Outils de test

MOCK (programmation orientée objet)

En programmation orientée objet, les **mocks** (ou *Mock object*) sont des objets simulés qui reproduisent le comportement d'objets réels de manière contrôlée. Un programmeur crée un **mock** dans le but de tester le comportement d'autres objets, réels, mais liés à un objet inaccessible ou non implémenté. Ce dernier est alors remplacé par un mock.

Le concept a été utilisé pour la première fois lors de la conférence XP 2000, dans un article de Tim Mackinnon, Steve Freeman et Philip Craig intitulé *Endo-Testing: Unit Testing with Mock Objects.* Une définition de l'adjectif *mock* étant *not real but appearing or pretending to be exactly like something*, on pourra lui préférer celle d'**objet factice**.

Dans un test unitaire, les mocks peuvent simuler le comportement d'objets réels et complexes et sont utiles à ce titre quand ces objets réels sont impossibles à utiliser. On peut citer les cas suivants :

- Remplacer un comportement non déterministe (l'heure ou la température ambiante).
- Si l'objet a des états difficiles à reproduire (une erreur de réseau par exemple).
- Si l'initialisation de l'objet est longue (ex : création d'une base de données).
- Si l'objet n'existe pas ou si son comportement peut encore changer.
- S'il est nécessaire d'inclure des attributs et des méthodes uniquement à des fins de test.

Par exemple, un programme d'alarme qui produira une sonnerie à une heure donnée. Pour le tester, le programme devra attendre l'heure prévue afin de vérifier que la sonnerie se produit. Si un mock est utilisé, celui-ci pourra être déréglé en vue de simuler l'heure de déclenchement de la sonnerie.

Behavior Driven Development

Behavior Driven Development (ou **BDD**) est une méthode Agile qui encourage la collaboration entre les développeurs, les responsables qualités, les intervenants non-techniques et les entreprises participant à un projet de logiciel. Il a été conçu en 2003 par Dan North comme une réponse au Test Driven Development.

Le processus BDD met en avant le langage naturel et les interactions dans le processus de développement logiciel. Les développeurs utilisant le BDD utilisent leur langue maternelle en combinaison avec le langage du domaine Domaine Driven Design pour décrire l'objectif et le bénéfice de leur code. Cela permet aux développeurs de se concentrer sur les raisons pour lesquelles le code doit être créé, plutôt que les détails techniques, et minimise la traduction entre le langage technique dans lequel le code est écrit et le domaine de la langue parlée par les entreprises, les utilisateurs, les intervenants, la gestion de projet…

Pratiques BDD

Les pratiques de BDD comprennent :

- La participation des parties prenantes dans le processus par le biais de l'extérieur dans le développement de logiciels outside-in software development
- L'utilisation d'exemples pour décrire le comportement de la demande, ou d'unités de code
- Automatisation de ces exemples pour fournir rapidement des commentaires et des tests de non-régression
- Dans le test logiciel, l'utilisation du mot «devrait» contribue à clarifier la responsabilité et permet la remise en cause de la fonctionnalité du logiciel
- L'utilisation de «vérifier que» permet de différencier les résultats obtenus dans le champ d'application du code en question en provenance des effets secondaires d'autres éléments du code.
- Enfin, l'utilisation de «Mocks» en remplacement des modules de code qui n'ont pas encore été écrits

Model-based testing

Le model-based testing (MBT) est une activité qui dérive model-based design, qui permet à partir de la modélisation du système sous test (SUT) de générer des tests. Le modèle est une représentation abstraite et partielle des comportements attendus d'un logiciel ou d'un système.

Sur la base de ce modèle, des cas de test peuvent être dérivées, sous la forme de "suite de tests". Ces suites de tests ne sont pas directement exécutables, car elles n'ont pas le même niveau d'abstraction que le code exécutable. Cela demande souvent une intervention manuelle.

Une fois les cas de tests exécutés, une comparaison est possible entre le comportement réel du logiciel (le logiciel développé) et le comportement attendu (décrit dans le modèle).

Les outils MBT sont des outils qui peuvent automatiser le design des tests fonctionnels (tests boîte noire).

Avantages et inconvénients du Model Based Testing

L'intégration d'un processus de validation MBT prend du temps et demande des formations pour les ingénieurs afin d'acquérir de nouvelles compétences, surtout lorsqu'ils développent et testent encore « manuellement ». La mise en place d'un outil MBT demande un certain investissement de la part des entreprises, notamment pour celles qui n'utilisent pas encore la modélisation graphique (UML) pour leur développement (MBD) et/ou test (MBT) de logiciels. L'utilisation d'outils MBT entraîne une modification des pratiques déjà mises en place au sein de l'entreprise.

Par ailleurs, la modélisation graphique atteint rapidement ses limites en terme d'expressivité, et il n'est pas rare de devoir rajouter manuellement des annotations et/ou de

s'aider d'une notation ou d'un langage (comme OCL, par exemple) pour rajouter de la sémantique au modèle.

Cependant, certains utilisateurs affirment que l'utilisation des MBT peut être un réel retour sur investissement avec un gain de productivité et une qualité augmentée.

En effet, l'automatisation des tests a des avantages directs pour les équipes responsables des tests :

- Si le modèle est bien fait, évite des cas de test mal conçus, défectueux ou manquants, du même coup, accroît la couverture de test
- Réduit les coûts pour les tests (tests de non régression)
- Améliore la qualité du processus de test
- Réduction des délais d'exécution des tests

et également des avantages indirects pour les utilisateurs du système d'information :

- Diminue les efforts de maintenance des jeux de tests
- Renforce la qualité de la documentation des exigences
- Crée une plateforme commune pour les designers et les testeurs

L'automatisation permet d'exécuter des tests à un coût marginal très faible, après un investissement initial significatif et en conception ou maintenance

JUnit

JUnit est un framework de test unitaire pour le langage de programmation Java. Créé par Kent Beck et Erich Gamma, JUnit est certainement le projet de la série des xUnit connaissant le plus de succès.

JUnit définit deux types de fichiers de tests. Les **TestCase** sont des classes contenant un certain nombre de méthodes de tests. Un TestCase sert généralement à tester le bon fonctionnement d'une classe. Une **TestSuite** permet d'exécuter un certain nombre de TestCase déjà définis. JUnit est intégré par défaut dans les environnements de développement intégré Java tels que BlueJ, Eclipse et NetBeans.

XUnit

Le terme générique **xUnit** désigne un outil permettant de réaliser des tests unitaires dans un langage donné (dont l'initiale remplace « x » le plus souvent). L'exemple le plus connu est JUnit pour Java.