Introduction au test logiciel

Frédéric Dadeau

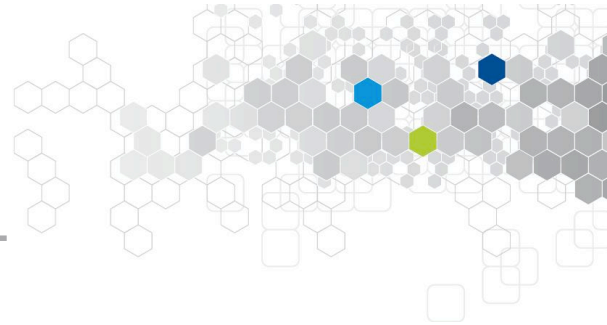
Département Informatique des Systèmes Complexes – FEMTO-ST

Bureau 405C

frederic.dadeau@univ-fcomte.fr

Licence 3 – Année 2023-2024

Le test de logiciels, une activité simple ?



Le test de logiciels, une activité simple ? Ou pas ...

Le test de logiciels, une activité simple ? Ou pas ...

Deux situations ...

1. Je teste un programme que **j'ai écrit** ...
   - je connais le code, donc je sais quels points du programme je dois exercer
   - il est difficile de calculer le résultat attendu (à refaire manuellement)

Le test de logiciels, une activité simple ? Ou pas ...

Deux situations ...

1. Je teste un programme que **j'ai écrit** ...
   - je connais le code, donc je sais quels points du programme je dois exercer
   - il est difficile de calculer le résultat attendu (à refaire manuellement)
2. Je teste un programme que **quelqu'un d'autre a écrit** ...
   - je ne connais pas le code, je ne sais pas exactement comment exercer le programme
   - néanmoins, j'ai une description du résultat attendu (verdict plus facile à établir)

Le test de logiciels, une activité simple ? Ou pas ...

Deux situations ...

1. Je teste un programme que **j'ai écrit** ...
   - je connais le code, donc je sais quels points du programme je dois exercer
   - il est difficile de calculer le résultat attendu (à refaire manuellement)
2. Je teste un programme que **quelqu'un d'autre a écrit** ...
   - je ne connais pas le code, je ne sais pas exactement comment exercer le programme
   - néanmoins, j'ai une description du résultat attendu (verdict plus facile à établir)

... qui illustrent les problèmes de l'activité de test

- comment choisir la **technique** de test ⇒ **boîte blanche** ou **boîte noire** ?
- comment obtenir le **résultat attendu** ? ⇒ problème de l'**oracle** du test
- comment savoir quand **arrêter la phase de test** ? ⇒ critère d'**arrêt**

Plan du cours

Qu'est-ce que tester ?

Comment tester ?

Que tester ?

Références

Motivations du test

Erreur dans les spécifications, la conception, ou le programme

⇓

Défaut (faute) dans le logiciel

⇓

Défaillance (anomalie de fonctionnement) du programme lors de l'exécution du défaut

Empêcher les dysfonctionnements

Quelques exemples de dysfonctionnements de systèmes critiques :

- Machines de radiothérapie Thérac-25 (1985/6 – gestion du clavier)
- Missile Patriot (1991 – erreurs d'arrondis → perte de précision de l'heure)
- Ariane 5 (1996 – portage nombre 64 bits → 16 bits non re-vérifié)
- Ver Samy (a.k.a. JS.Spacehero) propagé sur MySpace (oct 2005 – faille XSS)
- Faille de sécurité Heartbleed (avril 2014 – bug dans l'observation)
- etc.

Voir l'exposé "La chasse aux bugs" de Gérard Berry (collège de France)
http://gdr-gpl.cnrs.fr/node/292

Motivations du test

Erreur dans les spécifications, la conception, ou le programme

⇓

Défaut (faute) dans le logiciel

⇓

Défaillance (anomalie de fonctionnement) du programme lors de l'exécution du défaut

Coût d'un bug

- Coût des bugs informatiques estimé à 60 milliards de dollars par an
- 22 milliards pourraient être économisés si les procédures de test de logiciel étaient améliorées [a]

a. Source NIST – National Institute of Standards and Technology

Qualité du logiciel

Tester s'inscrit dans une démarche de **qualité des logiciels**.

Méthodes de vérification et de validation

- **Validation** : est-ce que le logiciel réalise les fonctions attendues ?
- **Vérification** : est-ce que le logiciel fonctionne correctement ?

Méthodes de vérification et de validation (V&V)

- Test **statique** : revue de code, de spécifications, de documents de conception
- Test **dynamique** : exécution du code pour s'assurer d'un fonctionnement correct
- Vérification **symbolique** : Run-Time Checking, exécution symbolique d'un programme
- Vérification **formelle** : preuve ou model-checking d'un model formel

Définitions du test

*Le test est **l'exécution ou l'évaluation d'un système** ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il **répond à ses spécifications** ou **identifier les différences entre les résultats attendus et les résultats obtenus**.*

IEEE Glossary of Standard Software Engineering Terminology

*Tester, c'est exécuter le programme **dans l'intention d'y trouver des anomalies ou des défauts**.*

G. Myers – The Art of Software Testing

*Testing can **reveal the presence of errors but never their absence**.*

Edsgar G. Dijkstra – Notes on Structured Programming

Méthodes de test

Test statique

Traite le texte du logiciel **sans l'exécuter** sur des données réelles.

Quelques exemples ...

- Lectures croisées/inspections : vérification collégiale d'un document (programme ou spécification du logiciel)
- Analyse d'anomalie : corriger de manière statique les erreurs (typage impropre, code mort, variable non initialisée, etc.)

Méthodes de test

Test statique

Traite le texte du logiciel **sans l'exécuter** sur des données réelles.

Avantages

- Méthodes efficaces et peu coûteuses
- entre 60% et 95% des erreurs sont détectées au cours de contrôles statiques

Inconvénient

- Ne permet pas de valider le comportement d'un programme au cours de son exécution.

Bilan : les méthodes de test statique sont **nécessaires**, mais pas **suffisantes**.

Méthodes de test

Test dynamique

Repose sur l'**exécution effective** du logiciel pour un sous-ensemble bien choisi du domaine de ses entrées possibles.

Quatre activités successives

1. **Sélection** d'un jeu de tests : choisir un sous-ensemble des entrées possibles du logiciel
2. **Exécution** du jeu de test
3. **Dépouillement** des résultats : consiste à décider du succès ou de l'échec du jeu de test (verdict) : échec, succès, inconclusif
4. **Évaluation** de la qualité et de la pertinence des tests effectués : déterminant pour la décision d'arrêt de la phase de test.

Méthodes de test

Test dynamique

Repose sur l'**exécution effective** du logiciel pour un sous-ensemble bien choisi du domaine de ses entrées possibles.

Deux méthodes (et demies)

- **Test structurel** : jeu de test sélectionné en s'appuyant sur une analyse du code du logiciel (test boîte blanche ou boîte de verre)
- **Test fonctionnel** : jeu de test sélectionné en s'appuyant sur les spécifications (test boîte noire)
- **Test boîte grise** : combinaison des deux approches précédentes (boîte blanche + boîte noire).

Méthodes de test

Test dynamique

Repose sur l'**exécution effective** du logiciel pour un sous-ensemble bien choisi du domaine de ses entrées possibles.

Dans tous les cas, les méthodes de test dynamique consistent à :

- **Exécuter le programme** sur un ensemble fini de données d'entrée
- **Contrôler** la correction des valeurs de sortie d'un programme par rapport à ce qui est attendu

La pratique du test

Le test appartient à une activité de **validation** du logiciel :

> *Est-ce que le logiciel a été développé correctement en fonction des différentes exigences ?*

Activité peu populaire en entreprise

Difficultés d'ordre psychologique ou culturel

- le développement de logiciels est un processus constructif : on cherche à établir des résultats corrects
- le test de logiciels est un processus destructif : un bon test est un test qui trouve une ou des erreurs
- mal perçu : mauvais programmeur $\rightarrow$ testeur

La pratique du test

Le test appartient à une activité de **validation** du logiciel :

Est-ce que le logiciel a été développé correctement en fonction des différentes exigences ?

Activité peu populaire en entreprise

La pratique du test

Cependant, le test est une **activité centrale** : il est le vecteur principal de l'amélioration de la qualité du logiciel.

Actuellement, le **test dynamique** est la méthode la plus diffusée.

Il représente jusqu'à 60% des efforts de développement d'un logiciel.

Coût moyen de l'activité de test se répartit :

- 1/3 durant les phases de développement
- 2/3 durant les phases de maintenance

Test de logiciels – Une auto-évaluation

Soit la spécification suivante :

Un programme prend en entrée trois entiers. Ces trois entiers sont interprétés comme représentant les longueurs des cotés d'un triangle. Le programme rend un résultat précisant s'il s'agit d'un triangle scalène, isocèle ou équilatéral (ou une erreur si les données ne correspondent pas aux longueurs d'un triangle).

G. Myers – The Art of Software Testing

Donner un ensemble de cas de tests que vous pensez adéquats pour tester ce programme.

Test de logiciels – Une auto-évaluation

Avez-vous un cas de test pour ...

1. Cas scalène valide (1,2,3 et 2,5,10 ne sont pas valides)
2. Cas équilatéral valide
3. Cas isocèle valide (2,2,4 n'est pas valide)
4. Cas isocèle valide avec les trois permutations (e.g. 3,3,4 ; 3,4,3 ; 4,3,3)
5. Cas avec une valeur à 0
6. Cas avec une valeur négative
7. Cas où la somme de deux entrées est égale à la troisième entrée
8. Trois cas pour le test 7 avec les trois permutations

Test de logiciels – Une auto-évaluation

Avez-vous un cas de test pour ...

...

9 Cas où la somme de deux entrées est inférieure à la troisième entrée

10 Trois cas pour le test 9 avec les trois permutations

11 Cas avec les trois entrées à 0

12 Cas avec une entrée non entière

13 Cas avec un nombre erroné de valeurs (par ex. 2 entrées, ou 4)

14 Pour chaque cas de test, avez-vous défini le résultat attendu ?

Et la liste est loin d'être exhaustive ...

Test de logiciels – Une auto-évaluation

Chacun de ces 14 tests correspond à un défaut constaté dans des implantations de cet exemple du triangle.

La moyenne des résultats obtenus pour un ensemble de développeurs expérimentés est de ....

Test de logiciels – Une auto-évaluation

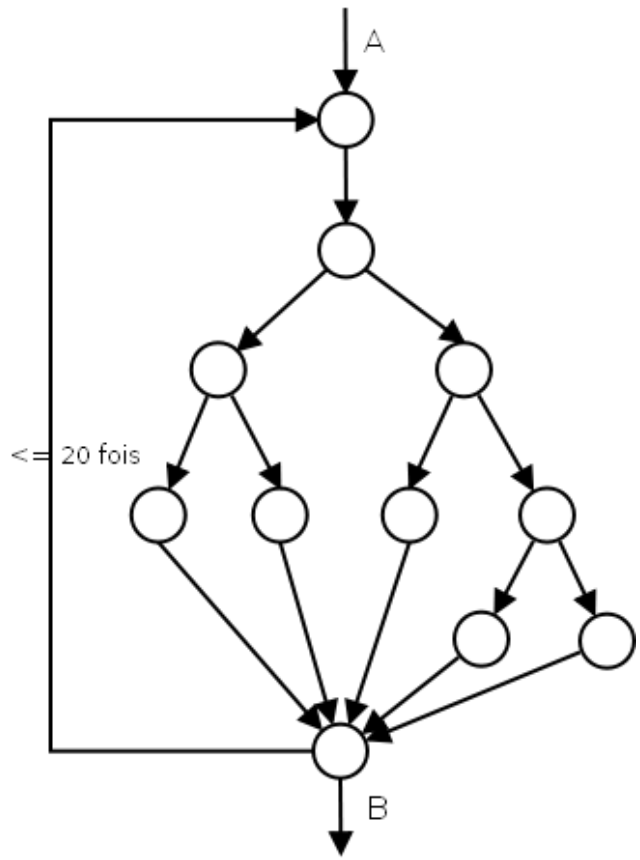
Chacun de ces 14 tests correspond à un défaut constaté dans des implantations de cet exemple du triangle.

La moyenne des résultats obtenus pour un ensemble de développeurs expérimentés est de 7.8

⇒ **le test est une activité complexe, a fortiori sur de grandes applications**

Test structurel (boîte blanche)

Les données de test sont produites à partir de l'**analyse du code source**



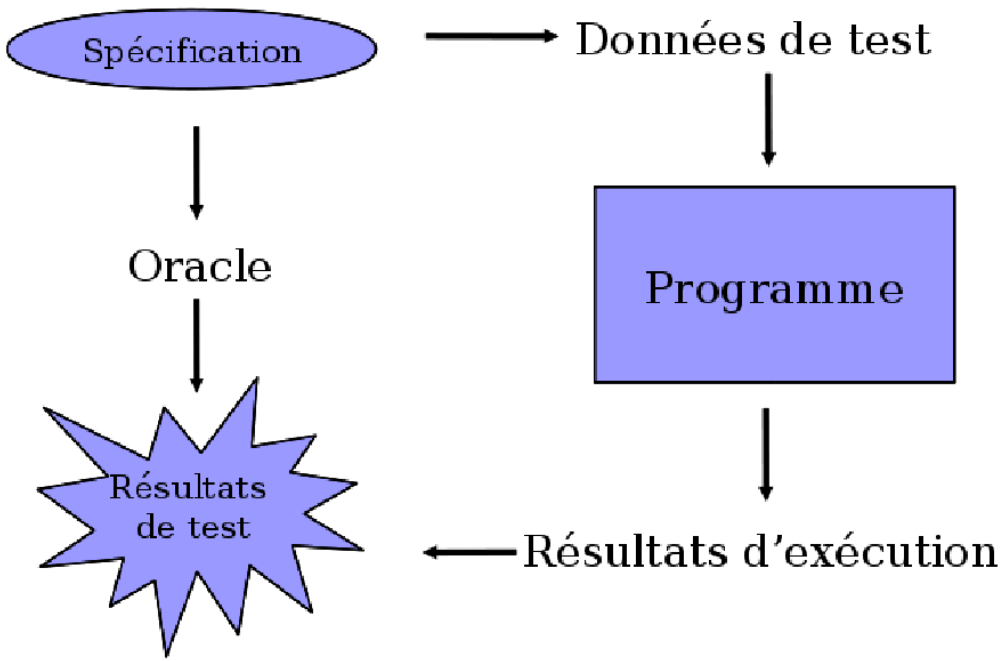
Graphe de flot de contrôle d'un programme

Critères de sélection de tests

- toutes-les-instructions
- ...
- tous-les-chemins

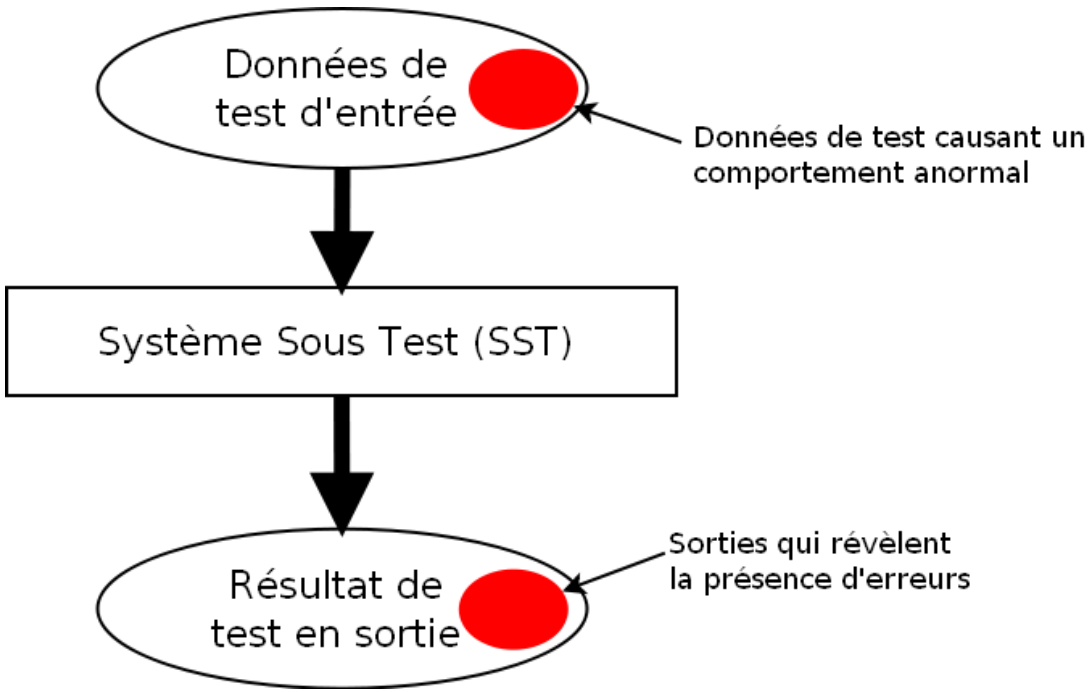
Test fonctionnel (boîte noire)

Test de conformité **par rapport à une spécification** (pas nécessairement formelle, mais suffisamment détaillée).



Test fonctionnel (boîte noire)

Le système sous test est vu comme une **boîte noire**.



Complémentarité test structurel/fonctionnel

Les deux approches peuvent (doivent) être utilisées de manière **complémentaire**.

Complémentarité boîte blanche/boîte noire

Soit le programme suivant, supposé calculer la somme de deux entiers.

```
function somme(x,y : integer) : integer
begin
    if x = 600 and y = 500 then
          somme := x - y
     else
          somme := x + y
     end if
end
```

Une analyse de la spécification uniquement détectera difficilement les valeurs x = 600 et y = 500 qui causent une anomalie de fonctionnement. Une approche basée sur le code isolera aisément ces valeurs.

Complémentarité test structurel/fonctionnel

En examinant ce qui a été réalisé, on ne prend pas forcément en compte ce qui aurait dû être fait :

- les approches **structurelles** détectent plus facilement les **erreurs commises** (division par 0, déréférencement de pointeur null, etc.)
- les approches **fonctionnelles** détectent plus facilement les **erreurs d'omission ou de spécification** (accès accordé au lieu d'être refusé, etc.)

La difficulté du test structurel consiste en la définition de l'Oracle du test. La difficulté du test fonctionnel consiste en la définition de données de test pertinentes.

Difficultés du test

Pas de test exhaustif

Le **test exhaustif** est en général impossible à réaliser : tester tous les chemins d'exécution possibles, avec toutes les données d'entrée possibles.

- En test fonctionnel, l'ensemble des données d'entrée est en général infini ou très grande taille (par exemple : une procédure avec 5 entrées sur 8 bits admet 2^{40} valeurs différentes en entrée)
- En test structurel, le parcours du graphe de flot de contrôle conduit à une forte explosion combinatoire (par exemple : le nombre de chemin logique dans le graphe en diapo 13 est supérieur à $10^{14} = 5^{20} + 5^{19} + \ldots + 5^{1}$

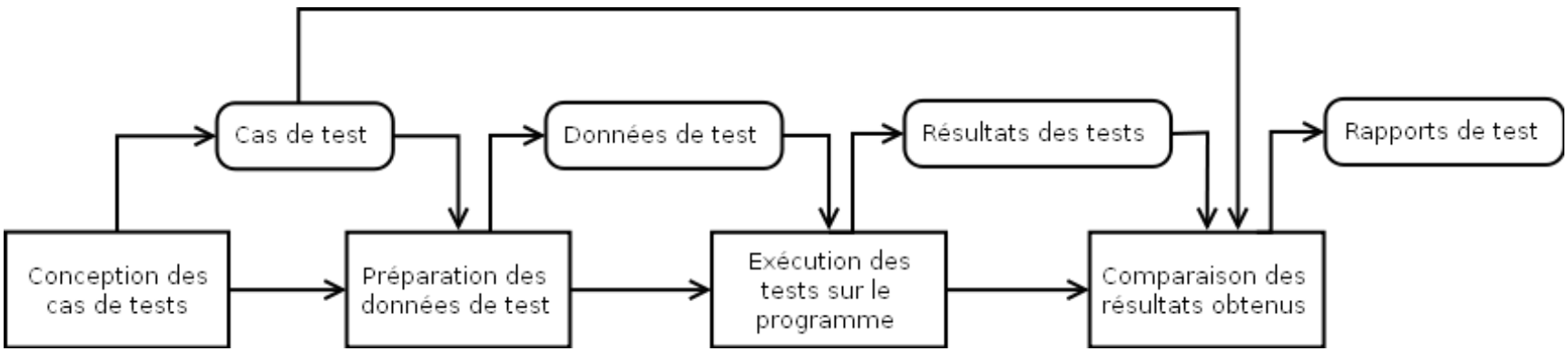
Un premier bilan

$\Rightarrow$ le test est une méthode de vérification **partielle** de logiciels

$\Rightarrow$ la qualité du test dépend de la **pertinence du choix des données** de test

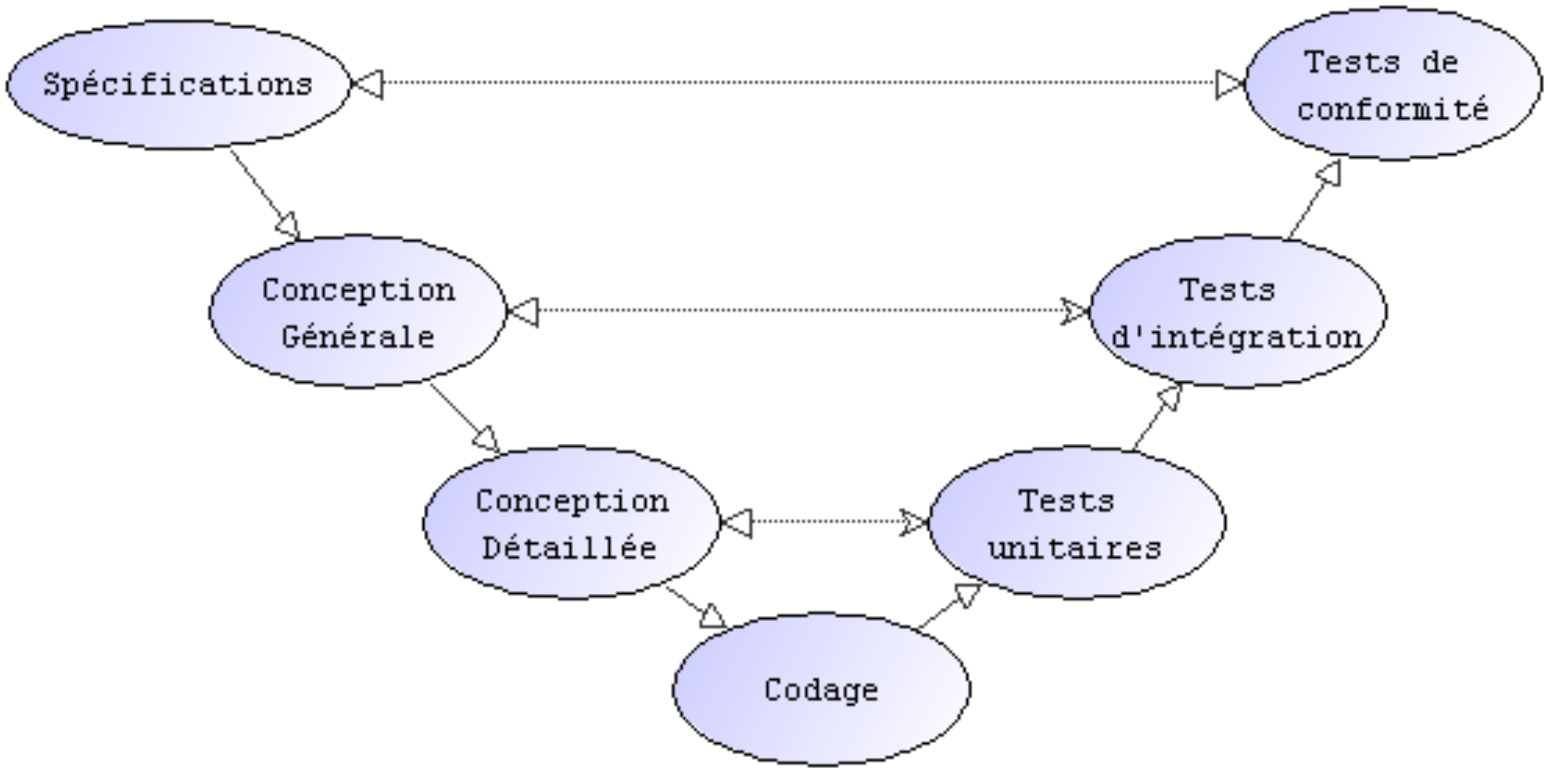
Test et cycle de vie

Le processus du test peut se résumer par le schéma suivant :



Test et cycle de vie du logiciel

Le test s'intègre à **chaque étape** du cycle de vie du logiciel (dit en V).



Types de test

Le type du test renseigne l'objectif mené.

Test unitaire

Test de procédures, de modules, de composants.
(Coût : 20% du coût total du développement initial correspondant)

Test d'intégration

Test du bon comportement lors de la composition des procédures, modules, ou composants.
(Coût d'un bug dans cette phase : 10 fois celui d'un bug unitaire)

Test de recette/de conformité

Validation de l'adéquation aux spécifications.
(Coût d'un bug dans cette phase : 100 fois celui d'un bug unitaire)

Types de test (suite)

Test nominaux / de bon fonctionnement

Vérifier que le résultat obtenu est conforme aux résultats attendus, en entrant des données valides au programme (test-to-pass).

Test de robustesse

Vérifier que le programme réagit correctement à une utilisation non-conforme, en entrant des données invalides (test-to-fail).

Test de performance

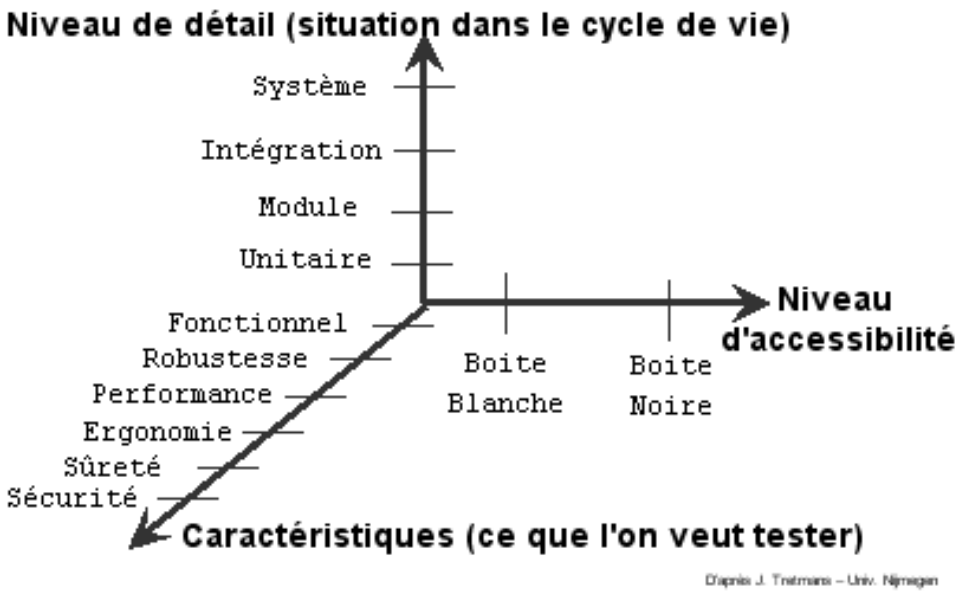
- load testing : test avec montée en charge
- stress testing : test du système soumis à des demandes de ressources anormales

Test de non-régression

Vérifier que les corrections ou évolutions du code n'ont pas créé d'anomalies nouvelles.

Test et cycle de vie du logiciel

Une vision de test en 3 dimensions (proposée par Jan Tretmans – Univ. Nijmegen, NL)



D'autres dimensions sont possibles :

- phase du développement considérée
- qui fait le test ?
- but du test ?

Stratégie de test

En début de projet, on définit un **Plan de Test et de Validation** (PTV).

Celui-ci comprend :

- Objectifs du test
- Techniques de test
- Phases de test et planning

Il s'adapte à la méthode de développement utilisée.

Test et méthodes agiles

Utilisation d'outil dédiés

- Exécution des tests : JUnit, TestNG, GoogleTest, etc.
- Automatisation : ANT, Maven, Gradle, etc.
- Contrôle : Jenkins, Travis, Bamboo, etc. (intégration continue - `https://stackify.com/top-continuous-integration-tools`)

Test Driven Development = Test-First Design + Refactoring

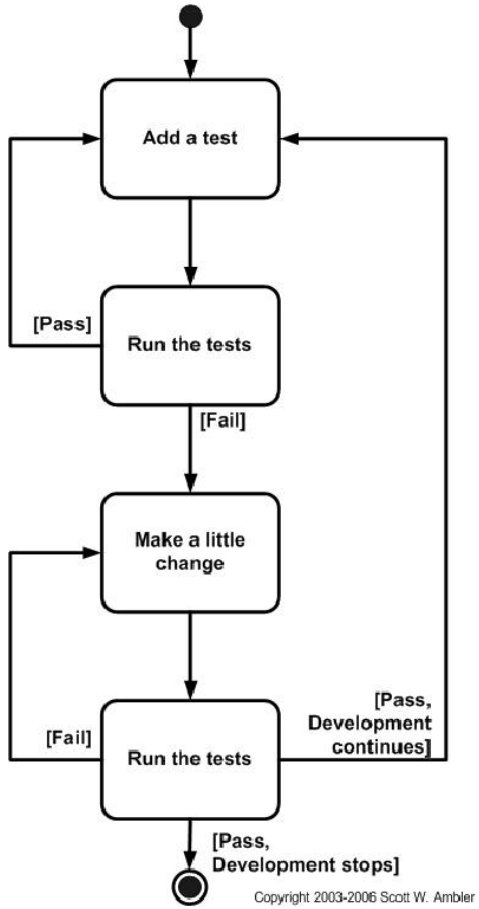
- développer les tests en premier
- développer le code correspondant
- refactoriser (refactoring)

Test et méthodes agiles

Test-First Design (TFD)

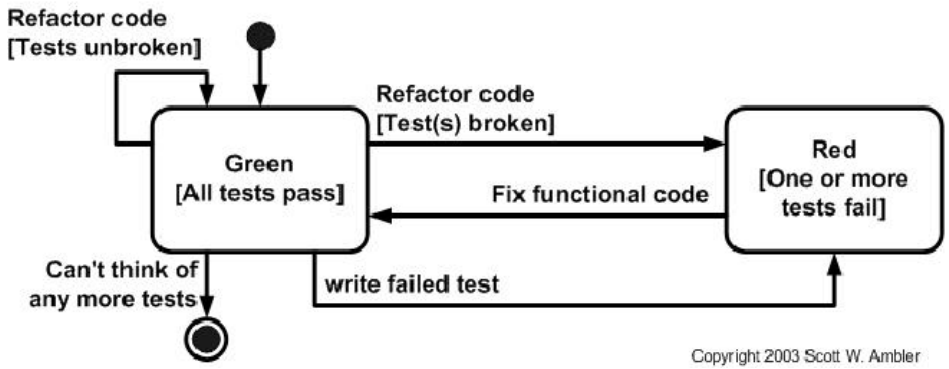
> *The first step is to quickly add a test, basically just enough code to fail. Next you run your tests, often the complete test suite although for sake of speed you may decide to run only a subset, to ensure that the new test does in fact fail. You then update your functional code to make it pass the new tests. The fourth step is to run your tests again. If they fail you need to update your functional code and retest. Once the tests pass the next step is to start over.*

`http://www.agiledata.org/essays/tdd.html`

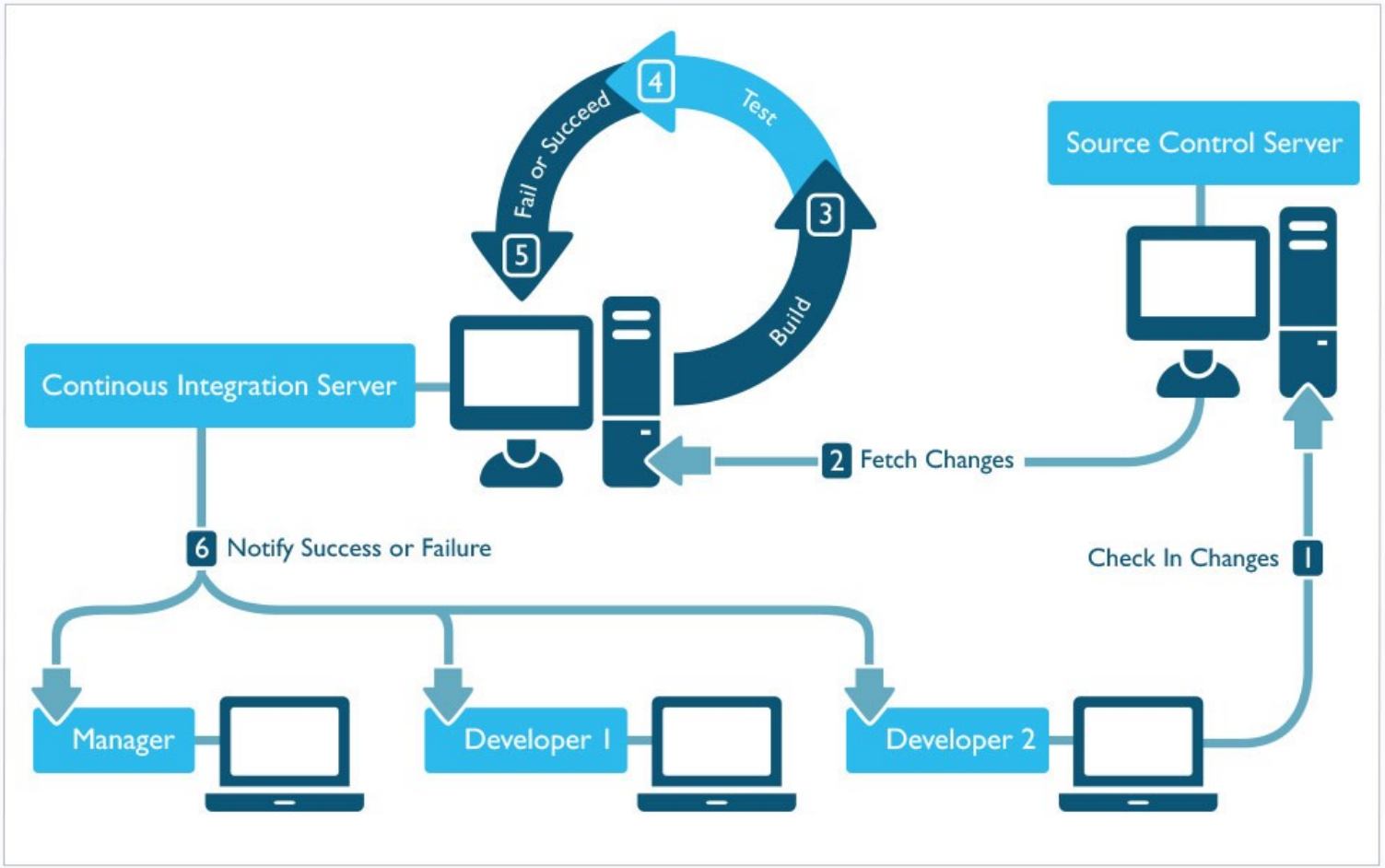


Test et méthodes agiles

Refactoring avec des outils de test unitaires (comme JUnit)



Test et intégration continue



Outils de test pour la programmation

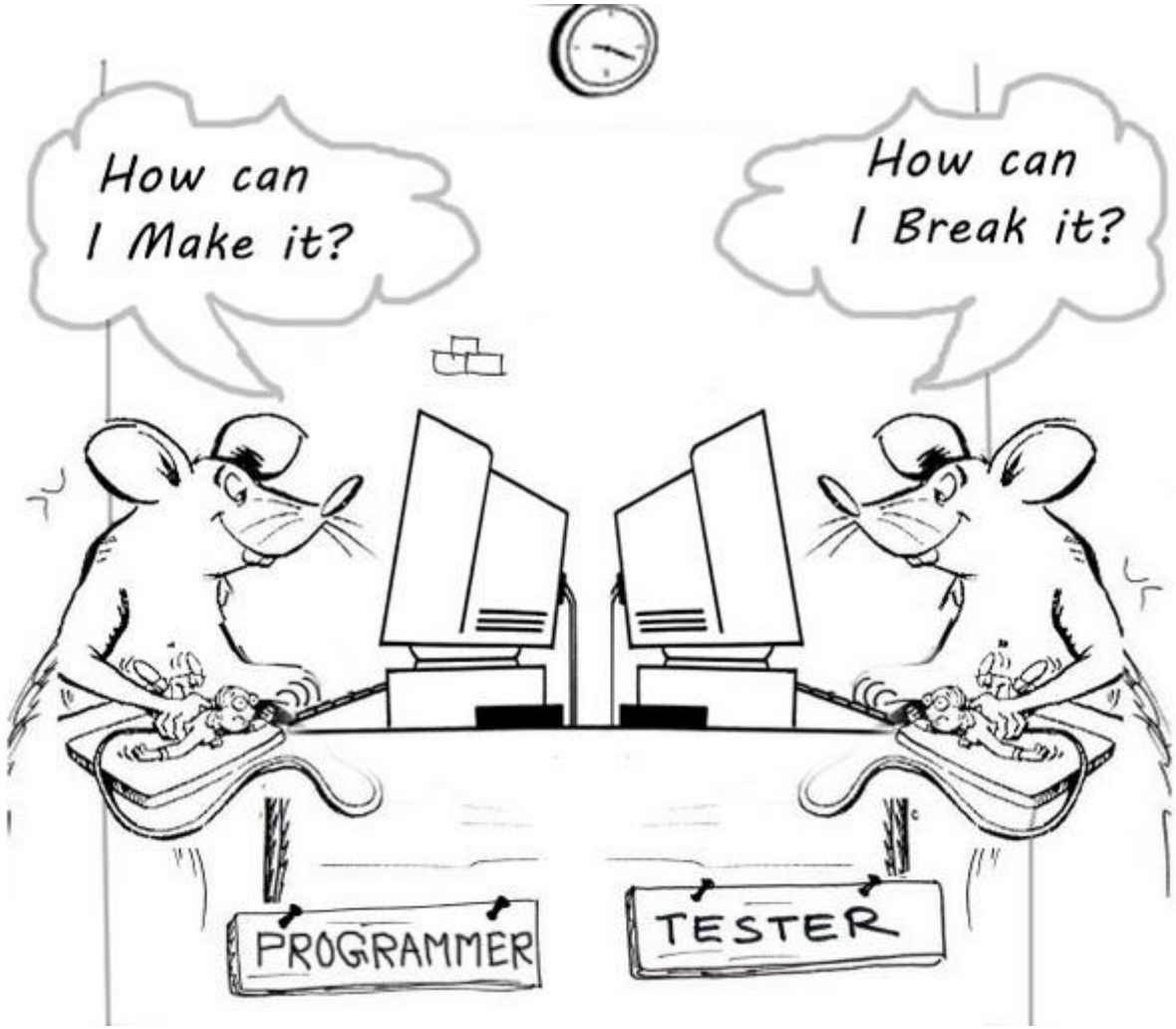
- En CM : le test structurel ou test boîte blanche
- En TD : exercices de calcul de test structurels
- En TP (pour cette partie) : prise en main...
  - d'un environnement d'exécution de tests (JUnit, Mockito),
  - d'un outil de mesure de couverture (JaCoCo),
  - d'un outil d'automatisation de production de logiciels (Maven),
  - d'un environnement de développement intégré (IntelliJ IDEA),
  - d'un debugger
  - d'un environnement de gestion de version (GitLab),
  - d'un environnement d'intégration continue (GitLab CI) et
  - d'un environnement de test d'applications Web (Selenium)

+ méthodes agiles (F. Peureux)

Quelques références ...

- Livre de référence du cours : *Le test des logiciels* – Hermès, 2000. S. Xanthakis, P. Régnier, C. Karapoulios
- Autres ouvrages :
  - *Software Engineering* – Addison-Wesley, 6th ed. 2001. I. Sommerville
  - *Test logiciel en pratique* – Vuibert Informatique, 2002. J. Watkins
  - *The Art of Software Testing* – John Wiley & Sons Inc, 1979. G. Myers
  - *Practical Model-Based Testing* – Morgan Kaufmann, 2006. M. Utting et B. Legeard
- Plus d'une centaine d'ouvrages sur le test de logiciels
- Liens web : `http://www.faqs.org/faqs/software-eng/testing-faq/`

Pour résumer



Quelques citations sur le test

- *Software Testers : Always looking for trouble.*
- *Software Testing is Like Fishing, But You Get Paid.*
- *Software Testers : "Depraved minds...Usefully employed." Rex Black*
- *Software Testing : Where failure is always an option.*
- *Software Testers don't break software ; it's broken when we get it.*
- *To err is human ; to find the errors requires a tester.*
- *If developers are so smart, why do testers have such job security ?*
- *A good tester has the heart of a developer...in a jar on the desk.*
- *If your software works, thank a tester.*
- *In God we trust. For the rest, we test.*