Concepts et applications de l'AOP

Deuxième partie : Impacts de l'AOP sur l'architecture et la qualité du logiciel

Félix-Antoine Bourbonnais

Université Laval, Québec

25 juin 2009

Plan

« Mocks »

- But : Dans un test unitaire, on souhaite tester le module en isolation
- Solution : Remplacer les modules utilisés par le module testé par des faux
- Un « Mock » est un faux module qui simule des réponses désirées pour le test

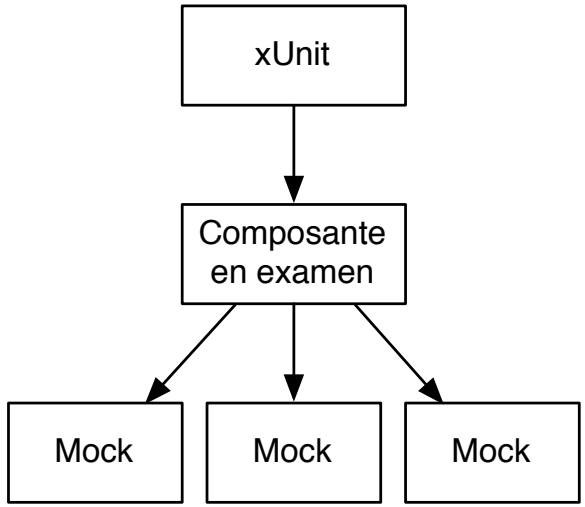

Fig. tirée : Contrôle de la qualité et métriques du logiciel, GLO-21525, F.-A. Bourbonnais et G.-E. Legendre, 2008.

Tester avec de l'AOP

Injection de « Mocks »

- Posons la classe `Afficheur` qui doit être testée unitairement :

```
1  class Afficheur {
2      private LED led;
3      private Display display;
4
5      public Afficher() {
6          led = new LED();
7          display = new Display()
8      }
9      public void afficher(String texte, boolean isError) {
10         display.print(texte)
11         led.off();
```

- `Afficheur` ne doit pas allumer réellement les lumières et afficher à l'écran pendant les tests.

Problème

Comment injecter le « Mock » uniquement pendant les tests ?

Tester avec de l'AOP (suite)

Injection de « Mocks »

- Solution traditionnelle (en Java) :

```
1  class Afficheur {
2      // [...]
3
4      public Afficher() {
5          led = new LED();
6          display = new Display()
7      }
8
9      Afficher(LED mockLED, Display mockDisplay) { /* Visibilite: Package */
10         led = mockLED;
11         display = mockDisplay;
12     }
13
14     // [...]
```

Tester avec de l'AOP (suite)

Injection de « Mocks »

- Solution AOP 1 :
  - Intercepter l'appel des constructeurs et retourner un Mock
  - Cette solution remplace LED et Display pour tous les tests unitaires.

```
1  public aspect MocksForPeripheriquesDansUTests {
2
3      protected pointcut newLEDouDisplayPC() :
4          (
5             call( LED.new(..) ) ||
6             call( Display.new(..) )
7          ) &&
8          cflow( within(*Test) );
9
10     Object around() : newLEDouDisplayPC() {
11         Class<?> type = thisJoinPoint.getSignature().getDeclaringType();
12         return mock(type);
13     }
14 }
```

Tester avec de l'AOP

Vérifier les appels

1 Vérifier si une méthode a bien été appelée exactement N fois

```
1  #[...]
2  def _is_called(self,*args, **kw):
3      self._CALLED = self._CALLED + 1
4      yield aspects.proceed
5
6  def setup(self):
7      self._CALLED = 0
8
9  def teardown(self):
10     aspects.peel_around(wwbxml.load_xml)
11
12 def testResetDocumentLoadsTheXML(self):
13     aspects.with_wrap(self._is_called, wwbxml.load_xml)
14
15     p2.resetDocument("<root/>")
16     self.assertEquals(self._CALLED, 1)
```

Tester avec de l'AOP (suite)

Vérifier les appels

2 Vérifier quelle méthode est appelée en fonction du résultat de l'analyse du « parser »

```
1  Aspect:
2
3      >>> def is_run(*args, **keyw):
4      ...     global _groups, _in
5      ...     ret = yield aspects.proceed
6      ...     name = yield aspects.get_wrapped
7      ...     print name
8      ...     _groups = args[1]
9      ...     yield aspects.return_stop(ret)
10
11 If the group bold is found, the method named _on_bold should be called
12 but not _on_unknown:
13
14     >>> null = aspects.with_wrap(is_run, parser.Parser._on_bold)
15     >>> null = aspects.with_wrap(is_run, parser.Parser._on_unknown)
16
17     >>> m = re.search(r'(?P<bold>\*\*)', 'a**b')
18
19     >>> parser.updateDOM(m)
20     <unbound method Parser._on_bold>
```

Tester avec de l'AOP

Autres exemples

1. Bloquer des appels dans un Framework
2. Simuler l'accès à des fichiers ou au réseau
3. Simuler l'envoi d'une exception
   - Exemple : Simuler IOExeception d'un périphérique
4. Vérifier et comparer des traces de programmes
5. Simuler des appels de méthodes et de services
6. Injecter des assertions (Design by contract)
7. Simuler des entrées/sorties
8. Tester des sorties du programme (ex. : System.out)
9. Etc.

Tester de l'AOP

- Il peut être difficile de tester de l'AOP
  1. Le flot de contrôle est perturbé
  2. Il peut être difficile de planifier quand les aspects vont s'exécuter
  3. Des aspects qui n'étaient pas prévus peuvent s'ajouter (même dynamiquement)
  4. Les outils sont moins développés
  5. Ordre d'application des aspects
- Pour tester unitairement un aspect, il faut s'assurer que :
  1. Le code injecté (advice) fait bien ce qu'il doit faire
  2. Le code est injecté (pointcuts) aux bons endroits (join point)

Tester de l'AOP (suite)

Comment tester unitairement ?

1. Faire des « Mocks » et vérifier si l'aspect s'exécute à la bonne place (principe du pot de miel)
   - Méthodes où l'aspect devrait s'exécuter
   - Méthodes où l'aspect ne devrait PAS s'exécuter
2. Utiliser un aspect pour vérifier le comportement de l'aspect testé
3. Tester le résultat de l'exécution
   - Techniques classiques
   - Les outils sont parfois mal adaptés

Astuce

Mettre le moins de code possible dans les « advices » et appeler le code dans une classe autonome. Il est plus facile de tester une classe traditionnelle. (Peut varier en fonction du langage)

Tester de l'AOP (suite)

Autres tests

- Il faut aussi tester l'intégration des aspects
- Tout autre type de test que l'on fait normalement

Développement vs production

1. Aspects de développement
2. Aspects de production

Plan

Réusinage

- Problème :
  - Le nom des méthodes et classes peuvent changer
  - Il y a des chances que les « pointcuts » ne correspondent plus
  - Il est possible que les changement aients des répercutions sur le code de l'aspect
  - Ces répercutions peuvent être difficiles à anticiper
- Solutions :
  - Tester ! Tester ! Tester !
  - De bons tests unitaires facilitent toujours le réusinage (AOP ou non)
  - Être conscient du danger et vérifier (se fier à son instinct)
  - Coder intelligemment les aspects !

Attention !

Les aspects ne doivent pas être un prétexte pour ne pas réusiner !

Design Patterns

- Plusieurs patrons ont des versions AOP
- Recherches à UBC
- Bibliothèques (libraries) d'aspects disponibles
- Parfois plus simple, parfois plus complexe que la version OO
- Souvent plus génériques que la version OO

```
1  @Singleton //<<< Singleton version AOP
2  public class LockManager {
3      //[...]
4  }
5
6  public class Main {
7  // [...]
8        LockManager lock = new LockManager(); //Retourne l'instance!
```

Assurer l'intégrité des objets

1 Exemple : le « Object Dirty State »

- Indique si l'objet a changé depuis la dernière sauvegarde

```
1  public privileged aspect DirtyState {
2      declare warning :
3          call(* setDirty(boolean))
4          && within(AbstractModelObject+)
5          : "Dirty state is manged by an aspect now";
6
7      private boolean AbstractModelObject.isDirty = false;
8
9      public boolean AbstractModelObject.isDirty() {
10         return isDirty;
11     }
12
13     // [...]
14
15     protected pointcut simplePropertyChanged(AbstractModelObject inst, Object val) :
16         set(* AbstractModelObject+.*)
17         && !within(AbstractModelObject)
18         && !set(* isDirty)
19         && !set(* nbOwners)
20         && !withincode(new(..))
21         && target(inst)
22         && args(val);
```

Assurer l'intégrité des objets (suite)

```
23
24      protected pointcut collectionPropertyChanged(AbstractModelObject inst, Object obj) :
25          ( call(* Collection+.add*(..))
26              || call(* Collection+.clear*(..))
27              || call(* Collection+.retain*(..))
28              || call(* Collection+.remove*(..))
29              || call(* Collection+.set*(..))
30              || call(* Collection+.put*(..))
31              || call(* Collection+.trim*(..))
32          )
33          && withincode(* AbstractModelObject+.*(..))
34          && !within(AbstractModelObject)
35          && !withincode(new(..))
36          && this(inst)
37          && target(obj);
38
39      before(AbstractModelObject inst, Object newVal) : simplePropertyChanged(inst, newVal) {
40          // [...]
41          if (oldValue != newVal) {
42              inst.setDirty(true);
43          }
44      }
45
46      after(AbstractModelObject inst, Object obj) returning(Object res) : collectionPropertyChanged(inst, obj) {
47          // [...]
```

Assurer l'intégrité des objets (suite)

2. Envoyer automatiquement une notification aux observateurs (patron Observer)

Pourquoi ?

Il est fréquent d'oublier d'ajouter « setDirty(true) » ou « notifyObservers() » à la fin de toutes les méthodes de tout le modèle !

Diagnostic

- Aspects de développement
- On peut se servir des aspects pour vérifier si les conditions sont respectées pendant l'exécution
- But : Détecter des erreurs !

Exemples tirés d'un projet réel :

- S'assurer qu'un observateur ne s'enregistre qu'une seule fois sur le même objet
- S'assurer que tous les observateurs enlèvent leurs références sur un objet qui doit être détruit
- Lorsqu'un élément d'un modèle est supprimé, il doit également détruire ses enfants
  - Règle : « Celui qui le possède, le détruit »

Questions

Questions

Questions ?

Merci,