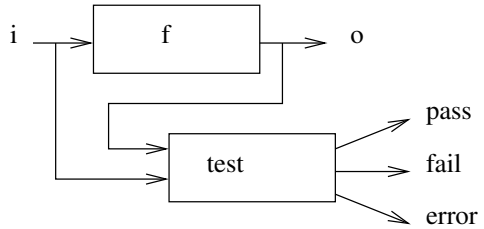
Quoi tester : rudiments d'approche fonctionnelle

Il est bien plus difficile trouver des cas de test pertinents que de maîtriser JUnit. On ne verra pas en SVL de techniques d'*analyse des risques*. On aborde ici une stratégie de test connue : l'*approche fonctionnelle* dans sa version la plus simple (de bon sens).



Modèle fonctionnel

- Méthode à tester = *fonction* f d'*entrées* i et de *sorties* o
- Test = *observateur* des entrées et sorties de f
- L'*oracle* vérifie que $f(x) = y$ pour une *donnée de test* (x, y)
- Il émet un *verdict* dans {pass, fail, error} :
  - pass : l'oracle répond vrai, test OK qui *passe* ;
  - fail : l'oracle répond faux, test KO qui *échoue* ;
  - error : comportement inattendu, *test inconclusif*.

Convient très bien pour les langages fonctionnels type Lisp.

Modèle fonctionnel objet

- Entrées au sens large : *paramètres* de la méthode et *état* de l'objet. Exemple pour `getLevel()` : paramètre `x` + état (lowLevel, midLevel, highLevel).
- Sorties au sens large : *résultat* de la méthode et *levées d'exceptions*

Choix des objectifs de test : approche systématique On identifie grâce à la spécification :

- le *domaine* de la méthode (= les valeurs possibles prises par ses entrées) ;
- les différents sous-comportements de la méthode (comment les sorties varient en fonction des entrées) ;
- un sous-domaine par comportement.

On a effectué une *partition du domaine* d'entrée par les comportements. Un sous-comportement = un *objectif de test*.

On choisit pour chaque couple (sous-domaine,sous-comportement) :

- une donnée de test *représentative* du sous-domaine ;
- des données de test « aux bornes ».

Stratégie de test implicite qui permet de réduire la combinatoire des tests :

- si la donnée représentative fait passer/échouer le test, les autres données du sous-domaine feront de même ;
- on suspecte qu'on a pu se tromper aux bornes (*modèle de fautes*).

Exemple : Pour l'objectif de test : "retourne `Level.Mid`"

$]x \in$ `getLowLevel()`, `getMidLevel()`$[$ avec lowLevel = 2, midLevel = 10.

Données de test pour l'oracle `lm.getLevel(x) == Level.Mid` :

- donnée représentative : x = 6, en plein milieu
- donnée borne inf : x = 3
- donnée borne sup : x = 20

Données pour l'oracle "négatif" `lm.getLevel(x) != Level.Mid` :

- dépassement de borne inf : x = 2
- dépassement de borne sup : x = 21