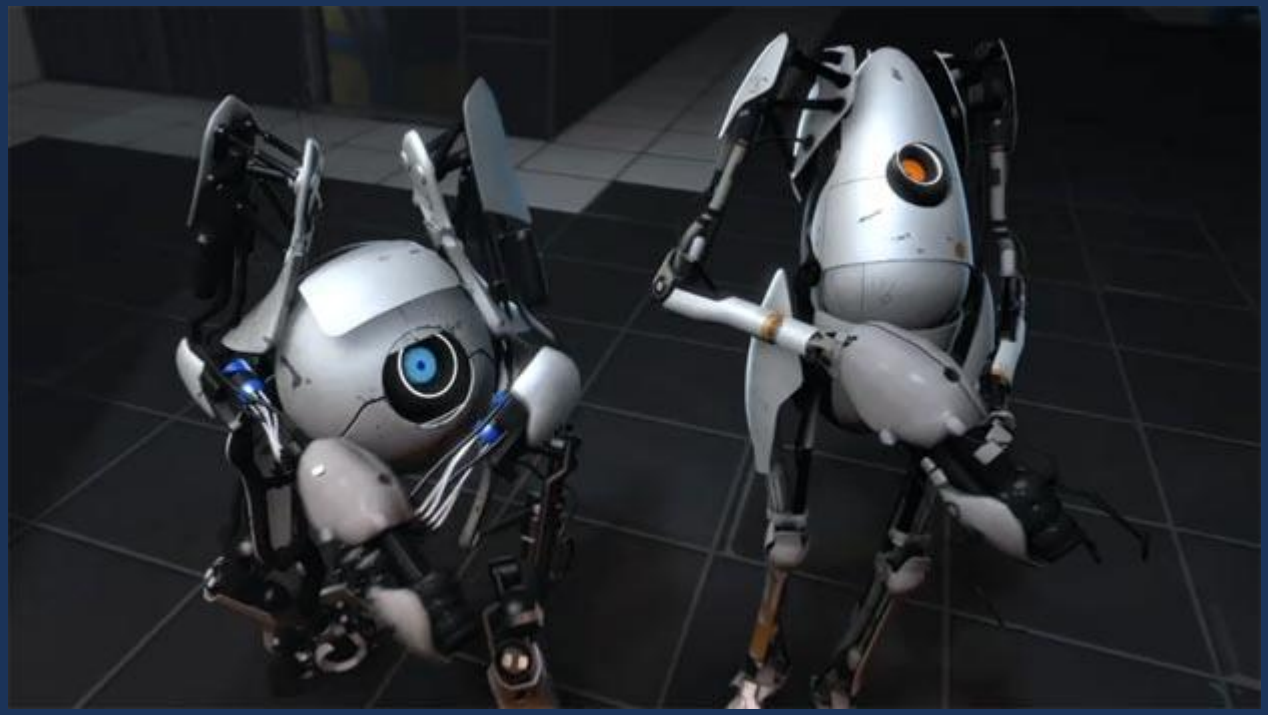
TESTS UNITAIRES

420-V31-SF – Programmation de Jeux Vidéo III

fparadis@cegep-ste-foy.qc.ca

Avant de publier, tester

Parce que pour faire un système éprouvé et rigoureux, il faut tester.



Tests unitaires

Il s'agit pour le programmeur de tester un module, indépendamment du reste du programme, ceci afin de s'assurer qu'il répond aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances. Cette vérification est considérée comme essentielle, en particulier dans les applications critiques

Couverture de code

Les tests unitaires s'accompagnent couramment d'une vérification de la couverture de code (évaluation de la couverture structurelle), qui consiste à s'assurer que le test conduit à exécuter l'ensemble (ou une fraction déterminée) des instructions présentes dans le code à tester…

Couverture de code

On s'assure qu'une ligne de code est testée

On s'assure que toutes les conditions et branchements sont possibles.

if / else, switch : doit aller dans toutes les possibilités

Une bonne couverture de code: 80% et plus on est en business

Certaines lignes de code sont durs à tester de manière unitaire: l'affichage, les fonctions void, etc.

Comment écrire des tests.

Dans tous les cas vos tests devraient couvrir un maximum de code (schéma de couverture)

Tests de régression.

L'ensemble des tests unitaires doit être rejoué après une modification du code afin de vérifier qu'il n'y a pas de régression (l'apparition de nouveaux dysfonctionnements).

Test de régression: après une modification à une classe, on refait rouler les tests unitaires afin de s'assurer que tout les tests passent toujours.

Si un test ne fonctionne plus correctement après une modification, on est en situation de régression.

Tests unitaires

Dans les applications non critiques, l'écriture des tests unitaires a longtemps été considérée comme une tâche secondaire.

Cependant, plusieurs manières de faire, spécialement le développement dirigé par les tests ("Test-driven developpement" ou TDD), ont remis les tests unitaires, appelés « tests du programmeur », au centre de l'activité de programmation.

Tests unitaires

À quelque part, un test unitaire n'est pas très différent d'un jeu d'essai

Dans le langage des tests unitaire, les variables d'entrées sont souvent appelées pré-conditions et les variables de sorties: post-conditions

On veut tester…

la fiabilité (pré et post-conditions valides)

la robustesse (pré-conditions invalides et traitement approprié, donc post-conditions valides également)

Les post-conditions devraient toujours être valides, si ce n'est pas le cas, alors le test échoue.

Comment écrire des tests.

Exemple de test de fiabilité

Entrez un âge (entre 0 et 120 ans);

- La méthode retourne true si le nombre entrée est entre 0 et 120 ans, false sinon.

Vous entrez 25 vous avez true

Vous entrez -15 vous avez false

Comment écrire des tests.

Exemple de test de robustesse

Entrez un âge (entre 0 et 120 ans);

- Vous entrez alors une donnée totalement invalide dans le but de faire planter le système.
- Exemple : vous entrez à la place d'un nombre la string suivante "Demander l'âge d'une dame est impoli!"
- Le système ne devrait pas planter et effectuer un traitement approprié.
- Évidemment, le test devrait donner false.

Comment écrire des tests (Important).

Si une pré-condition a une portée de valeurs définie, alors on doit tester.

- *La limite inférieure*
- *La limite supérieure*
- *Un cas quelque part au milieu*
- *Un cas juste avant la limite inférieure*
- *Un cas juste après la limite supérieure*

Dans notre exemple de l'âge, on devrait faire le test avec -1, 0, 30 (par exemple), 120 et 121

Comment écrire des tests.

Si une pré-condition a des conditions d'échec précises, alors il faut simplement les tester une à une.

Si "Toutes les situations" passent, sauf quelques cas, tester ces quelques cas.

Comment écrire des tests.

Si vos pré-conditions peuvent être invalides de multiples façons, très important de tester l'écart du cas correct d'une seule façon à la fois.

Tester une pré-condition invalide de plus d'une façon rend le diagnostic de l'erreur naturellement plus difficile pour rien (devient difficile de déterminer quel cas fait échouer).

Exemple, si dans un seul test on vérifie plusieurs âges à la fois, on ne saura dire lequel fait échouer le test.

Comment écrire des tests.

Chaque classe devrait avoir sa classe de test, portant le même nom que la classe d'origine et ajoutez-lui test

Chaque méthode publique doit avoir au moins une méthode dédiée à son cas précis. Si cette méthode est unique, lui donner le même nom, plus le mot "Test" à la fin.

Si une méthode a plusieurs méthodes de test, on lui donne [NomDeLaMéthode]_[DescriptionDeCeQu'onTeste]

Les méthodes privées n'ont pas à être testés directement. De toute façon, elles le seront indirectement quand appellées par des méthodes publiques.

Tests sur un mot de passe

Mot de passe doit avoir de 5 à 8 caractère alphanumériques et se terminer par un chiffre.

Test du constructeur par défaut

Test du constructeur: Password(thePassword)

- cas valide 1: cas standard
- cas valide 2: cas limite, 5 caractères
- cas valide 3: cas limite, 8 caractères
- cas valide 4: caractères tous alphanumériques.

Tests sur un mot de passe

Test du constructeur: Password(thePassword)

- cas invalide 1: mot de passe vide
- cas invalide 2: mot de passe < 5 caractères
- cas invalide 3: mot de passe > 8 caractères
- cas invalide 4: mot de passe ne se termine pas par 1 chiffre
- cas invalides 5 à 7: mot de passe avec un caractère non-alphanumérique

Exemple

Voir TP1 Prog mobile

Normalement, en TDD, le programmeur détermine lui-même ses tests. Ici on a fourni les tests

Si tous les tests passent, la méthode est considérée valide.

420-V31-SF – Programmation de Jeux Vidéo III

Tests unitaires dans Visual C++ 2015

fparadis@cegep-ste-foy.qc.ca

Tests dans Visual C++ 2015

Nouveau projet ->Visual C++ -> Test ->
Projet de test unitaire natif

Même nom que la solution + Test

Un fichier de test par projet. Un fichier de test peut contenir plusieurs classes de test

Pourrait avoir un fichier par classe, mais on va garder les choses simples

Tests dans Visual C++ 2015

ProjetDeTest->Dépendances de Build -> Dépendance du projet:

Cocher votre projet à tester

ProjetDeTest->Ajouter->Référence->[Votre projet à tester]

ProjetDeTest->Propriétés->Éditeur de liens->entrée->dépendances supplémentaires:

Ajouter le fichier .obj de votre classe à tester

Path:
../[NomDuProjet]/[Debug ou Release]/votreFichier.obj

Tests dans Visual C++ 2015

Inclure les classes à tester avec le path relatif précis

Nom de la classe de test dans les parenthèses de TEST_CLASS

Nom des méthodes de test dans les parenthèses de TEST_METHOD