Introduction aux tests du logiciel

F.X. Fornari
xavier.fornari@esterel-technologies.com
P. Manoury
pascal.manoury@pps.univ-paris-diderot.fr

2014

Contenu du cours

- Le test logiciel: Nous en faisons, mais qu'est-ce précisément ?

Contenu du cours

- Le test logiciel: Nous en faisons, mais qu'est-ce précisément ?
- Quand et comment fait-on du test ?

Contenu du cours

- Le test logiciel: Nous en faisons, mais qu'est-ce précisément ?
- Quand et comment fait-on du test ?
- But du cours: sensibiliser au test en général:
  - Le sujet est vaste
  - les applications variées

Le Logiciel

- Q0: Qu'est-ce que du logiciel ?

Le Logiciel

- Q0: Qu'est-ce que du logiciel ?
- des besoins, exprimés par un client
- une gestion de projet
  - cycle de vie (classique, agile, ...) avec ses délivrables
  - les outils et l'environnement de développement
- une spécification: la définition du logiciel en réponse aux besoins
- une conception: globale, architecturale, détaillée. Les composants et leurs échanges sont définis
- du code
- un *produit*

Le Logiciel

- Q0: Qu'est-ce que du logiciel ?
- des besoins, exprimés par un client
- une gestion de projet
  - cycle de vie (classique, agile, ...) avec ses délivrables
  - les outils et l'environnement de développement
- une spécification: la définition du logiciel en réponse aux besoins
- une conception: globale, architecturale, détaillée. Les composants et leurs échanges sont définis
- du code
- un *produit*
- ... des tests !

Tester un logiciel

- Q1: Que veut dire “tester un logiciel” ?

Tester un logiciel

- Q1: Que veut dire “tester un logiciel” ?
- C’est valider sa conformité,

Tester un logiciel

- Q1: Que veut dire “tester un logiciel” ?
- C’est valider sa conformité,
- Par rapport à des exigences
  C’est-à-dire par rapport à l’ensemble de la spécification, et de la conception du logiciel

Tester un logiciel

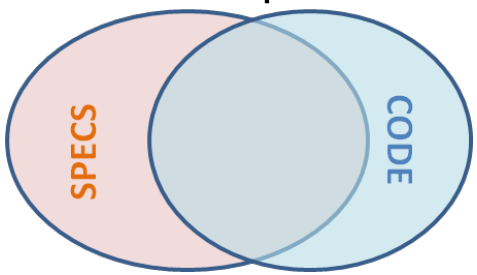
- Q1: Que veut dire “tester un logiciel” ?
- C’est valider sa conformité,
- Par rapport à des exigences
  C’est-à-dire par rapport à l’ensemble de la spécification, et de la conception du logiciel
- En fonction de critères.
  Par exemple, domaines d’entrées, taux d’erreurs acceptable, ...

- Q2: Y-a-t-il plusieurs types de tests ?

- Q2: Y-a-t-il plusieurs types de tests ?
- Il y a différentes méthodes:
  - Test boîte noire-blanche,
  - Prédiction d'erreur (on sait ce qu'on cherche)
  - Tests automatiques, ...
- il y a différents types de tests: fonctionnels, non-fonctionels, structurels

Types de tests

- Fonctionnels:
  - *est-ce que le logiciel est conforme à la spécification ?*
  - liés à la spécification, à la qualité, à la performance, à l'interfaçage, ...
- Non-Fonctionnels:
  - *est-ce que son usage est conforme ?*
  - liés à la configuration, la compatibilité, à la documentation, au *stress*, ...
- Structurels:
  - *est-ce que le codage est correct ?*
  - fautes d'implémentation, ou fonctions non prévues.



- Q3: Y-a-t-il des alternatives ?

- Q3: Y-a-t-il des alternatives ?
  - méthodes formelles:
    - *model-checking*, SAT, ...,
    - méthode par raffinements (B)
    - Interprétation abstraite (estimation d'erreur, runtime-error, ...)

    Ces méthodes sont de plus en plus employées. Problème: adéquation modèle réalité, passage à l'échelle

- Q3: Y-a-t-il des alternatives ?
  - méthodes formelles:
    - *model-checking*, SAT, ...,
    - méthode par raffinements (B)
    - Interprétation abstraite (estimation d'erreur, runtime-error, ...)

    Ces méthodes sont de plus en plus employées. Problème: adéquation modèle réalité, passage à l'échelle
  - relecture de code: détection d'erreurs statiques, mais pas dynamiques.
    - C'est lourd, mais ça favorise la connaissance du code.
    - Les règles de codage sont importantes, et peuvent être vérifiées par des outils

- Q3: Y-a-t-il des alternatives ?
  - méthodes formelles:
    - *model-checking*, SAT, ...,
    - méthode par raffinements (B)
    - Interprétation abstraite (estimation d'erreur, runtime-error, ...)

    Ces méthodes sont de plus en plus employées. Problème: adéquation modèle réalité, passage à l'échelle
  - relecture de code: détection d'erreurs statiques, mais pas dynamiques.
    - C'est lourd, mais ça favorise la connaissance du code.
    - Les règles de codage sont importantes, et peuvent être vérifiées par des outils
  - analyse de sécurité: validation d'une architecture. C'est l'identification en amont de tous les problèmes potentiels.

- Q3: Y-a-t-il des alternatives ?
  - méthodes formelles:
    - *model-checking*, SAT, ...,
    - méthode par raffinements (B)
    - Interprétation abstraite (estimation d'erreur, runtime-error, ...)

    Ces méthodes sont de plus en plus employées. Problème: adéquation modèle réalité, passage à l'échelle
  - relecture de code: détection d'erreurs statiques, mais pas dynamiques.
    - C'est lourd, mais ça favorise la connaissance du code.
    - Les règles de codage sont importantes, et peuvent être vérifiées par des outils
  - analyse de sécurité: validation d'une architecture. C'est l'identification en amont de tous les problèmes potentiels.
- Q4: Coût du test ?

- Q3: Y-a-t-il des alternatives ?
  - méthodes formelles:
    - *model-checking*, SAT, ...,
    - méthode par raffinements (B)
    - Interprétation abstraite (estimation d'erreur, runtime-error, ...)

    Ces méthodes sont de plus en plus employées. Problème: adéquation modèle réalité, passage à l'échelle
  - relecture de code: détection d'erreurs statiques, mais pas dynamiques.
    - C'est lourd, mais ça favorise la connaissance du code.
    - Les règles de codage sont importantes, et peuvent être vérifiées par des outils
  - analyse de sécurité: validation d'une architecture. C'est l'identification en amont de tous les problèmes potentiels.
- Q4: Coût du test ?
  - 30 à 40% du coût de développement, voire plus. Mais un bug peut coûter encore plus cher...

Un métier à part entière

- Seule activité dans le cycle de développement où l'on peut voir toutes les fonctionnalités d'un produit.
- Différent des développeurs spécialisés,

Un métier à part entière

- Seule activité dans le cycle de développement où l'on peut voir toutes les fonctionnalités d'un produit.
- Différent des développeurs spécialisés,
- C'est une activité créatrice:
  - il faut imaginer les scenarii pouvant mettre le logiciel en défaut,
  - il faut imaginer les bancs de tests, les environnements de simulations (logiciel de conduite de train, de controle des commandes de vol $\Rightarrow$ comment faire ?)
  - demande rigueur et compétence

Un métier à part entière

- Seule activité dans le cycle de développement où l'on peut voir toutes les fonctionnalités d'un produit.
- Différent des développeurs spécialisés,
- C'est une activité créatrice:
  - il faut imaginer les scenarii pouvant mettre le logiciel en défaut,
  - il faut imaginer les bancs de tests, les environnements de simulations (logiciel de conduite de train, de controle des commandes de vol $\Rightarrow$ comment faire ?)
  - demande rigueur et compétence
- Mais c'est une activité mal perçue:
  - le testeur est en fin de chaîne $\Rightarrow$ retards
  - certains tests sont répétitifs
  - mal considéres par les développeurs

Un métier à part entière

- Seule activité dans le cycle de développement où l'on peut voir toutes les fonctionnalités d'un produit.
- Différent des développeurs spécialisés,
- C'est une activité créatrice:
  - il faut imaginer les scenarii pouvant mettre le logiciel en défaut,
  - il faut imaginer les bancs de tests, les environnements de simulations (logiciel de conduite de train, de controle des commandes de vol $\Rightarrow$ comment faire ?)
  - demande rigueur et compétence
- Mais c'est une activité mal perçue:
  - le testeur est en fin de chaîne $\Rightarrow$ retards
  - certains tests sont répétitifs
  - mal considéres par les développeurs

C'est pourtant une activité essentielle de R&D

Typologie des logiciels

Tentative de classification:

- Transformationnels: logiciel de paie, compilateur, ...
- Interactif: Windows manager, WEB, DAB, ...
- Réactifs: comportement cyclique (chaîne de production)
- Réactifs sûrs: en milieu contraints: nucléaire, avionique, défense, ...

Pour chacune de ces classes, des contraintes différentes sont à gérer.

Exemples de contraintes

- Base de données
- Web
- Compilateur
- Interface graphique
- Code embarqués
- OS

Exemples de contraintes

- Base de données
  - volume des données, intégrité
- Web
  - disponibilité, multi-navigateur, liens,
- Compilateur
  - test du langage d'entrée, des optimisations, du code généré, ...
- Interface graphique
  - multi-threading, sequences d'actions, *undo*, vivacité
- Code embarqués
  - tests sur hôte, et tests sur cibles, traçabilité
  - avionique, nucléaire, ... : exhaustivité, proche du 0 défaut
  - téléphone mobile: time-to-market très court !
- OS
  - ...

Quelques principes de base

P1: Indépendance

un programmeur ne doit pas tester ses propres programmes
Mais il vaut mieux faire ses tests avant de délivrer, et aussi les conserver !

P2: Paranoïa

Ne pas faire de tests avec l'hypothèse qu'il n'y a pas d'erreur (code trivial, déjà vu, ...) $\Rightarrow$ bug assuré !
Conseil: un test doit retourner *erreur* par défaut. Le retour *ok* doit être forcé.

P3: Prédiction

La définition des sorties/résultats attendus doit être effectuée avant l'exécution des tests. C'est un produit de la spécification.

- c'est nécessaire pour des développements certifiés
- les données sont fournies parfois au niveau système (ex: Matlab), mais les résultats seront différents à l'implementation.

P4: Vérification

- Il faut inspecter minutieusement les résultats de chaque test.
- Mais aussi la pertinence des tests
- (DO-178B): le process assure un "tuyau" propre, mais il faut quand même des filtres $\equiv$ vérification
- C'est la séparation de l'exécution et de l'analyse.

P5: Robustesse

Les jeux de tests doivent être écrits avec des jeux valides, mais aussi invalides ou incohérentes: on ne sait jamais ce qui peut arriver

P6: Complétude

Vérifier un logiciel pour vérifier qu'il ne réalise pas ce qu'il est supposé faire n'est que la moitié du travail. Il faut aussi vérifier ce que fait le programme lorsqu'il n'est pas supposé le faire

En cas de bug ?

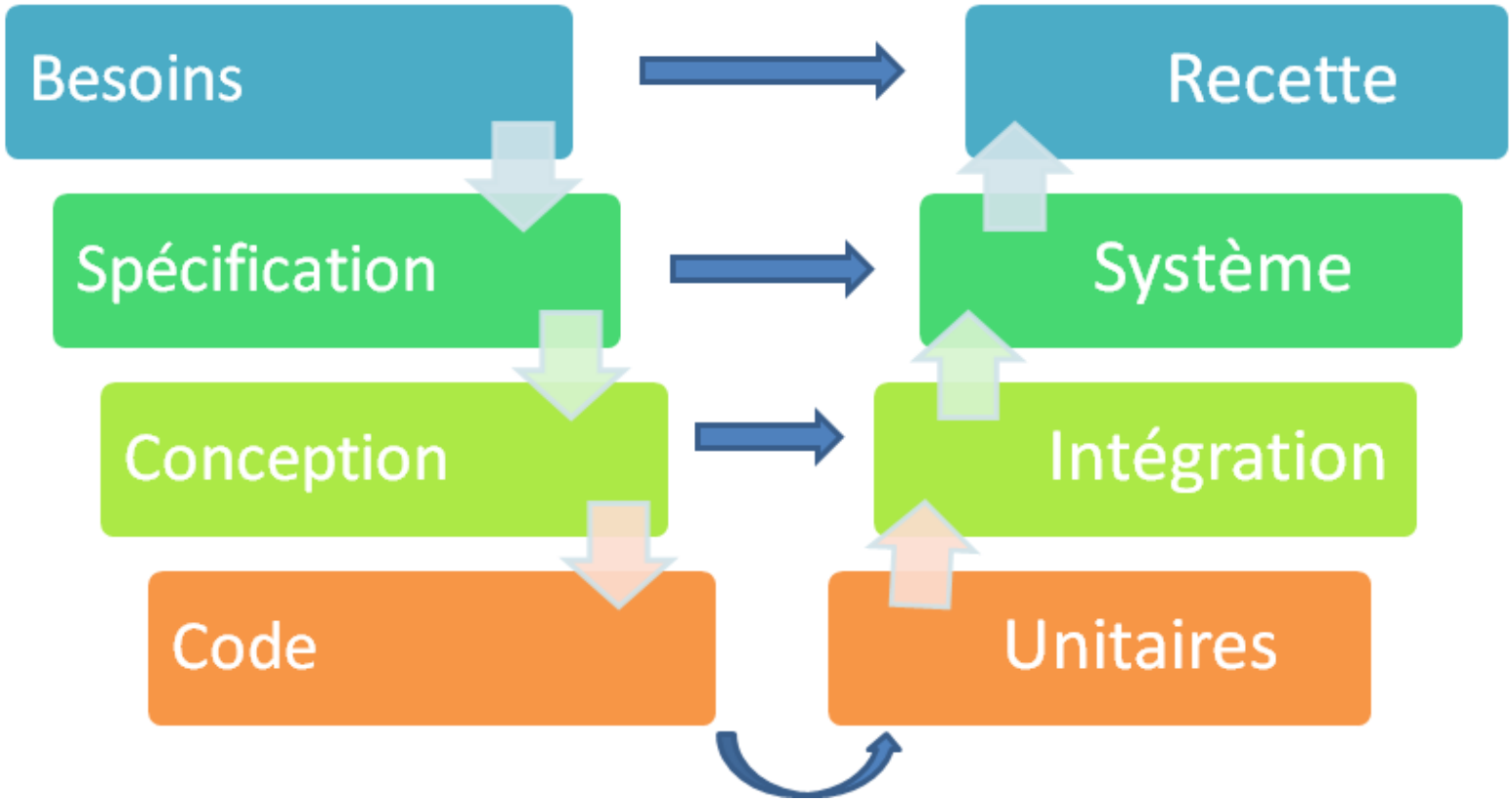
- Vérifier que le test est bien correct (P4)
- Vérifier que le problème n'est pas déjà répertorié (base de bugs par exemple)
- Etablir un rapport de bug

En cas de bug ?

- Vérifier que le test est bien correct (P4)
- Vérifier que le problème n'est pas déjà répertorié (base de bugs par exemple)
- Etablir un rapport de bug
  - donner un synopsis succint et précis
  - donner une description claire, avec tous les détails de reproduction du bug
  - si possible, essayer de réduire l'exemple.
    Renvoyer un "tas" en disant "ça marche pas" ... ne marche pas.

Quand commencer ?

Le test commence de suite ! Ici, le cycle en V. mais aussi approches incrémentale, agiles, ...



Les différents niveaux

- Tests de recette: test de réception du logiciel chez le client final

Les différents niveaux

- Tests de recette: test de réception du logiciel chez le client final
- Tests intégration système: test de l'intégration du logiciel avec d'autres logiciels

Les différents niveaux

- Tests de recette: test de réception du logiciel chez le client final
- Tests intégration système: test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme: test d'acception du logiciel avant livraison (nouvelle version par exemple)

Les différents niveaux

- Tests de recette: test de réception du logiciel chez le client final
- Tests intégration système: test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme: test d'acception du logiciel avant livraison (nouvelle version par exemple)
- Tests Integration: test de l'intégration des différents composants (avec ou sans hardware)

Les différents niveaux

- Tests de recette: test de réception du logiciel chez le client final
- Tests intégration système: test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme: test d'acception du logiciel avant livraison (nouvelle version par exemple)
- Tests Integration: test de l'intégration des différents composants (avec ou sans hardware)
- Tests Unitaires: tests élémentaires des composants logiciels (une fonction, un module, ...)

Les différents niveaux

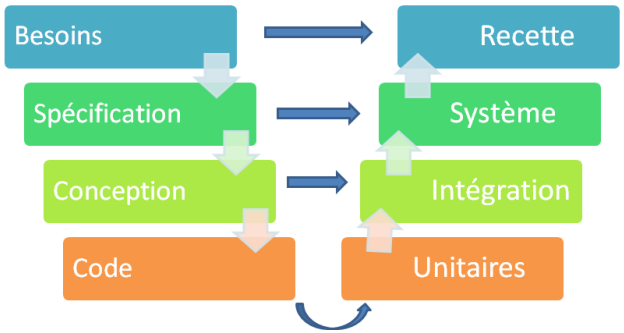
- Tests de recette: test de réception du logiciel chez le client final
- Tests intégration système: test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme: test d'acception du logiciel avant livraison (nouvelle version par exemple)
- Tests Integration: test de l'intégration des différents composants (avec ou sans hardware)
- Tests Unitaires: tests élémentaires des composants logiciels (une fonction, un module, ...)
- Tests de non-régression

Les différents niveaux

- Tests de recette: test de réception du logiciel chez le client final
- Tests intégration système: test de l'intégration du logiciel avec d'autres logiciels
- Tests systeme: test d'acception du logiciel avant livraison (nouvelle version par exemple)
- Tests Integration: test de l'intégration des différents composants (avec ou sans hardware)
- Tests Unitaires: tests élémentaires des composants logiciels (une fonction, un module, ...)
- Tests de non-régression
- "Smoke-tests": jeu réduit de tests pour valider un *build* avant de passer à la validation

La planification

- Ces différents niveaux de tests doivent être planifiés (c'est l'objet du plan projet).
- Théoriquement, on prépare les tests en même temps que le développement correspondant au niveau.



- On peut aussi le faire incrémentalement, ou en *pipeline*. Demande une gestion de projet très fine.
- En Extreme-Programmming: paire de développeurs/paires de testeurs.
- De toute façon, il y a rebouclage permanent (bugs, changements de specs): il faut s'adapter !

Tests fonctionnels

- Objectif: Analyse du comportement
  - conformité, comportement nominal, aux limites, robustesse

Tests fonctionnels

- Objectif: Analyse du comportement
  - conformité, comportement nominal, aux limites, robustesse
- Basés sur les spécifications, qui donnent le comportement attendus.
  - Attention: il ne suffit pas de donner des tests pour chaque exigence, mais il faut aussi comprendre les interactions possibles.

Tests fonctionnels

- Objectif: Analyse du comportement
  - conformité, comportement nominal, aux limites, robustesse
- Basés sur les spécifications, qui donnent le comportement attendus.
  - Attention: il ne suffit pas de donner des tests pour chaque exigence, mais il faut aussi comprendre les interactions possibles.
- Il faut mettre en place des stratégies.
  - De même qu'une exigence doit pouvoir être codée, elle doit pouvoir être testée $\Rightarrow$ il faut une relecture de la spécification par les testeurs

Tests fonctionnels

- Objectif: Analyse du comportement
  - conformité, comportement nominal, aux limites, robustesse
- Basés sur les spécifications, qui donnent le comportement attendus.
  - Attention: il ne suffit pas de donner des tests pour chaque exigence, mais il faut aussi comprendre les interactions possibles.
- Il faut mettre en place des stratégies.
  - De même qu'une exigence doit pouvoir être codée, elle doit pouvoir être testée $\Rightarrow$ il faut une relecture de la spécification par les testeurs
- Différentes méthodes seront abordées dans la suite de ce cours

Tests structurels

- Objectif: Détecter les fautes d'implémentation

Tests structurels

- Objectif: Détecter les fautes d'implémentation
- Détecter si:
  - détection de cas de "plantage"
  - le logiciel n'en fait pas trop

Tests structurels

- Objectif: Détecter les fautes d'implémentation
- Détecter si:
  - détection de cas de "plantage"
  - le logiciel n'en fait pas trop
- Niveau architectural & code:
  - Analyse des flots de contrôle, de données, condition logiques et des itérations.
  - Dépend de l'implémentation. Selon les cas, le testeur a accès ou non au code.

Tests structurels

- Objectif: Détecter les fautes d'implémentation
- Détecter si:
  - détection de cas de "plantage"
  - le logiciel n'en fait pas trop
- Niveau architectural & code:
  - Analyse des flots de contrôle, de données, condition logiques et des itérations.
  - Dépend de l'implémentation. Selon les cas, le testeur a accès ou non au code.
- De même qu'une exigence fonctionnelle doit pouvoir être codée, une exigence de codage doit pouvoir être testée

Tests structurels

- Objectif: Détecter les fautes d'implémentation
- Détecter si:
  - détection de cas de "plantage"
  - le logiciel n'en fait pas trop
- Niveau architectural & code:
  - Analyse des flots de contrôle, de données, condition logiques et des itérations.
  - Dépend de l'implémentation. Selon les cas, le testeur a accès ou non au code.
- De même qu'une exigence fonctionnelle doit pouvoir être codée, une exigence de codage doit pouvoir être testée
- Différentes méthodes seront abordées dans la suite de ce cours

Documents

Le test s'inscrit dans le cycle de vie du projet. Les documents suivant font partie des éléments nécessaires pour le test:

- Plan Projet: c'est le plan global du projet, servant de référence pour l'ensemble
- Spécification / Conception: les documents permettant l'élaboration du code et des tests associés
- Plan de tests: ce document doit décrire:
  - L'organisation du tests: équipes, environnement
  - Les stratégies: en fonction des éléments d'objectifs du projet, de spécifications, d'architectures, différentes stratégies de tests sont à envisager.
  - Ex WEB: test de l'interface, test de la logique "business" (authentification, transaction,...), test de la base, test de performances, de stress, ...
  - Les critères d'arrêt des tests: 100% de couverture de code n'est pas 100% des possibilités.

- Un standard de test. Assure une homogénéïté d'écriture et de validation.
- Les rapports de tests. Il faut être en mesure de donner un état précis de ce qui a été testé, et des résultats.
- En particulier, il faut pouvoir indiquer:
  - quelles sont les exigences couvertes ?
  - et comment les a-t-on couvertes (combien de tests/exigences, tests normaux, aux limites, de robustesse)
- La traçabilité entre les exigences et les tests doit être aussi assurée. En particulier, cela permet d'avoir une connaissance de l'impact d'une modification de spécification.

Organisation du test

L'organisation du test recouvre différents aspects:

- Humain: il faut définir qui fait quoi (Plan de test)
  - Comment est constituée une équipe de validation
  - Comment elle interagit avec le développement
  - Comment le département Qualité (s'il existe) interagit
  - Quelle est l'interaction avec l'utilisateur final
- Technique: le comment
  - Comment met-on en place la base (ou les bases) de tests
  - L'environnement matériel éventuel (hôtes, mais cartes embarquées, autre matériel, réseau, ...). Il faut être au plus près de la mise en œuvre finale.
  - Quel outillage pour les tests automatiques ? Pour les mesures de tests ?
  - Utilisation d'outils maison ou COTS: s'est-on assuré de leur validité ?

Conclusion

- Le test est une activité importante

Conclusion

- Le test est une activité importante
- Demande compétence, rigueur, créativité

Conclusion

- Le test est une activité importante
- Demande compétence, rigueur, créativité
- Le test ne s'improvise pas: il faut le penser dès le début

Conclusion

- Le test est une activité importante
- Demande compétence, rigueur, créativité
- Le test ne s'improvise pas: il faut le penser dès le début
- C'est une activité structurée, dans le cadre du projet
  - C'est un travail collectif entre le développent et la validation
  - Une bonne base de tests, avec une regression simple à mettre en œuvre est ausi très appréciée du dev !

Conclusion

- Le test est une activité importante
- Demande compétence, rigueur, créativité
- Le test ne s'improvise pas: il faut le penser dès le début
- C'est une activité structurée, dans le cadre du projet
  - C'est un travail collectif entre le développent et la validation
  - Une bonne base de tests, avec une regression simple à mettre en œuvre est ausi très appréciée du dev !
- Ce n'est pas l'apannage de l'*industrie*, mais aussi des projets OpenSource (ex: Perl, Linux, ...) ... et aussi de vos développements !

Références


John Watkins.
Test Logiciel en pratique
Vuibert, 2002.


Glenford J. Myers.
The Art of Software Testing 2^{nd} Ed.
John Wiley & Sons, Inc. 2004

Et les livres traitant de “Software Testing” sur Amazon par ex.