Eléments pratiques de test des Hiérarchies et Frameworks

Notes de cours
Christophe Dony
Master Info Pro - Université Montpellier-II

1 Introduction

1.1 Définitions

Génie Logiciel No 18, Mars 1990. EC2.

IEEE Software , 6(3), May 1989. Numéro spécial.

- erreur : commise par le développeur, conduit à un défaut.
- défaut : imperfection dans le logiciel conduit à une panne ;
- panne : comportement anormal (résultat incorrect ou programme non fiable) d'un programme.
- Un programme est dit correct lorsqu'il correspond à ses spécifications.
- Un programme est fiable s'il s'exécute sans pannes, ceci ne signifie pas qu'il est correct (instruction non exécutée ou résultat non conformes aux spécifications).
- Tests : essai du logiciel visant à tester s'il est fiable et correct
- Test de défauts : découverte des défauts.
- mise au point : phase de correction des défauts constatés lors des tests

Phases de test (extraits) :

- **test unitaire** : test d'un composant
- test des modules ou test d'intégration : test d'un ensemble de composants
- test d'acceptation (ou test alpha) : test du système dans les conditions définies par l'acheteur.
- test beta : test en situation d'installation.

1.2 Problèmes

1. Exhaustivité : Impossibilité de tester toutes les données d'entrée possibles.
2. Automatisation : re-tester simplement après chaque modification.
3. Découpage : modulatité
4. Couverture : tester le maximum de cas, na pas tester n fois la même chose.

2 Pratique du test de défauts

cf. (Howden 87 Functional Program Testing and Analysis, Mc-Graw-Hill)

2.1 test fonctionnels (ou test de boite noire)

- vérification de la conformité aux spécification. Le logiciel ou le composant testé est considéré comme une boite noire.

- tests réalisés par des équipes autres que celles ayant réalisé le logiciel.

2.2 Fabrication de jeux de test

Exemple : une procédure de recherche dans un tableau ordonné.

- Entrées : Tableau ordonné T, élément recherché L.
- préconditions : tableau ordonné, élément appartient au type des éléments du tableau.
- Sortie : rend l'index auquel L est rangé dans le tableau, nil sinon.

Principe de **couverture** : séparer les jeux de données en classes d'équivalence en se basant sur les spécifications et sur l'expérience.

Les tests appartenant à la même classe d'équivalence testents la même chose.

2.3 Définition de critères

Les classes d'équivalence sont construites à partir de critères,

Origine	**Critère**	**nom**
Préconditions :		
	Tableau ordonné	TO
	Tableau non ordonné	TNO
	Elément du bon type	ETY
	Élément d'un mauvais type	EMTY
Postconditions		
	Element recherché dans le tableau	ET
	Element recherché pas dans le tableau	ENT
Expérience		
	tableau de taille paire	TP
	tableau de taille impaire	TI
Tests aux limites :		
	Tableau de 0 élément	T0
	Tableau de 1 élément	T1
	Tableau de taille quelconque	TQ
	Elément en première position	E1
	Elément en dernière position	En.
	Element en position quelconque	EQ

2.3.1 Classes d'équivalence

Une classe d'équivalence est un sous-ensemble de l'ensemble de tous les jeux de test possibles qui contient tous les test testant la même chose.

1. Eliminer les critères déjà testés par le compilateur, par exemple TO, TNO, ETY ou EMTY.
2. supprimer les cas incompatibles comme : TPxT1.
3. Croiser les critères. Dans l'exemple, il y a 36 cas potentiellement différents : "(TP TI) x (TQ TO T1) x (ET ENT) x (EQ E1 En)". En enlevant les incompatibilités, il reste 11 cas effectifs (11 classes d'équivalence) à tester pour les tests de boite noire.
   C1 : TPxTQxETxEQ
   C2 : TPxTQxETxE1
   C3 : TPxTQxETxEn
   C4 : TPxTQxENT
   C5 : TPxT0xENT
   C6 : TIxTQxETxEQ
   C7 : TIxTQxETxE1
   C8 : TIxTQxETxEn
   C9 : TIxTQxENT
   C10 : TIxT1xETxE1
   C11 : TIxT1xENT

2.4 test structurels (ou tests de boite blanche)

Ajout de nouvelles classes d'équivalence par analyse du code et prise en compte de la structure du programme.

- Prise en compte des algorithmes utilisés. Exemple, une recherche dichotomique induit de nouveaux critères : clé recherchée est au milieu du tableau.
- Test des chemins : Enumération des chemins possibles d'un programme (graphe de flux)

2.5 Automatisation des tests

- Voir JUnit

- Automatiser la comparaison entre résultat attendu et résultat obtenu

- N méthodes par classes testant chacune les différentes fonctionnalités dont une méthode de classe aggrégeant l'utilisation des autres.

- Une méthode par Application.

- Repasser tous les tests après une modification (test de régression).

2.6 Généralisation au test d'une classe

Les critères sont définis par toute combinaison de valeur des attributs d'un objet qui lui confère un état particulier.

A mettre en relation avec les diagrammes d'état UML.

Il doit y avoir pour une méthode, un test pour chacun des états possible de l'objet.

2.7 Résumé

Combiner les tests fonctionnels (boite noire) et les tests structurels (boire blanche).

Les tests fonctionnels ne permettent pas de mettre en évidence les défauts dus à une mauvaise structure du programme ou à la particularité de certains algorithmes.

Les tests structurels ne permettent pas de détecter les divergences par rapport aux spécifications