A la chasse aux bugs : comment rendre l'informatique plus sûre

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages
Académie des sciences, Académie des technologies
gerard.berry@college-de-france.fr
http://www.college-de-france.fr/site/gerard-berry/index.htm

Montpellier, GDR GPL, 14 juin 2017

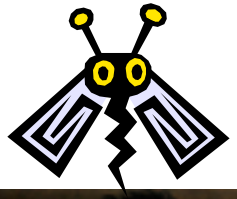
As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought.

Debugging had to be discovered.

I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.

Maurice Wilkes, 1949

Homme / ordinateur, un gouffre à combler

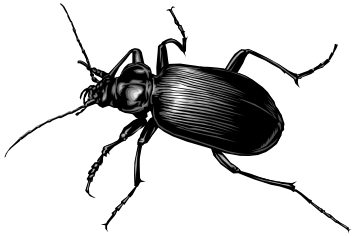


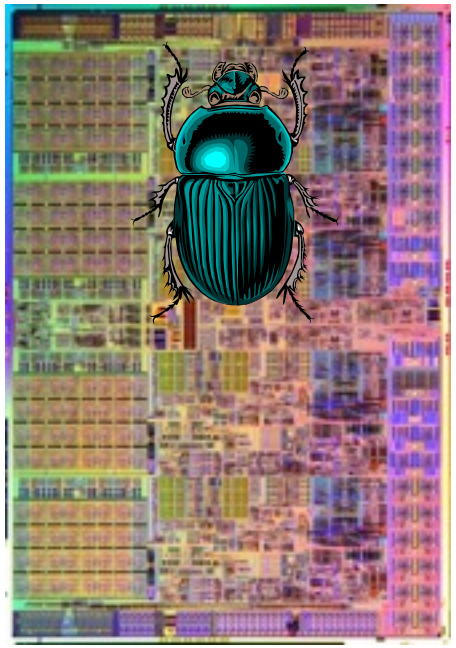
Intuition

Rigueur

Lenteur

Maîtrise ?





Stupidité

Exactitude

Rapidité

L’ordinateur est le plus extraordinaire
amplificateur d’erreurs
jamais construit !

Un bug n'est pratiquement jamais une erreur de la machine ou du logiciel

C'est une erreur humaine, faite par ceux qui ont conçu le circuit ou écrit le programme !

Les petits bugs, même non fonctionnels, créent ces trous de sécurité qui font les délices des hackers

Un des grands enjeux de la science informatique est de prévenir, éliminer ou contrôler les bugs

Rapport sur un contrôle moteur de Toyota

There are a large number of functions that are overly complex. By the standard industry metrics some of them are untestable, meaning that it is so complicated a recipe that there is no way to develop a reliable test suite or test methodology to test all the possible things that can happen in it. Some of them are even so complex that they are what is called unmaintainable, which means that if you go in to fix a bug or to make a change, you're likely to create a new bug in the process. Just because your car has the latest version of the firmware -- that is what we call embedded software -- doesn't mean it is safer necessarily than the older one….And that conclusion is that the failsafes are inadequate. The failsafes that they have contain defects or gaps. But on the whole, the safety architecture is a house of cards. It is possible for a large percentage of the failsafes to be disabled at the same time that the throttle control is lost.

Michael Barr, expert de la justice américaine
(rapport de 800 pages)

Prise de contrôle à distance de Jeep Cherokee

https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Leur code est un cauchemar : un logiciel qui laisse les hackers envoyer des commandes par l'autoradio de la Jeep vers son moteur, son tableau de bord, sa direction, ses freins, sa transmission, tout ça depuis un PC quelconque qui peut être à l'autre bout du pays…

Il ne suffit plus de vérifier la fonctionnalité.
Les trous de sécurité informatique viennent
le plus souvent de micro-bugs non fonctionnels

cf. Internet Explorer, Java, Acrobat Reader,

chantage sur les hôpitaux aux USA,
SCADA Siemens, ouverture des BMW,
prise de contrôle à distance de Jeeps Cherokee, etc.

Trous de sécurité des SmartThings de Samsung

The easiest way to turn your home into a smart home

- Boîte de commande locale, serveur sécurisé dans le nuage pour la mise en place et la modification de l'installation
- Code serveur fermé, mais API ouverte aux apps externes
- Mauvais design de la logique fondée sur des capacités et du système de messagerie
- Les applications obtiennent trop de capacités et de messages
- Hacks : ouverture de la porte, injection de faux codes PIN émission de fausses alarmes, allumage du four, etc.
- Corriger pourrait poser des problèmes aux applis existantes !

Earlence Fernandes, Jaeyeon Jung, and Atul Prakash
Security Analysis of Emerging Smart Home Applications
In Proceedings of 37th IEEE Symposium on Security and Privacy, May 2016

Comment nourrir les bugs

- Ne pas comprendre les spécificités de l'informatique
  - bugs vus comme des « pannes de logiciels » - FAUX !
  - logiciel vu comme « mécanique plus légère et plus souple »
- Employer des raisonnements valables ailleurs mais pas en informatique
  - redondance par simple duplication du système
  - calculs de « probabilités de bugs » - MEFIANCE
- Mal spécifier, mal documenter, mal maintenir
  - mauvaises méthodes de travail, mauvais outils
- Limiter les tests à la « marche normale »
  - les problèmes sont dans les coins et dans le non prévu
  - les trous de sécurité sont dans le non-fontionnel

Les bugs sont des pannes des hommes
pas des pannes des machines !

De la conception à l'implémentation

analyse

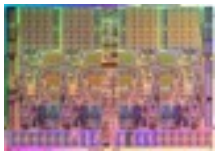
spécification

programmation

Nid à bugs

intégration

implémentation



Les problèmes des spécifications informelles

- Contradiction ou incohérence
  - page 12 : X est une variable entière positive
  - page 33 : si X est négatif, ...
- Sous-spécification
  - oublier de préciser les contraintes d'environnement du programme
  - oublier de spécifier précisément les cas d'erreurs
  - en cas de survenue d'alarmes multiples et rapprochées, le logiciel devra réagir en conséquence
- Sur-spécification
  - donner des détails inutiles compliquant la réalisation
    ex. : caractéristiques physiques irrelevantes,
    exigence d'un outillage de développement inadapté

Il est impossible de réaliser une bonne application
à partir de spécification incohérentes, ou même laides

Comment réduire les bugs

- Utiliser de bonnes méthodes de design
  - spécifications, langages et déboguers à base formelle
  - caractérisant l'environnement autant que de l'application
- Rendre tout visible et explorable (→ traçable)
  - procédures de développement, spécification, code, documentation, tests et résultats de test, etc.
- Faire des revues indépendantes
  - processus de certification
  - communautés open source
- Tester systématiquement
  - les composants, leur intégration, la non-régression
  - l'application globale sur la cible réelle ou par simulation

21e siècle : vérifier formellement
avec des outils automatiques ou semi-automatiques

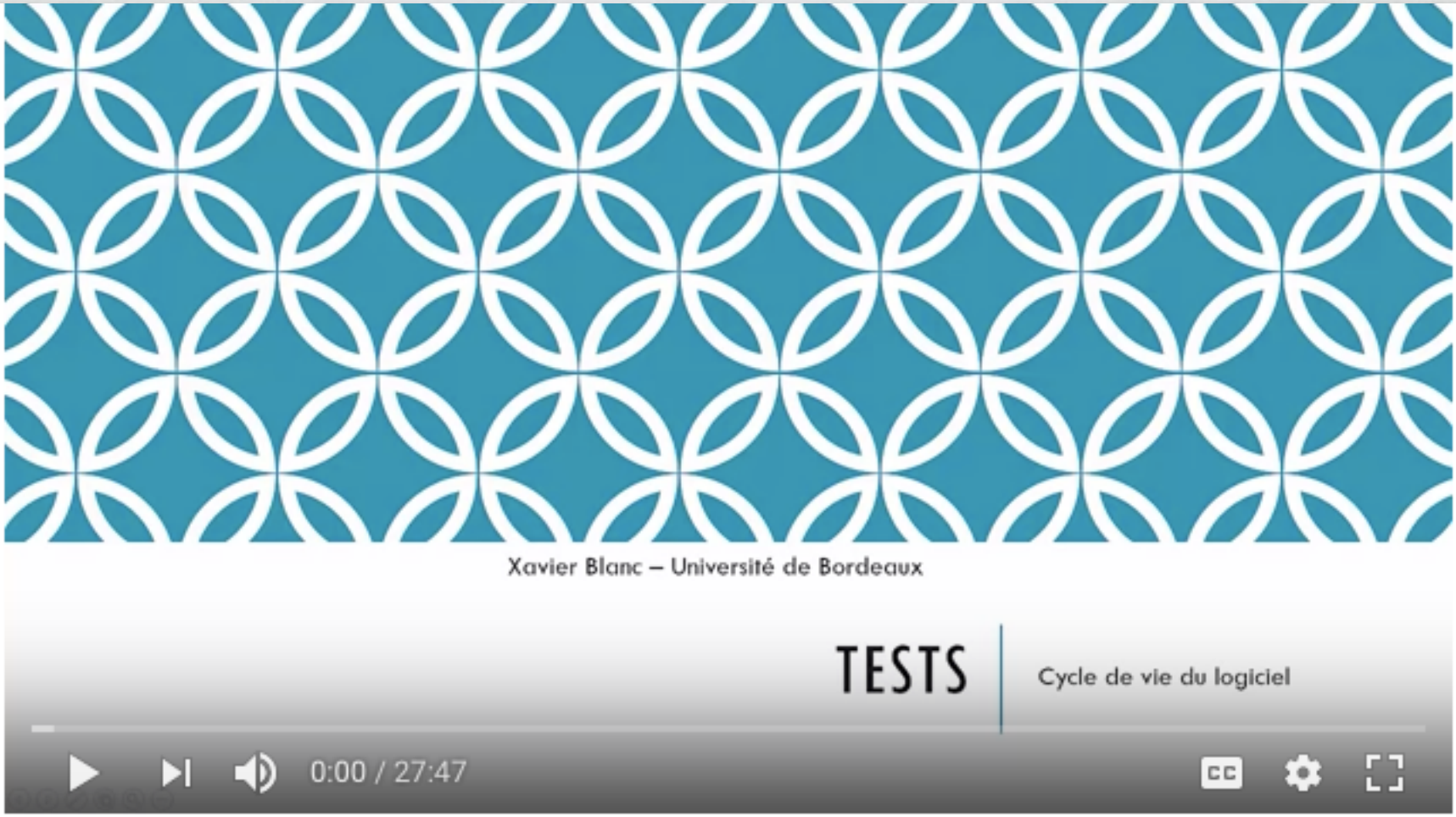
Tests logiciels

Kent Beck « Une fonctionnalité sans test automatique n'existe tout simplement pas»

Penser que les tests [et le refactoring] ralentissent le développement
c'est comme penser que prendre des voyageurs ralentit le bus
David Evans

Merci à tous ceux qui ont rendu leurs cours et exposés
disponibles sur le web & dans les livres,
voir Biblio.

M. Blay-Fornarino
blay@unice.fr,
IUT Département Informatique


Xavier Blanc – Université de Bordeaux
TESTS
Cycle de vie du logiciel
0:00 / 27:47
CC

Le testeur n'est pas un justicier solitaire

http://fr.slideshare.net/crochas/le-test-nest-pas-quune-histoire-agile-france-2016-63179563

http://fr.slideshare.net/crochas/le-test-nest-pas-quune-histoire-agile-france-2016-63179563

Industrial Robotics Evolves Very Fast!

Industrial robots are now complex cyber-physical systems
(motion control and perception systems, multi-robots sync.,
remote control, Inter-connected for predictive maintenance, ...)

They are used to perform safety-critical tasks in complete autonomy
(high-voltage component, on-demand painting with color/brush change, ..)

And to collaborate with human co-workers

Testing Robotic Systems is Crucial and Challenging

- The validation of industrial robots still involve too much human labour
- *"Hurry-up, the robots are uncaged!"*: Failures are not anymore handled using fences
- Robot behaviours evolve with changing working conditions
- Today, industrial robots can be taught by-imitation. Tomorrow, they will learn by themselves

More automation in testing

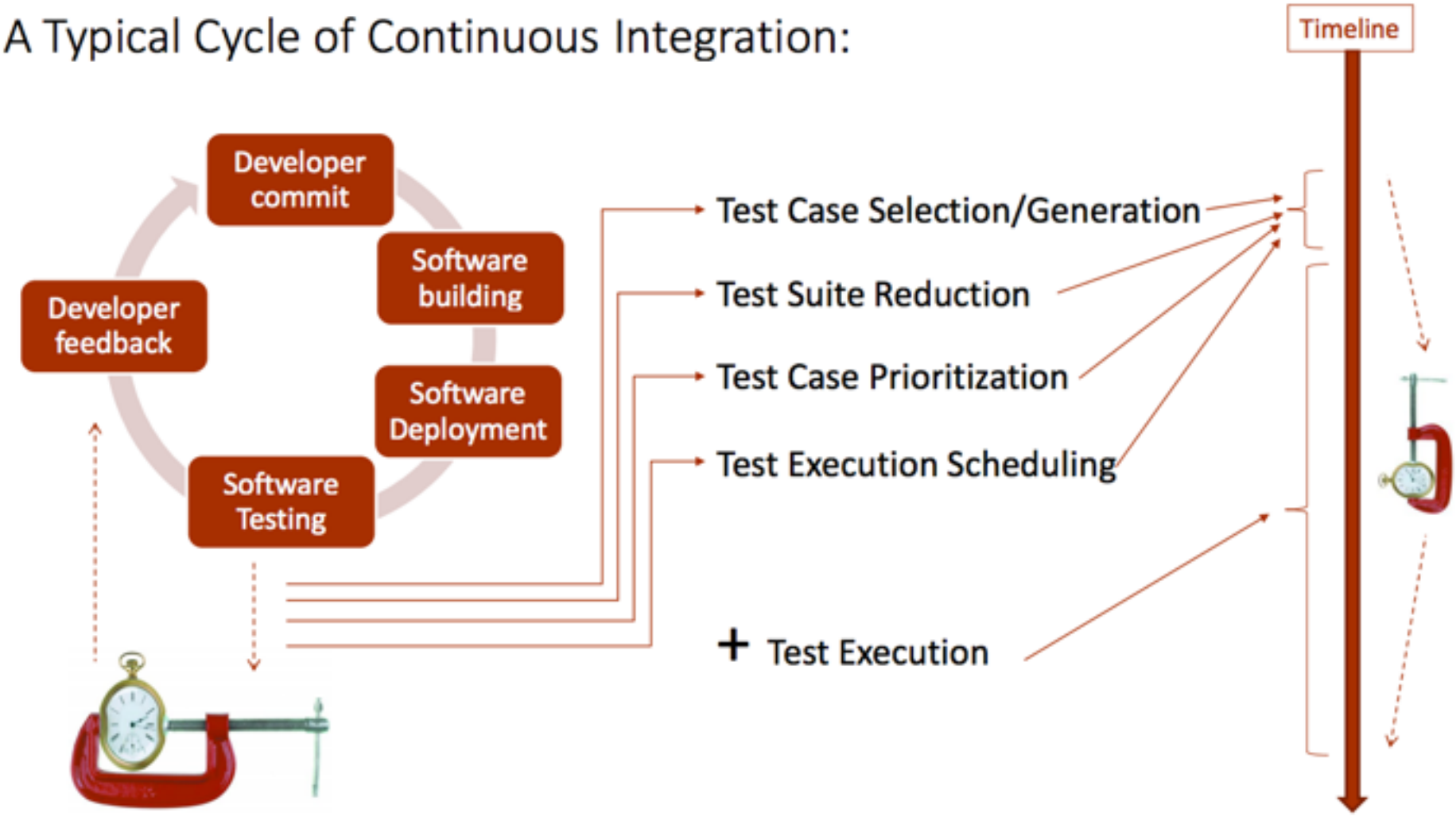
More diversity in testing

More efficiency in testing

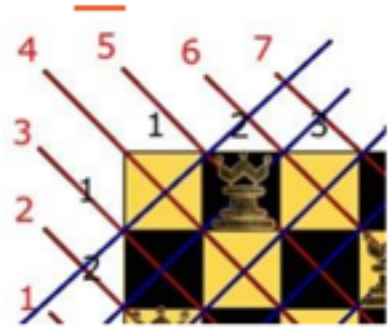
How Software Development of Industrial Robots Has Evolved...

From....	To...
Single-core, single application system	Multi-core, complex distributed system
All source code maintained by a small team located at the same place	Subsystems developed by distinct teams located at distinct places in the world
Manual system testing only handled in a single place, on actual robots	Automated software testing handled in a continuous integration process

A Typical Cycle of Continuous Integration:



Our Focus : Artificial Intelligence for Testing of Robotic Systems



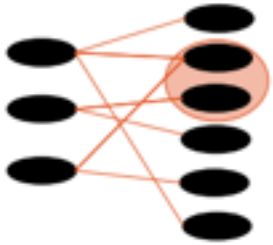
Constraint Modelling

1. Automatic Test Case Generation

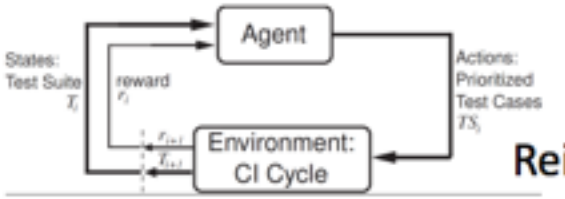
2. Test Suite Reduction

3. Test Execution Scheduling

4. Test Case Prioritization



Global Constraints



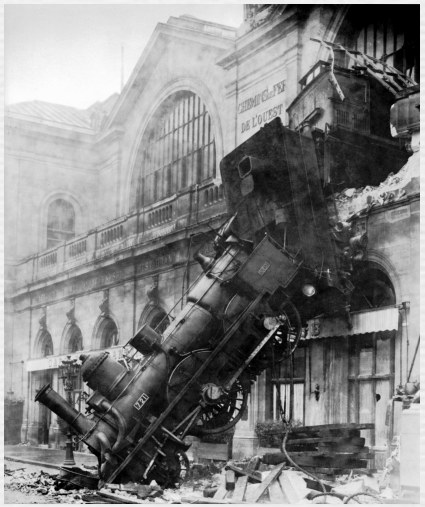
Reinforcement Learning

Constraint-based Scheduling

Bibliographie

- Programmation par les tests, ESIREM, Céline ROUDET
- Comment écrire du code testable, Conférence Agile France 2010, Florence CHABANOIS
- Reflexion on Software Quality and Maintenance, Alexandre Bergel, Chili
- An Introduction to Test-Driven Development (TDD), Craig Murphy
- Tests et Validation du logiciel, http://home.nordnet.fr/~ericleleu
- Test à partir de modèles : pistes pour le test unitaire de composant, le test d'intégration et le test système, Yves Letraon
- Les tests en orienté objet, J. Paul Gibson http://www-inf.int-evry.fr/cours/CSC4002/Documents
- Mocks and Stubs, Martin Fowler
- Introduction au test du logiciel, Premiers pas avec JUnit, Mirabelle Nebut
- Écrire du code testable Par Aurélien Bompard

Des bugs et des tests



- Objectifs
  - **Savoir où sont les bugs,** pour
    - Les corriger ou
    - les documenter et donner des contournements :
      - de version en version on voit les corrections apparaître (même dans le hard il y a des versions);
      - Eviter la mauvaise image de la découverte du bug par le client
  - **Eviter le update qui régresse**, c'est pire qu'un bug présent, ...
  - Oser améliorer son code sans avoir peur d'introduire de «nouveaux» bugs
  - Retour sur les exigences(bugs show product usage)
- En conclusion : 80% du code sert à tester les cas d'erreur (principe de Pareto)

Tests...

- unitaire
- integration
- GUI
- non regression
- coverage
- load
- stress
- performance
- scalability
- volume
- usability/utilisateurs
- security
- recovery
- L10N/l18N (langue?, symbole)
- accessibility
- Installation/configuration
- Documentation
- Platform
- samples/tutorial
- code inspections

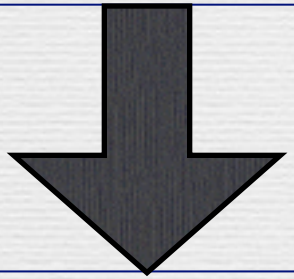
Qui teste ?

- L'utilisateur
- Les collègues en charge du test (s'il y en a)
- Le **développeur** : il a le devoir de fournir un code le plus clair et le mieux testé possible....
- La **machine**

Qu'est-ce qu'on teste et comment ?

Quoi ?

- Fonctionnalité
- Sécurité / intégrité
- Utilisabilité
- Cohérence
- Maintenabilité
- Efficacité
- Robustesse
- Sûreté de



Comment ?

- **Test statique :**
  - relecture / revue de code
  - analyse automatique (vérification de propriétés, règles de codage ...)
- **Test dynamique :**
  - exécution du programme, et observation du comportement en fonction des valeurs en entrée.

Une spécification exprime ce qu'on attend du système, elle prend différentes formes en fonction de ce qui est ciblé : cahier des charges, use cases, données numériques, ...

S'organiser pour tester

- Module contenant les tests, indépendant du code « réel »
- Faire des tests indépendants les uns des autres
  - -Pas de tests imbriqués
  - -Un test qui échoue ne fait pas échouer les tests suivants
- Automatiser les tests
  - -Maven : compiler et lancer automatiquement tous les tests
  - -Utiliser un Environnement de tests : framework JUnit
  - -Tests Unitaires/Tests de non régression
- Réutiliser les tests et leurs résultats comme documentation

Construire des suites de tests

- A field can accept integer values between 20 and 50.
- What tests should you try?

A Test Ideas List for Integer-Input Tests

- Common answers to the exercise would include:

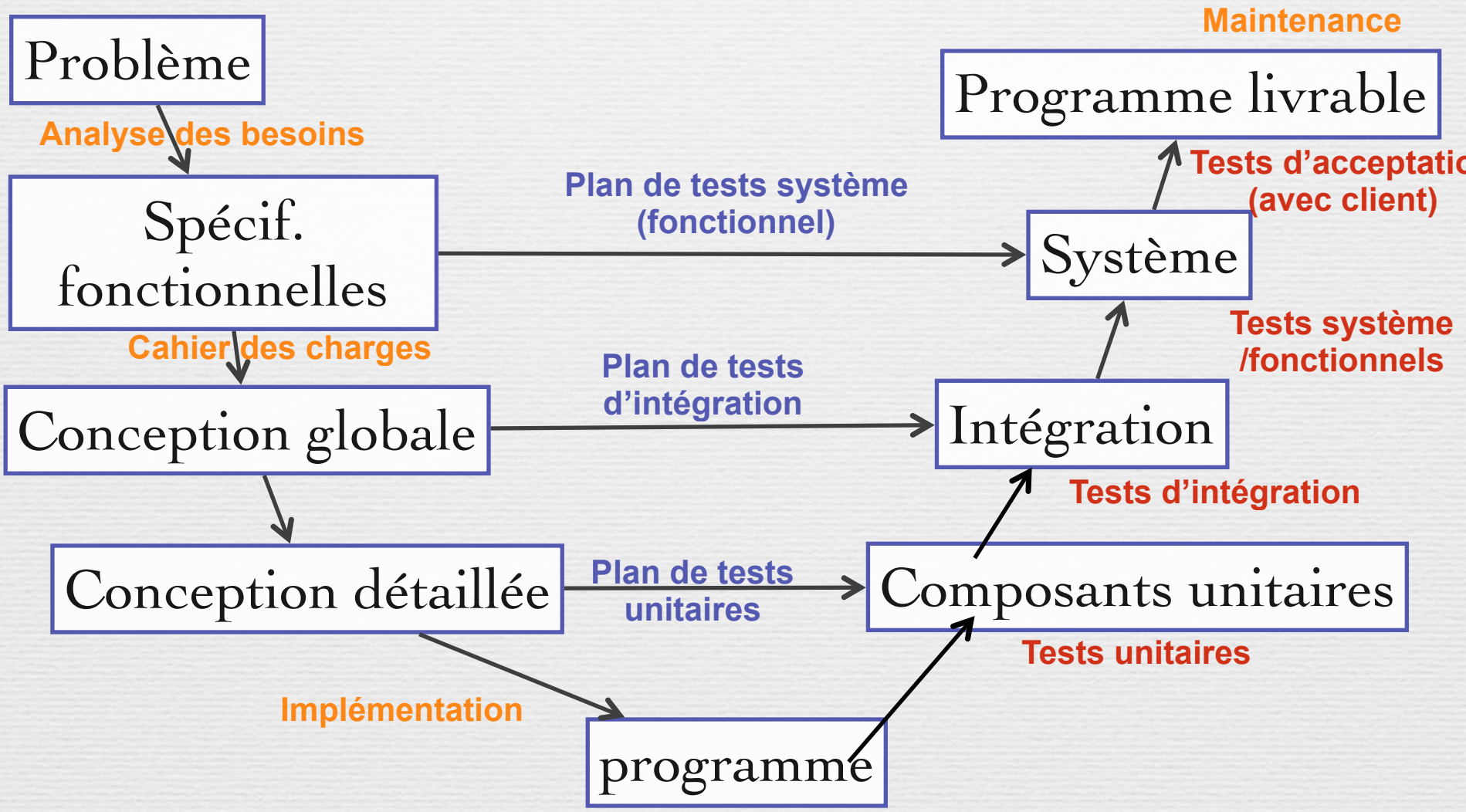
Test	Why it's interesting	Expected result
20	Smallest valid value	Accepts it
19	Smallest -1	Reject, error msg
0	0 is always interesting	Reject, error msg
Blank	Empty field, what's it do?	Reject? Ignore?
49	Valid value	Accepts it
50	Largest valid value	Accepts it
51	Largest +1	Reject, error msg
-1	Negative number	Reject, error msg
4294967296	2^32, overflow integer?	Reject, error msg

Types de Tests

Types de tests

- Tests en boîte noire: par ex. Tests système/fonctionnels
  - -Utilise la description des fonctionnalités du programme
  - -Provenant des spécifications (scenarii, uses cases)
- Tests en boîte blanche: par ex Tests structurels
  - -Utilise la structure interne du programme
- Tests de (non) régression (après modifications)
  - -Correction et évolution ne créent pas d'anomalies nouvelles
- Tests de robustesse
  - -Cas de tests correspondant à des entrées non valides
- Tests de performance (application intégrée dans son environnement)
  - -load testing : résistance à la montée en charge
  - -stress testing : résistance aux demandes de ressources anormales

Hiérarchisation des tests dans le cycle en V



Diagrammes UML et Tests

- Niveau Application (spécification)
  - Diagramme des cas d'utilisation
  - Diagramme de classes : test des associations, des agrégations
    - multiplicité,
    - création, destruction
  - Diagramme de séquence : test de séquences
    - construction d'un graphe de flot
- Niveau Classes (conception détaillée)
  - Classes détaillées
  - Diagrammes de machine à états

Tests Système/Fonctionnel

Tests d'intégration

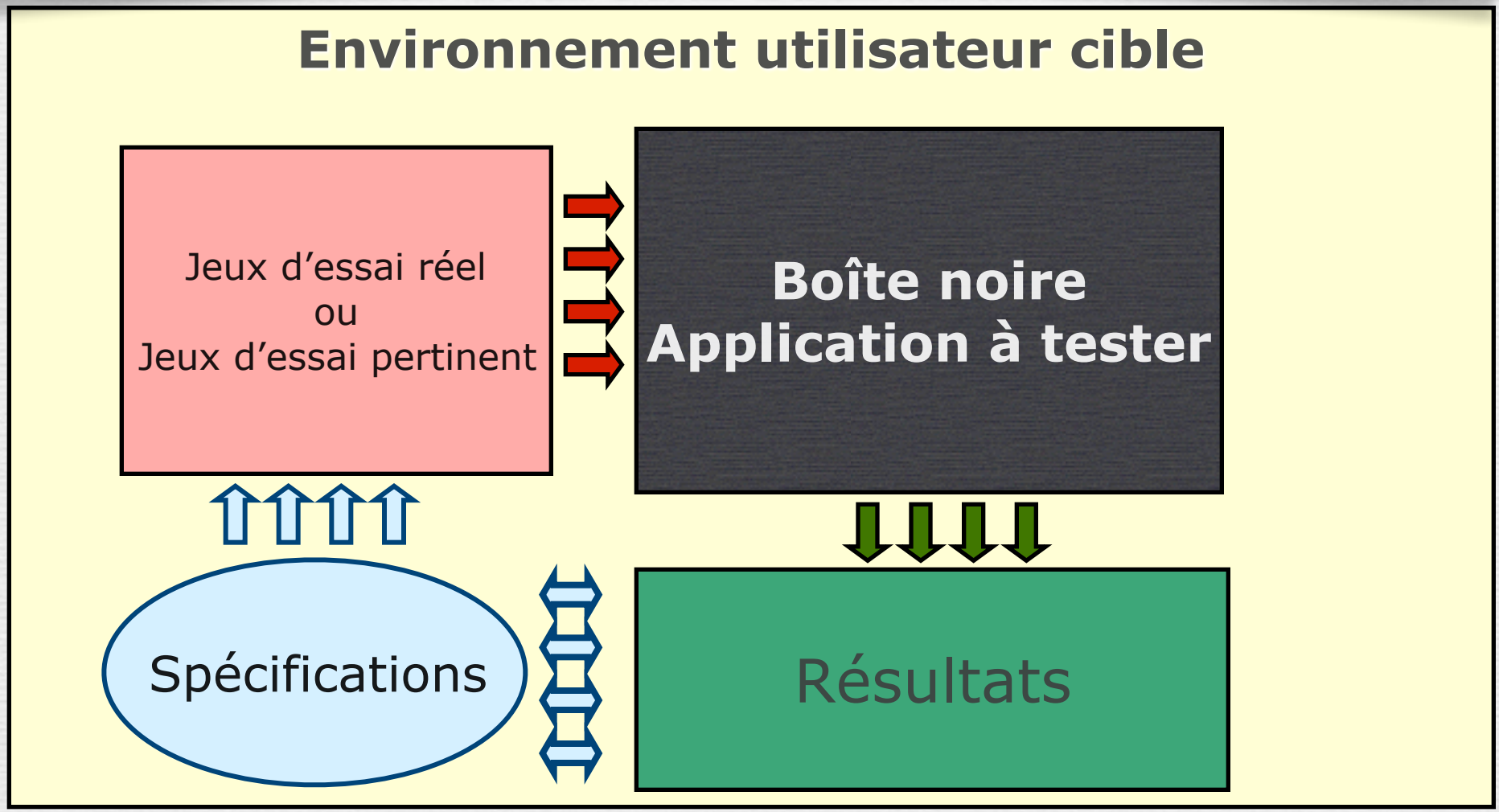
Tests d'intégration

Tests unitaires

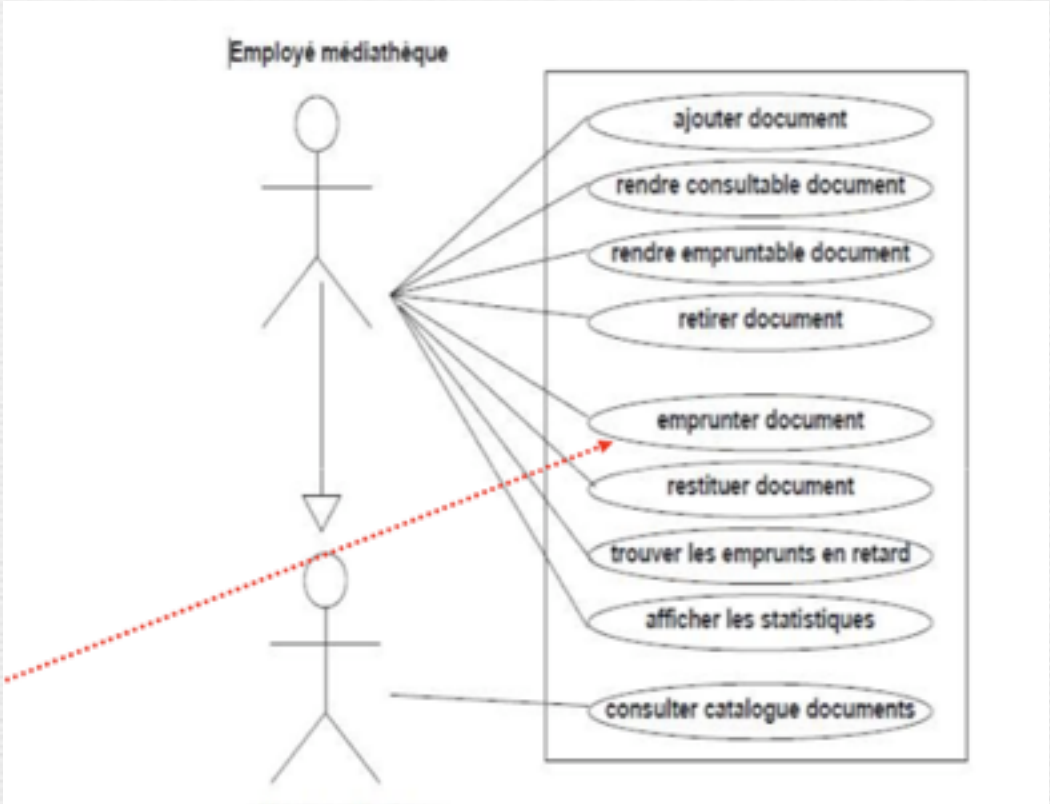
Tests Fonctionnels

Test Système (fonctionnel)

Vérifier que les fonctions correspondant aux attentes sont *bien atteintes*



Tests fonctionnels & UML



Tests fonctionnels & UML

Données d'entrée

client: peut être inscrit ou non;

emprunts: déja effectués par le client

- existe-t-il un emprunt en retard ?
- le nombre d'emprunts déja effectués correspond-il au nombre maximum de ce client ?

document:

- existe?
- empruntable ou consultable?,
- déjà emprunté ou disponible?

Tests fonctionnels & UML

Données de sortie
`Emprunt` accepté ou refusé.

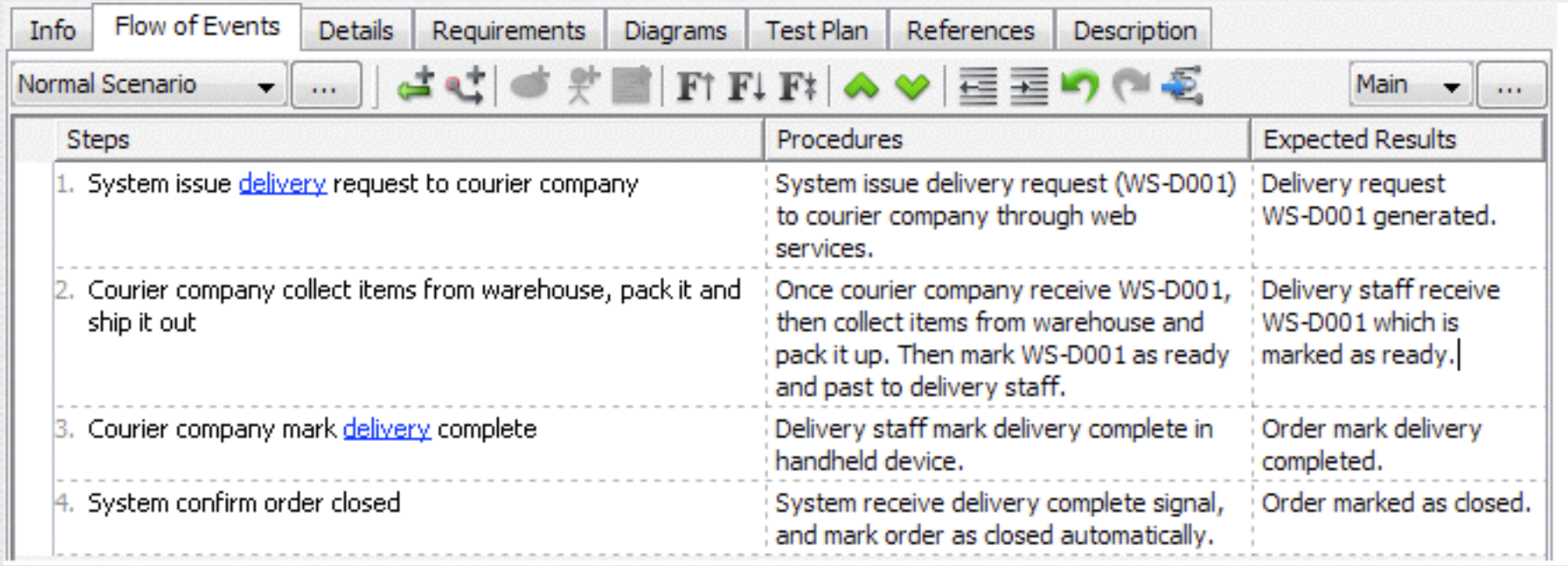
Remarque: la définition des jeux de tests de validation pour le cas d'utilisation `emprunter document` permet de soulever au moins les questions suivantes (à poser au client):

- un abonné qui n'est pas à jour de sa cotisation peut-il tout de même emprunter un document?
- doit-il être considéré comme un client au tarif normal tant qu'il n'a pas renouvelé son abonnement?
- ou doit-il se réabonner avant de pouvoir emprunter un document?

D'une manière générale, la préparation des jeux de tests permet de lever les ambiguïtés et réparer des oublis.

Use cases et Tests

- Pour chaque cas d'utilisation, sélectionner les scénarios et définir les tests correspondants.



Info | Flow of Events | Details | Requirements | Diagrams | Test Plan | References | Description

Normal Scenario ... Main ...

Steps	Procedures	Expected Results
1. System issue delivery request to courier company	System issue delivery request (WS-D001) to courier company through web services.	Delivery request WS-D001 generated.
2. Courier company collect items from warehouse, pack it and ship it out	Once courier company receive WS-D001, then collect items from warehouse and pack it up. Then mark WS-D001 as ready and past to delivery staff.	Delivery staff receive WS-D001 which is marked as ready.
3. Courier company mark delivery complete	Delivery staff mark delivery complete in handheld device.	Order mark delivery completed.
4. System confirm order closed	System receive delivery complete signal, and mark order as closed automatically.	Order marked as closed.

http://www.visual-paradigm.com/product/vpuml/tutorials/testingprocedure.jsp

Tests fonctionnels & US

En tant que passager PRIMO, **je veux** annuler ma réservation,...

Seules les taxes « Autres » me sont remboursées moins les frais administratifs

En tant que passager BUSINESS FLEX,, **je veux** annuler ma réservation, ...

L'ensemble du billet m'est remboursé à 100% moins les frais administratifs

Tests fonctionnels...

http://seleniumhq.org/docs/

http://watir.com

Une démonstration...

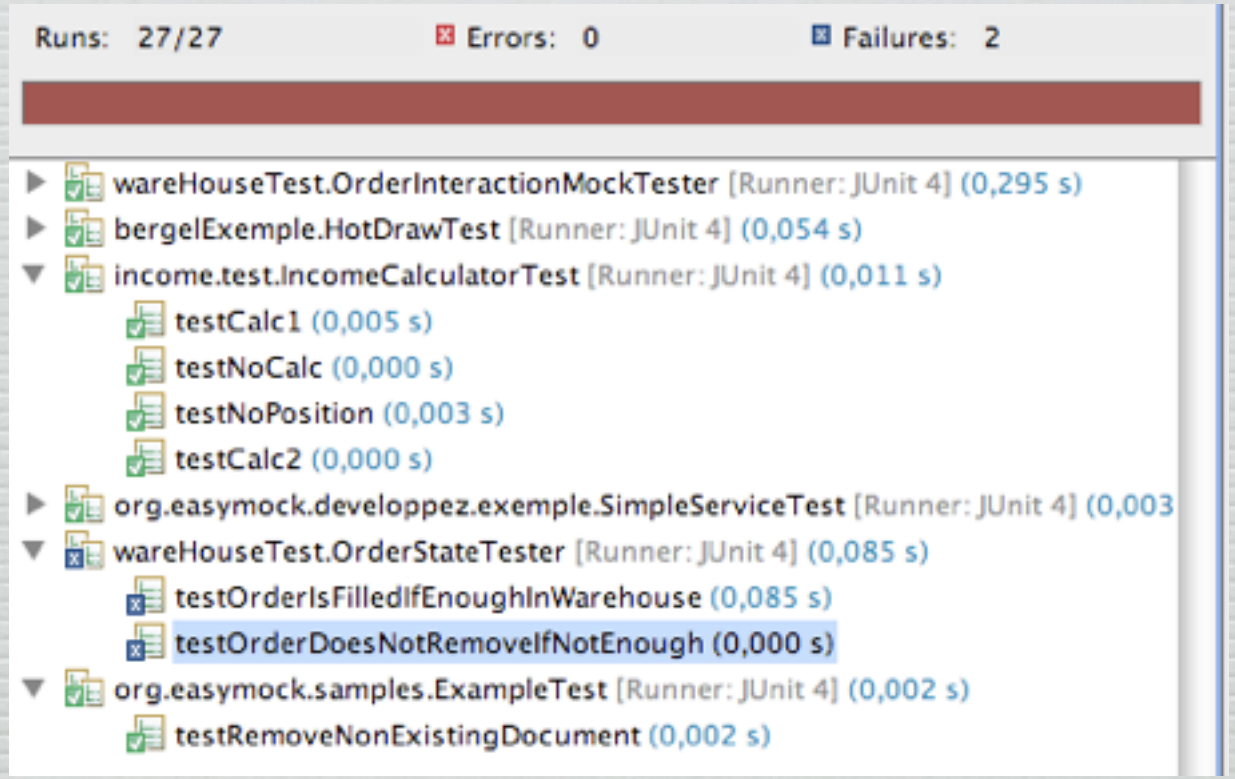
http://www.opensourcetesting.org

Prise en compte de la variabilité

« Au sein de la société, nos utilisions donc les principaux navigateurs utilisés : Google Chrome (49,8% des utilisateurs) , Safari (14,9%) , Internet Explorer/ Edge (14,8%) et Mozilla Firefox (7,7%) . »

Tests Unitaires

exemple JUnit

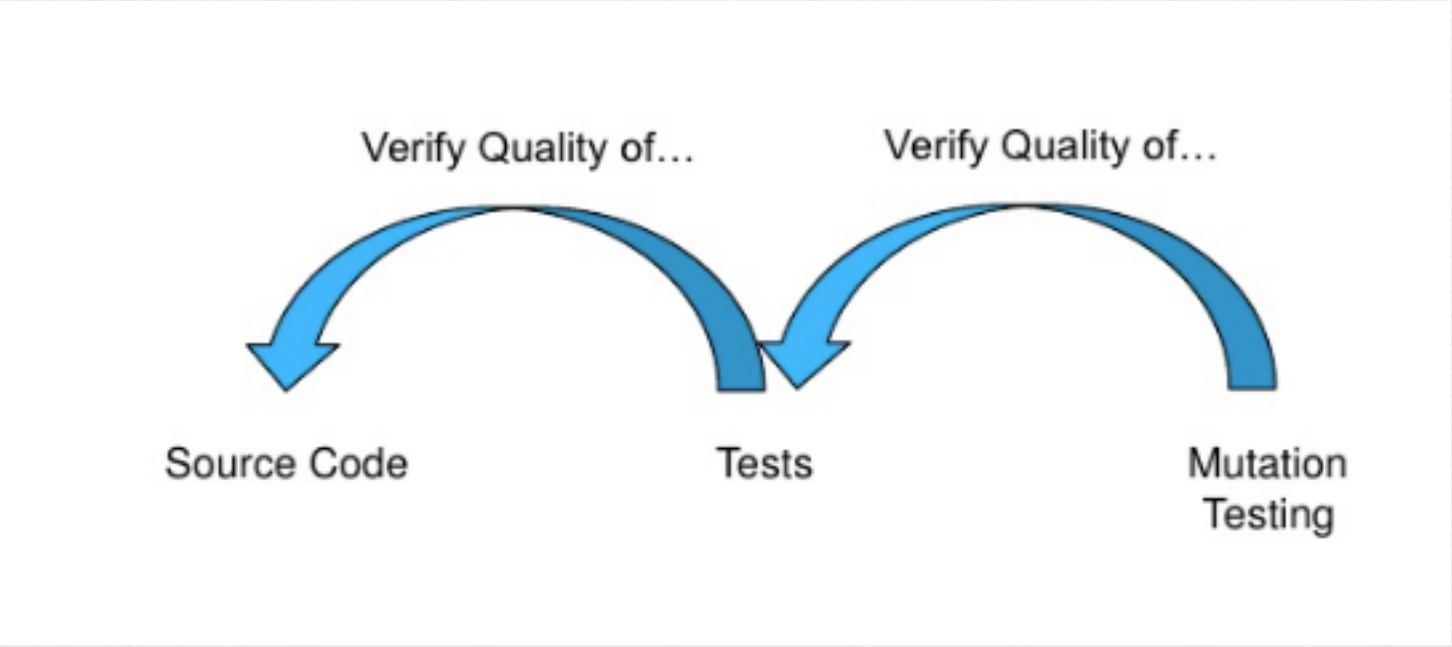


Tests Unitaires
(déjà étudiés en AP)

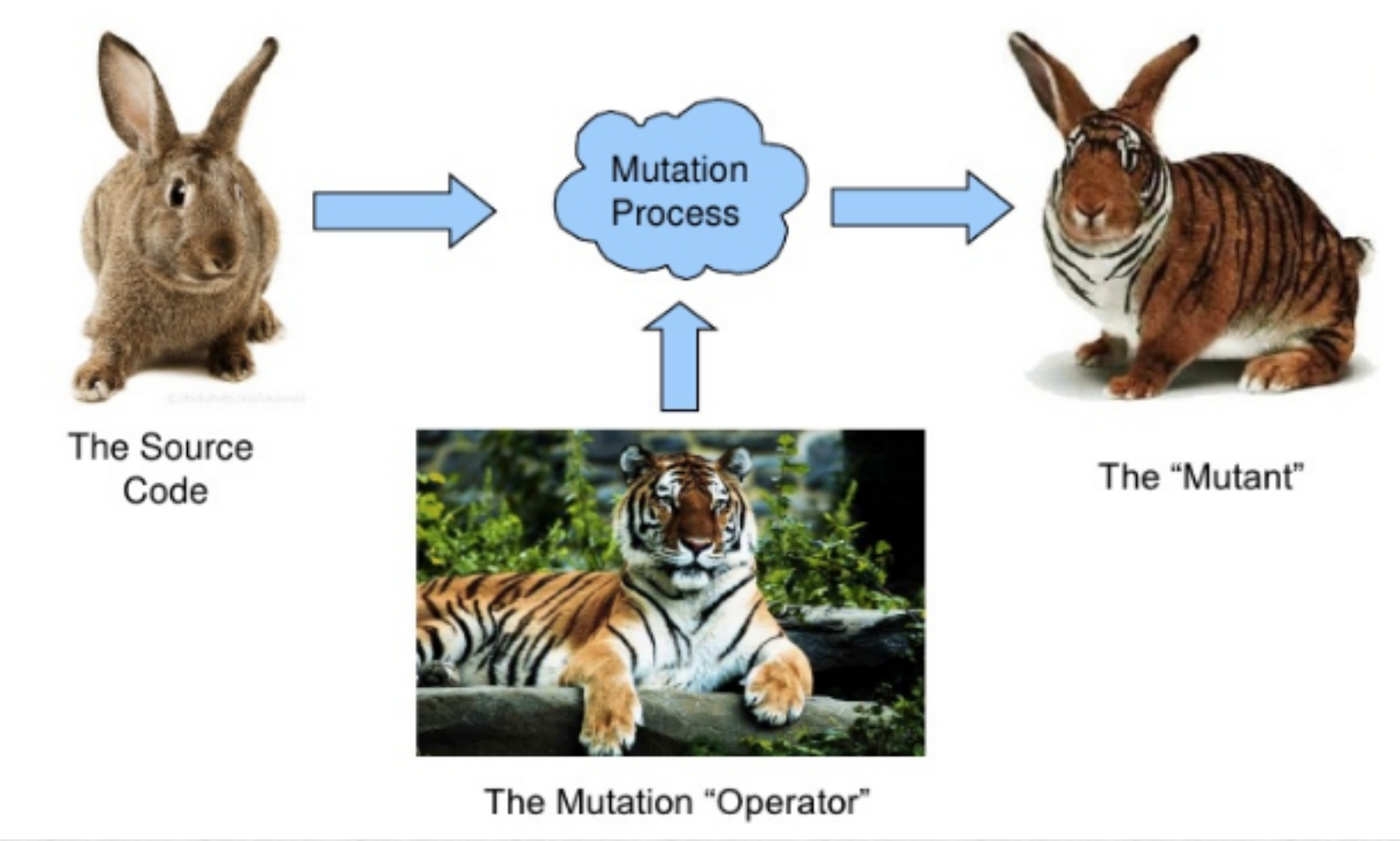
- Tester une unité logicielle isolée du reste du système
  - Plus petit composant compilable
  - Pour un langage procédural : unité de test = procédure
  - Pour un langage objet : unité de test = classe
  - Test de **l'interface**
  - Les unités sont-elles suffisamment spécifiées ?
  - Le code est-il lisible, maintenable ...?
- Importance de la notion de contrats (pré/post)

Tests par mutation

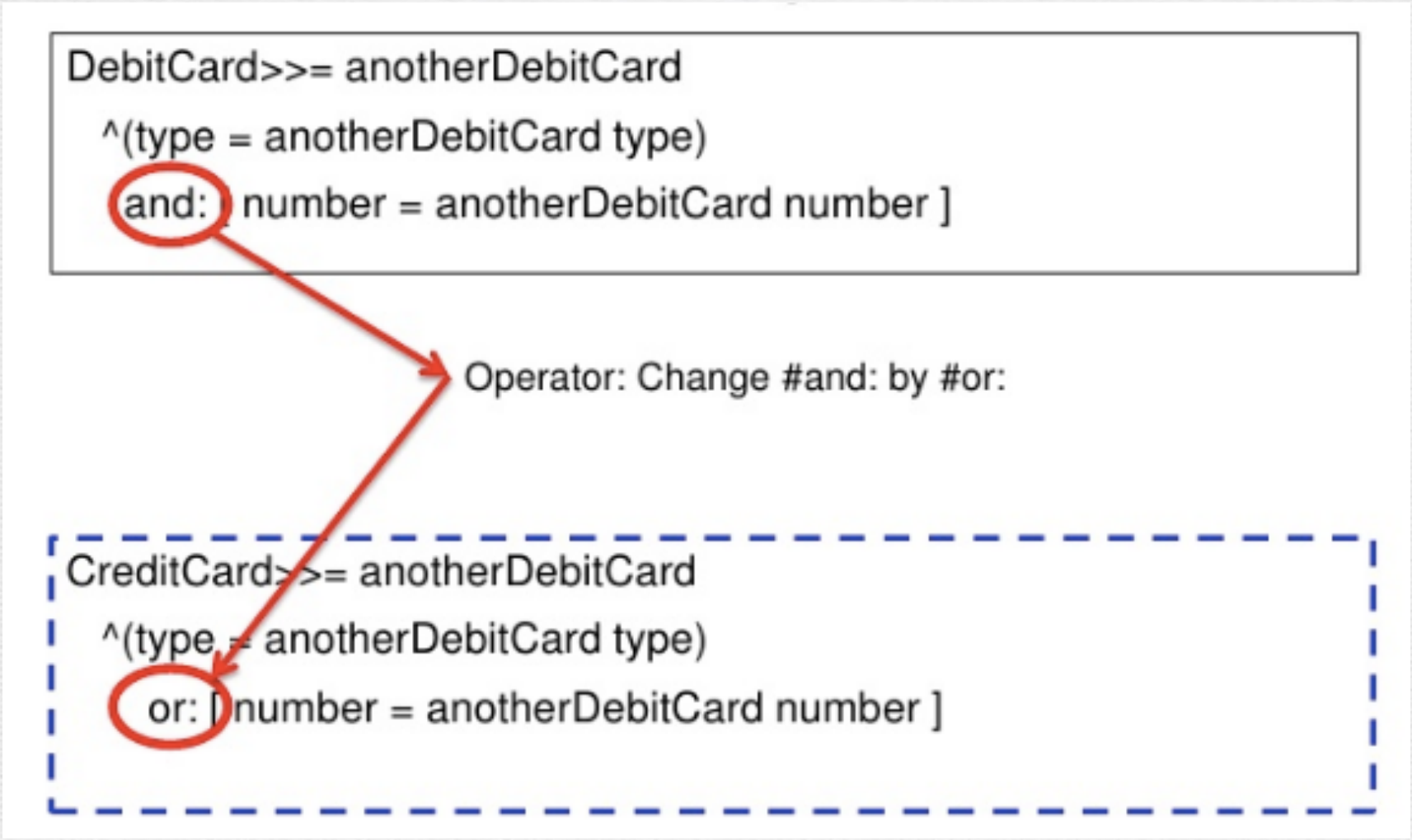
- Technique pour vérifier la qualité des tests



Comment cela fonctionne ?



Transformations?



Tuer les mutants ?

The "Mutant"

A Killer
tries to kill the Mutant!

The Test Suite

All tests run → The Mutant Survives!!!

A test fails or errors → The Mutant Dies

Tests d'intégration (voir cours dédié)

✓Différents modules d'une application peuvent fonctionner unitairement, leur intégration, entre eux ou avec des services tiers, peut engendrer des dysfonctionnements.

✓Il est souvent impossible de réaliser les tests unitaires dans l'environnement cible avec la totalité des modules à disposition.

➡Les tests d'intégration ont pour objectif de créer une version complète et cohérente du logiciel (avec l'intégralité des modules testés unitairement) et de garantir sa bonne exécution dans l'environnement cible.

The Practical Dev a retweeté

The Practical Dev @ThePracticalDev · 14 janv.

2 unit tests. 0 integration tests

Autres tests...

Usability Testing

- Purpose: Find out is the system is really usable by its intended audience

- Why?
  - -System is built by developers … but used by Business Users
  - -Even minimal UI changes can confuse business
  - -users with years of experience of “doing it this way”
  - -System has to face real-life usage

Usability Testing

- How: Almost impossible to automate
- Tips:
  - Involve ergonomic specialists early in the project
  - Use reusable, standardized UI components
  - Take performance into account: a slow responding system won't be accepted easily
  - Have Business Users test early on UI mock

Usability Testing

- **Tools**: Eye tracking

Extraits d'un rapport de tests utilisateurs

Méthodologie de test

➔ Déroulement des tests :

6 entretiens individuels, d'un durée moyenne de **1h30** chacun, ont été réalisés du **18 au 26 juin 2014** à Nice et Sophia Antipolis. Lors de chaque passation, l'interface a été enregistrée via le **logiciel Morae** et le comportement des utilisateurs a été filmé grâce à une webcam. Les extraits vidéo les plus pertinents seront communiqués au laboratoire I3S.

Chaque test s'est déroulé en **trois étapes** :

1. Entretien pré-test (questions sur le profil du participant),
2. Test de l'interface (plusieurs tâches à réaliser),
3. Entretien post-test (questionnaire de satisfaction et évocation).

➔ Tâches réalisées :

Tâche A : « Vous avez besoin de diffuser des informations sur un écran lors d'un événement. Vous souhaitez diffuser un diaporama photo avec fondu ainsi que de brèves actualités. »

Tâche B : « Vous pouvez maintenant retourner en page d'accueil. »

Tâche C : « Vous souhaitez modifier votre écran d'affichage. Vous voulez ajouter un calendrier avec le diaporama photos (dans la même zone). »

Tâche D : « Vous souhaitez construire un écran dans le cadre de votre travail, selon vos envies et besoins. Vous êtes libre. Comment vous y prenez-vous ? »

Profil des participants

UTILISATEURS

Nathalie	Françoise	Sophie	Laurence	Olivier	Clément
47ans	35ans	36ans	41ans	37ans	30ans
Secrétaire	Chargé de communication	Chargé d'affaires	Ingénieur d'Etudes, Responsable administratif	Chargé de projets	Animateur socioéducatif
depuis plus de 10 ans	depuis plus d'1 an	depuis plus de 5 ans	depuis plus de 5 ans	depuis 4 ans	depuis plus de 5 ans

Tous pourraient être amenés à configurer un diffuseur d'informations dans le cadre de leur travail.

Profil des participants

INTERNET

Des testeurs connectés en permanence (4/6)

sinon plusieurs heures par jour

Niveau en

ANGLAIS

3 avancés
1 intermédiaire
2 débutants

Connaissances en

GRAPHISME

1 beaucoup
3 un peu
2 pas du tout

Connaissances en

PROGRAMMATION

Aucun
0

3/6 utilisent des comparateurs de prix, des outils de configuration en ligne

Classification des principaux problèmes

	Fréquence	Gravité	Resultats (sévérité)
Absence d'aperçu	4	3	Très haut
Incompréhension du mode de sélection	4	3	Très haut
Manque d'explications	4	3	Très haut
Vocabulaire confus	4	3	Très haut
Absence de plan	4	2	Haut
Undo complexe	4	2	Haut
Choix inutiles	3	2	Moyen
Icônes peu parlantes	3	2	Moyen
Contenu de la rubrique « help »	3	2	Moyen
Impossibilité de supprimer une configuration	2	2	Moyen
Affordance de la colonne de droite	4	1	Moyen
Langue anglaise	1	3	Bas

Test de charge

- Volume des données traitées
- Nombre d'utilisateurs simultanés
  - Exemple de Système RESARAIL qui permet d'obtenir les horaires, les prix, effectuer la réservation et émettre des billets SNCF

Nécessité d'une bonne estimation des besoins dès le début et prise en compte de leur évolution

Prenons un exemple

*« XX ayant des problèmes de ralentissements importants voir de crash assez régulièrement il a été décidé de lancer une campagne de **tests de performance**, … Trois principales étapes composent la campagne de tests : la création de scénarios fonctionnels, le scripting de ces scénarios et les tirs de performance.*

Rapport d'apprentissage, 2016, Kévin Detti

Création de tests fonctionnels

La première chose à faire pour débuter la campagne est d'écrire des scénarios fonctionnels qui seront joués par l'outil de tirs. Il faut donc définir les scénarios les plus fidèles à une utilisation classique de l'outil, mais aussi les plus consommateurs afin de comprendre pourquoi l'application subit des ralentissements. »

Rapport d'apprentissage, 2016, Kévin Detti

A quoi cela correspond-il d'après vous?

Suite à cela, j'ai pu commencer la rédaction de quatre scénarios, selon le plan indiqué par l'équipe de tests, c'est-à-dire une suite d'instructions pas à pas avec une capture d'écran à chaque étape, le temps d'attente moyen d'un utilisateur entre chaque page, le nombre d'utilisateurs effectuant ce scénario en période normale et en période de pic d'activité, etc.
Ensuite, il faut constituer un jeu de données : des comptes utilisateur de tests, des données qui seront consommées pendant les tests, etc.

Rapport d'apprentissage, 2016, Kévin Detti

Et ce n'est pas toujours simple !

Un des problèmes rencontré a été l'insuffisance du jeu de données pour un des scénarios. Nous avions environ 140 données, mais selon nos calculs pour l'ensemble de la campagne de test, 7000 données auraient été nécessaires. Créer des données à la main étant bien trop long et fastidieux, nous avons dû trouver une solution rapide et efficace… nous avons convenu de créer un nouveau scénario qui serait joué en masse et qui permettrait de créer les données nécessaires.

Rapport d'apprentissage, 2016, Kévin Detti

Tests de performance en entreprise

La phase des tirs se déroule en 3 étapes :

- Un tir nominal, soit environ 1h30 en utilisation classique, puis une montée en charge pour simuler un pic d'activité,
- Un tir de stress soit environ 1h30 en simulation de pic d'activité, puis une montée en charge jusqu'au crash de l'application,
- Un tir d'endurance, soit 8h d'utilisation classique et plusieurs montées en charge pour simuler des pics d'activité au cours de la journée.

Limites du test

- L'espace des entrées -La *complexité*
- Les séquences d'exécution -La *complexité*
- Sensibilité aux fautes –Transformation entre le *fini* et *l'infini*
- 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
- Comment être sûr qu'un système de test est correct … il faut le tester?

Tests Statiques

A voir ultérieurement

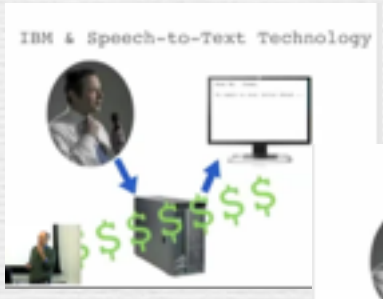
next

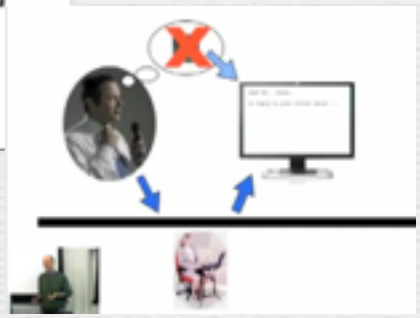
Encore plus en amont, le «preprototype» ;-)

Pretotyping	Prototyping
• **Investment:** hours, days	• **Investment:** days, weeks
• **Main Q:** Would we use it?	• **Main Q:** Can we build it?
• **Deliverable:** [Working] pretotype	• **Deliverable:** Working prototype

Palm «pre»

- now beats later
- doing beats talking
- simple beats complex
- commitment beats committees





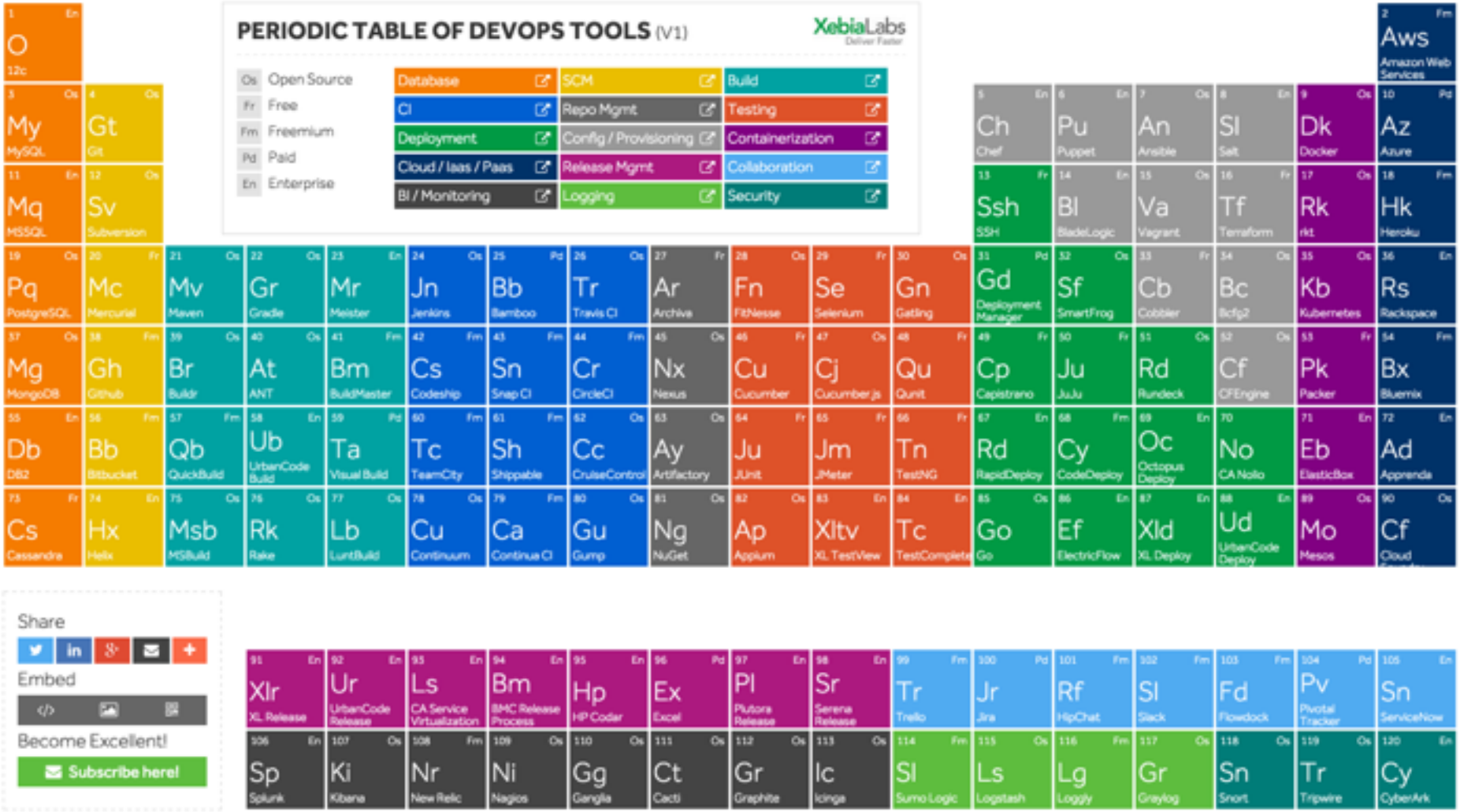
http://www.pretotyping.org/

"Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead." -- Martin Fowler

Les Tests sont un tuteur

http://emmanuelchenu.blogspot.com/


DevOps…
Tester
en continu.
C’est aujourd’hui.



https://xebialabs.com/assets/files/infographics/periodic-table-of-devops.png