UPMC – Master Informatique – STL – MI047

Cours Composant
5. Test basé sur le Modèles (MBT)

©Frédéric Peschanski

UPMC - LIP6 - RSR - APR

19 février 2014

Plan du cours

1. Introduction au test logiciel
2. Catégorisation des tests
3. Processus de test MBT
4. Anatomie d'un cas de test
5. Critères de couverture

Cours basé sur :

- *Practical Model-based Testing*. Hutting et Legeard. Morgan kaufman (2004)
- *Test MBT automatique basé sur CSP*. Thèse de Hakim Belhaouari. LIP6 (2010)

Défaillances logicielles

Défaillances logicielles : exemples

- Bug du réseau BouyguesTel le 17 novembre 2005
  - panne pendant env. 24H
  - coût estimé à plus de 8 millions d'euros
- Bug d'Ariane 5 en 1996
  - dépassement de capacité suite au portage du code d'Ariane 4
  - coût estimé à plus de 450 millions d'euros

$\Rightarrow$ coût exorbitant de certaines défaillances logicielles

Le test logiciel

Le test est un outil incontournable pour augmenter la **qualité du logiciel**, et d'ainsi diminuer les probabilités de défaillance.
$\Rightarrow$ entre 40% et plus de 60% du coût de développement d'un logiciel.

Définition du SWEBOK1 IEEE (2004)

Le test logiciel consiste en la vérification dynamique du comportement d'un programme sur un nombre fini de cas de tests, sélectionnés convenablement à partir du domaine d'exécution (généralement infini), vis-à-vis du comportement attendu.

- les tests portent sur les implémentations
- les tests ne sont pas exhaustifs : ils ne permettent de s'assurer de l'absence de bug
- l'objectif est de déterminer les «meilleurs» tests : ceux le plus susceptibles de générer des défaillances
- on doit connaître à l'avance les résultats attendus des tests : notion d'oracle

Quelques définitions

SUT (*system under test*) : système à tester

Défaillance (*failure*) : comportement non-désiré ou inattendu du SUT

Erreur ou faute *(fault)* : cause d'une défaillance

Tester : tenter de produire, en amont du développement, des défaillances sur un logiciel pour en détecter des fautes.

Debugger : trouver puis corriger des fautes à partir de rapports de défaillances

Catégorisation des tests

- Granularité des tests
  - composant = test unitaire, assemblage = test d'intégration, système = test de système
- Objectif des tests
  - Fonctionnalités = test fonctionnel ou test de conformité
  - Test extra-fonctionnel : robustesse, performances, disponibilité, responsiveness, non-régression, etc.
- Source des tests
  - Spécifications / modèles = test boîte-noire
  - Code source = test boîte-blanche ou test structurel
  - (également test boîte-grise)
- Modèle d'exécution des tests
  - Test manuel, Test assisté ou Test automatique
  - Test hors-ligne (classique) ou Test en-ligne (contrats).
- etc.

Le test basé sur les modèles

Le test basé sur les modèles (MBT)

- test fonctionnel
- basé sur les spécifications / modèles
  ⇒ test boîte-noire
- objectif d'automatisation = test assisté et/ou automatique
- approche classique = test hors-ligne

Méthodologie du Test MBT

Méthodologie du Test MBT

1. Objectifs de test : **critères de couverture**
   ⇒ plan de test : ensemble des objectifs assurant une certaine couverture
2. Pour chaque objectif de test :
   - détermination d'au moins un cas de test
     ⇒ objectif atteint
   - objectif non-atteignable : impossibilité de cas de test
   - objectif non-atteint : pas de cas de test trouvé (mais pas de preuve d'impossibilité)
3. rédaction des scripts de test : versions exécutables des cas de test
4. exécution des script sur le SUT dans son environnement de test
   ⇒ harnais (rapport) de test (*test harness*)

Exemple : feu de signalisation

service : TrafficLight
types : boolean, enum Color RED, ORANGE, GREEN
observators :
    color : [TrafficLight] → Color
    blinking : [TrafficLight] → boolean
    failed : [TrafficLight] → boolean
Constructors :
    init : → [TrafficLight]
Operators :
    change : [TrafficLight] → [TrafficLight]
        pre change(L) **require** ¬failed(L)
Observations :
[invariants]
    if failed(L) **then** blinking(L)
    if blinking(L) **then** color(L) = ORANGE
[init]
    color(init()) = ORANGE
    blinking(init())
[change]
    if color(L) = RED **then** color(change(L)) = GREEN
    else if color(L) = ORANGE **then** color(change(L)) = RED
    else color(L) = GREEN ∧ color(change(L)) = ORANGE
    blinking(change(L)) = false

Anatomie d'un cas de test

Anatomie d'un cas de test :

- **Description** : objectif de test en langage informel et identification précise du cas de test.
- **Préambule** : conditions initiales du test sous forme d'un état accessible
- **Contenu** : opérations à réaliser pour le test sous forme d'opérations et d'observation sur l'état initial
- **Oracle** : comportement attendu du logiciel pour ce test sous forme d'une observation d'état
- **Postambule** : rapport du test pour le harnais de test

Exemple : objectif et cas de test

Objectif de test : le feu est orange clignotant à l'allumage

$\Rightarrow$ objectif atteignable

Cas de test : `TrafficLight::testInit`

- **Conditions initiales** : vide
- **Opérations** : $L_0 \stackrel{\text{def}}{=} init()$
- **Oracle** :

  color(L_0) = ORANGE
  blinking(L_0) = true
- **Rapport** :
  - color(L_0) $\neq$ ORANGE
    blinking(L_0) = true
    $\Rightarrow$ le feu n'est pas orange lors de l'allumage
  - color(L_0) = ORANGE
    blinking(L_0) $\neq$ true
    $\Rightarrow$ le feu ne clignote pas lors de l'allumage
  - color(L_0) $\neq$ ORANGE
    blinking(L_0) $\neq$ true
    $\Rightarrow$ le feu ne clignote pas et n'est pas à l'orange lors de l'allumage

Exemple : script de test

```
public class TrafficLightTest {
    ...
    // Cas de test : TrafficLight::testInit
    @Test
    public void testInit() {
        // conditions initiales
        TrafficLightService L0 = new TrafficLight();
        // Operations
        L0.init();
        // Oracle
        assertTrue("Le feu n'est pas orange lors de l'allumage",
                   L0.color() !=TrafficLight.ORANGE
                   && L0.blinking()==true);
        assertTrue("Le feu ne clignote pas lors de l'allumage",
                   L0.color()==TrafficLight.ORANGE
                   && L0.blinking()==false);
        assertTrue("Le feu ne clignote pas et n'est pas à l'orange lors de l'allumage",
                   L0.color() !=TrafficLight.ORANGE
                   && L0.blinking()==false);
```

Couverture des tests

Important : les tests ne permettent de statuer sur l'absence de faute dans le logiciel.
⇒ la notion de couverture permet de fournir des garanties sur les tests.
⇒ un jeu de test sans information de couverture ne sert à rien.

Critères de couverture basés sur le code source (test boîte blanche) :

- conditions d'alternatives (if, switch) ou de boucle
- branches d'alternatives
- localisations (morceau de code)
- valeurs d'un paramètre
- combinaisons de valeurs des paramètres d'une fonction/méthode
- etc.

⇒ la couverture porte sur les Objectifs de test
⇒ plan de test

Couverture des tests MB

Dans le test MBT, les critères de couverture sont basés sur :
les modèles et/ou les spécifications (test boîte noire).
⇒ les critères dépendent du langage de modélisation/spécification.
⇒ dans ce cours : l'unité fonctionnelle est le service.
⇒ premier critère : nombre de services couverts

Critères de couverture pour un service donné (liste non-exhaustive) :

- couverture des préconditions
- couverture des postconditions
- couverture des transitions
- couvertyre des scénarios (*use cases*)
- couverture des états
- couverture des données
- etc.

Critères de couverture : préconditions et postconditions

Critère : Couverture des préconditions

Pour chaque précondition :

- au moins un test positif : la précondition passe
  Oracle : pas d'exception levée et la condition est vérifiée.
- au moins un test négatif : la précondition est invalidée
  Oracle : une exception est levée.

Critère : Couverture des postconditions

Pour chaque postcondition (observation ou invariant) :

- au moins un test positif : la postcondition est observable
  Oracle : la postcondition est vraie.

Exemple : couverture des préconditions

Préconditions de TrafficLight :

- **Objectif 1** : précondition de change : ¬failed(L)
  - ⇒ objectif non atteignable

Invariants de TrafficLight :

- **Objectif 2** : invariant **if** failed(L) **then** blinking(L)
  ⇒ objectif non atteignable
- **Objectif 3** : invariant **if** blinking(L) **then** color(L)==ORANGE
  ⇒ objectif atteignable
  - **Initial** vide
  - **Cas de test** : $L_0 \stackrel{\text{def}}{=}$ init()
  - **Oracle** : blinking(L)==true et color(L)==ORANGE
- etc.

Critères de couverture : conditions logiques 1/2

Condition $P \vee Q$

- Cas positifs ($P \vee Q = \mathit{true}$)
  - au moins un cas de test pour pouvoir observer $P = \mathit{true}$ et $Q = \mathit{false}$
  - au moins un cas de test pour pouvoir observer $P = \mathit{false}$ et $Q = \mathit{true}$
  - au moins un cas de test pour pouvoir observer $P = \mathit{true}$ et $Q = \mathit{true}$
- Cas négatifs ($P \vee Q = \mathit{false}$)
  - au moins un cas de test pour pouvoir observer $P = \mathit{false}$ et $Q = \mathit{false}$

Condition $P \wedge Q$

- Cas positifs ($P \wedge Q = \mathit{true}$)
  - au moins un cas de test pour pouvoir observer $P = \mathit{true}$ et $Q = \mathit{true}$
- Cas négatifs ($P \wedge Q = \mathit{false}$)
  - au moins un cas de test pour pouvoir observer $P = \mathit{false}$ et $Q = \mathit{false}$
  - au moins un cas de test pour pouvoir observer $P = \mathit{false}$ et $Q = \mathit{true}$
  - au moins un cas de test pour pouvoir observer $P = \mathit{true}$ et $Q = \mathit{false}$

Critères de couverture : conditions logiques 2/2

Condition $P \implies Q$ (ou **if** P **then** Q)

- Cas positifs ($P \implies Q = \mathit{true}$)
  - au moins un cas de test pour pouvoir observer $P = \mathit{true}$ et $Q = \mathit{true}$
  - au moins un cas de test pour pouvoir observer $P = \mathit{false}$ et $Q = \mathit{true}$
  - au moins un cas de test pour pouvoir observer $P = \mathit{false}$ et $Q = \mathit{false}$
- Cas négatifs ($P \implies Q = \mathit{false}$)
  - au moins un cas de test pour pouvoir observer $P = \mathit{true}$ et $Q = \mathit{false}$

Condition **if** C **then** P **else** Q

- Cas positifs
  - au moins un cas de test pour $C = \mathit{true}$ et $P = \mathit{true}$
  - au moins un cas de test pour $C = \mathit{false}$ et $Q = \mathit{true}$
- Cas négatifs
- au moins un cas de test $C = \mathit{true}$ et $P = \mathit{false}$
- au moins un cas de test pour $C = \mathit{false}$ et $Q = \mathit{false}$

Alternatives et tests

Remarque

Les conditions alternatives
if C **then** P **else** Q
permettent de structurer les spécifications et de faciliter l'élaboration des cas de tests.

On préférera donc écrire :
if C **then** P **else** Q
plutôt que son équivalent logique :
$(C \implies P) \wedge (\neg C \implies Q)$

Exemple : couverture de postcondition

Objectifs de test : postcondition de change

if color(L) = RED **then** color(change(L)) = GREEN
else if color(L) = ORANGE **then** color(change(L)) = RED
else color(L) = GREEN $\wedge$ color(change(L)) = ORANGE

- **Objectif 1** : Le feu passe du rouge au vert en fonctionnement normal.
  - **Initial** : $L_0 \stackrel{\text{def}}{=}$ change(init())
  - **Test** : $L_1 \stackrel{\text{def}}{=}$change($L_0$)
  - **Oracle** : color(L_1) == GREEN est vrai
- **Objectif 2** : Le feu passe du orange au rouge en fonctionnement normal.
  - **Initial** : $L_0 \stackrel{\text{def}}{=}$ change(change(init()))
  - **Test** : $L_1 \stackrel{\text{def}}{=}$change($L_0$)
  - **Oracle** : color(L_1) == RED
- etc.

Critères de couverture : transitions et états

Critère : Couverture des transitions

Pour chaque transition (constructeur ou opérateur) :

- au moins un test positif : la transition peut-être franchie
  Oracle : postconditions et invariants.
- au moins un test négatif si la transition est gardée (précondition)
  Oracle : une exception est levée.

⇒ variantes : chemins = paires de transitions, triplets, scénarios utilisateur, tous les chemins (vérification exhaustive), etc.

Critère : Couverture des états

Pour chaque état remarquable (i.e. atteignable et intéressant)

- au moins un test positif :l'état est atteignable
  Oracle : postconditions et invariants.

⇒ variante : tous les états atteignables (vérification exhaustive).

Critères de couverture : données

Contexte : un paramètre nécessite d'être valué (ex. opération, invariant paramétré, etc.)

Critère : Tests limites

- au moins un test positif dans les limites
- au moins un test positif aux limites
- au moins un test négatif hors limite

Critère : Test aléatoire

- Test de robustesse : génération de valeurs «purement» aléatoires
- Test uniforme : génération de valeurs aléatoires uniformes sur une distribution probabilité connue
- Test statistique : génération de valeurs aléatoires selon une distribution statistique connue
- Test symbolique : les valeurs respectent des contraintes symboliques

Exemple : Tests limites

structureRappel invariant du compte-bancaire :

peutPrelever(C,s) $\stackrel{\text{min}}{=}$ solde(C)-s $\geq$ limite(C)

$\Rightarrow$ le paramètre s doit être valué

- **Objectif 1** : dans les limites
  - **Cas de test** : $C_1 \stackrel{\text{def}}{=}$depot(init("C1",1,1000),500)
  - **Paramètre** : $s_1 \stackrel{\text{def}}{=}$solde($C_1$)-limite($C_1$) div 2 (division entière)
  - **Oracle** : peutPrelever(C_1,s_1) = solde(C_1)-s_1 $\geq$ limite(C_1)
- **Objectif 2** : aux limites
  - $s_2 \stackrel{\text{def}}{=}$solde($C_1$)-limite($C_1$) (même oracle)
- **Objectif 3** : hors limites
  - $s_3 \stackrel{\text{def}}{=}$solde($C_1$)-limite($C_1$)+1 (même oracle)
- etc.

Métriques de couverture

Pour «mesurer» une couverture de tests selon un critère donné :

reached nombre d'objectifs atteints

unreachable nombre d'objectifs non-atteignables (+ justification)

unknown nombre d'objectifs non-atteints (mais on ne sait pas s'ils sont atteignables)

Définition : la couverture en pourcentage est :

$$coverage \stackrel{\text{def}}{=} \frac{reached}{reached + unreachable + unknown} \times 100$$

Exemple : 35 reached — 12 unreachable — 3 unknown

$$coverage \stackrel{\text{def}}{=} \frac{35}{35 + 12 + 3} \times 100 = \frac{35}{50} \times 100 = 70\%$$

Fin