

Coding Week

École Centrale
Casablanca
2026

Casablanca, Maroc

CODING WEEK

Livrer du logiciel robuste



Tests, seams, reproductibilité à A4, Coding Week 1A

Hermann Agossou

Doctorant, UTER MID@S, ECC / UH2C

¹École Centrale Casablanca

Le test comme preuve

01

Le test comme preuve

Rappel : d'où venaient les tests au vert

Deux questions sur le refactoring

L'étape rouge

En A2, que faisiez-vous d'une étape qui laissait la suite au rouge ?

Et ces tests-là

Qui les avait écrits ? Et comment savait-on qu'ils testaient la bonne chose ?

LA DEUXIÈME QUESTION N'AVAIT PAS DE RÉPONSE

En A2, la suite de tests vous était fournie et vous ne faisiez que l'exécuter. Aujourd'hui, c'est vous qui l'écrivez et qui devez prouver qu'elle détecte quelque chose.

Le bug est corrigé, et maintenant

Qu'est-ce qui vous dit qu'il ne reviendra pas

AVANT

'parse_amount' transmet directement
"12,50" à 'Decimal'.

- ▶ Résultat observé dans le défaut
fourni : 'Decimal("0")'

APRÈS

Remplacer la virgule par un point avant la
conversion.

- ▶ Résultat attendu :
'Decimal("12.50")'

LA QUESTION INCONFORTABLE

Le programme marche de nouveau. Qu'est-ce qui vous garantit que quelqu'un ne réintroduira pas ce bug le mois prochain sans que personne ne s'en aperçoive ?

Test, seam, test de caractérisation

Le vocabulaire avant l'outillage

test

Un énoncé exécutable de ce que le code doit faire.

seam

Un endroit où l'on peut changer le comportement sans éditer à cet endroit.

point d'activation

L'endroit où l'on décide lequel des comportements sera utilisé.

test de caractérisation

Un test qui fige ce que le code fait aujourd'hui, pas ce qu'il devrait faire.

Le test comme preuve

Beck : rouge, vert, refactorer

Le test qui échoue vient d'abord

WORKFLOW 01

Rouge

Écrire un test qui échoue. C'est la preuve qu'il détecte quelque chose.

WORKFLOW 02

Vert

Le faire passer de la manière la plus simple, même laide.

WORKFLOW 03

Refactorer

Supprimer la duplication introduite. C'est ici que la conception arrive.

LES DEUX RÈGLES

- ▶ N'écrire du code fonctionnel qu'après avoir vu un test échouer.
- ▶ Éliminer la duplication.

Voir: [1] Beck (2002)

Le test comme spécification

Ce que le code doit faire, pas ce qu'il fait

TEST ÉCRIT APRÈS

Il confirme ce que le code fait déjà.

- ▶ Y compris ses bugs
- ▶ Il passe toujours du premier coup
- ▶ Il ne démontre pas qu'il détectait le défaut initial

TEST ÉCRIT AVANT

Il énonce ce que le code doit faire.

- ▶ Il échoue d'abord, pour une raison précise
- ▶ Il survit à la réécriture de l'implémentation
- ▶ Il reste lisible par quelqu'un d'autre

Un test est la seule forme d'énoncé qui survit à son auteur, à son langage et à son implémentation.

Le test comme preuve

Écrire le test qui aurait attrapé le bug

Avant de corriger, pas après

WORKFLOW 01

Test

Exiger que `parse_amount("12,50")` vaille `Decimal("12.50")`.

WORKFLOW 02

Correctif annulé

Échec : obtenu `Decimal(0)`. C'est la preuve rouge.

WORKFLOW 03

Correctif réappliqué

Un test passe. C'est la preuve verte.

VÉRIFICATION

Exécutez le test sur le code non corrigé et confirmez qu'il échoue pour la raison annoncée, puis réappliquez le correctif et confirmez qu'il passe. Un test déjà vert avant le correctif n'a jamais capturé le défaut.

Le test comme preuve

Terminer le test commencé

L'assertion est à vous

Appel	Obtenu avec le défaut	Attendu
Montant '100', remise '0.10'	'Decimal("81")'	'Decimal("90")'

À COMPLÉTER

Écrivez l'assertion, exécutez-la sur le code non corrigé, et notez le message d'échec exact. Si votre premier essai est vert, c'est que vous l'avez écrit contre le correctif.

Le code des autres

02

Feathers : du code sans tests est du legacy

Et le dilemme que cela crée

LA DÉFINITION

Du code legacy est du code sans tests. Ni son âge ni son auteur n'entrent en compte : c'est l'absence de tests qui rend le changement risqué.

LE DILEMME

Pour changer du code en sécurité, on voudrait des tests. Pour poser des tests, il faut souvent changer le code. C'est ce cercle que le seam ouvre.

Voir: [2] Feathers (2004)

Les seams : changer sans éditer sur place

Et le point d'activation qui décide

AVANT : RÉSEAU DIRECT

'taux_du_jour' choisit lui-même le client HTTP.

- ▶ Le test dépend du réseau réel

APRÈS : SEAM D'OBJET

'taux_du_jour' reçoit le client comme paramètre.

- ▶ Le paramètre devient le point d'activation

VÉRIFICATION

Substituez un stub à travers le seam et vérifiez que le vrai collaborateur n'est jamais appelé. S'il s'exécute quand même, le seam n'est pas là où vous le croyiez.

Trouver un seam dans du code inconnu

Exemple entièrement traité

01 1. Test de caractérisation

Figurer ce que le code fait aujourd'hui, sans juger si c'est correct.

02 2. Introduire le seam

Un paramètre, une valeur par défaut : aucun appelant existant ne change.

03 3. Substituer un stub

Le test ne dépend plus du réseau.

04 4. Rejouer la caractérisation

Toujours vert : le comportement a été conservé.

VÉRIFICATION

Le test de caractérisation, écrit avant toute modification, doit rester vert après l'introduction du seam. Il vérifie la conservation du comportement couvert par ce test ; il ne prouve rien sur les cas absents.

À vous : mettre sous test

Sans modifier le comportement

CONSIGNE

Prenez la fonction distribuée, écrivez son test de caractérisation, introduisez un seam, et mettez-la sous test sans changer son comportement observable.

- ▶ Le test de caractérisation s'écrit **AVANT** toute modification
- ▶ Un appelant existant ne doit pas avoir à changer
- ▶ Rendez les deux exécutions : avant et après

Wilson et al. : le minimum pour rejouer

Écrit pour des scientifiques, pas des ingénieurs logiciel

LE MINIMUM, ET RIEN DE PLUS

- ▶ Un répertoire par projet, avec les étapes écrites pour le rejouer.
- ▶ Les dépendances et les commandes consignées, pas laissées dans un historique de shell.
- ▶ Tout ce dont un collaborateur a besoin sous gestion de version.

Voir: [3] Wilson et al. (2017)

Ce n'est pas un idéal d'ingénierie : c'est le seuil en dessous duquel un travail n'est pas rejouable.

Déclarer et isoler ses dépendances

Le lecteur suivant est peut-être une machine

DÉPENDANCES IMPLICITES

ñ Ça marche chez moi. ž

- ▶ Paquets installés à la main, un jour, on ne sait plus quand
- ▶ Version jamais notée
- ▶ Personne d'autre ne peut relancer

DÉPENDANCES DÉCLARÉES ET ISOLÉES

Le projet porte ce dont il a besoin.

- ▶ Un fichier de dépendances versionné
- ▶ Un environnement isolé, pas la machine
- ▶ La configuration hors du code

CE QUI A CHANGÉ RÉCEMMENT

Le prochain lecteur de votre projet est peut-être un assistant. Un assistant qui ne sait pas lancer votre projet depuis un clone vierge ne peut pas vous aider dessus.

Voir: [4] Wiggins (2011)

Cloner à vide et tout relancer

Chaque commande inventée est un défaut

WORKFLOW 01

Cloner

Partir d'un répertoire neuf, sans état local implicite.

WORKFLOW 02

Suivre

Exécuter uniquement les étapes écrites dans le README.

WORKFLOW 03

Noter

Chaque commande improvisée devient un défaut à corriger.

Commandes inventées

= défauts du README

VÉRIFICATION

Clonez dans un répertoire vide, sur une machine où rien n'est préinstallé, suivez uniquement les étapes écrites et notez chaque commande qu'il a fallu inventer. Chacune est une ligne manquante dans les instructions.

Ce qui reste quand l'outillage change

Trois critères, pas trois outils

Le test est une spécification

Il survit à l'implémentation, au langage et à l'auteur.

Le seam rend testable sans réécrire

Changer le comportement sans éditer sur place.

Rejouable depuis un état vierge

Un projet qui ne tourne que chez son auteur n'est pas fini.

CE QUI N'ÉTAIT QUE DE L'OUTILLAGE

pytest, les fixtures, les environnements virtuels, la CI. En A5 : quand une contribution arrive plus vite qu'on ne peut la relire, c'est le test qui tient encore le comportement.

Références

03