Tests unitaires

Principes

Le concept de tests n'est pas une nouveauté. Depuis toujours, tester le code fait partie intégrante de l'activité quotidienne d'un développeur. **Attention** ! On parle bien ici de tests (au sens d'une démarche dédiée à amener un code vers un niveau de qualité optimal) par opposition à de la mise au point (au sens d'une pratique destinée à rendre un code simplement opérant).

Dans un certain nombre de contextes de développement aujourd'hui, cette activité se place au cœur du processus de conception (**Méthodes Agiles** ou encore **TDD**). Il y a plusieurs raisons à cela :

- Concevoir le test d'un service, avant même d'avoir codé le service à tester : favorise la modularité (petites unités à tester) et la concision (le développeur n'implémente que l'essentiel) ;
- Plus un bogue est détecté tôt, plus facile sera sa correction et moins il coûtera ;
- Disponibilité d'outils de tests en open source ;
- Bonne intégration de ces outils dans les ateliers de génie logiciel (ex. Win'Design) ;
- Qualité générale du code développé. Facile à maintenir et à tester ;
- Possibilité de concevoir des batteries de tests que l'on peut rejouer à la demande ou automatiquement (intégration continue), ce qui participe d'une démarche « industrielle ».

Typologie des tests

Tests unitaires : destinés à vérifier la conformité des unités élémentaires de programme (procédures, fonctions, méthodes).

Tests de non-régression : lorsqu'on intervient sur un code qui passait positivement une batterie de tests auparavant, il faut nécessairement s'assurer que cette même batterie de tests continue de produire un résultat positif après intervention. C'est ce qu'on appelle des tests de non-régression car, en effet, parfois on pourrait corriger une anomalie ou améliorer une fonctionnalité avec succès mais au prix d'un changement notable dans le comportement de détail. Ce qui ne serait pas tolérable.

Tests d'intégration : destinés à vérifier la conformité de composants distincts que l'on vient d'assembler entre eux. Au préalable, ces composants auront été testés séparément via des tests unitaires. Ici, c'est la bonne communication ou coopération entre ces composants qui est visée, essentiellement.

Tests fonctionnels : destinés à vérifier la conformité d'un ensemble de composants en référence à un ou plusieurs cas d'utilisation. Ici, on ne soucie pas de l'implémentation technique sous-jacente, on déroule des scenarii « métier » à partir des interface utilisateur. Ces tests valident le bon comportement du logiciel en matière de service rendu.

Tests de charge : destinés à vérifier la capacité d'une application à supporter une charge supplémentaire. Il pourra s'agir d'une charge d'utilisateurs ou de données qui auront un impact sur les performances et la volumétrie du stockage.

Tests de vulnérabilité : tests destinés à vérifier le niveau de sécurité d'un composant ou d'une application.

Tests unitaires

Un test unitaire est un programme qui vérifie le bon fonctionnement d'une unité fonctionnelle élémentaire (une procédure, une fonction ou une méthode) au travers de situations déduites des spécifications de l'unité testée : à partir de données particulières en entrée, le test sollicite l'unité et confronte les données réellement obtenues avec celles théoriquement attendues. Il en déduit alors un état de succès ou d'échec.

La **couverture des tests** désigne le rapport entre le nombre de situations testées et le nombre de situations possibles. Par principe, la couverture des tests n'est jamais complète car il n'est pas raisonnable, et souvent pas envisageable, d'imaginer tester toutes les situations possibles. On se « contentera » donc de recenser les cas de tests les plus remarquables.

Prudence

Attention ! La réussite des tests ne permet évidemment pas de conclure au bon fonctionnement du logiciel (Test informatique).

On essaye cependant, avec rigueur mais assez empiriquement, de faire en sorte que si un bogue est présent, le test le mette en évidence, notamment en exigeant une bonne couverture des tests :

- Couverture en points de programme : chaque point de programme doit avoir été testé au moins une fois ;
- Couverture en chemins de programme : chaque séquence de points de programme possible dans une exécution doit avoir été testée au moins une fois (impossible en général) ;
- Couverture fonctionnelle : chaque fonctionnalité doit être vérifiée par au moins un cas de test.

Selon la complexité du logiciel, des séquences de vérification globale peuvent s'avérer nécessaires. Celles-ci mettent en jeu la maîtrise d'ouvrage et toutes les composantes du projet, au-delà du logiciel lui-même (processus, organisation, formation, accompagnement du changement) : réception, qualification, certification, homologation, simulation, VABF (vérification d'aptitude au bon fonctionnement) … les termes varient selon les contextes.

Exemples d'environnements de tests unitaires

PHP : PHPUnit, Atoum, SimpleTest JAVA : JUnit .NET : NUnit