

Génie Logiciel et GESTION DE PROJETS INFORMATIQUES

J Paul Gibson, INF, TSP, paul.gibson@telecom-sudparis.eu

Merci à: J-L Raffy, Daniel Millot, Daniel Ranc

POURQUOI UN PROJET?

« *Notre* » Objectif - Transformation: élèves/étudiants en ingénieurs

Typiquement

- seul
- sans *vrai* client
- un problème isolé, abstrait
- bien défini : toutes les données sont disponibles
- une (meilleure) solution « unique »

Typiquement

- en équipe
- au moins un *vrai* client
- des problèmes liés, concrets
- mal définis : on ne connaît pas les informations nécessaires
- ayant de nombreuses solutions possibles

Le projet: vous allez *jouer le rôle* des ingénieurs

PLANNING

I. Principes Fondamentaux

(De La Gestion De Projet)

30mins

II. Genie Logiciel

(Dans Les Projets Informatiques)

II.1 Cycle de vie d'un logiciel

II.2 Le cycle en V

II.3 Cycle de vie – les alternatifs

II.4 Le cycle en V: plus détaillé

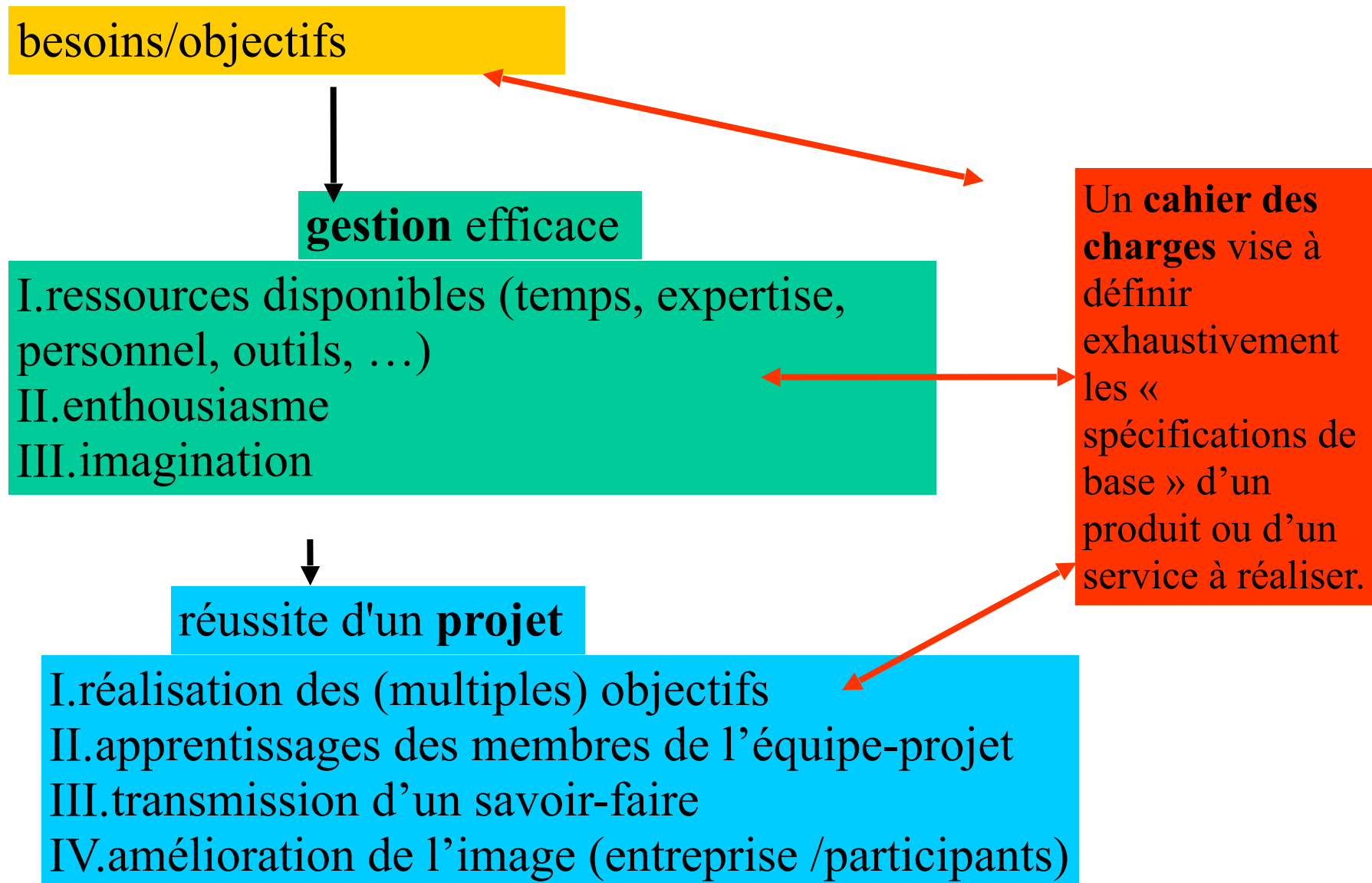
30mins

III. Gestion de Projets Informatiques

30mins

IV. Étude de Cas – Bibliothèque

90mins



Cahier des charges (pour chaque projet)

A) *Contexte et Historique*

B) Description de la demande

I. Les objectifs

II. Produit(s) du projet

III. Les fonctions du produit

IV. Critères d'acceptabilité et de réception

C) Contraintes

I. Contraintes de coûts

II. Contrainte de délais

III. Contraintes techniques

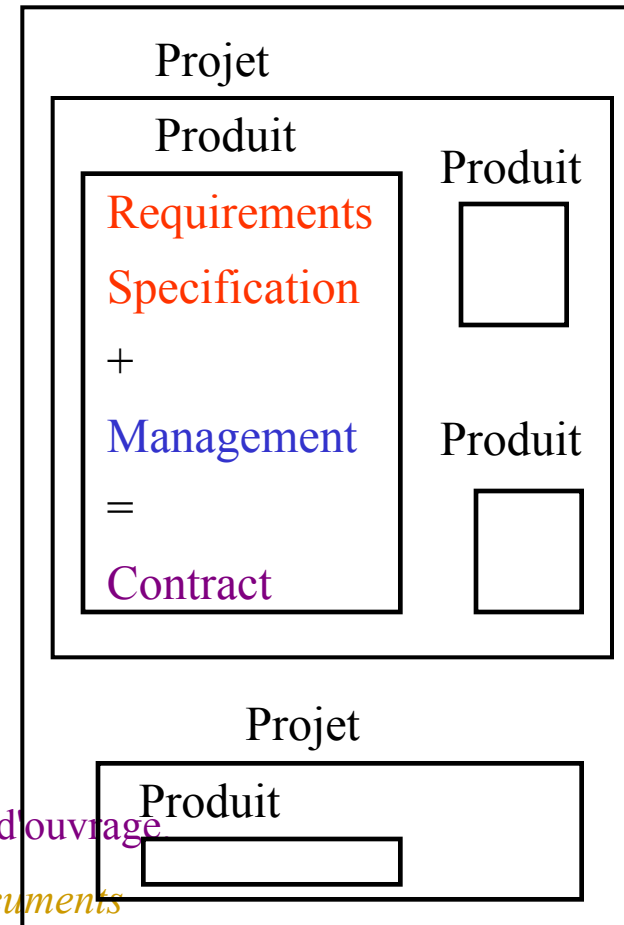
IV. Clauses juridiques, etc.

D) Déroulement du projet- Planification des Phases et Ressources

E) Authentification - Date et signature du chef de projet et du maître d'ouvrage

F) *Annexes - Lister et joindre au cahier des charges les éventuels documents que le client peut mettre à disposition.*

Domaine/Contexte



Qu'est-ce qu'un projet?

Suite d'actions que l'on se propose d'accomplir pour aboutir à une fin

Un projet est toute activité réalisée :

- I.une seule fois,
- II.doté d'un début et d'une fin déterminés
- III.vise à créer un produit ou un savoir unique

Complexité d'un projet :

- I.# de personnes
- II.# organisations/groupes/clients
- III.# de fonctions/ contraintes
- IV.durée de travail

Qu'est-ce que la gestion de projet?

La gestion de projet est *l'utilisation d'un savoir, d'outils et de techniques* en vue de satisfaire (ou de dépasser) les exigences et les attentes des parties prenantes à l'égard d'un projet

Le gestionnaire de projet doit avoir *la capacité de concilier des exigences contradictoires* telles que :

- I.les ressources disponibles et les attentes;
- II.les priorités différentes des parties prenantes;
- III.les besoins définis et rester à la portée du projet;
- IV.la qualité et le temps;

Caractéristiques des projets réussis

I. Les objectifs « SMART »

I. Un plan de projet bien établi et réaliste

I. Une communication réelle et continue

I. Une perte minimale de temps (et de concentration)

I. Le soutien des intervenants (pour toute la durée du projet)

Les objectifs « SMART »

I.Spécifique: L'objectif décrit précisément

I.Mesurable: Il sera possible de juger objectivement de l'atteinte de l'objectif.

I.Ambitieux: Atteindre l'objectif implique un effort. Ceci s'exprime à travers l'objectif lui-même.

I.Réaliste: Un objectif de projet doit être réaliste dans le sens qu'il doit être atteignable avec les moyens disponibles. On ne peut juger de cela qu'en connaissant le contexte, la durée et les ressources du projet.

I.Défini dans le Temps: En principe, un objectif de projet doit être atteint à la fin du projet. Un objectif intermédiaire doit être atteint au plus tard au moment du pointage d'étape.

Les objectifs « SMART+ »

+flexible

Les objectifs sont formulés clairement de façon à éviter tout désaccord ultérieur.

Mais, que se passera-t-il, par exemple, si l'un des intervenants ajoute un nouvel objectif au projet (ou change/supprime un objectif déjà existant).

Un plan de projet détaillé vous aide à affronter ce type de situation et à vous en tenir aux objectifs généraux du projet.

La planification? Pourquoi?

Le plan de mise en oeuvre d'un projet contribue à la maîtrise et à la mesure de sa progression

Le plan de projet vous aidera à faire face aux changements susceptibles de survenir

Le plan de projet peut être considéré comme un contrat (ou un accord légal)

Le plan est une validation (partielle) des objectifs

La planification? Les limites?

Il est impossible de tout prévoir

On ne peut pas toujours maîtriser toutes les variables impliquées

En voulant trop planifier, on peut en venir à étouffer le potentiel de créativité de l'entreprise et de ses membres

La planification? Les critiques?

La planification comporte un biais en faveur des objectifs d'ordre économique au détriment des aspects humains, sociaux et écologiques reliés à l'entreprise.

L'utilisation du calcul, de l'analyse et de moyens instrumentaux propres à la planification se fait au détriment de la réflexion, de l'intuition et de la synthèse.

La planification peut être une source de rigidité et donc devenir un obstacle important à la souplesse et à la capacité d'adaptation de l'entreprise et nuire ainsi au développement d'idées innovatrices.

La planification a tendance à favoriser la centralisation et la concentration de la prise de décision au niveau de la haute direction.

Qu'est-ce qui constitue un plan de projet?

Puisqu'il n'y a pas deux projets pareils, il n'existe pas non plus deux plans de projet qui soient identiques.

Pour qu'il soit d'une utilité maximum, votre plan de projet doit être pertinent, compréhensible et tenir compte de l'importance et de la complexité de votre projet unique.

Le plan de projet devrait être remis au gestionnaire de projet, au promoteur, à chaque partenaire et à tous les principaux membres du personnel du projet. Il s'agit d'un outil de grande valeur qui peut permettre d'éviter la confusion quant à la portée du projet et les malentendus sur les responsabilités, les échéanciers ou la gestion des ressources.

Qu'est-ce qui constitue un plan de projet? (pour PRO3600)

Le mandat de projet - autorise le gestionnaire du projet à mener le projet et à attribuer les ressources nécessaires

Le calendrier d'activités - fractionner un projet et répartir toutes les tâches nécessaires à son achèvement. Certaines tâches sont subordonnées à d'autres et ne peuvent être amorcées que lorsque d'autres sont terminées. (*Task Graphs*)

L'horaire de travail - pour chaque tâche il faut estimer le temps de travail et préciser les dates d'exécution de l'activité (*Gantt charts/diagramme de Gantt*)

La matrice de responsabilités - pour chaque tâche il faut partager le travail entre les partenaires (1 partenaire avec responsabilité)

Le budget du plan de projet - l'estimation des coûts pour chaque tâche

Les étapes importantes, avec les dates cibles - marquent un événement significatif dans le cours du projet, habituellement l'achèvement d'un produit (composant/artefact) livrable important. Les vraies mesures de progrès.

La stratégie de gestion du risque -il faut d'abord les identifier

Qu'est-ce qui constitue un **bon** plan de projet?

Il faut un plan **cohérent**

La quantité de détails fournis dans votre plan de projet dépendra de vos besoins. Votre plan peut être très bref ou très détaillé. Il vous incombe de mettre au point un plan **approprié** à la nature, à la taille et à la complexité de votre projet. (Prévoyez du temps pour des activités de gestion de projet – en règle générale, 10 % du temps total du projet)

Si votre organisme et vos partenaires décident que la modification du plan est justifiée et possible, le gestionnaire de projet devrait obtenir l'accord écrit de tous les intervenants quant au plan révisé. Tel changement/décision doit être **bien documenté** (et **crédible**) et faire partie du plan.

Une communication réelle et continue

Le « **flash reports** » - un document bref dont l'objectif est de fournir une vision macro et synthétique du projet à une date donnée (chaque jour/semaine/mois) :

les tâches réalisées et les décisions importantes (depuis le dernier flash), les points à discuter, les risques liés au projet et les prochaines réunions, et les actions importantes à venir.

Ca peut être comme un blog (sur le web/wiki)

Le Compte-Rendu de Réunion (CRR) - le résumé des discussions relatives à une assemblée ou une réunion, qui comprend également les décisions prises et les actions à mettre en œuvre:

LES INFORMATIONS BASIQUES, LES SUJETS ABORDÉS ET LES DISCUSSIONS,
LES ACTIONS QUI EN DÉCOULENT, LA PROCHAINE RÉUNION

*Un CRR doit être précédé (qq jours avant) par l'envoi d'un
Ordre Du Jour (ODJ)*

Une communication réelle et continue

Le "Change Request" - une "demande de changement / de modification" émanant du client, qui a des impacts en terme de charge de travail et de planning:

I.travail supplémentaire

II.remettre en cause tout ou partie du travail effectué

III.décrire le nouveau fonctionnement souhaité par le client (ce qui permettra de mettre à jour les spécifications)

IV.valider les décalages du planning et les coûts liés à ces demandes (si le client valide le « Change Request »)

V.trace écrite de toutes les demandes supplémentaires qu'a faites le client.

Ils ne sont pas censés être trop nombreux dans un projet

Une communication réelle et continue

TRELLO - <https://trello.com>



Une perte minimale de temps (et de concentration)

Attention!

La **procrastination** - la tendance pathologique à remettre systématiquement au lendemain quelques actions. Cette tendance apparaît souvent au cours des études dès que la personne doit gérer elle-même son activité et prendre la responsabilité de sa production.

Le **perfectionnisme** - la tendance d'une personne à estimer inacceptable un travail qui ne s'approche pas de la perfection. Comme il est rare de pouvoir atteindre la perfection autrement que par essais et erreurs et que la personne n'accepte pas l'idée de produire un résultat imparfait, elle contourne le problème en ne faisant rien.

En voulant **trop planifier**, on risque une perte de temps importante

Le soutien des intervenants (pour toute la durée du projet)

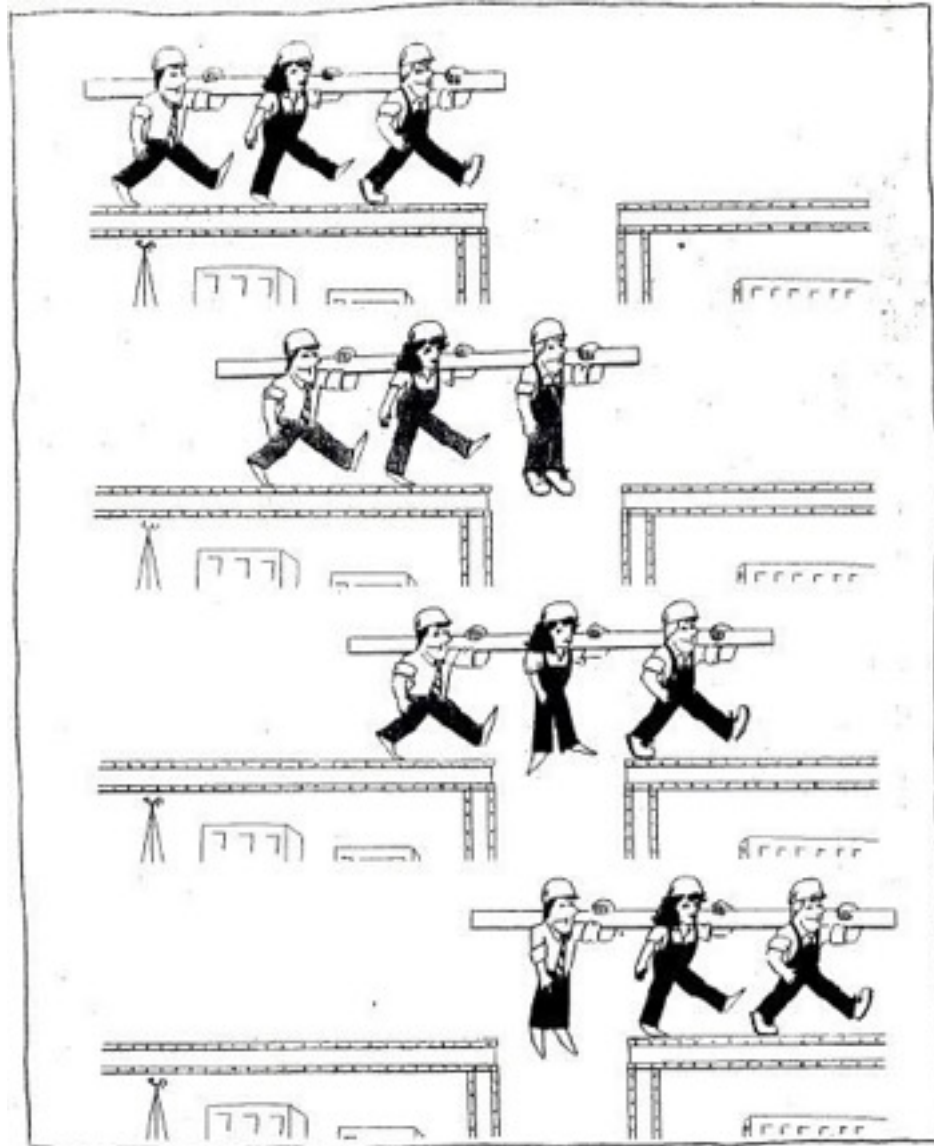
« Les hommes se plaisent à penser qu'ils peuvent se débrouiller seuls, mais l'homme, le vrai, sait que rien ne vaut le soutien et les encouragements d'une bonne équipe. »

[Tim Allen. Acteur américain (1953-)]

« Le travail d'équipe est essentiel. En cas d'erreur, ça permet d'accuser quelqu'un d'autre. »

[Anon]

Le soutien des intervenants (pour toute la durée du projet)



Définition:

Le Génie Logiciel est une **discipline d'ingénierie** qui est concernée par tous les aspects de la production du logiciel (de la spécification à la maintenance)

« It should be noted that no **ethically-trained software engineer** would ever consent to write a “DestroyBaghdad” procedure. Basic professional ethics would instead require him to write a “DestroyCity” procedure, to which “Baghdad” could be given as a parameter. »

[Nathaniel S. Borenstein, 1957 -] *(MIME protocol pour email)*

Cycle de vie d'un logiciel

Le « **cycle de vie d'un logiciel** » (*software life-cycle*), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition.

L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la **validation** du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la **vérification** du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

L'origine de ce découpage provient du constat que **les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt** et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Cycle de vie d'un logiciel

Le cycle de vie du logiciel comprend généralement a minima les activités suivantes :

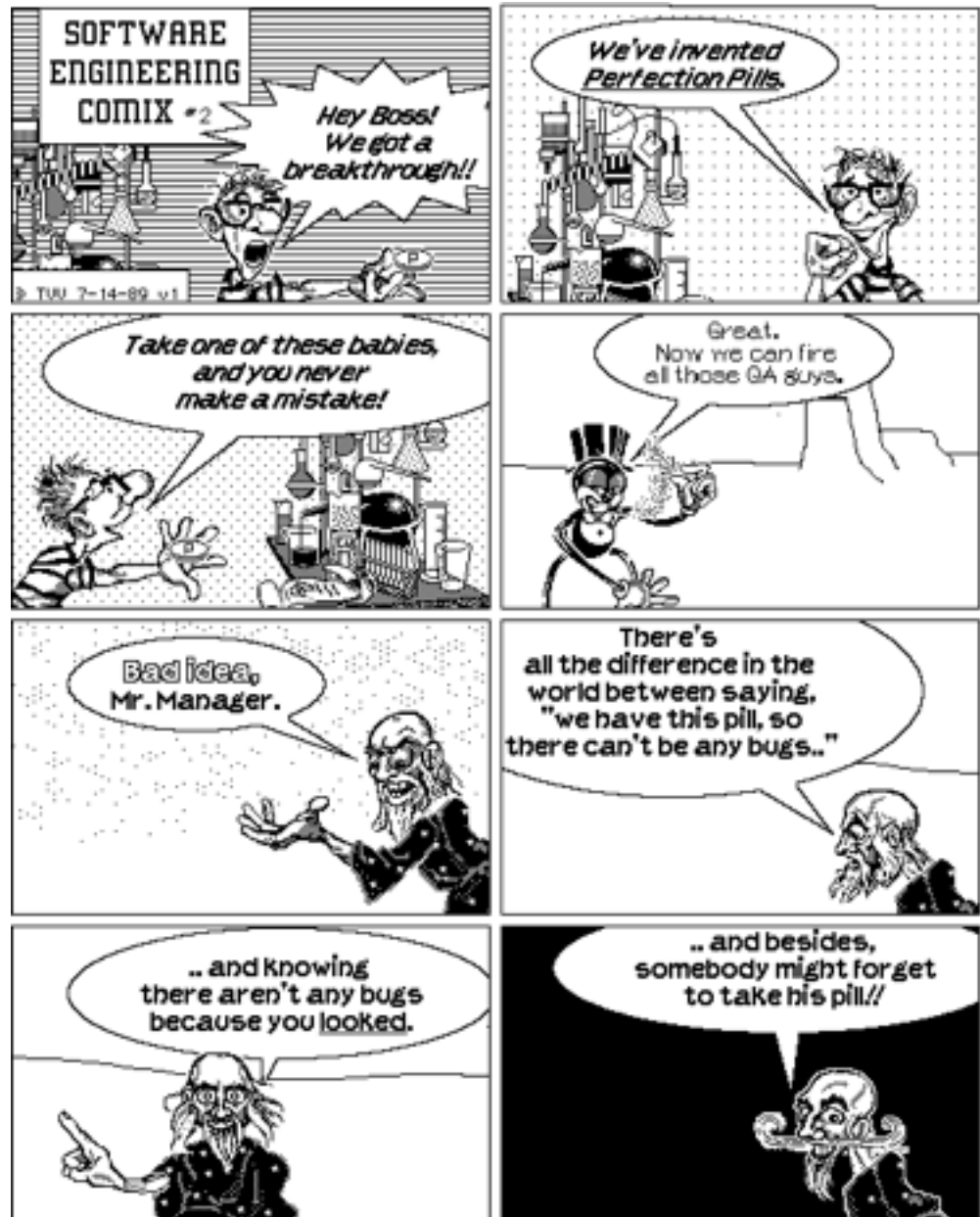
- I. Définition des objectifs,
- II. Analyse des besoins et faisabilité (et **validation**)
- III. Conception générale (et **vérification**)
- IV. Conception détaillée (et **vérification**)
- V. Codage (Implémentation ou programmation),
- VI. Tests** unitaires
- VII. Intégration et *tests d'intégration*
- VIII. Qualification et *tests de validation* - la conformité du logiciel aux spécifications.
- IX. Documentation (pour chaque activité)
- X. Mise en production,
- XI. Maintenance

La séquence et la présence de chacune de ces activités dans le cycle de vie dépend du choix d'un modèle de cycle de vie entre le client et l'équipe de développement.

Pourquoi **tester**?

« L'erreur est humaine, le pardon est divin »

[Alexander Pope 1688 – 1744]



Cycle de vie d'un logiciel: gestion des versions

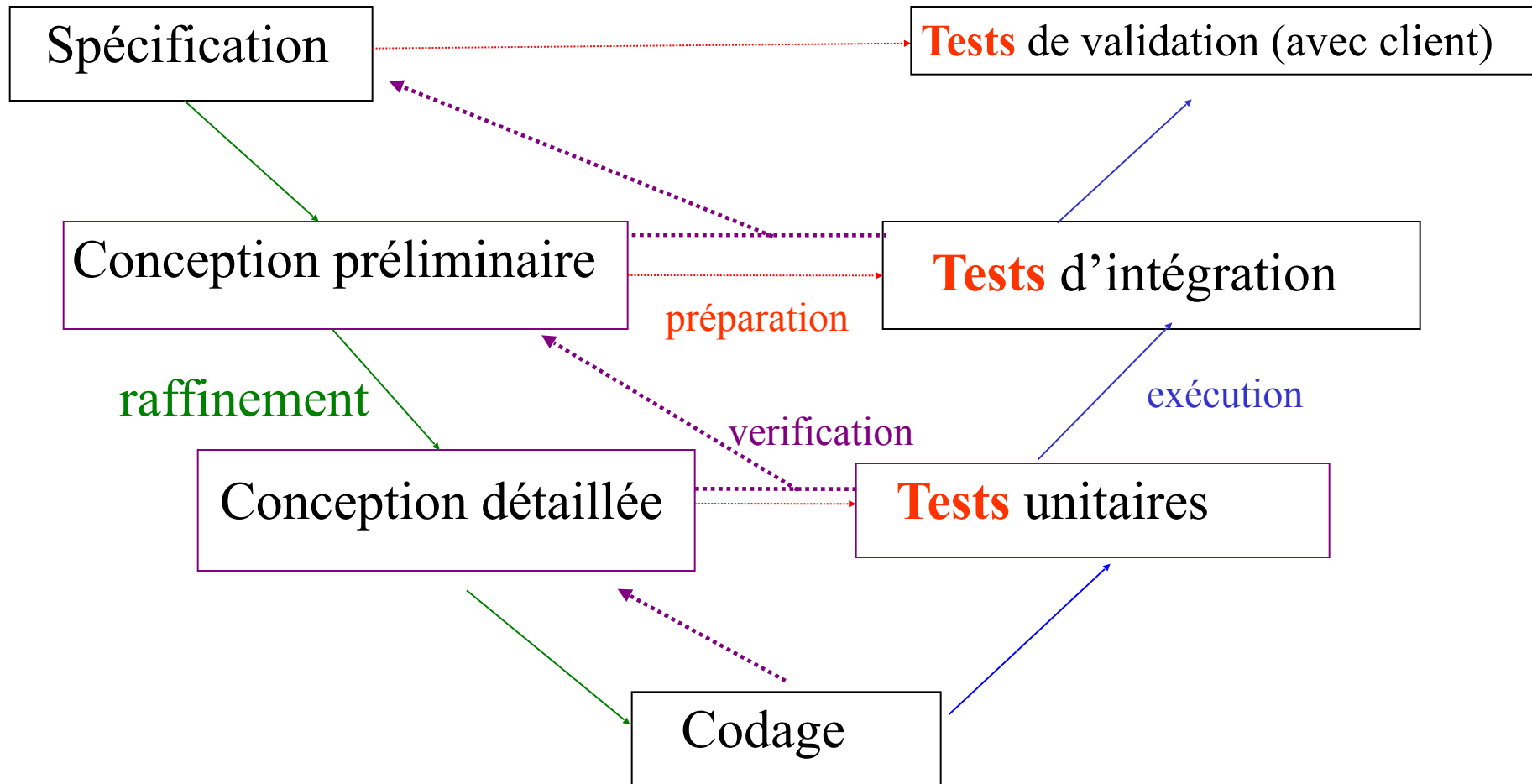
La **gestion de versions** (*revision/version control*) est une activité qui consiste à maintenir l'ensemble des versions d'un logiciel. Elle est surtout concernée par le code source; mais elle peut être utilisée pour tout type de document informatique:

- I. Définition des objectifs,
- II. Analyse des besoins et faisabilité (et validation)
- III. Conception générale (et vérification)
- IV. Conception détaillée (et vérification)
- V. Codage (Implémentation ou programmation),
- VI. Tests unitaires
- VII. Intégration et *tests d'intégration* ...

Il faut maintenir une cohérence (de versions) entre le code, les tests, les spécifications ...

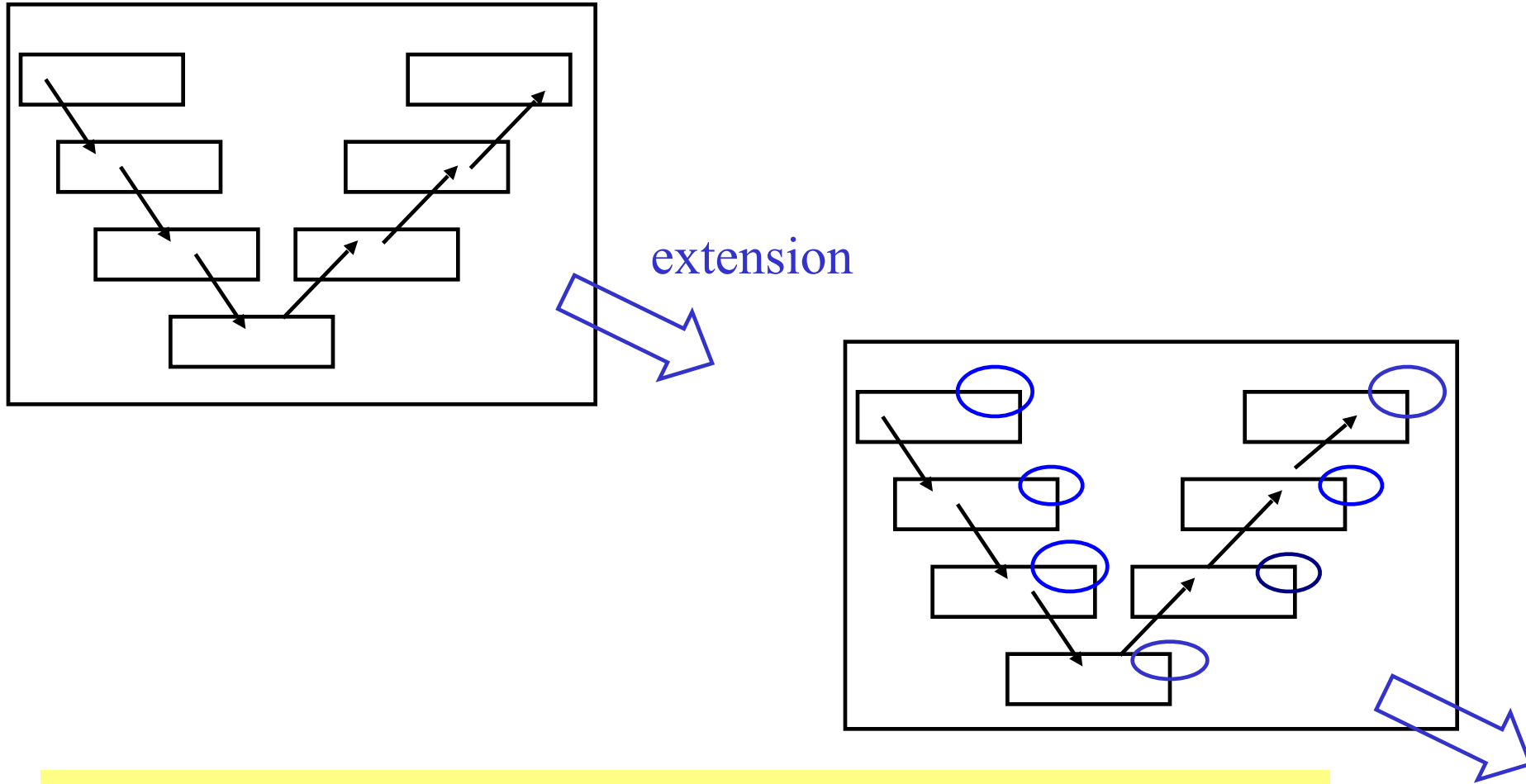
Les logiciels de contrôle de versions sont nombreux: CVS, RCS, SVN, GIT...

Le cycle en V



Il s'agit *peut-être* de la *meilleure approche* pour vos projets?

Le cycle en V – **incremental**



Il s'agit *peut-être* de la *meilleure approche* pour vos projets?

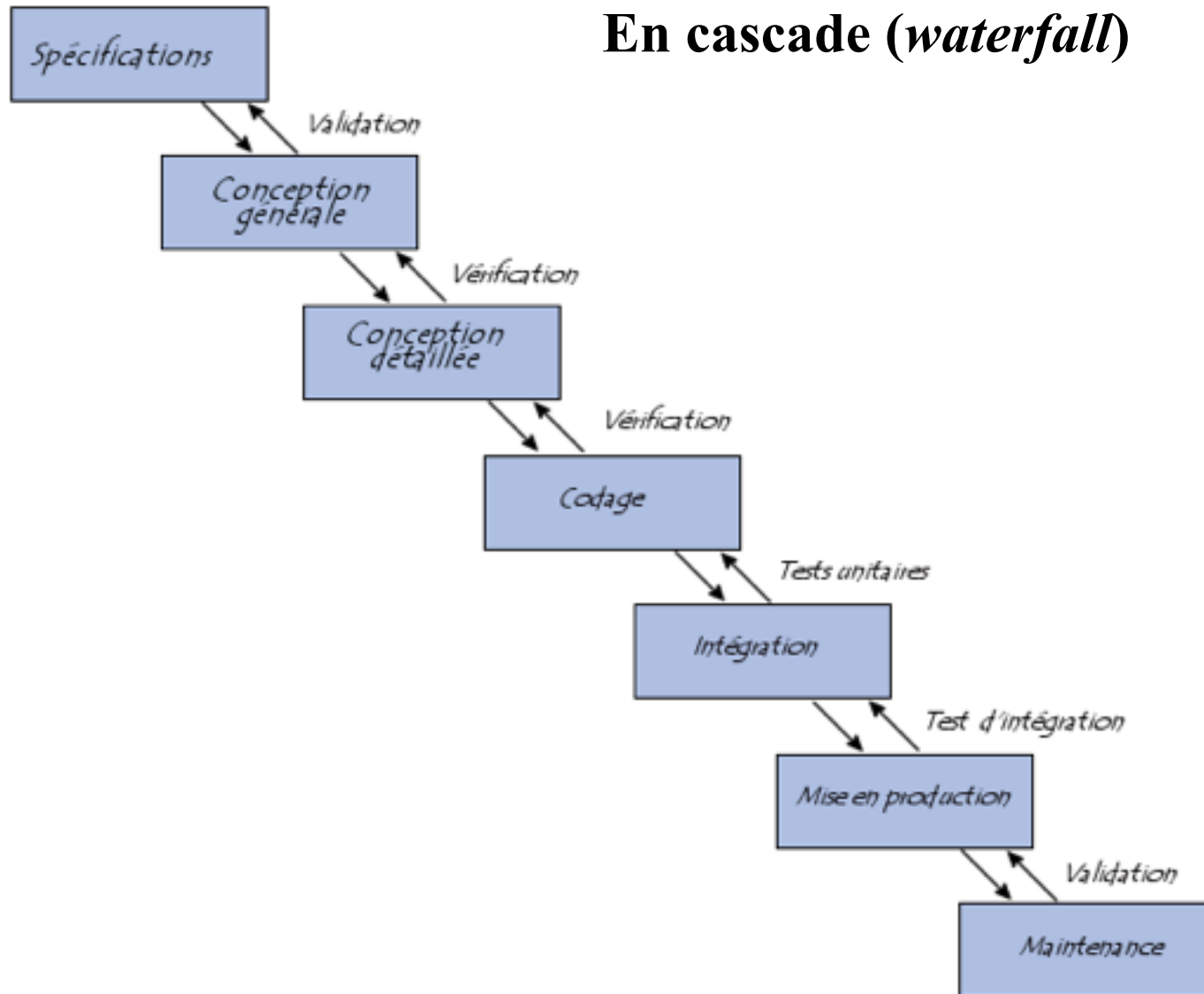
Le cycle en V: Attention - Les travaux ne sont jamais séquentiels

Considérer que les boîtes du V correspondent aussi à des phases d'un cycle de vie revient à dire qu'elles se déroulent en séquence stricte : d'abord toute la Spécification puis toute la Conception puis tout le Codage puis tout le Test

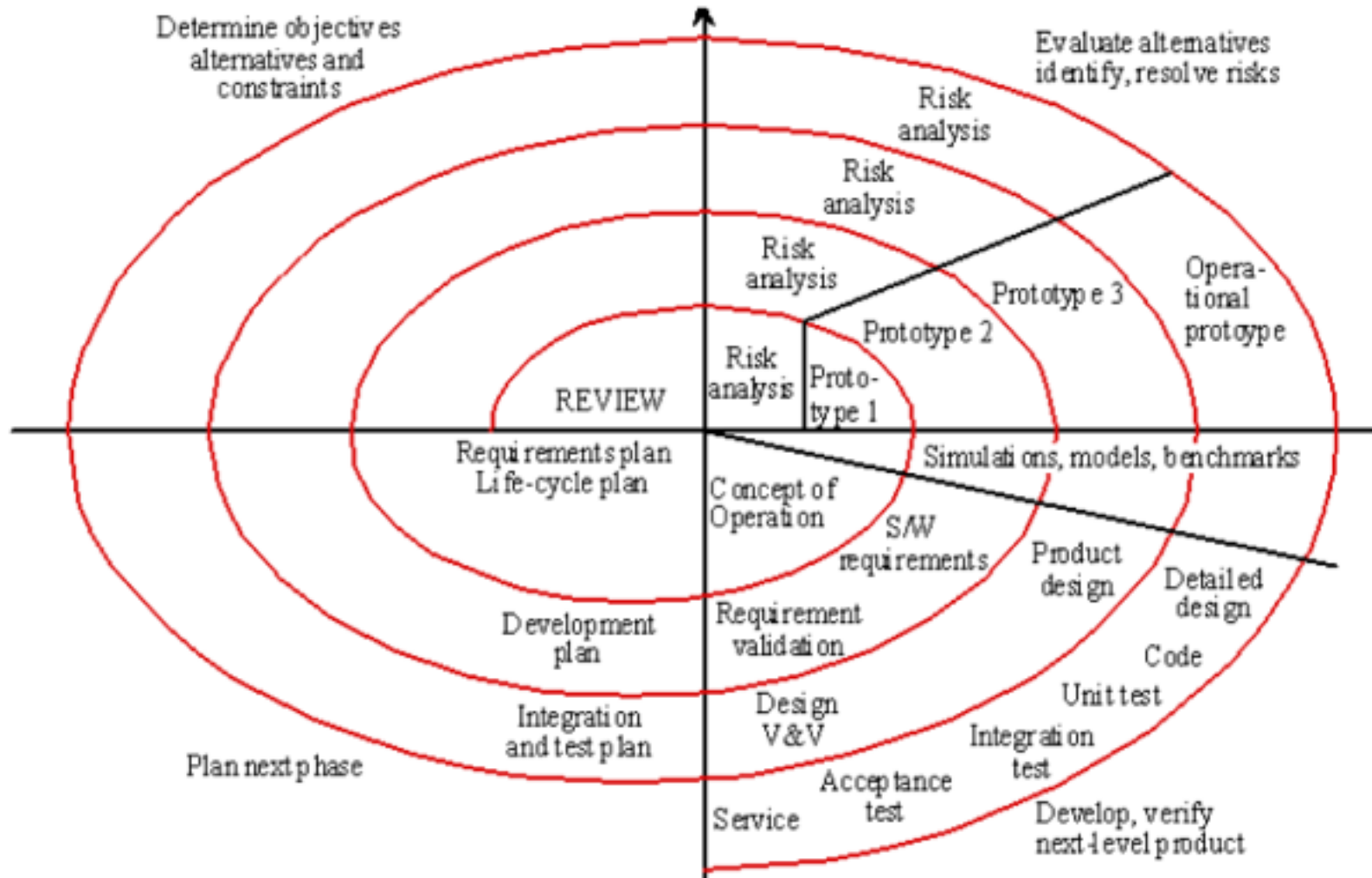
Evidemment aucun logiciel n'a jamais été développé de façon strictement séquentielle.

Par exemple, si on trouve un bug (et on en trouve) dans une "phase" de **test**, le minimum est de refaire des travaux de codage (ou refaire la spécification, ou recoder le test).

En cascade (*waterfall*)



En spirale (Boehm 1988)



Les méthodes “agiles” (réactivité, souplesse) de développement concernent le développement de projet, en réaction aux méthodes traditionnelles (lourdeur).

Parmi ces méthodes, on trouve :

- ASD (Adaptative Software Development)
- Scrum
- Crystal
- FDD (Feature Driven Development)
- DSDM (Dynamic Systems Development Method)
- AM (Agile Modeling)
- XP (eXtreme Programming)

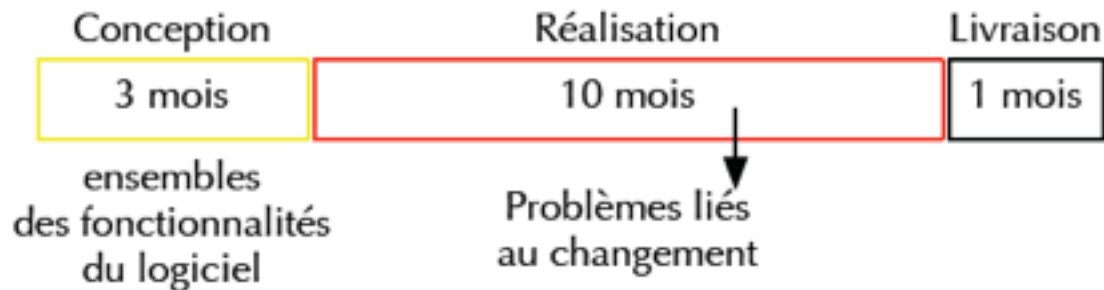


Les méthodes “agiles”: pourquoi?

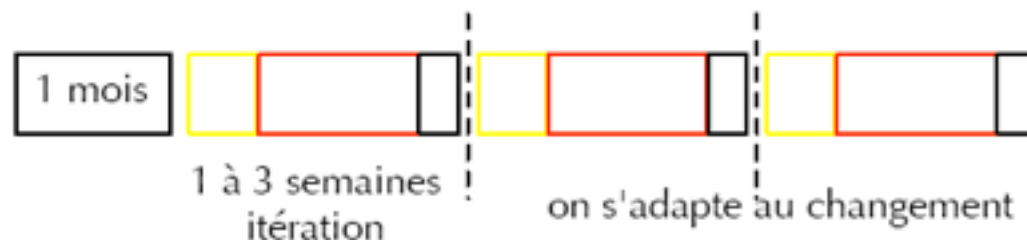
XP (Kent Beck):

- *Le client est intégré à la conception du logiciel et peut influencer sur son évolution*
- *Le client peut utiliser le logiciel et définir de nouvelles fonctionnalités*
- *L'ordre d'implantation des fonctionnalités n'est pas contraignant*

Les méthodes traditionnelles



Les « méthodes agiles »



Il s'agit *peut-être* d'une *approche bien adaptée* à vos projets (?)

Spécification

Quelles sont les fonctions que doit réaliser le logiciel?

- Fonctions à remplir par le logiciel
- Performances requises
- Contraintes de réalisation
- Exigences de Qualité requises
- Interfaces du logiciel avec son environnement humain, matériel ou logiciel
- Préparer les tests de validation

Conception préliminaire

Comment réaliser le logiciel ?

- Choisir une solution
- Définir de manière descendante la structure du logiciel
- Définir les fonctions nominales, les flux de données
- Définir les données (signification, code, type, domaine, appellation mnémonique, ...)
- Préparer les tests d'intégration

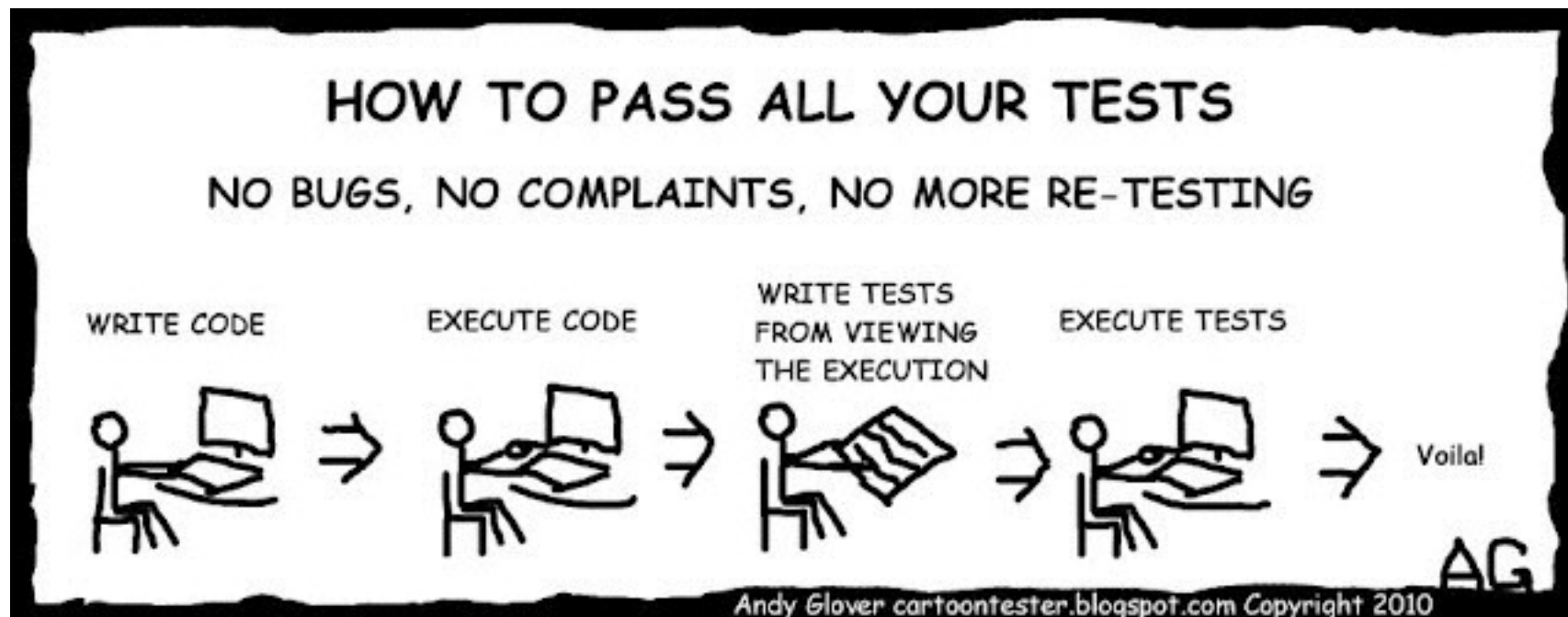
Conception détaillée

- Pour chaque fonction
 - décomposer en traitements élémentaires ou éléments logiciels
 - préciser l'interface de chaque élément
 - préciser et affiner les données d'entrée et de sortie
 - figer les contraintes d'appel et d'enchaînement
 - décrire le traitement nominal et la prise en compte du traitement des exceptions
 - conception d'algorithmes : testabilité, conditions de fin, caractéristiques d'emploi
 - Préparer les tests unitaires

Codage

- Traduire la conception détaillée en code lisible et intelligible
- Inclure les commentaires
- Prévoir les tests unitaires

Comment tester votre système ??



TESTS : DEFINITION

"Technique de contrôle consistant à s'assurer, au moyen de l'exécution d'un programme que son comportement est conforme à des données préétablies"

Tests unitaires/ tests d'intégration

- Exécuter les tests
- Comparer les résultats obtenus aux résultats attendus
- Déterminer le taux de couverture
- Si l'élément n'est pas conforme, engager la procédure de modification prévue (d'abord, il faut trouver le « bug »)

Conception préliminaire – les tests d'integration - pourquoi faire

“2 unit tests, zero integration tests”



Risque sans tests d'intégration: trouver des problèmes après l'ensemble du système est construit, en utilisant des composants fiables qui ont été testés individuellement, (qui peut être coûteux à réparer)

<http://i.imgur.com/qSN5SFR.gifv>

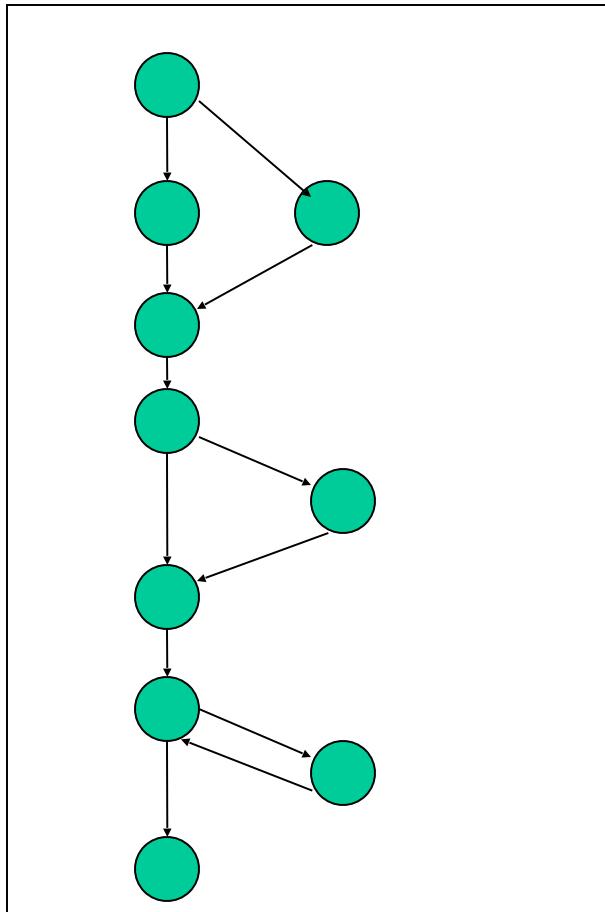
LES TECHNIQUES DE TESTS

- LES TESTS STRUCTURELS (boîte blanche)
- LES TESTS FONCTIONNELS (boîte noire)
- LES TESTS DE ROBUSTESSE
- LES TESTS DE PERFORMANCE
- LES TESTS DE NON REGRESSION
- LES TESTS DE PORTABILITÉ ...

Tests de validation

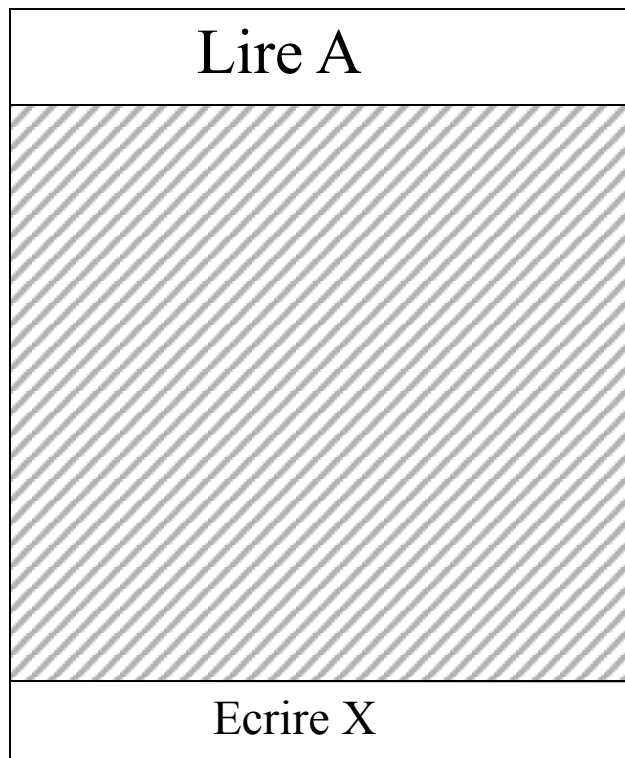
- Installer les moyens nécessaires à la mise en œuvre des tests (plate-forme de validation)
- Exécuter les tests (conformité du logiciel aux **spécifications du logiciel**)
- Recette
- Qualification
- Communication avec le client

TESTS STRUCTURELS



```
Debut
Si A
    alors X = 1
    sinon X = 2
FinSi
Si X=1
    alors X = X -2
FinSi
Tant que X > 0 et X < 4
    faire X = X +1
FinTantQue
Fin
```

TESTS FONCTIONNELS



Entrée : A vrai
Sortie : $X = -1$

Entrée : A faux
Sortie : $X = 4$

Les limites du test

- L'espace des entrées (et des états du système)
 - Les séquences d'exécution
 - Sensibilité aux fautes
-
- Complexité
- Continu
vs
Discret

Espace des entrées: exemple

- Lire 3 valeurs entières. Ces trois valeurs sont interprétées comme représentant les longueurs des côtés d'un triangle. Le programme imprime un message qui établit que le triangle est isocèle, équilatéral ou scalène.

1 triangle = 3 segments

1 segment = 2 points

1 écran = 1024 x 768 points = 786432 points

1 segment = 786432 x 786432 = 618475290624

1 triangle = 1855425871872

$> 1,8 \cdot 10^{12}$

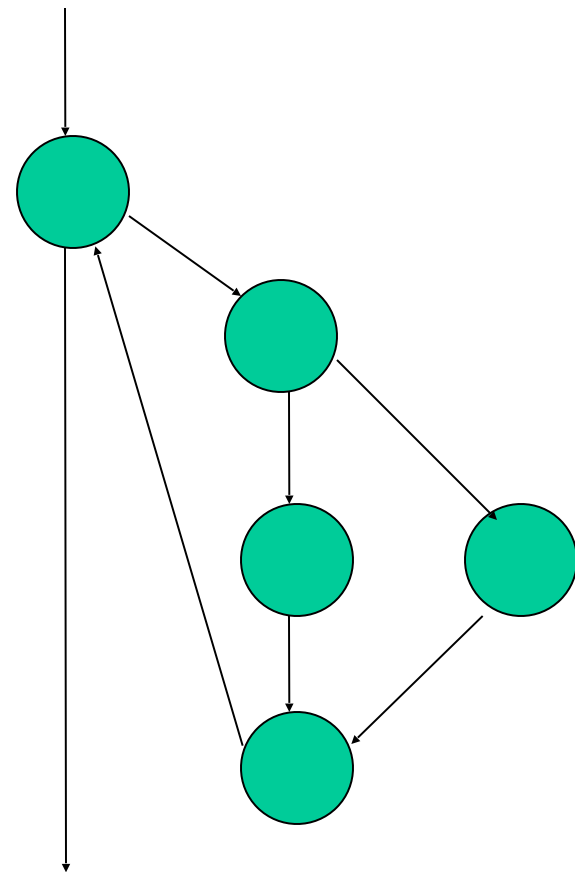
Les séquences d'exécution

```
for (int i=0; i<n; ++i) {  
  if (a.get(i) == b.get(i))  
    x[i] = x[i] + 100;  
  else  
    x[i] = x[i] / 2;  
}
```

Les séquences d'exécution

Nbre d'itérations	Nbre de chemins
1	3
2	5
3	9
10	1025
20	1 048 577
60	$> 1,15 \cdot 10^{18}$

INT/LOR/GLCP/J-L RAFFY



Sensibilité aux fautes

```
short scale(short j) {  
j = j - 1; // devrait être j = j+1  
j = j/30000;  
return j;  
}
```

Six valeurs sont « buggy »

Autres limitations

- Tester un programme permet de montrer la présence de fautes mais en aucun cas leur absence
- Les tests basés sur une implémentation ne peuvent pas révéler des omissions car le code manquant ne peut pas être testé
- On ne peut jamais être sûr qu'un système de test est correct

Génie Logiciel dans les projets informatiques n'est pas seulement le choix d'un cycle de vie ... il y a aussi le problème de gestion



The management choice:

Managers who don't understand software ... or software engineers who don't understand management?



Le cycle de vie (typique) d'un projet informatique?



Comment le client a exprimé son besoin



Comment DAS l'a compris



Comment l'analyste EDI l'a conçu



Comment le programmeur EDI l'a développé



Ce que les beta testeurs ont reçu



Comment le consultant l'a décrit



Comment le projet fut documenté



Ce que PMI a installé



Ce que le client a payé



Ce qui a été supporté



La pub faite par le Marketing



Quand le produit a été livré



Ce dont le client avait réellement besoin



L'impact de l'effet papillon sur le produit



Le Plan de Crise



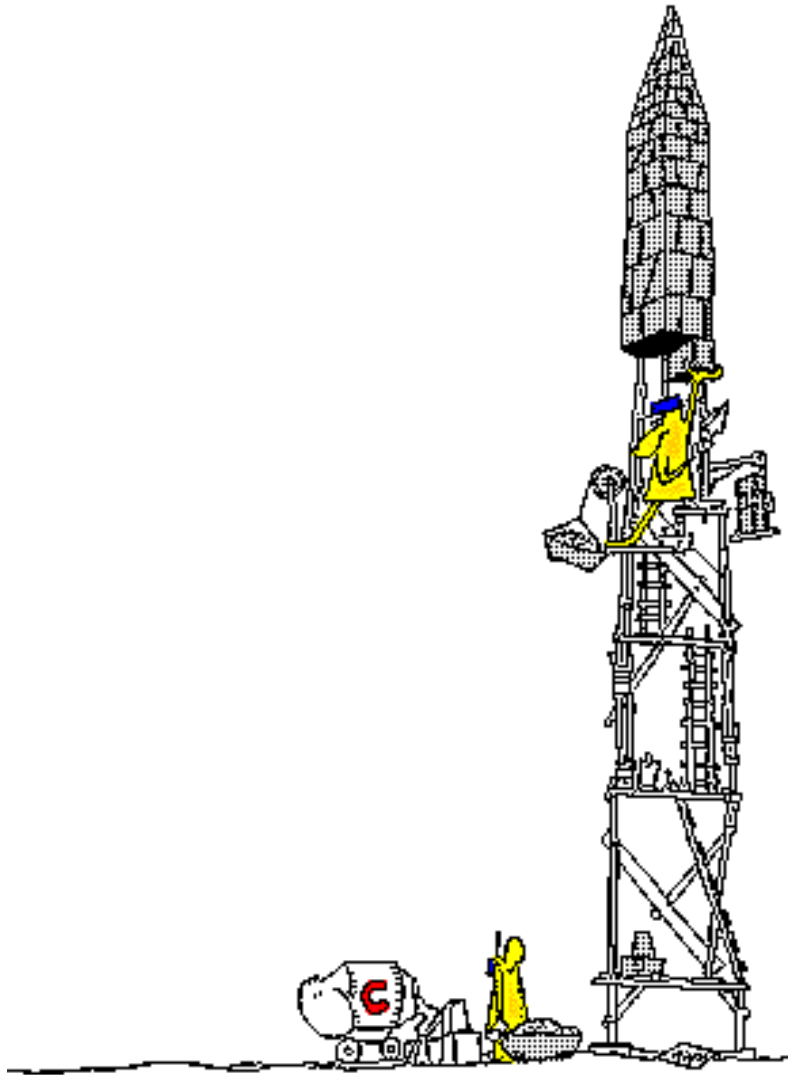
La version Open Source



Comment le produit a réagi à la charge

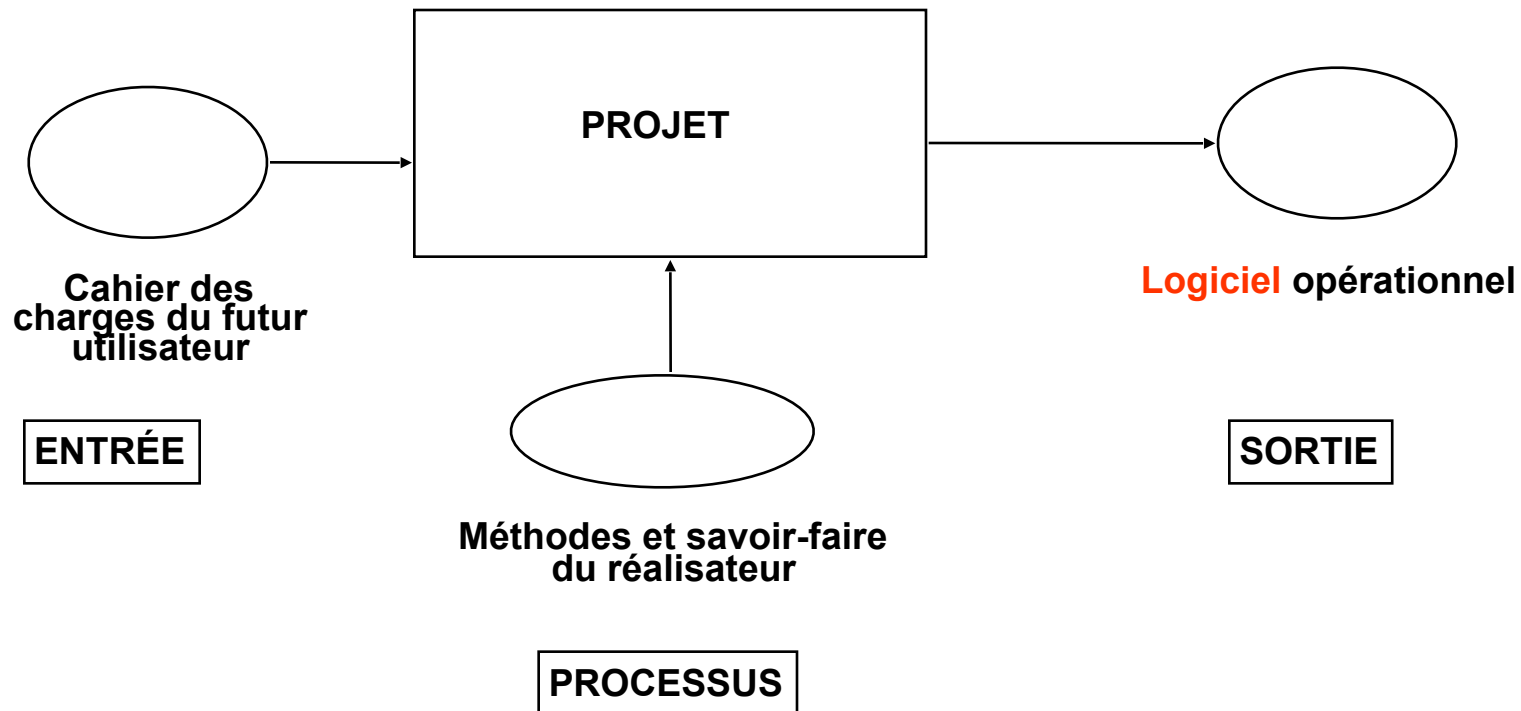


Comment les correctifs ont été appliqués



Avec les logiciels on peut construire notre système de façon *Top Down !!!*

Projet informatique



Spécificités d'un projet informatique:

- Résultats immatériels
- Importance cruciale des phases de conception
- Intangibilité : absence de visibilité explicite d'avancement du projet
- Processus de développement du logiciel pas toujours bien maîtrisé
- Grands projets logiciels parfois sans précédent - on expérimente grandeur nature, risques mal analysés
- Pérennité des technologies

Ingénierie de projets / environnement

- Environnement d'un projet informatique (1)

CHANTIER

- Ouvrage : Produit à réaliser, comportant le code, la documentation, les jeux de tests
- Oeuvre : Ensemble des actions à effectuer pour mener à bien l'ouvrage

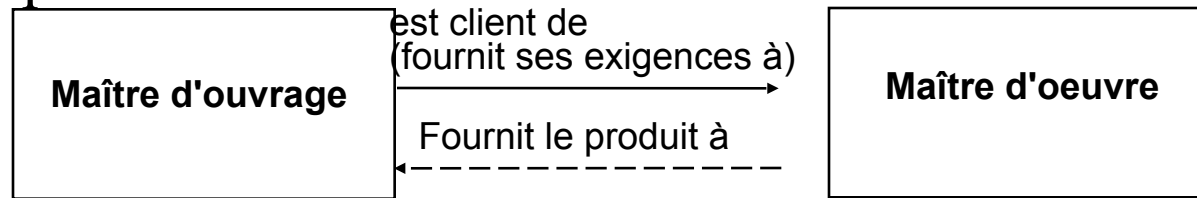
ACTEURS

- Maître d'ouvrage : Commanditaire, client
- Maître d'oeuvre : Réalisateur de l'ouvrage, fournisseur

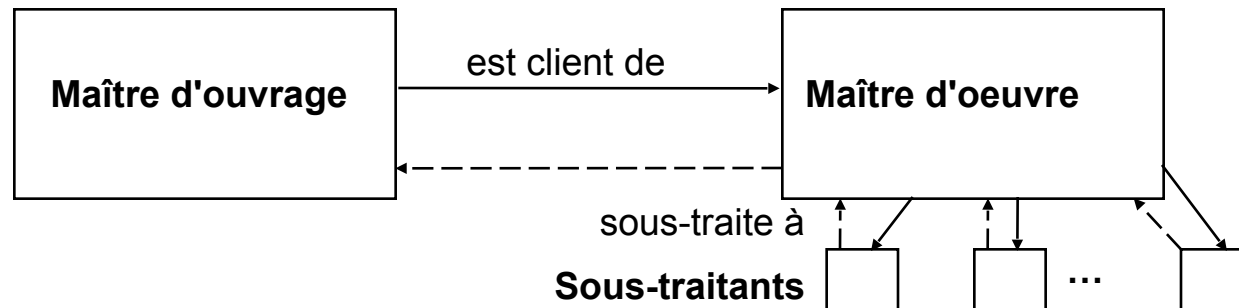
Ingénierie de projets / environnement

- Environnement d'un projet informatique (2)

- Simple

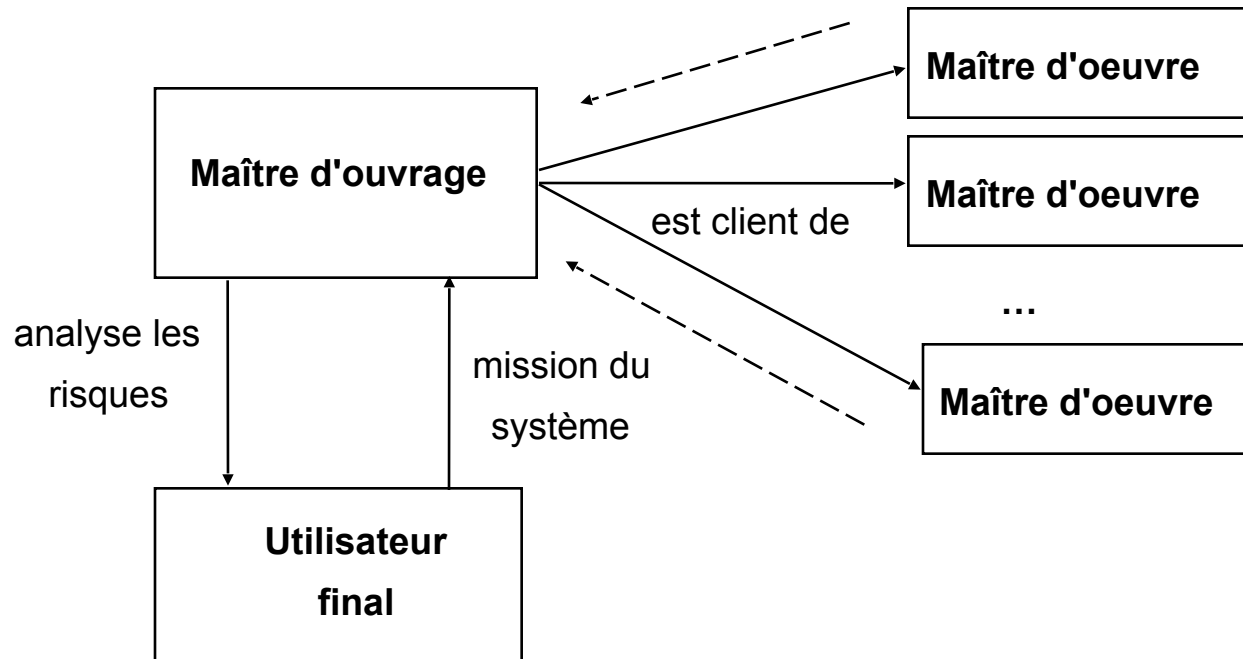


- Avec sous-traitants



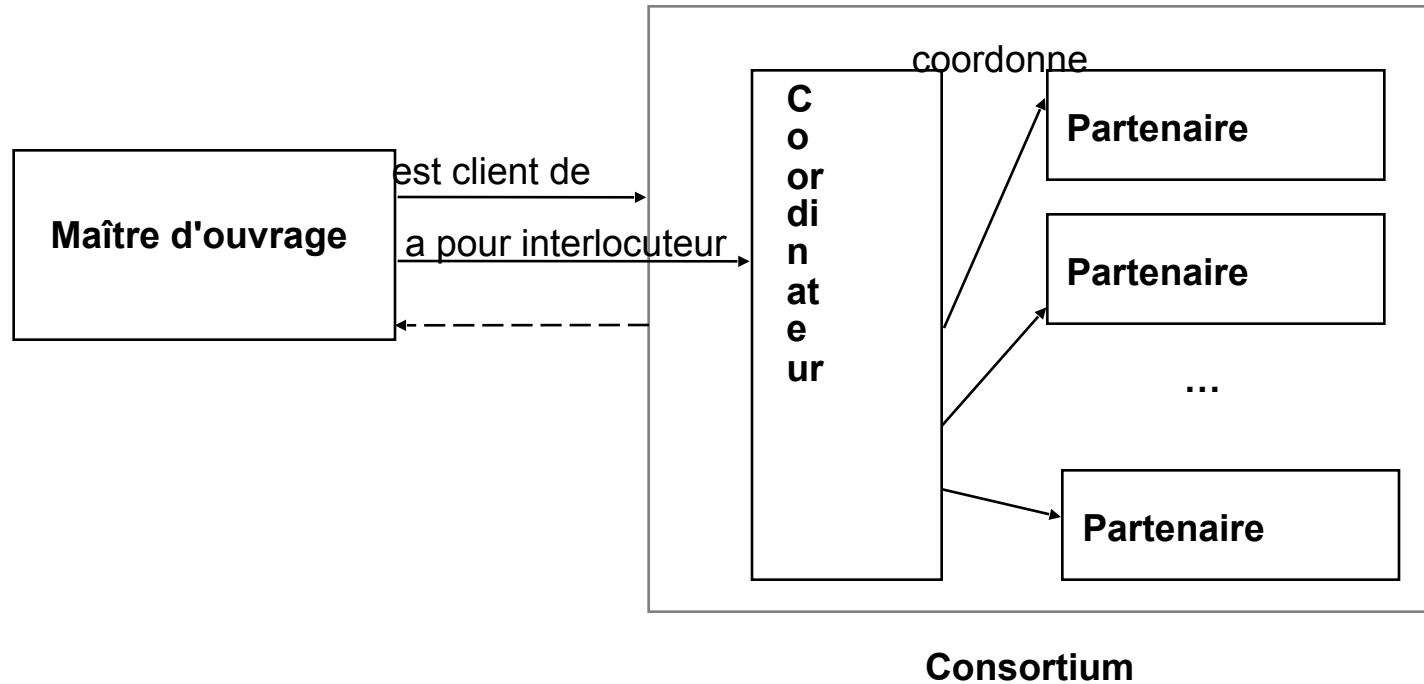
Ingénierie de projets / environnement

- Multi-fournisseurs et utilisateur final



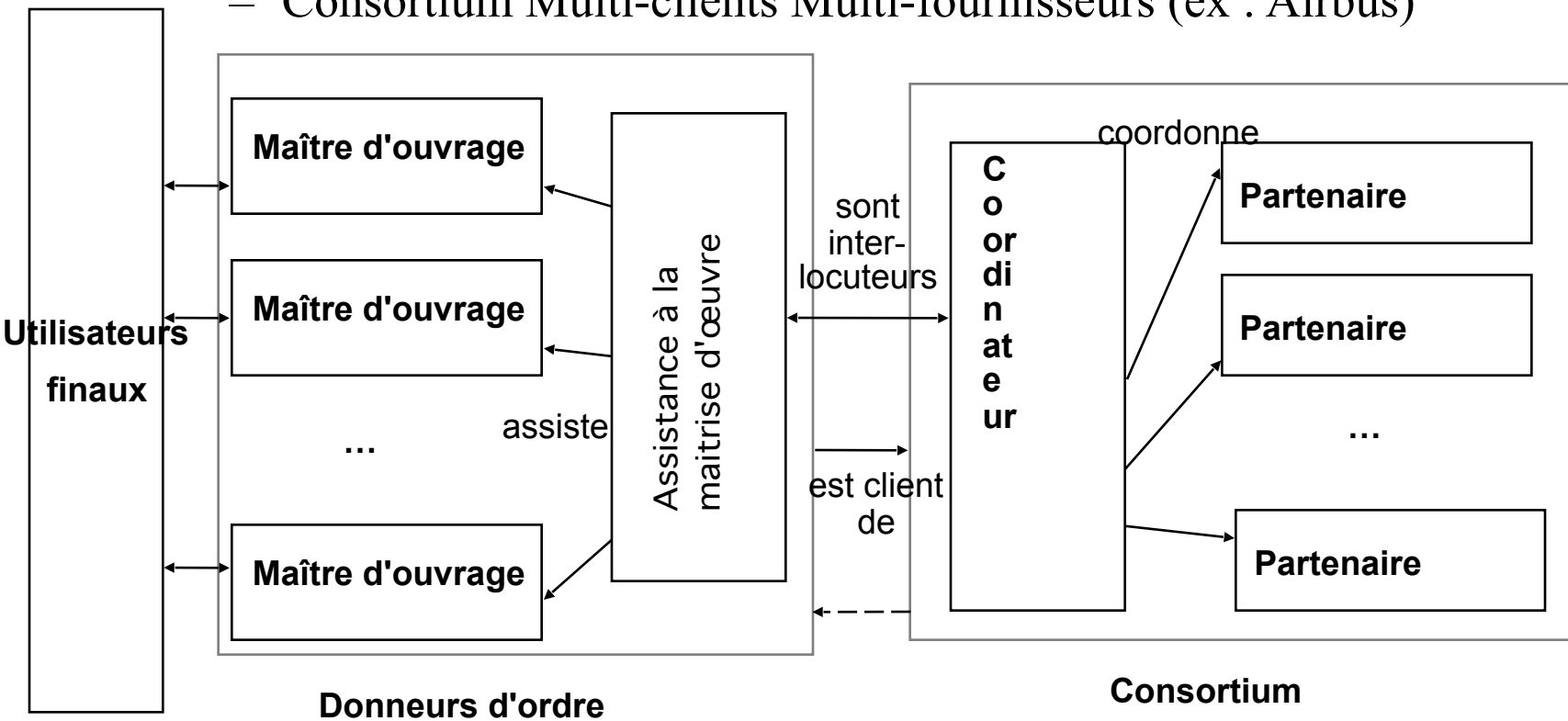
Ingénierie de projets / environnement

- Coopératif (ex : marchés publics)



Ingénierie de projets / environnement

- Consortium Multi-clients Multi-fournisseurs (ex : Airbus)



Ingénierie de projets / Conclusion

- Ingénierie de projet complexe
 - Nombreux acteurs
 - Complexité des logiciels
 - découpage en sous-projets
 - intégration des logiciels
 - Des problèmes à résoudre
 - allouer au mieux les ressources disponibles
 - éviter qu'un événement mineur grippe un énorme dispositif
 - analyse de risques
 - criticité des ressources
 - synchroniser au mieux différentes tâches dans le temps
 - interdépendantes mais qui peuvent être menées en //

Etude de cas - bibliothèque

Cahier Des Charges

A) *Contexte et Historique*

B) Description de la demande

Les objectifs

Produit(s) du projet - « bibliothèque »
Les fonctions du produit
Critères d'acceptabilité et de réception

C) Contraintes

Contraintes de coûts

Contraintes de délais

Contraintes techniques

Clauses juridiques, etc.

D) *Déroulement du projet- Planification des Phases et Ressources*

E) *Authentification - Date et signature du chef de projet et du maître d'ouvrage.*

F) *Annexes - Lister et joindre au cahier des charges les éventuels documents que le client peut mettre à disposition.*

Zoom (pour l'étude de cas)



Etude de cas - bibliothèque

1. Cahier Des Charges
2. Analyse
3. Cycle en V - les tests
4. Documentation
5. Débogueur
6. Conseils



Etude de cas - bibliothèque (des ouvrages)

Les fonctions du produit

Les fonctionnalités qui devront être offertes par l'application sont :

- *Afficher (lister) l'ensemble des ouvrages*
- *Ajouter un ouvrage*
- *Suppression des ouvrages*
- *La persistance des données et gestion de la session*

Contraintes techniques

- Les fonctionnalités décrites dans le sujet sont à implanter dans un premier temps avec une interface mode “terminal”,
- Le langage de développement choisi est le Java.

Critères d'acceptabilité et de réception

- L'application doit être **robuste** en cas de:
 - Terminaison imprévue
 - Erreurs d'entrée par l'utilisateur
- L'application doit être **performante** (interaction entre utilisateur et application en temps réel *sans délai*)
- L'interface de l'application doit être conforme à la maquette fournie ci-dessous :

```
-----  
Bibliothèque - mode terminal  
-----
```

```
1 - lister ouvrages  
2 - ajout  
3 - suppression  
4 - exit  
Sélection:
```

Les extensions (prévues)

I Interface - graphique

II Mode Web

III Fonctionnalité *Undo*— correction des erreurs d'utilisateur

IV Recherche *en utilisant plusieurs critères*

Extension I: Interface - graphique

conforme à la maquette :



Extension II - mode web

1. Une consultation en mode web à travers l'utilisation d'un navigateur
2. L'accès concurrent aux données de l'application côté serveur.
3. Cette extension devra être utilisable avec un client navigateur indépendant de l'OS. On utilisera un serveur Apache sous Linux.
4. Une maquette de l'IHM (interface homme-machine) de l'extension ???

Extension III - *undo*

1. La maquette de l'IHM (interface homme-machine) de l'extension envisagée est fournie ci-dessous.

```
-----  
Bibliothèque - mode terminal  
-----  
  
1 - lister ouvrages  
2 - ajout  
3 - suppression  
4 - undo  
5 - exit  
Sélection:
```

Extension IV - Recherche *en utilisant plusieurs critères*

Il faut communiquer avec l'utilisateur pour avoir plus d'information sur la fonctionnalité et l'interface.

S'il y a assez de temps, on peut développer quelques prototypes différents afin d'aider la communication avec l'utilisateur en démontrant des « options »

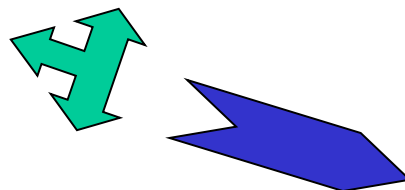
Analyse

Une relecture détaillée du **cahier des charges** et l'**énoncé du sujet initial** avec le "*client*" de l'application est souvent nécessaire pour :

- Donner des réponses à toutes les questions,
- Lever toutes les ambiguïtés.

Client(s)
Contexte du Problème

Analyste(s)
Spécification (Modèles)



Les documents de
spécification ont vocation à
être validés par le client.

L'objectif de cette phase est de décrire
précisément et exhaustivement les
fonctionnalités offertes par l'application

Analyse

Les fonctionnalités qui devront être offertes par l'application sont :

Ajouter un ouvrage, Supprimer un ouvrage, Afficher l'ensemble des ouvrages, La persistance des données

Questions:

- Qu'est-ce qu'un ouvrage ? Connaît-on le nombre maximum d'ouvrages ?
- Fonctionnalités offertes à qui (et quand)?
- Ajouter/Supprimer/ où?/ Afficher comment?
- Les données : quelle information sauvegarder (comment/où et quand)

Analyse - Spécification

Un ouvrage est en général composé des informations suivantes : le titre, l'auteur (principal), l'année d'édition, ...

La collection d'ouvrages est de taille quelconque.

Tous les utilisateurs ont le même privilège et ils n'ont pas à s'authentifier.

On considère qu'à un moment donné, il n'y a qu'un seul utilisateur de l'application.

L'affichage doit être sur l'écran (mode terminal)

Il faut créer (au moins) un fichier pour la collection, c'est-à-dire s'assurer à minima de la persistance des données. Le fichier doit être localisé sur le compte de l'utilisateur.

Analyse - Description des fonctionnalités

Ajouter un ouvrage –

On vérifiera au préalable que l'ouvrage n'existe pas déjà pour ne pas créer de doublon, et on refusera la création d'un ouvrage lorsque les informations sont incomplètes.

L'affichage doit être conforme à la maquette fournie ci-dessous :

```
1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 2
```

```
Ajout d'un ouvrage
```

```
-----
```

```
Entrer l'auteur: author
```

```
Entrer le titre: title
```

```
Entrer l'année: 2016
```

Analyse - Description des fonctionnalités

Supprimer un ouvrage -

Pour simplifier, on considère qu'on supprime tous les ouvrages trouvés selon tous les critères proposés, et pour simplifier l'implantation, un seul critère sera utilisé dans la suite, le nom de l'auteur.

L'affichage doit être conforme à la maquette fournie ci-dessous :

```
1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 3
```

```
Suppression des ouvrages d'un auteur
```

```
-----
```

```
Entrer l'auteur: author
```

```
1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
```

Analyse - Description des fonctionnalités

Lister ouvrages –

On affichera tous les ouvrages dans la bibliothèque (ordre alphabétique par auteur, puis titre, puis année).

L'affichage doit être conforme à la maquette fournie ci-dessous :

```
1 - lister ouvrages
```

```
2 - ajout
```

```
3 - suppression
```

```
4 - exit
```

```
Sélection: 1
```

```
Liste des ouvrages
```

```
-----
```

```
anyone, some title, 1
```

```
author1, title1, 2000
```

```
author1, title1, 2001
```

```
author1, title2, 1999
```

```
bill gates, i'm very rich, 2016
```

Analyse - Description des fonctionnalités

Persistence des données –

Au début de la session la liste d'ouvrages est cohérente avec le fichier `bibliotheque.data`:

```
sr.java.util.ArrayList<Ouvrage> srgestionOuvrages.Ouvrage lannee lauteur ljava/lang/String; ltitre ~xptanyonet  
some titles ~tauthor1 title1sq ~tauthor2 title2sq ~t  
bill gates  
i'm very rich
```

Liste des ouvrages

anyone, some title, 1
author1, title1, 2001
author2, title2, 2002
bill gates, i'm very rich, 2016

Comment tester la coherence?

1.afficher

2.exit

3.re-execute application

4.afficher

Analyse - Robustesse

Terminaison imprévue

Le fichier « bibliotheque.data » est mis à jour pendant l'exécution du système. Donc, si l'exécution est terminée pendant une opération (ajouter ou supprimer) nous ne perdons que l'effet de cette opération sans risque de perdre toutes les opérations précédentes.

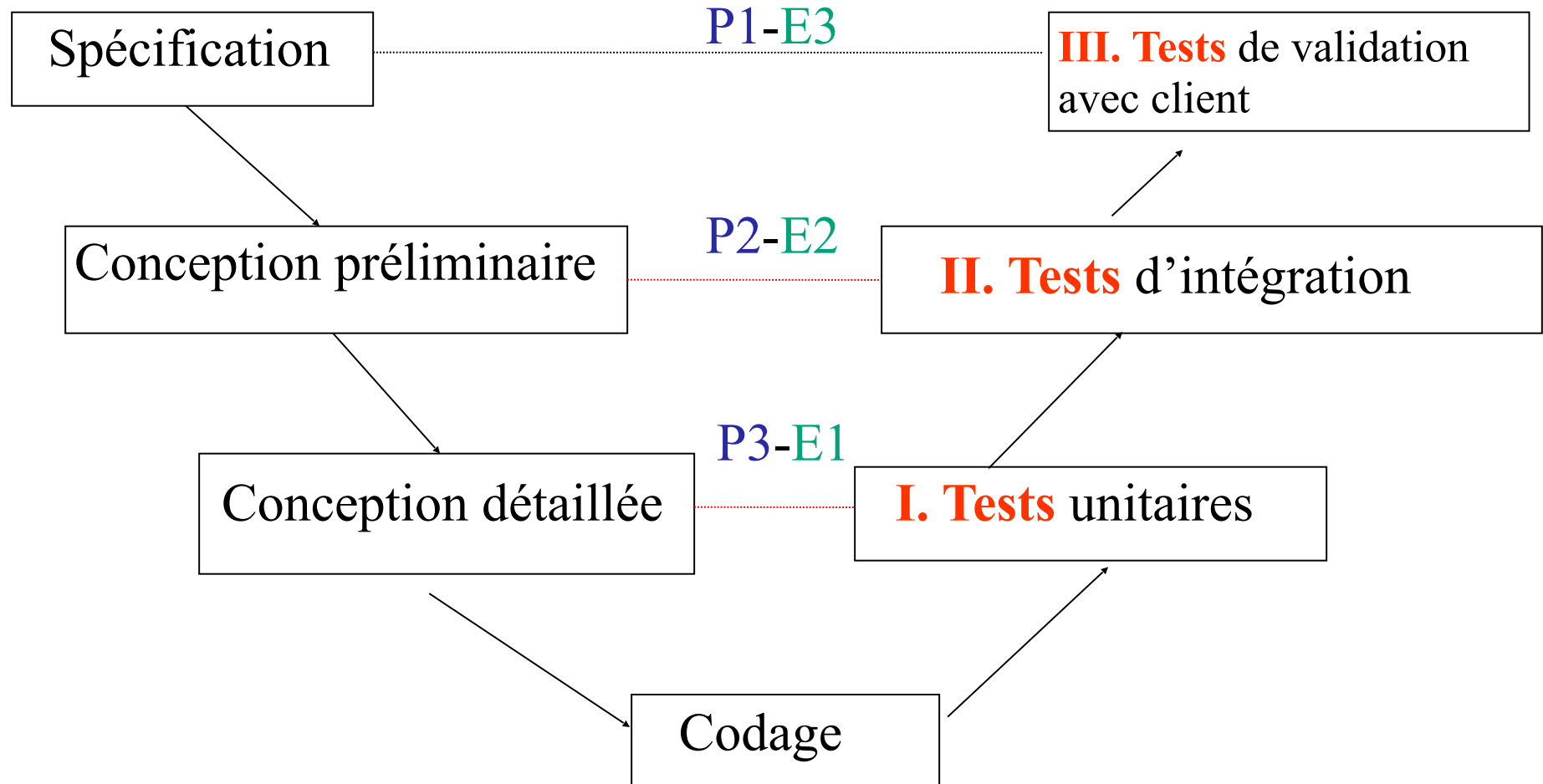
Erreurs d'entrée par l'utilisateur

Il faut afficher un message d'erreur et donner une 2ème chance à l'utilisateur

Analyse - Performance

L'application doit être **performante** (interaction entre utilisateur et application en temps réel *sans délai*):

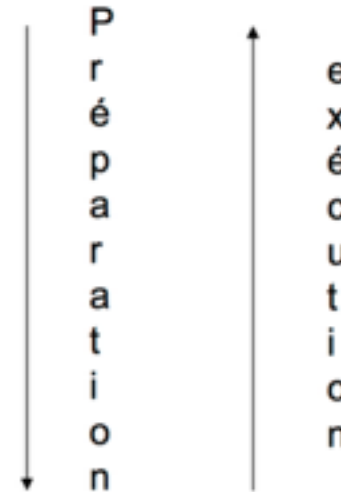
la collection est mis-à-jour sans prendre trop de temps, pour une collection raisonnable (e.g. 1 sec, collection de 30000 livres)

Séquence de **préparation** et **exécution** des **tests**

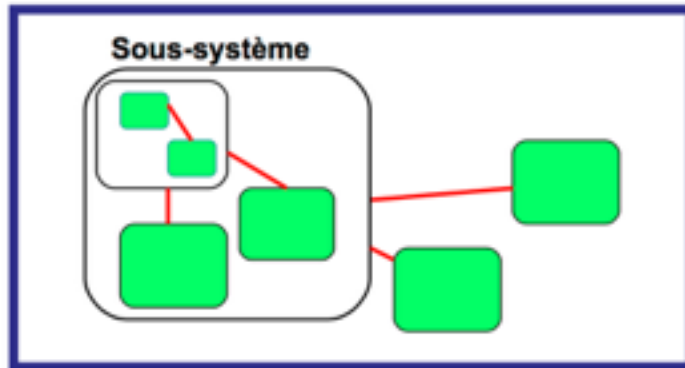
Séquence de **préparation** et **exécution** des **tests**

Niveaux de test - pour chaque (sous)système

- Tests de **validation**
- Tests d'**intégration**
- Tests **unitaires**



Système



Conception détaillée

Plan de tests de validation

Dès la spécification, on doit prévoir les tests de validation.

Exemple de tests à prévoir pour la **fonctionnalité** "Ajouter un ouvrage" :

Test 1

Donnée : un ouvrage déjà existant à ajouter

Résultat : message d'erreur : « L'ouvrage existe déjà »
la collection ne change pas

Test 2

Donnée : un ouvrage complet/valide (pas déjà existant) à ajouter

Résultat : la collection est mise à jour

Plan de tests de validation "Ajouter un ouvrage" :

Test 3 (robustesse - entrées)

Donnée : le *titre* est ""

Résultat : message d'erreur et 2ème chance

Test 4 (robustesse - entrées)

Donnée : l'*auteur* est ""

Résultat : message d'erreur et 2ème chance

Test 5 (robustesse - entrées)

Donnée : l'*année* n'est pas valide (eg AAA234, -10, 3000)

Résultat : message d'erreur : « donnée non cohérente... »
et 2ème chance – « votre choix : »

Analyse - Plan de tests de validation

Il faut tester (en plus):

Les fonctions - supprimer, afficher

Les limites - la collection d'ouvrages est de taille quelconque.

*Robustesse - terminaison imprévue et corruption de
bibliothèque.data*

Performance – mise à jour max. 1 sec, collection de 30000 livres

Tests de validation : **conception**, implémentation, exécution et résultats

Conception: 3 options complémentaires typiques –

- a. Test fait à la main avec l'interface utilisateur
- b. Test fait par simulation d'entrée au niveau de l'interface utilisateur (à l'aide d'opérations/facilités de l'OS, e.g. redirections/tubes en Unix)
- c. Test fait par simulation d'entrée au niveau de l'interface utilisateur (en utilisant du code écrit par les testeurs)

Tests de validation : conception, implémentation, exécution et résultats



```
Validation_Test_Bibliotheque.java
Bibliotheque ▸ src ▸ tests ▸ Validation_Test_Bibliotheque ▸
package tests;

import gestionOuvrages.Bibliotheque;

* <p> <b> A simple library (bibliotheque) </b> </p>
public class Validation_Test_Bibliotheque {

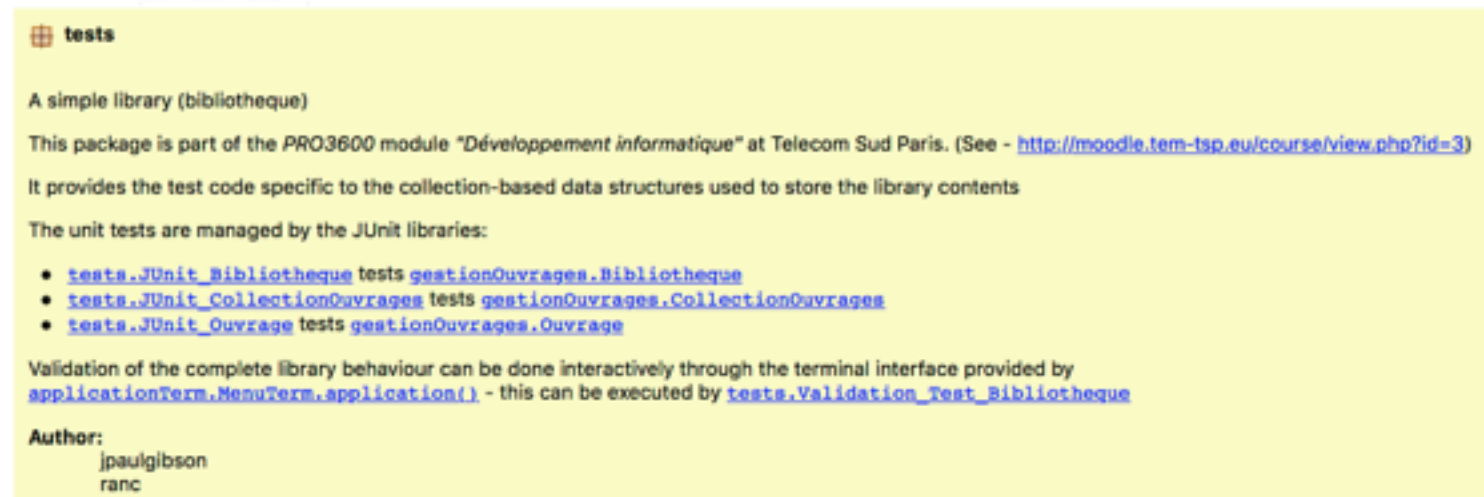
    public static void main (String [] args){

        Bibliotheque.init();
        MenuTerm.application();

    }

}
```

a. Test fait à la main avec l'interface utilisateur



tests

A simple library (bibliotheque)

This package is part of the PRO3600 module "Développement informatique" at Telecom Sud Paris. (See - <http://moodle.tem-tsp.eu/course/view.php?id=3>)

It provides the test code specific to the collection-based data structures used to store the library contents

The unit tests are managed by the JUnit libraries:

- [tests.JUnit_Bibliotheque](#) tests [gestionOuvrages.Bibliotheque](#)
- [tests.JUnit_CollectionOuvrages](#) tests [gestionOuvrages.CollectionOuvrages](#)
- [tests.JUnit_Ouvrage](#) tests [gestionOuvrages.Ouvrage](#)

Validation of the complete library behaviour can be done interactively through the terminal interface provided by [applicationTerm.MenuTerm.application\(\)](#) - this can be executed by [tests.Validation_Test_Bibliotheque](#)

Author:
jpaulgibson
ranc

Tests de validation : conception, implémentation, **exécution** et résultats

Test 1

Donnée :

un ouvrage déjà existant à ajouter

Résultat :

message d'erreur : « L'ouvrage existe déjà »
la collection ne change pas

Le test échoue partiellement

Valid Fichier non trouvé! Démarrage à vide.

Bibliothèque - mode terminal

1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 2

Ajout d'un ouvrage

Entrer l'auteur: author1
Entrer le titre: title1
Entrer l'année: 2001

1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 2

Ajout d'un ouvrage

Entrer l'auteur: author1
Entrer le titre: title1
Entrer l'année: 2001

1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 1

Liste des ouvrages

author1, title1, 2001

Tests de validation : conception, implémentation, **exécution** et résultats

Test 2

Donnée :

un ouvrage complet/valide (pas déjà
existant) à ajouter

Résultat :

la collection est mise à jour

Le test passe

```
-----  
Bibliothèque - mode terminal  
-----
```

```
1 - lister ouvrages  
2 - ajout  
3 - suppression  
4 - exit  
Sélection: 1
```

```
Liste des ouvrages  
-----
```

```
1 - lister ouvrages  
2 - ajout  
3 - suppression  
4 - exit  
Sélection: 2
```

```
Ajout d'un ouvrage  
-----
```

```
Entrer l'auteur: author1  
Entrer le titre: title1  
Entrer l'année: 2001
```

```
1 - lister ouvrages  
2 - ajout  
3 - suppression  
4 - exit  
Sélection: 1
```

```
Liste des ouvrages  
-----
```

```
author1, title1, 2001
```

Tests de validation : conception, implémentation, **exécution** et résultats

Test 3 (robustesse - entrées)

Donnée : le *titre* est ""

Résultat : message d'erreur et 2ème chance

```
1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 2
```

```
Ajout d'un ouvrage
-----
```

```
Entrer l'auteur:
Entrer le titre: title
Entrer l'année: 2000
```

```
Exception in thread "main" java.lang.IllegalArgumentException: The author must not be the empty string
    at gestionOuvrages.Ouvrage.<init>(Ouvrage.java:61)
    at gestionOuvrages.Bibliotheque.ajouterOuvrage(Bibliotheque.java:72)
    at applicationTerm.MenuTerm.ajouterOuvrage(MenuTerm.java:81)
    at applicationTerm.MenuTerm.application(MenuTerm.java:37)
    at tests.Validation_Test_Bibliotheque.main(Validation_Test_Bibliotheque.java:28)
```

Le test échoue

Tests de validation : conception, implémentation, **exécution** et résultats

Test 4 (robustesse - entrées)

Donnée : l'*auteur* est ""

Résultat : message d'erreur et 2ème chance

```
-----  
Bibliothèque - mode terminal  
-----
```

```
1 - lister ouvrages  
2 - ajout  
3 - suppression  
4 - exit  
Sélection: 2
```

```
Ajout d'un ouvrage  
-----
```

```
Entrer l'auteur: author
```

```
Entrer le titre:
```

```
Entrer l'année: 2000
```

```
Exception in thread "main" java.lang.IllegalArgumentException: The title must not be the empty string  
    at gestionOuvrages.Ouvrage.<init>(Ouvrage.java:63)  
    at gestionOuvrages.Bibliotheque.ajouterOuvrage(Bibliotheque.java:72)  
    at applicationTerm.MenuTerm.ajouterOuvrage(MenuTerm.java:81)  
    at applicationTerm.MenuTerm.application(MenuTerm.java:37)  
    at tests.Validation_Test_Bibliotheque.main(Validation_Test_Bibliotheque.java:28)
```

Le test échoue

Tests de validation : conception, implémentation, **exécution** et résultats

Test 5 (robustesse - entrées)

Donnée : l'*année* n'est pas valide (eg AAA234, -10, 3000)
Résultat : message d'erreur : « donnée non cohérente... »
et 2ème chance – « votre choix : »

Le test passe partiellement

```
1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 2
```

Ajout d'un ouvrage

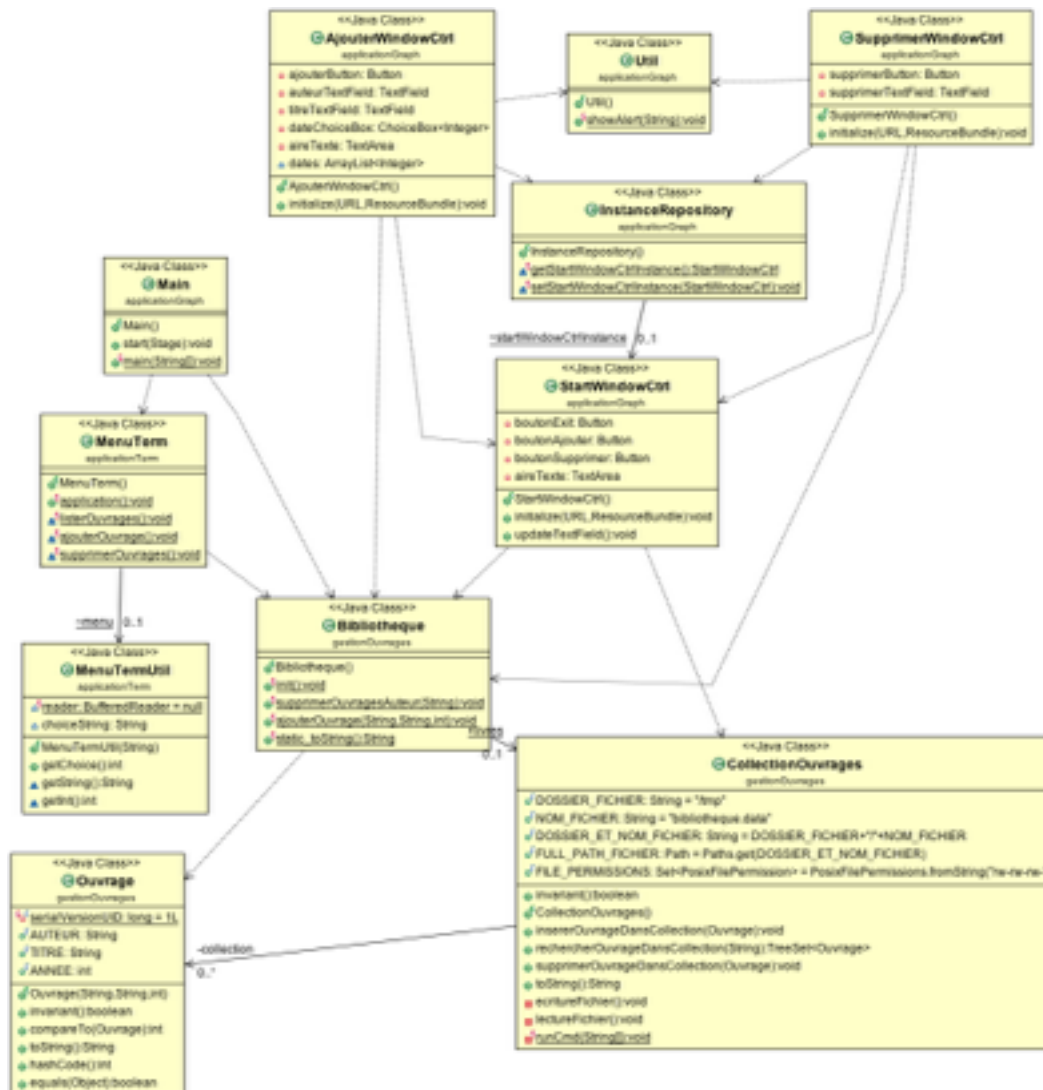
```
Entrer l'auteur: author1
Entrer le titre: title1
Entrer l'année: a
donnée non cohérente...
2000
```

```
1 - lister ouvrages
2 - ajout
3 - suppression
4 - exit
Sélection: 1
```

Liste des ouvrages

```
author1, title1, 2000
author1, title1, 2001
```

Conception préliminaire – les tests d'integration



Comment faire ?

Dans quel ordre?

Pour les projets simples, cette phase de test est souvent ignorée

Conception préliminaire

Les packages - On choisit ici de décomposer l'application en **quatre packages** :

▼ Bibliothèque

▼ src

▶ applicationGraph

▶ applicationTerm

▶ gestionOuvrages

▶ tests

▶ JUnit 4

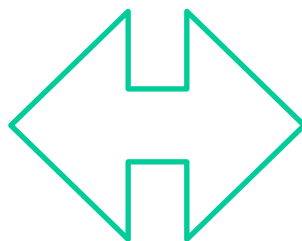
▶ JavaFX SDK

▶ Referenced Libraries

▶ JRE System Library [jdk1.8.0_66]

▶ doc

build.fxbuild



GUI

Console Interface

Structures de données

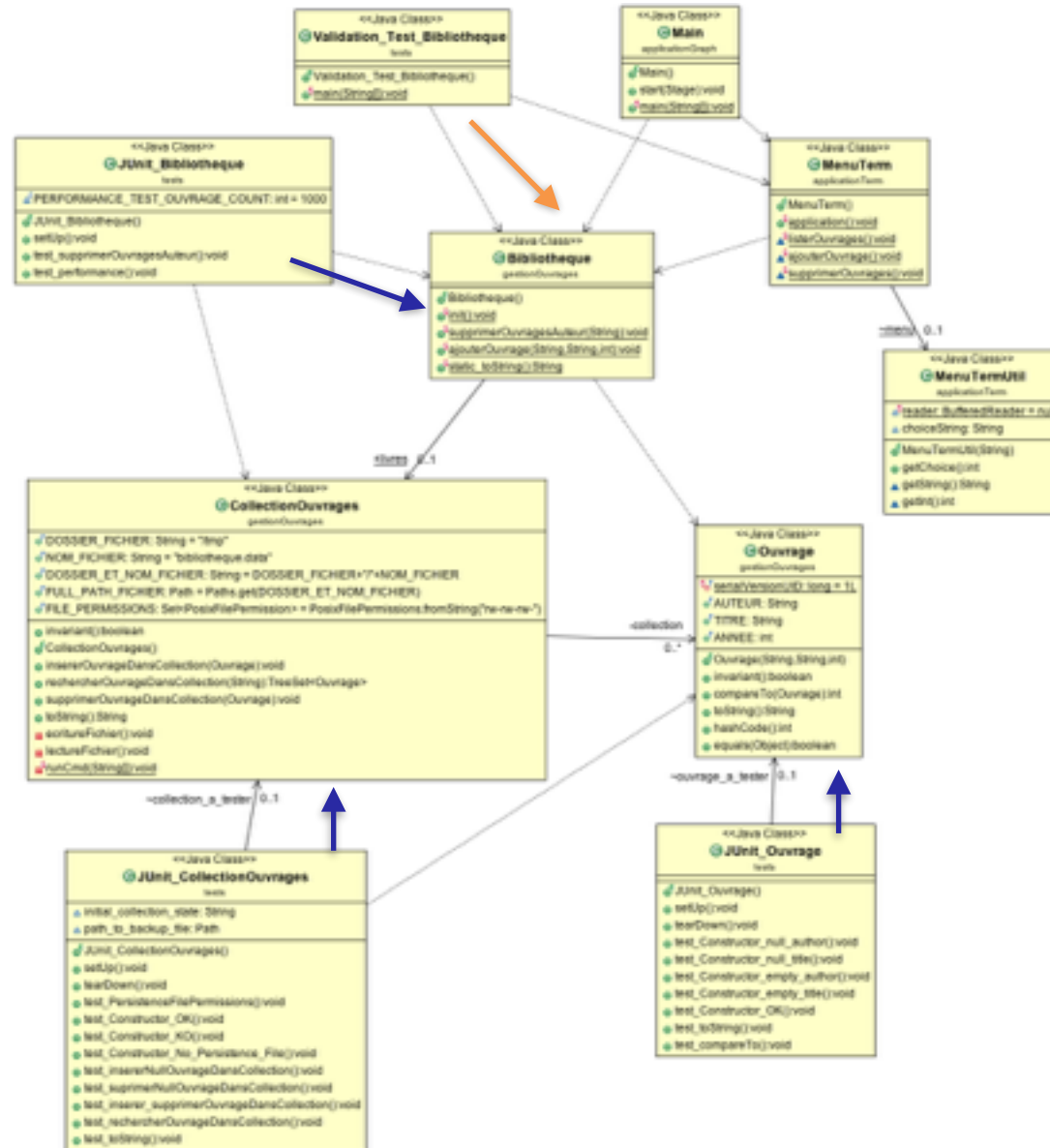
Tests

Decision a prendre: Ou sauvegarder le fichier **bibliothèque.data**?

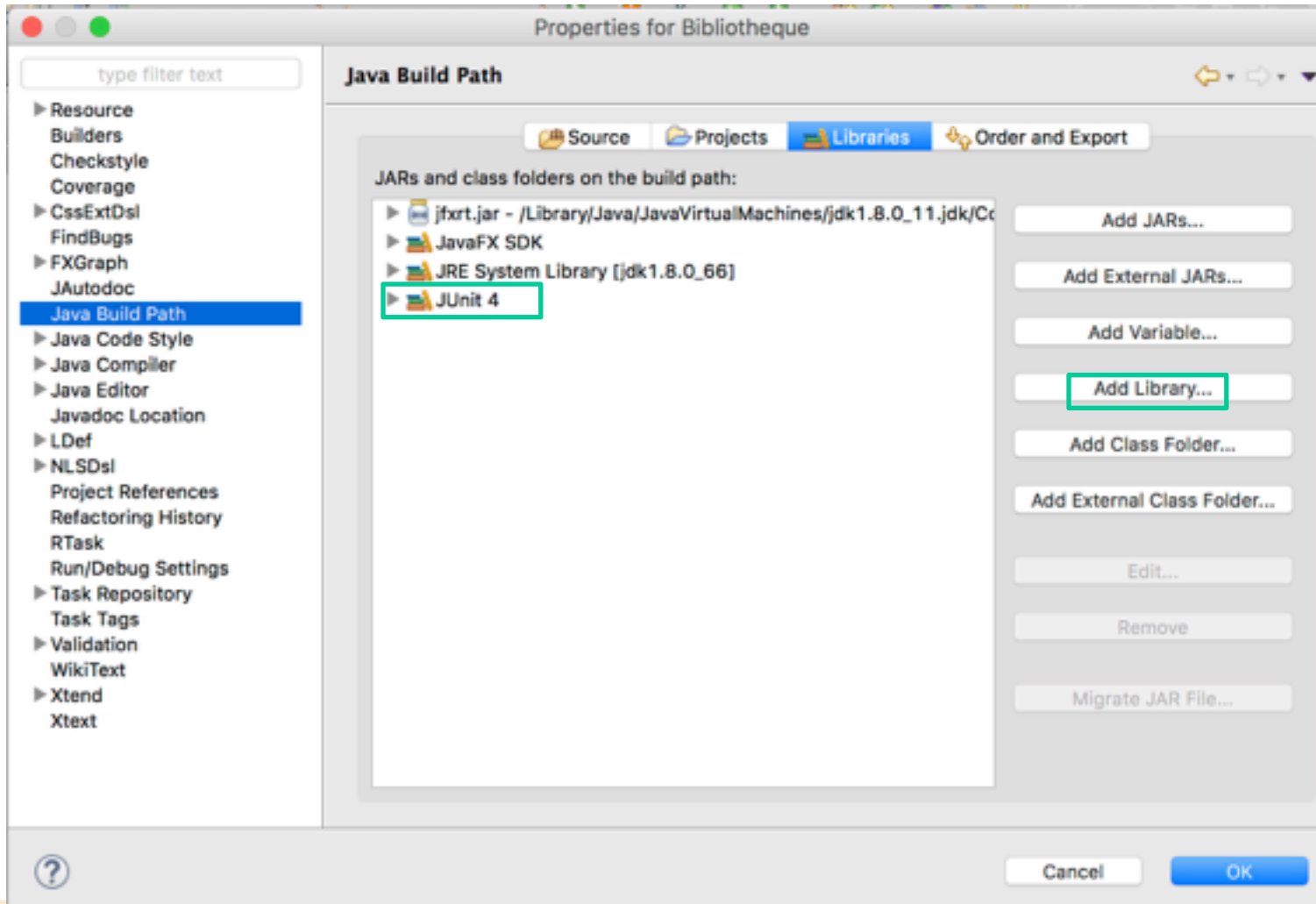
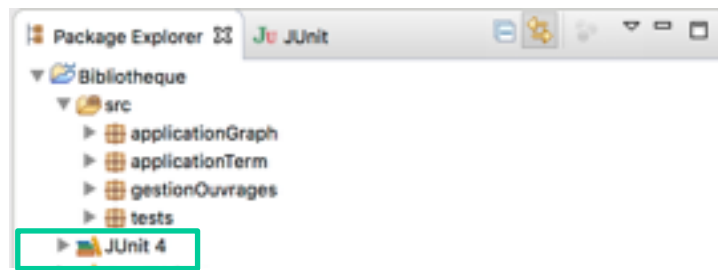
Pourquoi dossier - **/tmp** ?

Ou se trouve cette information dans notre implementation?

**Conception
détaillée**
– les **tests**
unitaires pour
les structures
de données, et
**test de
validation**
pour la version
“console”



JUnit (dans Eclipse)



```
package tests;

import gestionOuvrages.Ouvrage;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

+ * <p> <b> A simple library (bibliotheque) </b> </p>
public class JUnit_Ouvrage {

    Ouvrage ouvrage_a_tester;

+    * Initialise a default ouvrage (<b>"author, title, 2016"</b>) and check that it is in a safe state
-    @Before
    public void setUp(){

        ouvrage_a_tester = new Ouvrage("author","title", 2016);
        Assert.assertTrue(ouvrage_a_tester.invariant());
    }

-    /**
     * Recover the resources assigned during the setUp process
     */
-    @After
    public void tearDown(){

        ouvrage_a_tester = null;
    }
}
```

```
+    * Tests {@link gestionOuvrages.Ouvrage#Ouvrage(String, String, int)}  
-    @Test(expected=IllegalArgumentException.class)  
    public void test_Constructor_null_author() {  
  
        ouvrage_a_tester = new Ouvrage(null, "title", 2016);  
  
    }  
  
+    * Tests {@link gestionOuvrages.Ouvrage#Ouvrage(String, String, int)}  
-    @Test(expected=IllegalArgumentException.class)  
    public void test_Constructor_null_title() {  
  
        ouvrage_a_tester = new Ouvrage("author", null, 2016);  
  
    }  
  
+    * Tests {@link gestionOuvrages.Ouvrage#Ouvrage(String, String, int)}  
-    @Test(expected=IllegalArgumentException.class)  
    public void test_Constructor_empty_author() {  
  
        ouvrage_a_tester = new Ouvrage("", "title", 2016);  
  
    }  
  
- +    * Tests {@link gestionOuvrages.Ouvrage#Ouvrage(String, String, int)}  
-    @Test(expected=IllegalArgumentException.class)  
    public void test_Constructor_empty_title() {  
  
        ouvrage_a_tester = new Ouvrage("author", "", 2016);  
  
    }  
    }
```

```
* Tests {@link gestionOuvrages.Ouvrage#Ouvrage(String, String, int)}[]
@Test
public void test_Constructor_OK() {

    Assert.assertEquals("author", ouvrage_a_tester.AUTEUR);
    Assert.assertEquals("title", ouvrage_a_tester.TITRE);
    Assert.assertEquals(2016, ouvrage_a_tester.ANNEE);
}

* Tests {@link gestionOuvrages.Ouvrage#toString()}[]
@Test
public void test_toString(){

    Assert.assertEquals("author, title, 2016", ouvrage_a_tester.toString());
}

* Tests {@link gestionOuvrages.Ouvrage#compareTo(Ouvrage)}[]
@Test
public void test_compareTo(){

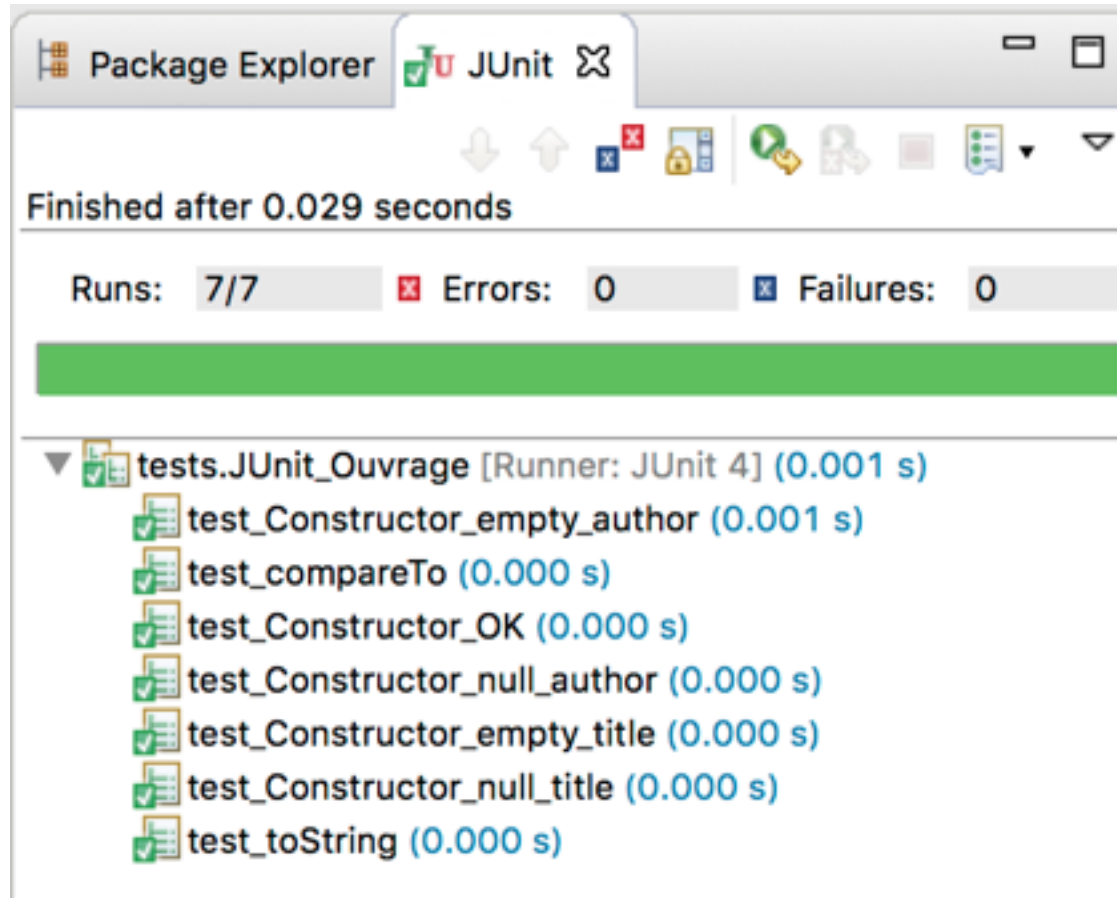
    Ouvrage ouvrage_to_compare;

    ouvrage_to_compare = new Ouvrage(ouvrage_a_tester.AUTEUR, ouvrage_a_tester.TITRE, ouvrage_a_tester.ANNEE);
    Assert.assertEquals(0, ouvrage_a_tester.compareTo(ouvrage_to_compare));

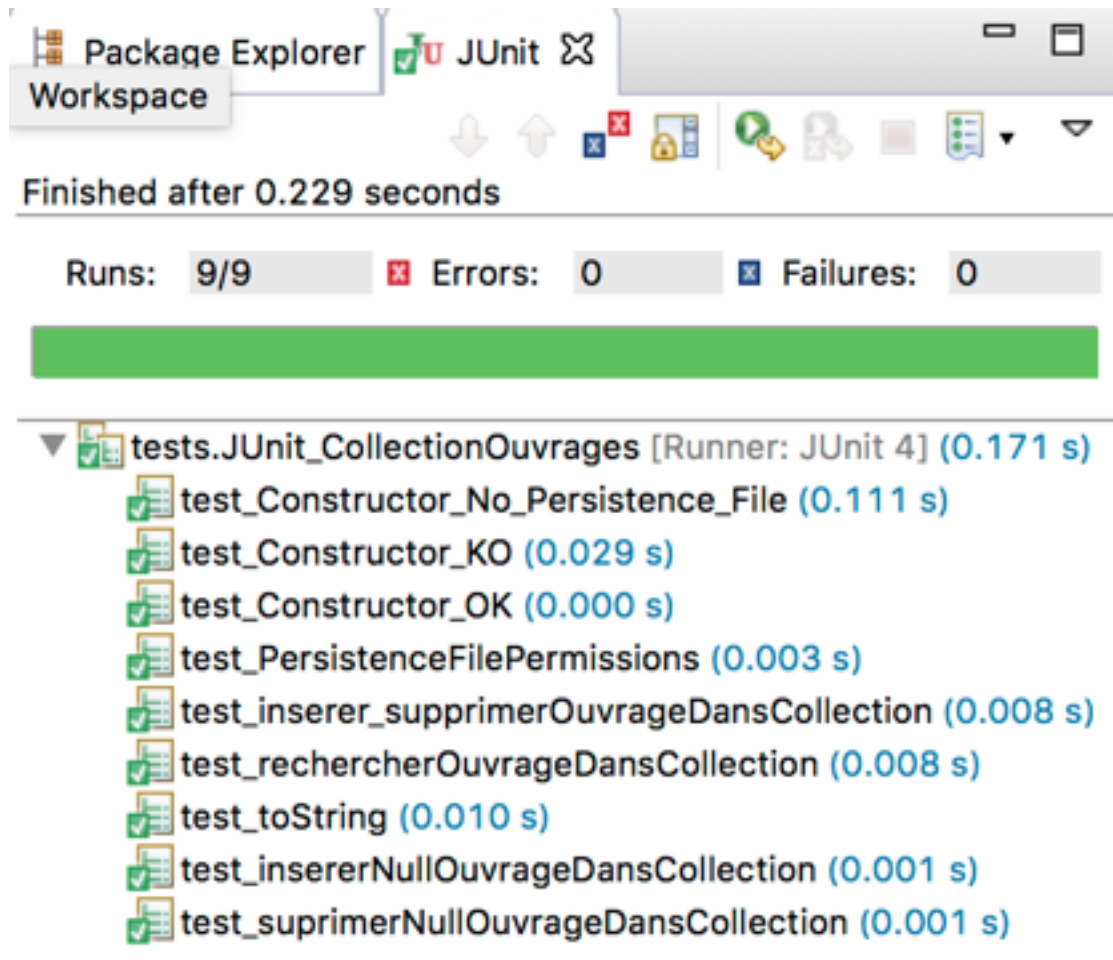
    ouvrage_to_compare = new Ouvrage(ouvrage_a_tester.AUTEUR, ouvrage_a_tester.TITRE+"DIFFERENT", ouvrage_a_tester.ANNEE);
    Assert.assertTrue(ouvrage_a_tester.compareTo(ouvrage_to_compare)<0);
    Assert.assertTrue(ouvrage_to_compare.compareTo(ouvrage_a_tester)>0);

    ouvrage_to_compare = new Ouvrage(ouvrage_a_tester.AUTEUR, ouvrage_a_tester.TITRE, ouvrage_a_tester.ANNEE+1);
    Assert.assertTrue(ouvrage_a_tester.compareTo(ouvrage_to_compare)<0);
    Assert.assertTrue(ouvrage_to_compare.compareTo(ouvrage_a_tester)>0);

    ouvrage_to_compare = new Ouvrage(ouvrage_a_tester.AUTEUR+"AFTER", ouvrage_a_tester.TITRE+"DIFFERENT", ouvrage_a_tester.ANNEE+1);
    Assert.assertTrue(ouvrage_a_tester.compareTo(ouvrage_to_compare)<0);
    Assert.assertTrue(ouvrage_to_compare.compareTo(ouvrage_a_tester)>0);
}
```



OK



OK

```

public class JUnit_Bibliotheque {

    /**
     * This is the number of books in the library for which we will test the performance of
     * a single addition or removal of an element<br>
     * When we set this to the desired performance level of 30000, the test {@link #test_performance()}
     * takes 20-30 minutes to execute (due to the time needed to keep updating the persistence file). <br>
     */
    final int PERFORMANCE_TEST_OUVRAGE_COUNT = 5000; // Should be 30000

    @Before
    public void setUp(){

        Bibliotheque.init();

    }

    public void test_performance(){

        String auteur="auteur";
        String titre ="titre";
        int annee = 0;

        int NUMBER_OF_PERFORMANCE_TESTS =10;

        // Add PERFORMANCE_TEST_OUVRAGE_COUNT elements to the library
        for (int ouvrage_count=0; ouvrage_count < PERFORMANCE_TEST_OUVRAGE_COUNT; ouvrage_count++){

            auteur = "auteur"+ouvrage_count;
            titre = "titre"+ouvrage_count;
            annee = ouvrage_count;

            // Check that adding each of last 10 elements takes less than 1 second
            if (ouvrage_count<PERFORMANCE_TEST_OUVRAGE_COUNT-NUMBER_OF_PERFORMANCE_TESTS){
                long startTime = System.currentTimeMillis();
                Bibliotheque.ajouterOuvrage(auteur, titre, annee);
                long endTime = System.currentTimeMillis();
                Assert.assertTrue(endTime-startTime<1000);}

        }

        for (int ouvrage_count=0; ouvrage_count < PERFORMANCE_TEST_OUVRAGE_COUNT; ouvrage_count++){

            // Check that removing each of last 10 elements takes less than 1 second
            if (ouvrage_count<PERFORMANCE_TEST_OUVRAGE_COUNT-NUMBER_OF_PERFORMANCE_TESTS){
                auteur = "auteur"+ouvrage_count;
                long startTime = System.currentTimeMillis();
                Bibliotheque.supprimerOuvragesAuteur(auteur);
                long endTime = System.currentTimeMillis();
                Assert.assertTrue(endTime-startTime<1000);
            }

        }

    }

}

```

tests.JUnit_Bibliotheque [Runner: JUnit 4] (2.029 s)

test_performance (1.952 s)

test_supprimerOuvragesAuteur (0.077 s)

KO

Conception préliminaire : les compromis peuvent poser les problèmes pour les tests

Exemple : Robustesse et Performance

Afin de limiter les accès fichier, la recherche sera effectuée sur la représentation interne de la collection d'ouvrages en mémoire et non pas dans le fichier.

Le fichier doit être mis à jour et il nous faut un « backup »

QUESTION: Comment intégrer au mieux ces deux contraintes dans la gestion de la mise à jour du fichier :

- Trop souvent => compromet les performances — **KO**
- Trop rarement => compromet la robustesse

Plan de tests de non-régression

eg, Extension III *undo*

Il faut ré-exécuter les tests déjà développés dans la version précédente

Les tests de chaque extension doivent être réussis avant que l'extension suivante ne soit intégrée dans le système.

Nous avons utilisé Javadocs

```
/**
 * Construct an ouvrage with the given author, title and year<br>
 * @param auteur is the author
 * @param titre is the title
 * @param annee is the year
 * @throws IllegalArgumentException if the author or title fields are null or empty
 */
public Ouvrage(String auteur, String titre, int annee) throws IllegalArgumentException
{
    if (auteur == null) throw new IllegalArgumentException("The author must not be null");
    if (auteur.equals("")) throw new IllegalArgumentException("The author must not be the empty string");
    if (titre == null) throw new IllegalArgumentException("The title must not be null");
    if (titre.equals("")) throw new IllegalArgumentException("The title must not be the empty string");

    AUTEUR = auteur;
    TITRE = titre;
    ANNEE = annee;
}
```

📄 gestionOuvrages.Ouvrage.Ouvrage(String auteur, String titre, int annee) throws IllegalArgumentException

Construct an ouvrage with the given author, title and year

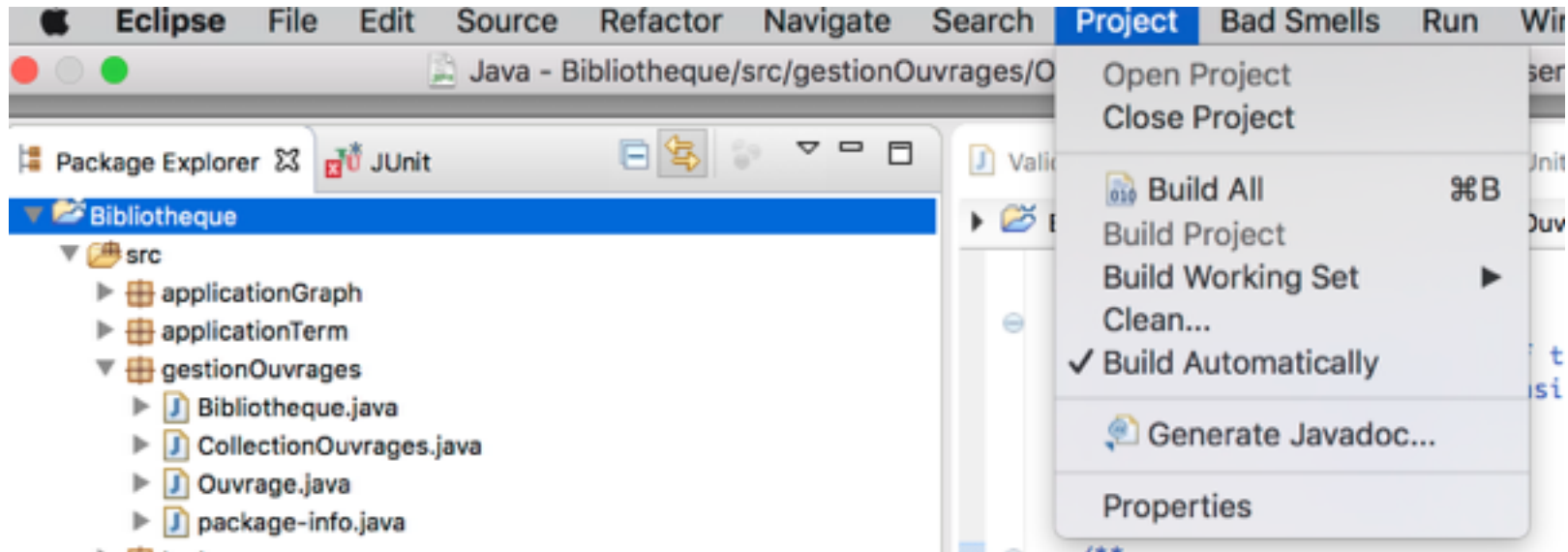
Parameters:

auteur is the author
titre is the title
annee is the year

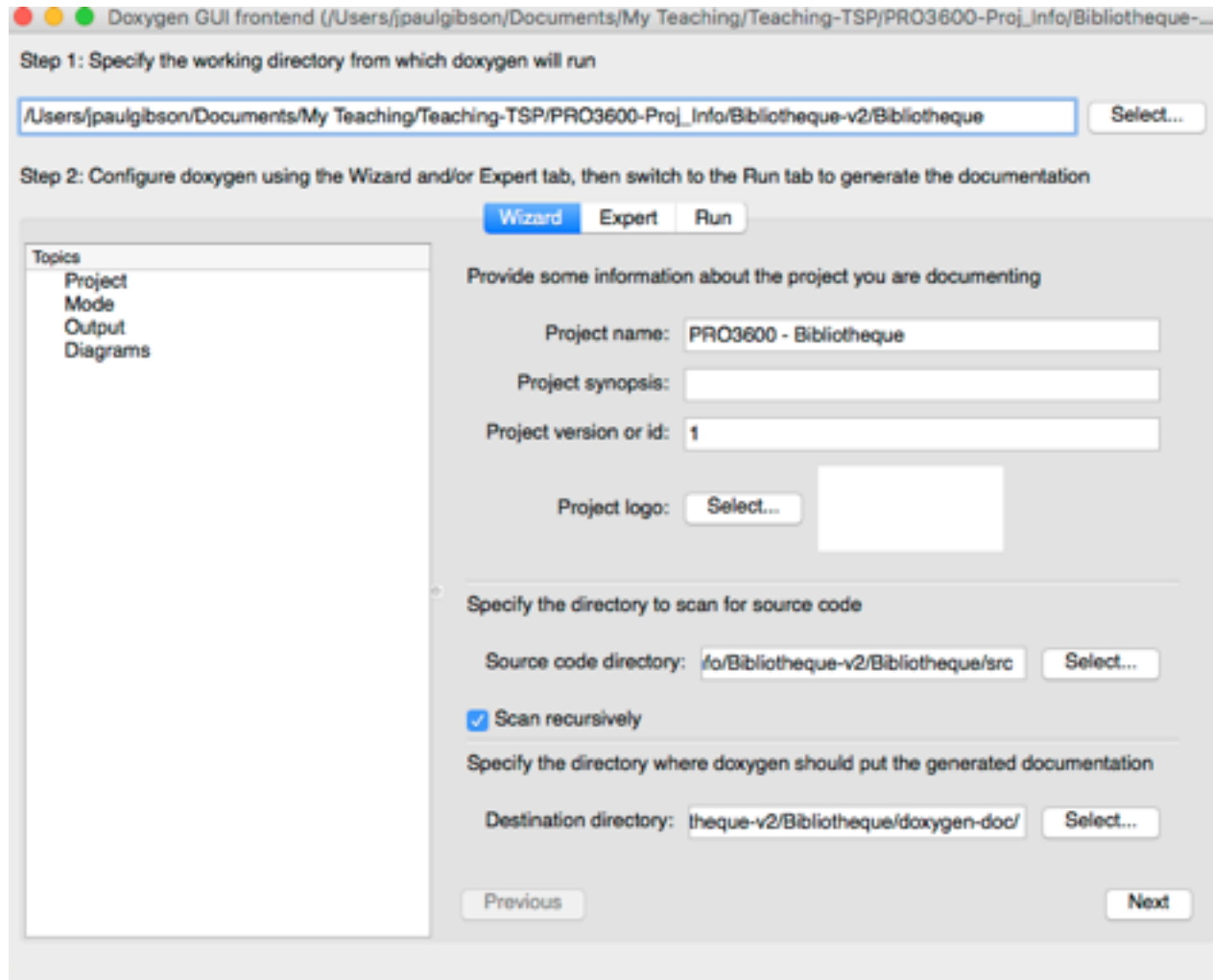
Throws:

[IllegalArgumentException](#) - if the author or title fields are null or empty

Nous avons utilisé **Javadocs**

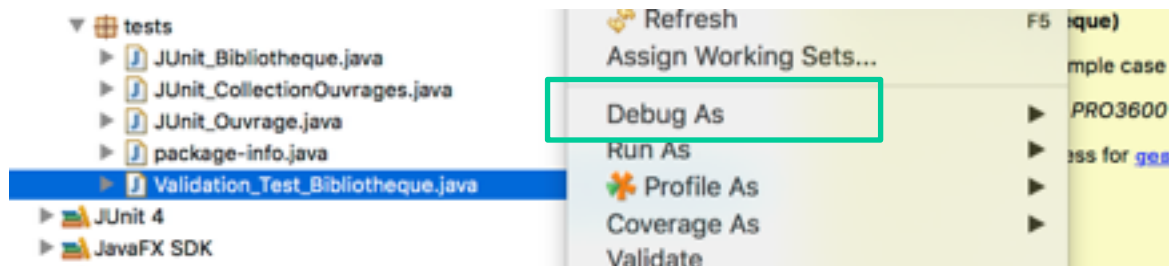


Il y a aussi:



Débogage/Debugging

Vous devez apprendre à utiliser un débogueur



Conseils : Développement en parallèle

Pour de gros projets, c'est assez courant mais ça demande des compétences et des outils d'intégration avancés pour tester et configurer le système

Pour de petits projets, c'est mieux de ne les développer en parallèle que si les composants sont indépendantes (ou si leurs interdépendances sont faciles à gérer)

Il y a un problème fondamental avec les interactions qui ne peut être résolu (en utilisant les techniques/outils actuels) pour des extensions multiples.

**CONSEIL FINAL POUR VOTRE PROJET: pas d'extensions en parallèle.
Ayez un nombre raisonnable d'extensions/incréments – 3 .. 6**

QUESTIONS?

"il faut
apprendre à
penser avant
d'apprendre à lire
car, après, il est
trop tard. "

[Gérard de
Rohan-Chabot
1930 – 1992]

CALVIN AND HOBBS

