Programmez par Contrat

Emmanuel CHENU
emmanuel.chenu@gmail.com
http://emmanuelchenu.blogspot.com/

Cet article présente une pratique largement sous-utilisée en génie logiciel: la programmation par contrat. Nous verrons notamment en quoi cette démarche contribue à mettre en pratique des principes du développement logiciel agile et Lean.

La Programmation par Contrat, honteusement vulgarisée

La programmation par contrat a été introduite par Bertrand Meyer dans son langage Eiffel et décrite dans son livre **[CPOO]**. Cette démarche met l'accent sur la documentation et le respect de contrats de modules afin de garantir que le logiciel ne fait ni plus ni moins que ce qu'on attend qu'il fasse. Un contrat précise les responsabilités entre le client et le fournisseur d'un morceau de code logiciel.

Le contrat d'une routine est sa **pré-condition** et sa **post-condition**.
La pré-condition est la condition qui doit être vérifiée par le client avant le lancement d'une routine donnée. Cette condition doit garantir que l'exécution de la routine est possible sans erreur.
La post-condition est la condition qui doit être garantie par le fournisseur après le déroulement d'une routine donnée. Cette condition doit garantir que la routine a bien réalisé son travail.

> *« Si vous promettez d'appeler [une routine] avec une **pré-condition** vérifiée, en retour, je promets de délivrer un état final dans lequel la **post-condition** est vérifiée. »* **[CPOO]**.

Voici l'exemple pédagogique de la racine carré, en Java:

```
static double sqrt(double num) {
   assert num >= 0.0 : "precond num must be positive";
   final double res = java.lang.Math.sqrt(num);
   assert (res * res) == num : "postcond";
   return res;
}
```

« Une violation de pré-condition est la manifestation d'un bogue chez le client. » **[CPOO]**.
L'appel suivant:

```
res = Math.sqrt(-4.0);
```

Provoque la fin de l'exécution est l'affichage du message suivant dans la console:

```
run:
java.lang.AssertionError: precond num must be positive!
   at exemple.Math.sqrt(Math.java:19)
   at exemple.Main.main(Main.java:34)
   Exception in thread "main"
   Java Result: 1
```

Le bogue se trouve chez l'appelant de **`exemple.Math.sqrt()`**.

« Une violation de post-condition est la manifestation d'un bogue chez le fournisseur. » **[CPOO]**.

L'interruption de l'exécution du logiciel et l'affichage du message suivant:

```
run:
java.lang.AssertionError: postcond!
   at exemple.Math.sqrt(Math.java:22)
   at exemple.Main.main(Main.java:33)
   Exception in thread "main"
   Java Result: 1
```

indiquent que le bogue se trouve dans le code de **`exemple.Math.sqrt()`**.

La Programmation par Contrat et le développement logiciel Lean

Envisageons la programmation par contrat du point de vue d'un praticien du développement logiciel Lean.

Dans le livre **[ILSD]**, les auteurs donnent un exemple de produit détrompé :

> *« A properly mistake-proofed system will not need inspection. My video cable is an example of mistake-proofing. I can't plug a monitor cable into a computer or video projector upside down because the cable and the plug are keyed. »* **[ILSD]**

Je me suis demandé comment transposer cet exemple pour détromper du code. Comment concevoir une classe afin qu'elle ne puisse pas être mal utilisée?

La programmation par contrat avec des assertions est une manière de détromper du code. Regardez le court exemple suivant (en Java). Le code illustre l'exemple donné par Tom et Mary Poppendieck. L'opération **plug()** permet de connecter un câble de moniteur à un vidéo projecteur.

```
public class VideoProjector {

public void plug(MonitorCable monitorCable) {
      assert monitorCable != null : "PreCond: monitorCable != null";
      // some code ...
   }
...
```

Si l'opération **plug()** est utilisée avec un paramètre **monitorCable** qui vaut **null**, alors l'assertion termine l'exécution et le message suivant est affiché:

```
run:
Exception in thread "main" java.lang.AssertionError: PreCond: monitorCable !=null
   at zeroinspection.VideoProjector.plug(VideoProjector.java:14)
```

Ainsi, le code des classes **VideoProjector** et **MonitorCable** est conçu pour que la collaboration de leurs instances soit détrompée. La pratique de la programmation par contrat contribue donc à une gestion préventive des défauts si chère au Lean.

Lorsque une opération est mal utilisée, sa pré-condition interrompt l'exécution du logiciel. Lorsque une opération est mal implémentée, sa post-condition interrompt le déroulement de l'application. Ainsi, le logiciel signale la cause d'un bogue et l'application ne peut être utilisée avant la correction du problème détecté. La programmation par contrat contribue donc à la pratique Lean du « Stop-The-Line! ».

La Programmation par Contrat et le développement logiciel Agile

Maintenant, envisageons la programmation par contrat du point de vue d'un développeur agile. Le Manifeste Agile **[AM]** préconise de favoriser:

« Un logiciel opérationnel plutôt qu'une documentation exhaustive. » **[AM]**

Si vous reprenez l'exemple de la racine carré, vous constaterez que le contrat d'une opération documente intégralement celle-ci. La pré-condition donne les contraintes sur l'appel de l'opération et la post-condition décrit le résultat obtenu. Par contre, cette documentation est du logiciel opérationnel puisque les contrats sont formalisés en code source et sont exercés à l'exécution du logiciel. Ainsi la programmation par contrat favorise un logiciel opérationnel intégrant une documentation exhaustive à jour.

D'ailleurs, Andrew Hunt et David Thomas, co-rédacteurs du Manifeste Agile **[AM],** recommandent la pratique de la programmation par contrat dans leur livre **[PP]**:

« Tip 31. Design with Contracts. Use contracts to document and verify that code does no more and no less than it claims to do. » **[PP]**

Les auteurs insistent sur le fait que la pratique de la programmation par contrat permet de vérifier le respect du Principe de Substitution de Liskov.

La Programmation par Contrat et le Principe de Substitution de Liskov

« Subclasses must be usable through the base class interface without the need for the user to know the difference. » **[DAH]**

La programmation par contrat peut être utilisée en plus des tests pour s'assurer qu'il existe une vraie relation « est une sorte de » entre une sous classe et sa classe de base. Il s'agit alors d'écrire les contrats de manière à ce qu'une opération redéfinie dans une sous classe n'exige pas plus et ne garanti pas moins que l'opération de sa classe de base.

La Programmation par Contrat pratiquée

J'ai eu l'occasion de pratiquer la programmation par contrat de manière systématique sur plusieurs projets avec du code en C++, Java, Ada95, Ada2005 et Ruby. Cette démarche accélère très sensiblement la mise au point du logiciel. Aussi, cette pratique se combine parfaitement avec le pilotage par les tests qui garantie que les assertions sont toutes très fréquemment exercées.

BIBLIOGRAPHIE

- **[CPOO]** Conception et Programmation Orientées Objet, Bertrand Meyer, ISBN-10: 2212091117
- **[PP]** The Pragmatic Programmer: From Journeyman to Master, Andrew Hunt et David Thomas, ISBN-10: 020161622X
- **[ILSD]** Implementing Lean Software Development: From Concept to Cash, Tom et Mary Poppendieck, ISBN-10: 0321437381
- **[AM]** Manifesto for Agile Software Development, http://agilemanifesto.org/
- **[DAH]** Data Abstractions and Hierarchy, Barbara Liskov, SIGPLAN Notices, 23(5), Mai 1988.