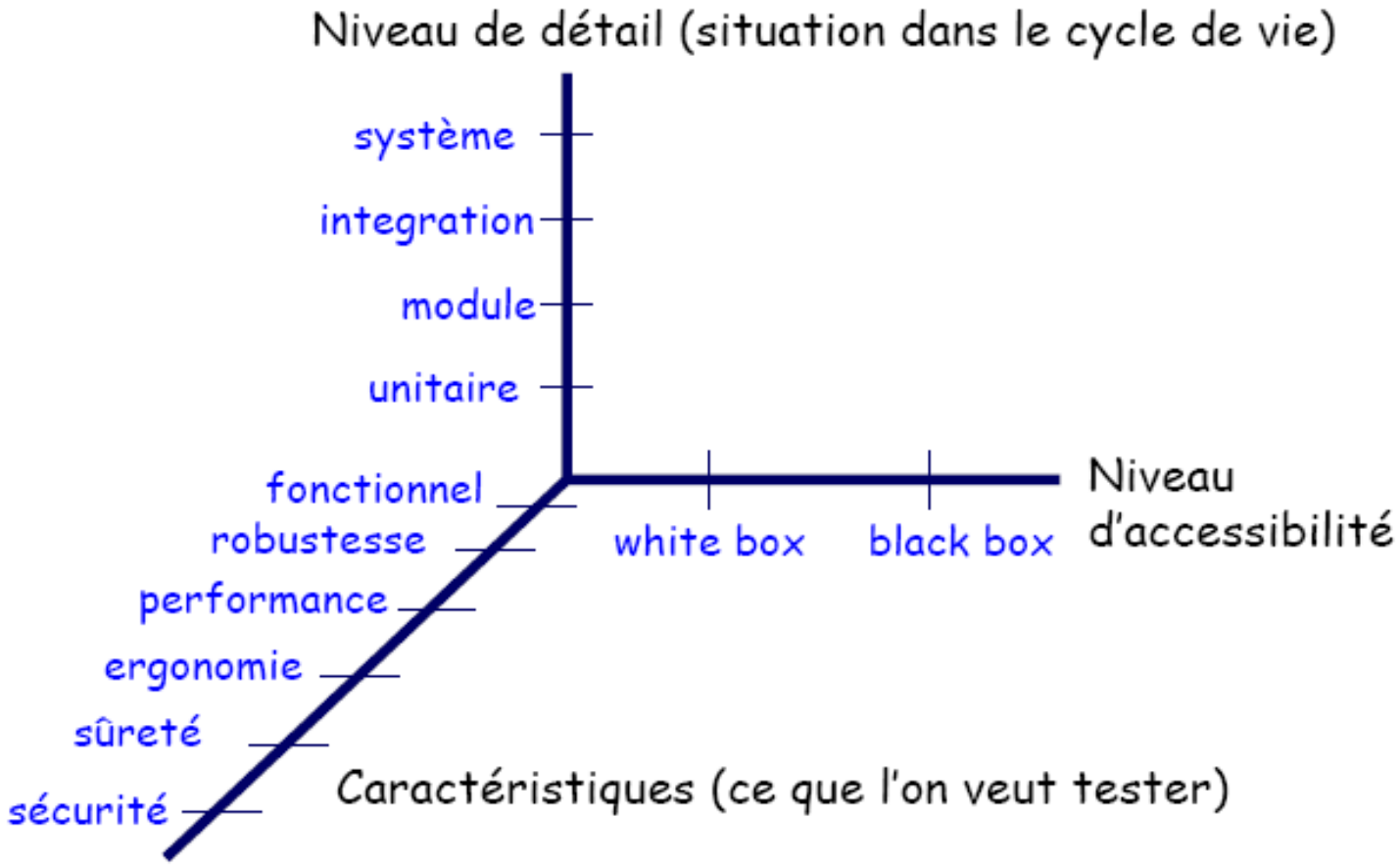
Le test est un processus "destructif" : le but est de mettre en défaut le logiciel !

Un test ne trouvant aucun bug est un échec !

Éviter l'approche "montrer que ça marche" !

Ainsi, il est préférable de confier l'activité de test à une équipe séparée de celle du développement.

Taxonomie



D'après J. Tretmans – Univ. Nijmegen

Types de tests

Les tests statiques

Pas d'exécution du logiciel

Vérification et validation basée sur les modèles, la documentation

Les tests dynamiques

Exécution du logiciel

Comparaison entre les résultats obtenus et les oracles (résultats spécifiés, attendus)

Niveaux des tests

Test unitaire

Classe, Méthode, Fonction

Détecter les erreurs de programmation (logique)

Test d'intégration

Sous-systèmes, Modules, Packages

Détecter les erreurs d'interfaces

Test système

Architecture, Interaction, Performance, Sécurité, stress

Tests d'intégration mais **en condition réelle**

Test de validation

Utilisation, Ergonomie, Fonctionnalité, non régression

Détecter les non-conformité au niveau du CDC avec utilisateurs réels

Techniques de tests

Tests boites noires : tests fonctionnels

- Pas d'accès au code
- Tests basés sur les entrées-sorties
- On parle aussi de tests orientés données

Tests boites blanches : tests structuraux

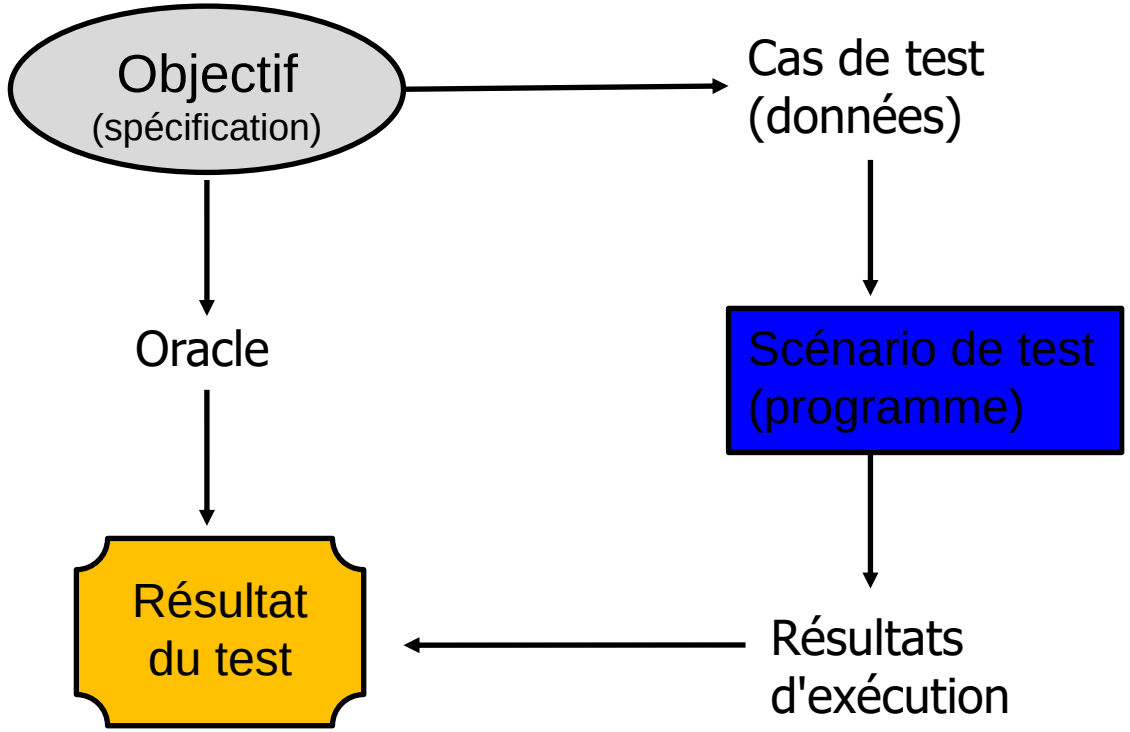
- Le code est accessible
- Basés sur la structure et la logique du code

Tests boite noire

Appelées aussi **tests fonctionnels**

Exemples de techniques

- Valeurs frontières
- Classes d'équivalence
- Tables de décisions
- Random



Tests boite blanche

Appelés aussi tests structuraux

On dispose du code source

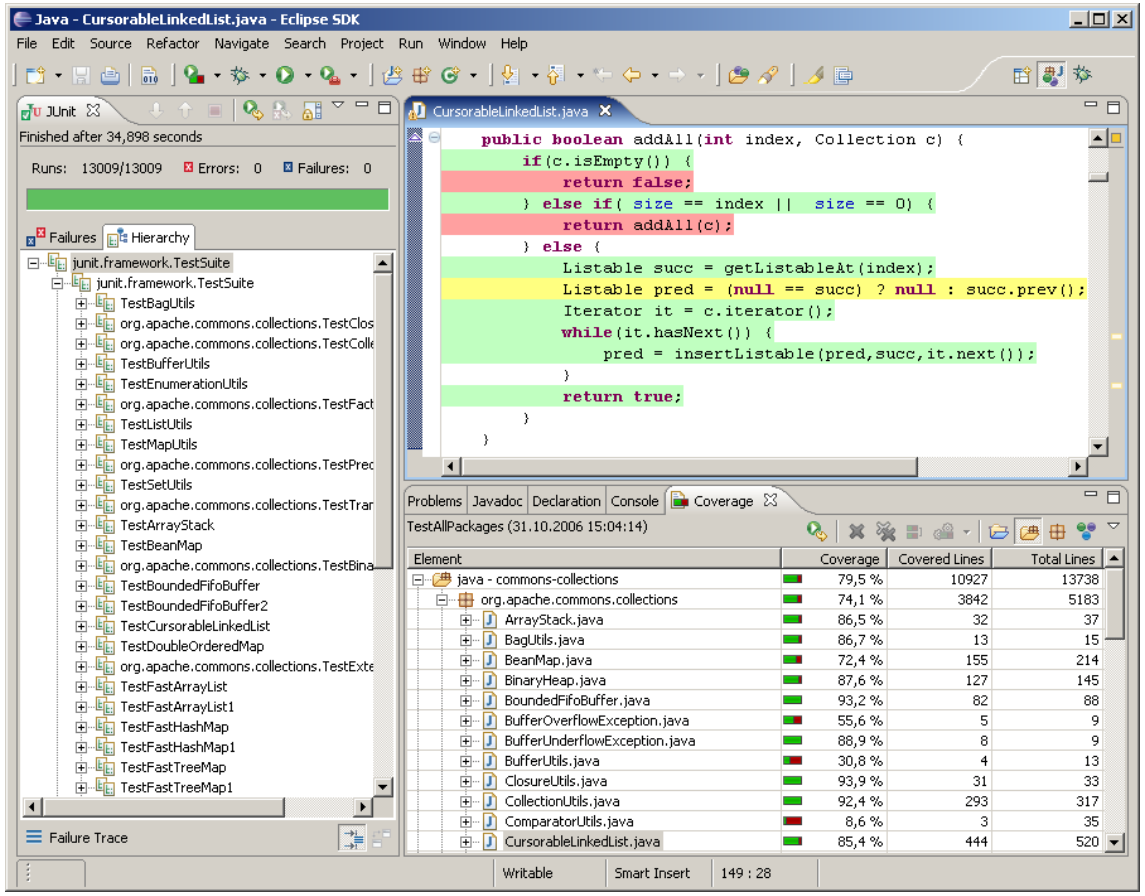
les tests tiennent compte de la structure du code source

Difficulté de couverture (#chemins possibles grand)

Exemples de techniques « boites blanches »

Chemin d'exécution, Flot de données, complexité cyclomatique

Tests boite blanche



Function Coverage

Statement Coverage

Condition Coverage

Path Coverage

2.5 Déploiement et maintenance

Déploiement

Intégration du logiciel développé dans son contexte d'utilisation réel

- Installation sur les machines de la société cliente
- Raccordement au réseau de la société cliente
- Utilisation de la base de données de la société

Ce contexte d'utilisation doit être décrit avec précision dans la spécification du logiciel afin d'éviter les mauvaises surprises lors de cette phase de déploiement

Déploiement

Phase généralement accompagnée de problèmes semblables à ceux rencontrés lors de l'intégration !

En effet, le déploiement est un cas particulier d'intégration

→ Continuous Integration / Continuous Deployement

C'est pour cette raison que dans le cycle de vie d'un logiciel, **il faut prévoir un délai important pour l'activité de déploiement.**

Stratégies de déploiement

Styles de transition

directe (« big bang ») **vs** progressive (2 systèmes en parallèle)

Volume de la transition

Transition complète : système entier

Transition modulaire

Aspects importants

Gestion des changements organisationnels

Gestion des risques

Politique ou stratégie de déploiement

Motivation des utilisateurs (trauma, écoute)

Formation

Activités Post-déploiement

3 activités importantes :

Le support

Formation, support online (niveaux 0 1 2 3), etc.

L'évaluation et la documentation

Gestion du projet, équipes, techniques et méthodologies

La maintenance

La partie la plus coûteuse dans le cycle de vie d'un logiciel

La maintenance

Fait partie du cycle de vie du logiciel !

Partie la moins développée en génie logiciel

Peu de théorie, de fondements, de littérature, etc.

Processus de développement, mais avec un système existant

Assurer la maintenabilité

Ce n'est pas une activité ponctuelle : elle se prépare durant les autres phases du projet :

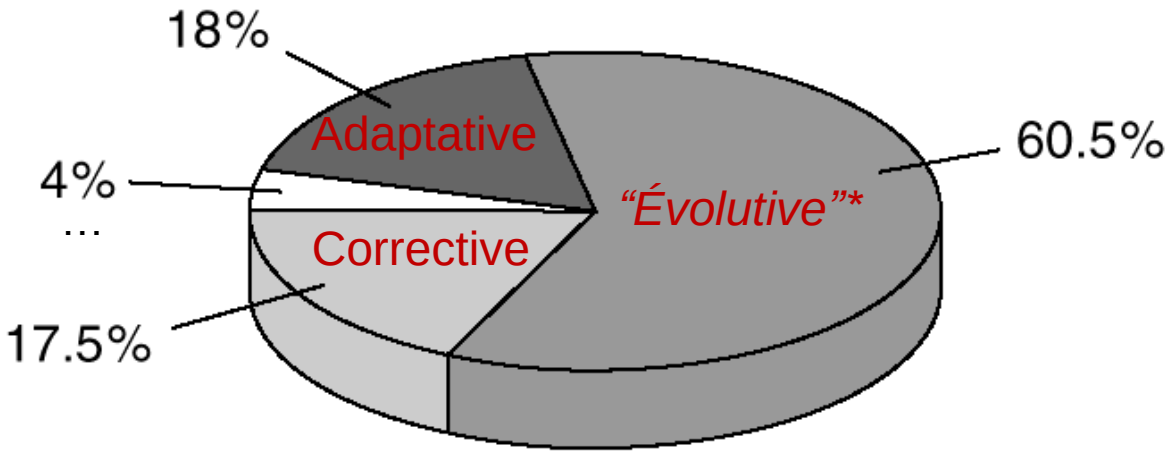
Dans la conception, maximiser l'abstraction, le masquage de l'information, la modularité, etc…

Dans l'implémentation, utiliser des noms de variable significatifs, des commentaires parlants, etc…

La documentation doit être juste, complète et doit refléter l'état courant et actuel du logiciel

Prendre en compte la pérennité des composants et technologies

Trois types de maintenance



** Pas de la maintenance au sens strict du terme*

Maintenance corrective (18%)

Correction d'erreurs

- de spécification
- de conception
- d'implémentation
- documentation
- ...

Maintenance adaptative (18%)

Répond à des changements de contexte :

- Porter sur un nouvel environnement ou un nouveau système
- Utiliser un nouveau compilateur ou langage de programmation
- Changement dans certaines variables de bases
  - Taille du code postal, passage au bitcoin, scanner de code-barres → QR code → NFC, bug de l'an 2000 …

- Suivre l'obsolescence des composants (IIE)

Maintenance évolutive (60%)

Nouvelles exigences : meilleure efficacité, ergonomie… :

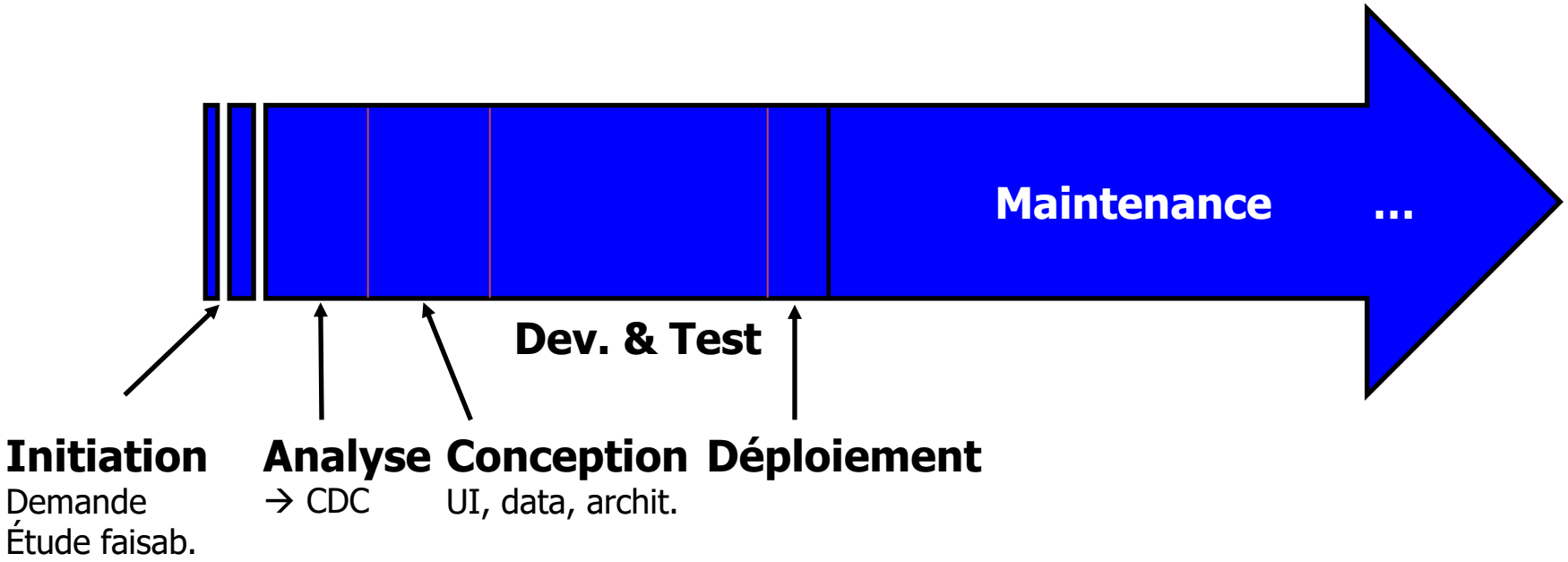
Ajout de nouvelles fonctionnalités

Amélioration des performances du logiciel

Amélioration de la maintenabilité du logiciel (refactoring, …)

En résumé

Cycle de vie d'un logiciel ou système :



Exercice

Formaliser l'analyse des besoins de votre projet P2