Cours CPS
5. Test MBT

©Frédéric Peschanski

UPMC - LIP6 - RSR - APR

15 février 2017

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 ainsi diminuer les risques de défaillance.
$\Rightarrow$ part importante du coût de développement

Définition du SWEBOK3 IEEE (2004–2014)

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

- 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

Test MBT vs. contrats

Important : complémentarité des approches

- les contrats sont un support pour la phase de conception
  - le MBT accompagne la campagne de test pour l'implémentation
- l'implémentation des contrats correspond à du test en-ligne
  - le test MBT (comme le reste du test) est hors-ligne
- la rupture d'un contrat peut entrainer la révision des spécifications.
  - les tests (MBT, unitaires, etc.) concernent uniquement l'implémentation.

$\Rightarrow$ En situation normale, les deux activités concernent des personnes différentes.

Un peu de terminologie

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
  - 50+ nuances de 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.

Couverture des tests

Les tests n'assurent pas la correction d'un logiciel mais en augment la qualité.
⇒ la couverture des tests est le critère principal de qualité (en développement classique).

Un jeu de test sans information de couverture ne sert à rien.

Exemple : Critères de couverture pour le test boîte blanche (couverture basée sur le code source) :

- passage par une instruction/expression donnée
- conditions d'alternatives (if, switch) ou de boucle
- etc.

⇒ plan de test = ensemble des objectifs de test nécessaires pour atteindre un certain ratio de couverture.

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

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

Le test basé sur les modèles (Model-Based Testing)

Le test basé sur les modèles (MBT)

- test fonctionnel
- basé sur les spécifications / modèles
  ⇒ test boîte-noire
  ⇒ Critères de couverture basés sur les spécifications
- objectif d'automatisation = test assisté et/ou automatique
  Rightarrow mais en CPS on restera en mode manuel
- test hors-ligne

Méthodologie du Test MBT

Méthodologie du Test MBT

1. Objectifs de test : **critères de couverture**
   $\Rightarrow$ plan de test : ensemble des objectifs assurant une certaine couverture
2. Pour chaque objectif de test :
   - détermination d'au moins un cas de test
     $\Rightarrow$ 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
   $\Rightarrow$ harnais (rapport) de test (*test harness*)

Exemple : feu de signalisation

type Color = **enum** { GREEN, ORANGE, RED }

service : TrafficLight
observators :
    color : [TrafficLight] $\rightarrow$ Color
    blinking : [TrafficLight] $\rightarrow$ boolean
    failed : [TrafficLight] $\rightarrow$ boolean
Constructors :
    init : $\rightarrow$ [TrafficLight]
Operators :
    change : [TrafficLight] $\rightarrow$ [TrafficLight]
        pre change(L) **require** $\neg$failed(L)
Observations :
[invariants]
    failed(L) $\Longrightarrow$ blinking(L)
    blinking(L) $\Longrightarrow$ color(L) = ORANGE
[init]
    color(init()) = ORANGE
    blinking(init())
[change]
    color(L) = RED $\Longrightarrow$ color(change(L)) = GREEN
    color(L) = ORANGE $\Longrightarrow$ color(change(L)) = RED
    color(L) = GREEN $\Longrightarrow$ 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

Remarque : la notion d'oracle est la caractéristique fondamentale du test. Un test = un comportement attendu.

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** :

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

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 :

- couverture des préconditions
- couverture des transitions
- couverture des états
- couvertyre des scénarios (*use cases*)
- couverture des données (en commun avec le test structurel)
- etc.

Critères de couverture : préconditions

Critère : Couverture des préconditions

Pour chaque précondition et pré-invariant[1] :

- au moins un test positif : la condition est validée
  Oracle : pas d'exception levée, et l'oracle "implique" la précondition.

Pour chaque précondition (sans objet pour les pré-invariants) :

- au moins un test négatif : la précondition est invalidée
  Oracle : une exception est levée.

1. précondition implicite dans tous les états autre qu'initiaux

Exemple : couverture des préconditions

Préconditions de TrafficLight :

- **Objectif 1** : précondition de change : $\neg$failed(L)
  - $\Rightarrow$ objectif non atteignable

Pré-invariants de TrafficLight :

- **Objectif 2** : invariant failed(L) $\implies$ blinking(L)
  $\Rightarrow$ objectif atteignable (état dans lequel blinking(L)=true).
- **Objectif 3** : invariant blinking(L) $\implies$ color(L)==ORANGE
  $\Rightarrow$ objectif atteignable
  - **Initial** vide
  - **Cas de test** : $L_0 \stackrel{\text{def}}{=}$ init()
  - **Oracle** : blinking(L)==true et color(L)==ORANGE (cela "implique" le pré-invariant)
- etc.

Critères de couverture : transitions

Critère : Couverture des transitions

Pour chaque transition (constructeur ou opérateur) :

- au moins un test : la transition peut-être franchie
  Oracle : postconditions et invariants.

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

$\rightarrow$ exemple(s) sur le feu rouge ?

Critères de couverture : états

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.

$\Rightarrow$ variante : tous les états atteignables (vérification exhaustive).

$\rightarrow$ exemple sur le feu rouge ?

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 «aléatoires»
- Test statistique : génération de valeurs aléatoires selon une distribution statistique connue
- Test uniforme : génération de valeurs aléatoires sur la distribution uniforme
- Test symbolique : les valeurs respectent des contraintes symboliques (ex. nombres premiers, satisfaction d'équations, etc.)

Exemple : Tests limites

Rappel invariant du compte-bancaire :

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

$\Rightarrow$ le paramètre s doit être valué

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

Fin