UNIVERSITÉ DU QUÉBEC À MONTRÉAL

FACULTÉ DES SCIENCES

# Rapport du projet de session

présenté à Monsieur François-Xavier Guillemette

par Alexandre Lauzon (LAUA09018003)

Guy Bertrand (BERG17016106)

Stelie Luc (LUCS10627807)

dans le cadre du cours

MGL7361 Principes et applications de la conception de logiciels

Groupe 10

31 octobre 2011

# TABLE DES MATIÈRES

# 1 JOURNAL DE BORD

## 1.1 Semaine du 19 septembre

- Installation d'Éclipse, de NetBeans et des plugins nécessaires pour git.
- Réalisation de tests pour se rendre confortable avec git.
- Revue des Design Patterns et création initiale des cas d'utilisation.

## 1.2 Semaine du 26 septembre

- Création de la classe « Assert » contenant une méthode assertEquals() qui permet de vérifier l'égalité de deux objets.
- Création d'une classe de cas de tests « TestCase » et d'une interface « TestIndividuel ».
- Première ébauche du build.xml pour utilisation avec ANT.

## 1.3 Semaine du 3 octobre

- Modification du code pour ajouter les méthodes setUp() et tearDown() dans l'interface « TestIndividuel ».
- Modifications apportées aux fichiers build.xml et README pour inclure de nouvelles instructions pour la compilation de l'application via ant.
- Emphase sur le design et création de la classe « Trace » qui permet d'afficher les résultats en format texte directement sur la console.
- Refactoring de notre code pour permettre l'utilisation de l'annotation « @Test » pour désigner une méthode comme un test unitaire.

L'utilisation des annotations « @Test » permet de créer plusieurs tests unitaires dans une même classe qui implémente l'interface « TestIndividuel ». La raison principale qui nous a poussés à utiliser cette structure est que cela donne la flexibilité nécessaire à

l'utilisateur de notre librairie pour spécifier ces propres noms de méthode de test unitaire.

### 1.4 Semaine du 10 octobre

- Création d'une nouvelle classe « TestSuite » et d'une nouvelle interface « Test ».
- Utilisation du design pattern Composite dans les classes « TestSuite », « TestCase » et l'interface « Test ».
- Renommer l'interface « TestIndividuel » par « Testable ».
- Refactoring de notre code pour enlever l'utilisation des annotations « @Test ».
- Réalisation de la première version du document Word.
- Création de tous les cas d'utilisation.

Nous avons décidé d'utiliser le design pattern « Composite » afin de créer une hiérarchie entre les classes « TestSuite » (Composite) et « TestCase » (Feuille). L'interface « Test » (Composant) a été créée pour regrouper les fonctionnalités similaires entre ces deux classes. De cette façon, cette structure permet l'exécution de plusieurs cas de test dans une suite de test et/ou plusieurs suites de test dans une suite de test!

Les méthodes getNbTest() et getNbTestFailed() ont également été ajoutées à l'interface « Test » afin de permettre à la classe « TestSuite » de connaître le nombre de tests exécutés (succès/échec) pour chacun des cas de test.

Pour ce qui est des annotations « @Test », nous avons décidé de les enlever suite à une discussion entre les membres de notre équipe. En effet, bien que l'utilisation des annotations apportait quelques avantages, nous avons également décelé que cette méthode n'était pas très claire et pouvait apporter de la confusion pour un nouvel utilisateur de notre librairie. Or, afin de rendre notre code plus intuitif, nous avons décidé de forcer l'utilisateur de notre librairie à créer une classe implémentant notre interface « Testable », et ce, pour chacun des tests unitaires qu'il désire tester. Ce faisant,

l'utilisateur devrait comprendre plus facilement que l'exécution d'un test unitaire doit obligatoirement être définit à l'aide de la méthode execute() et que le résultat doit être un booléen (« True » = Succès).

### 1.5 Semaine du 17 octobre

- Modification et déplacement de classes du programme client dans le package « application.test ».
- Modification apportée à la classe « Trace » :
    - Gestion des paramètres d'entrée :
        - Afficher la trace à l'écran ou dans un fichier texte;
        - Écrire la trace en format texte ou XML.
    - Utilisation d'un second design pattern : Singleton;
    - Validation de la structure du XML généré à l'aide du site web du W3C : http://validator.w3.org/#validate_by_input.
- Première ébauche du diagramme de classe dans le document Word.
- Correction du fichier build.xml pour permettre la compilation et l'exécution de l'application à partir de ant.

Suite à l'ajout de la fonctionnalité qui permet de sauvegarder la trace dans un fichier texte, nous avons décidé de modifier la classe « Trace » pour qu'elle devienne un « Singleton ». L'utilisation de ce design pattern nous permet d'assurer la création et l'utilisation d'un seul fichier texte lors de l'exécution de plusieurs cas de tests et/ou suite de tests. Par la fait même, on s'assure que la trace soit complète et de regrouper tous les résultats d'exécution à un seul endroit.

### 1.6 Semaine du 24 octobre

- Déplacement des méthodes setUp() et tearDown() de l'interface « Testable » à la classe « TestCase » :
    - La classe « TestCase » devient abstraite.
    - Modification des cas d'utilisation afin de les adapter à la nouvelle structure des classes.
    - Adaptation des diagrammes de classes à la nouvelle structure des classes.
    - Mise à jour du document Word.
- Finalisation du fichier build.xml.
- Finalisation du document Word pour la remise du 31 octobre.

En discutant avec le professeur, nous sommes venus à la conclusion que les méthodes setUp() et tearDown() n'étaient tout simplement pas définies au bon endroit! En effet, en définissant ces méthodes dans l'interface « Testable », cela ne permettait pas à l'utilisateur de notre librairie de définir les événements à exécuter entre chacun des tests unitaires compris dans un même cas de test.

Nous avons donc déplacé l'implémentation de ces méthodes dans la classe « TestCase ». Puis, afin de permettre à l'utilisateur de notre librairie de spécifier l'implémentation de ces méthodes, nous avons dû les déclarer comme des méthodes abstraites. Suite à ce changement, l'utilisateur de notre librairie devra créer une classe qui implémente la classe « TestCase » dans son programme client.

# 2 USE CASE

## 2.1 CU1 - Créer un test unitaire

### Description

L'acteur désire créer un nouveau test unitaire.

### Acteur

Développeur

### Précondition

La librairie de test unitaire « ca.uqam.mgl7361.a2011.gamma » est disponible et importée dans la classe principale du projet à tester.

### Postcondition

La classe du nouveau test unitaire est créée.

### Scénario principal

1. L'acteur crée une nouvelle classe à l'intérieur de son projet.
2. L'acteur importe l'ensemble des fichiers de la librairie de test unitaire contenue dans le package « ca.uqam.mgl7361.a2011.gamma ».
3. L'acteur définit sa nouvelle classe comme une implémentation de l'interface « Testable ».
4. L'acteur redéfinit (Override) à l'intérieur de sa nouvelle classe la méthode execute() de l'interface « Testable ».

### Scénario alternatif

Aucun

## 2.2 CU2 - Créer un cas de test

**Description**

L'acteur désire créer un nouveau cas de test.

**Acteur**

Développeur

**Précondition**

La librairie de test unitaire « ca.uqam.mgl7361.a2011.gamma » est disponible et importée dans la classe principale du projet à tester.

**Postcondition**

Le nouveau cas de test est créé.

**Scénario principal**

1. L'acteur crée une nouvelle classe à l'intérieur de son projet.
2. L'acteur importe l'ensemble des fichiers de la librairie de test unitaire contenue dans le package « ca.uqam.mgl7361.a2011.gamma ».
3. L'acteur définit sa nouvelle classe comme une implémentation de la classe abstraite « TestCase ».
4. L'acteur redéfinit (Override) à l'intérieur de sa nouvelle classe les méthodes setUp() et tearDown().
5. L'acteur ajoute le(s) test(s) unitaire(s) qu'il désire tester à l'intérieur de ce cas de test à l'aide de la méthode addTest().

**Scénario alternatif**

Aucun

### 2.3 CU3 - Créer une suite de test

**Description**

L'acteur désire créer une suite de test pour lui permettre d'exécuter tous les cas de tests.

**Acteur**

Développeur

**Précondition**

La librairie de test unitaire « ca.uqam.mgl7361.a2011.gamma » est disponible et importée dans la classe principale du projet de l'acteur.

**Postcondition**

La nouvelle suite de test est créée.

**Scénario principal**

1. L'acteur crée une variable de type « TestSuite » dans la classe principale de son projet.
2. L'acteur ajoute le(s) cas de test qu'il désire tester à l'intérieur de cette suite de test à l'aide de la méthode addTest() de la classe « TestSuite ».

**Scénario alternatif**

3. L'acteur ajoute la(les) suite(s) de test(s) qu'il désire tester à l'intérieur de cette suite de test à l'aide de la méthode addTest() de la classe « TestSuite ».

## 2.4 CU4 - Exécuter un cas de test

### Description

L'acteur désire exécuter un cas de test.

### Acteur

Développeur

### Préconditions

La librairie de test unitaire « ca.uqam.mgl7361.a2011.gamma » est disponible et importée dans la classe principale du projet de l'acteur.
Au moins un cas de test existe dans le projet à tester.
Le cas de test contient un ou plusieurs test(s) unitaire(s).

### Postcondition

Le cas de test a été exécuté et les résultats de l'exécution sont affichés.

### Scénario principal

1. L'acteur exécute le cas de test à l'aide de la méthode execute() de la classe qui implémente « TestCase » dans la classe principale de son projet.
2. Les résultats de l'exécution du cas de test sont affichés à la console ou dans un fichier (selon l'initialisation de la « Trace »).

### Scénario alternatif

Aucun

### 2.5 CU5 - Exécuter une suite de test

**Description**

L'acteur désire exécuter une suite de test.

**Acteur**

Développeur

**Préconditions**

La librairie de test unitaire « ca.uqam.mgl7361.a2011.gamma » est disponible et importée dans la classe principale du projet de l'acteur.
Au moins une suite de test existe dans le projet à tester.
La suite de test contient un ou plusieurs cas de test.

**Postcondition**

La suite de test a été exécutée et les résultats de l'exécution sont affichés.

**Scénario principal**

1. L'acteur exécute la suite de test à l'aide de la méthode execute() de la classe « TestSuite » dans la classe principale de son projet.
2. Les résultats de l'exécution de la suite de test sont affichés à la console ou dans un fichier (selon l'initialisation de la « Trace »).

**Scénario alternatif**

Aucun

## 2.6 CU6 - Modifier le comportement de la classe Trace

### Description

Par défaut, la classe « Trace » affiche les résultats en format texte directement sur la console. Il est également possible d'afficher les résultats en format XML et/ou dans un fichier texte. Ce cas d'utilisation permet donc à l'acteur de modifier le comportement par défaut de la classe « Trace ».

### Acteur

Développeur

### Précondition

La librairie de test unitaire « ca.uqam.mgl7361.a2011.gamma » est disponible et importée dans la classe principale du projet de l'acteur.

### Postcondition

Le comportement de la Trace a été modifié.

### Scénario principal

1. L'acteur modifie le comportement de la classe « Trace » à l'aide de la méthode Trace.getInstance().initialize(boolean, boolean) dans la classe principale de son projet :
    a. Le premier paramètre indique si l'acteur désire voir le contenu de la trace en format XML (« True ») ou en format texte (« False »).
    b. Le deuxième paramètre indique si l'acteur désire voir le contenu dans un fichier texte (« True ») ou directement sur la console (« False »).

### Scénario alternatif

2. Si l'acteur indique qu'il désire voir le contenu de la trace dans un fichier texte, il doit obligatoirement utiliser la méthode Trace.getInstance().close() à la toute fin de la

classe principale de son projet afin de permettre la sauvegarde du fichier créé par la classe « Trace ».

# 3 DIAGRAMMES DE CLASSES

## 3.1 Librairie de tests unitaires

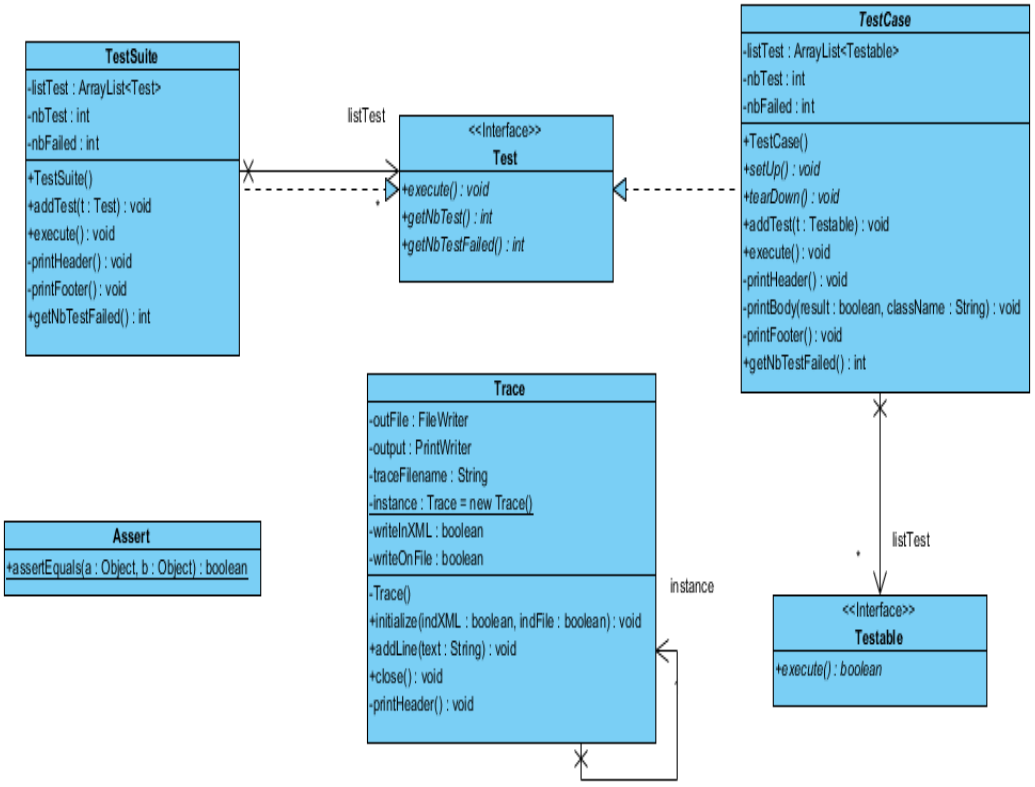


Figure 1: Diagramme des classes de la librairie de tests unitaires

## 3.2 Programme client

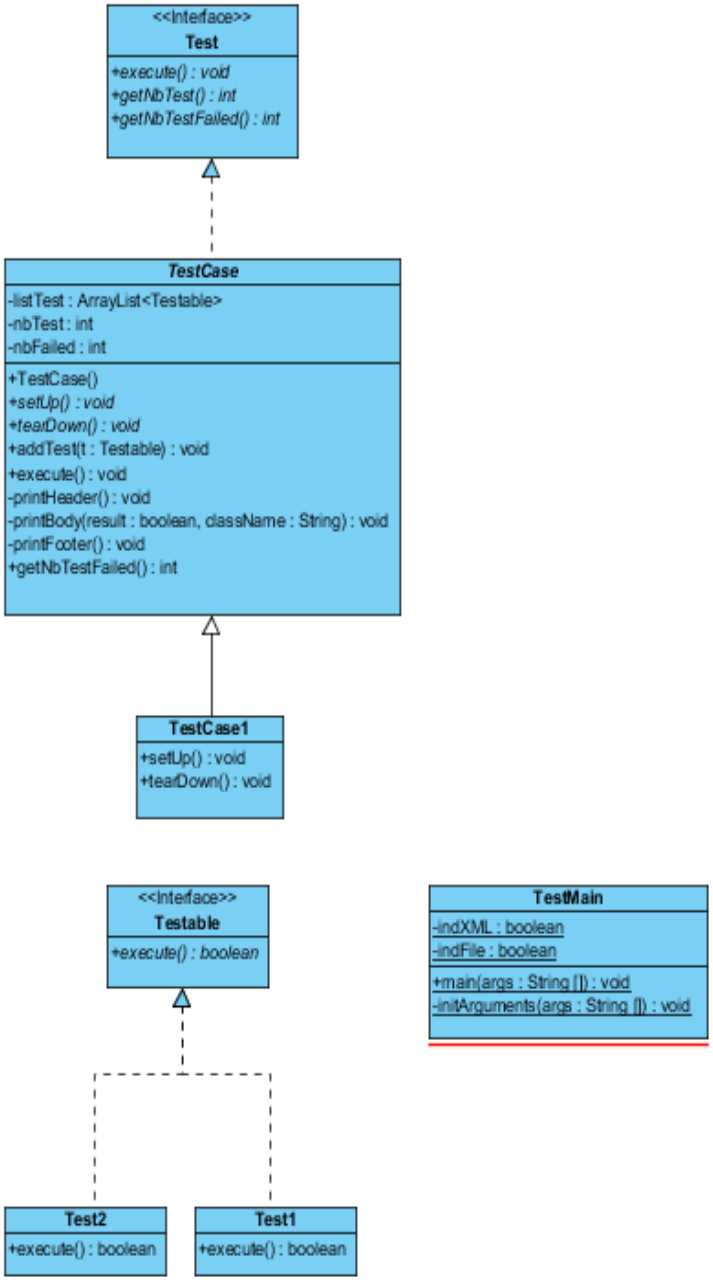


Figure 2: Diagramme des classes du programme client

# 4 INSTRUCTIONS POUR GIT HUB

$ git clone git://github.com/guybertrand/MGL7361-Gamma.git
ou
$ git clone https://guybertrand@github.com/guybertrand/MGL7361-Gamma.git

## 4.1 Informations générales

Dans GitHub, les sources de notre librairie de tests unitaires se retrouvent dans le répertoire « /src/ca/uqam/mgl7361/a2011/gamma ». Les sources du programme client se retrouvent pour leur part dans le répertoire « /srcTest/application/test ».

## 4.2 Compiler et exécuter

$ cd MGL7361-Gamma
$ ant

Les exécutables se retrouveront dans les sous-répertoires DIST (Gamma.jar) et DISTTEST (monApp.jar).

## 4.3 Exécuter seulement le code (paramètres par défaut)

$ cd MGL7361-Gamma
$ ant run

## 4.4 Exécuter à partir de la ligne de commande avec JAVA

$ java -cp ./distTest/monApp.jar:./dist/Gamma.jar application.test.TestMain

Par défaut, le résultat de l'exécution sera affiché à l'écran en format texte. Il est possible d'utiliser les paramètres suivants pour modifier le comportement de l'exécution :

- xml=true ➔ Afficher les résultats en format XML
- file=true ➔ Sauvegarder les résultats de l'exécution dans un fichier

Exemple :

1) Exécution et sauvegarde des résultats dans un fichier texte :

```
$ java -cp ./distTest/monApp.jar:./dist/Gamma.jar application.test.TestMain file=true
```

2) Exécution et affichage des résultats à l'écran en format XML :

```
$ java -cp ./distTest/monApp.jar:./dist/Gamma.jar application.test.TestMain xml=true
```

3) Exécution et sauvegarde des résultats dans un fichier en format XML :

```
$ java -cp ./distTest/monApp.jar:./dist/Gamma.jar application.test.TestMain xml=true file=true
```

### 4.5 Exécuter seulement le code (paramètres spécifiques)

```
$ cd MGL7361-Gamma
$ ant runParam -Darg1="xml=true" -Darg2="file=true"
```

### 4.6 Nettoyer et supprimer les répertoires créés par ant

```
$ cd MGL7361-Gamma
$ ant clean
```

## 5 DETAIL DU FICHIER BUILD.XML POUR UTILISATION AVEC ANT

Le fichier build.xml (pour ANT) sépare le code de notre librairie et celle du programme client dans des répertoires séparés et créé deux fichiers JAR distincts.

- Librairie Gamma.jar :
    - /src : Code source
    - /build : Class compilé
    - /dist : Fichier JAR nommé Gamma.jar

- Librairie code usager qui exécute des tests et séries de tests :
    - /srcTest : Code source
    - /buildTest : Class compilé
    - /distTest : Fichier JAR nommé monApp.jar

Par défaut, le fichier build.xml va exécuter l'application en utilisant les paramètres par défaut. Si vous désirez spécifier des paramètres spécifiques, vous devez utiliser l'option « ant runParam » tel que définie dans la section précédente.