R4.02 – Qualité de développement

Simon Robillard

8 février 2023

Section 1

Syllabus

Objectifs du cours

- comprendre comment intégrer la notion de qualité logicielle au processus de développement
- découvrir des outils permettant de tester efficacement des logiciels

Fonctionnement du cours : classe inversée

un seul CM (aujourd'hui) + 8 séances de TD/TP

- **chez vous**
  - des notes de cours à lire obligatoirement
  - des liens vers des ressources additionnelles à consulter facultativement
- **en classe**
  - une session de questions/réponses pour clarifier les notes de cours
  - un QCM sur les dernières notes de cours
  - TD et/ou TP

Accès aux ressources

- Tout le cours est sur Moodle

  `https://moodle.umontpellier.fr/course/view.php?id=29878`

- les slides et notes de cours seront mises en ligne au fur et à mesure
- le code des TP sera sur Gitlab

  `https://gitlabinfo.iutmontp.univ-montp2.fr/r402`

- Les intervenants
  - Simon Robillard (chargé de cours + TP Q3/Q4)
  - Nadjib Lazaar (TP Q1/Q2)
  - Gaëlle Hisler (TP Q5)
  - contact : `prenom.nom@umontpellier.fr`

Évaluation

- notes de QCM (40%)
- deux TP notés (30% + 30%)

Section 2

L'importance de la qualité logicielle

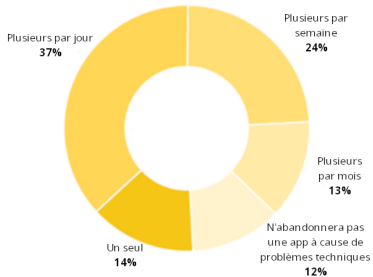
Les bugs, c'est grave ?

Les défauts dans des systèmes informatiques peuvent causer une perte de...

temps...	argent ...	données privées ...	ou pire ...
			
USS Yorktown	Pentium FDIV	Heartbleed	Lime scooters
arrêté pendant 3 heures	$475 millions	TLS compromis	blocage des freins à haute vitesse

Abandon des applications

- La qualité logicielle compte pour les utilisateurs
  - 88% des utilisateurs sont susceptibles d'abandonner des applications à cause de problèmes techniques
- Il y a encore des progrès à faire
  - 78% des utilisateurs remarquent des problèmes
  - 29% en rencontrent au moins un par semaine
- La qualité est donc aussi un élément important pour les logiciels non-critiques



Réponse à la question "à partir de combien de problèmes techniques êtes vous susceptibles d'abandonner une application ?"

Source : étude QualiTest sur les abandons d'applications

`https://qualitestgroup.com/news/survey-88-of-app-users-will-abandon-apps-based-on-bugs-and-glitches/`

Le coût de la mauvaise qualité logicielle

En 2020, aux États-Unis :

Le coût de la mauvaise qualité logicielle

En 2020, aux États-Unis :

$ 260 milliards

dûs aux **projets informatiques infructueux**

Le coût de la mauvaise qualité logicielle

En 2020, aux États-Unis :

$ 260 milliards

dûs aux **projets informatiques infructueux**

$ 520 milliards

dûs à la mauvaise qualité dans les **systèmes informatiques anciens**

Le coût de la mauvaise qualité logicielle

En 2020, aux États-Unis :

$ 260 milliards
dûs aux **projets informatiques infructueux**

$ 520 milliards
dûs à la mauvaise qualité dans les **systèmes informatiques anciens**

$ 1560 milliards
dûs à des **défaillances logicielles**

Le coût de la mauvaise qualité logicielle

En 2020, aux États-Unis :

$ 260 milliards
dûs aux **projets informatiques infructueux**

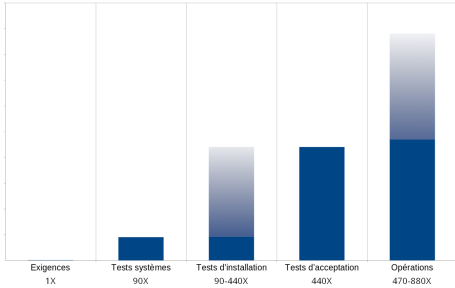
$ 520 milliards
dûs à la mauvaise qualité dans les **systèmes informatiques anciens**

$ 1560 milliards
dûs à des **défaillances logicielles**

Source : Rapport CISQ – *The Cost of Poor Software Quality in the US : A 2020 Report*
`https://www.it-cisq.org/the-cost-of-poor-software-quality-in-the-us-a-2020-report/`

Le coût d'une réparation suivant le moment de sa détection

- plus un défaut est **découvert tard** plus il est **coûteux à réparer**
- les conséquences en production peuvent être désastreuses : exemple des accélérateurs Toyota
  - 9 millions de véhicules rappelés coût estimé à 1,4 milliards €
  - amende de 1,2 milliards $
- il est crucial découvrir les défauts logiciels **le plus tôt possible**



Source : *BNR/NORTEL : path to improve product quality, reliability and customer satisfaction* (Barziuk, 1995)

Comment assurer la qualité logicielle ?

Il existe de nombreuses pratiques, qui ne sont pas mutuellement exclusives

- revues de code
  - tout nouveau code est examiné avant d'être intégré au logiciel
  - analyse automatique et/ou manuelle
  - peut utiliser le versionnage (*feature branch workflow*)

Comment assurer la qualité logicielle ?

Il existe de nombreuses pratiques, qui ne sont pas mutuellement exclusives

- revues de code
  - tout nouveau code est examiné avant d'être intégré au logiciel
  - analyse automatique et/ou manuelle
  - peut utiliser le versionnage (*feature branch workflow*)
- programmation en binôme
  - un "conducteur" et un "observateur"
  - échangent les rôles à intervalles régulier
  - permet une revue de code en continu et une diffusion des bonnes pratiques

Comment assurer la qualité logicielle ?

Il existe de nombreuses pratiques, qui ne sont pas mutuellement exclusives

- revues de code
  - tout nouveau code est examiné avant d'être intégré au logiciel
  - analyse automatique et/ou manuelle
  - peut utiliser le versionnage (*feature branch workflow*)
- programmation en binôme
  - un "conducteur" et un "observateur"
  - échangent les rôles à intervalles régulier
  - permet une revue de code en continu et une diffusion des bonnes pratiques
- **tests logiciels** : la technique la plus commune, au centre de ce cours

Comment assurer la qualité logicielle ?

Il existe de nombreuses pratiques, qui ne sont pas mutuellement exclusives

- revues de code
  - tout nouveau code est examiné avant d'être intégré au logiciel
  - analyse automatique et/ou manuelle
  - peut utiliser le versionnage (*feature branch workflow*)
- programmation en binôme
  - un "conducteur" et un "observateur"
  - échangent les rôles à intervalles régulier
  - permet une revue de code en continu et une diffusion des bonnes pratiques
- **tests logiciels** : la technique la plus commune, au centre de ce cours
- méthodes formelles
  - techniques d'analyse basées sur la logique/les maths/l'informatique théorique
  - certaines techniques automatiques, d'autres demandent une expertise
  - commence à se répandre en entreprise (ex : analyse statique chez Facebook)

Section 3

Définitions

Défauts, erreurs, défaillances

Plutôt que le terme vague de “bug”, on distinguera :

- **défaut logiciel** : existence statique d’un vice dans le logiciel.
  Concerne le *logiciel* lui-même.
- **erreur logicielle** : état d’exécution incorrect, causé par un défaut.
  Concerne une *exécution* du logiciel.
- **défaillance logicielle** : comportement incorrect manifesté par le logiciel à la suite d’une erreur.
  Concerne un *résultat* du logiciel (valeur calculée, action effectuée...).

Un défaut ne mène pas forcément à une erreur (en particulier si le code en question n’est pas exécuté), une erreur ne mène pas forcément à une défaillance.

L'état d'exécution d'un programme

L'état d'exécution d'un programme est caractérisé par

- les valeurs de toutes les variables accessibles
- le compteur de programme qui indique la prochaine instruction à exécuter

Programme

```
1 int[] a = {2, -1};
2 int i = 0;
3 while (i < a.length) {
4     if (a[i] < 0) {
5         a[i] = 0;
6     }
7     i++;
8 }
```

États d'exécution

Compteur	Variables
l.1	
l.2	$a \mapsto \{2, -1\}$
l.3	$a \mapsto \{2, -1\}, i \mapsto 0$
l.4	$a \mapsto \{2, -1\}, i \mapsto 0$
l.7	$a \mapsto \{2, -1\}, i \mapsto 0$
l.3	$a \mapsto \{2, -1\}, i \mapsto 1$
l.4	$a \mapsto \{2, -1\}, i \mapsto 1$
l.5	$a \mapsto \{2, -1\}, i \mapsto 1$
l.7	$a \mapsto \{2, 0\}, i \mapsto 1$
l.3	$a \mapsto \{2, 0\}, i \mapsto 2$
l.8	$a \mapsto \{2, 0\}, i \mapsto 2$

Est-ce-que ce programme est correct ?

```java
public static int f(Object[] a, Object o) {
    for (int i = 0; i < a.length; i++) {
        if (o.equals(a[i])) {
            return i;
        }
    }
    return -1;
}
```

Est-ce-que ce programme est correct ?

```java
public static int f(Object[] a, Object o) {
    for (int i = 0; i < a.length; i++) {
        if (o.equals(a[i])) {
            return i;
        }
    }
    return -1;
}
```

Cette question n'a pas de sens !
Un programme ne peut être correct ou incorrect que par rapport à une attente : sa **spécification**

Spécification

Un description du comportement attendu d'un programme en fonction de ses entrées.
La forme exacte de cette description dépend du projet

- user story
- documentation utilisateur
- documentation du code (ex : javadoc)
- document décrivant les exigences du système

Vérification & validation

- **Vérification** : processus pour déterminer si le résultat d'une étape de développement logiciel satisfait les exigences établies durant l'étape précédente
- **Validation** : processus pour déterminer si le produit final est conforme à l'usage attendu
- Les deux activités sont faîtes par des personnes différentes
  - la vérification demande une connaissance du système et de son processus de développement
  - la validation demande une connaissance du domaine d'application

Qu'est-ce qu'un test logiciel ?

- **Test** : procédure de vérification (partielle) d'un système
  - a pour but d'**identifier** des défaillances, afin que les défauts correspondants puissent être corrigés
  - peut être automatisé, en partie ou totalement
- **Cas de test** : scénario comprenant une entrée (valeurs, actions...) et une propriété à vérifier
- **Suite de tests** : ensemble de cas de tests destinés à vérifier le système ou un de ses composants

Section 4

Développement et qualité logicielle

Les niveaux de tests

Niveaux définis par Boris Beizer pour caractériser la maturité d'une organisation ou d'un développeur face aux test (1990)

Niveau 0 Niveau 1 Niveau 2 Niveau 3 Niveau 4

Niveau 0

- au niveau 0, le testeur pense que “tester c’est débugger”
- C’est FAUX !
- Lancer le programme sur quelques valeurs choisies arbitrairement n’est pas un moyen de construire des logiciels de qualité

Niveau 1

- Au niveau 1, le testeur pense que “tester c’est prouver qu’un programme est correct”
- Encore une fois... FAUX !
- Combien de tests pour prouver que la méthode `int abs(int x)` est correcte ?

Niveau 1

- Au niveau 1, le testeur pense que "tester c'est prouver qu'un programme est correct"
- Encore une fois... FAUX !
- Combien de tests pour prouver que la méthode `int abs(int x)` est correcte ?
  - 2^{32} ($>$ 4 milliards) si `int` est un entier 32 bits
  - 2^{64} ($\approx 100\times$ le nombre de secondes depuis le Big Bang) si `int` est un entier 64 bits
  - une infinité si `int` est un entier multiprécision (ex : entiers Python ou la classe `BigInteger` de Java)
- Il est impossible de tester exhaustivement un programme, les tests ne sont donc **pas une méthode pour prouver qu'un programme est correct**
- *"Tester des programmes peut être un moyen très efficace de révéler des bugs, mais est irrémédiablement inadapté pour en démontrer l'absence."* (E.W. Dijkstra)

Niveau 2

- Au niveau 2, le testeur cherche à **découvrir des défauts logiciels**
- Les tests sont conçus pour maximiser les chances de découvrir des défauts
- Cette vision est correcte et utile, mais incomplète
  - Approche “adversariale” qui peut poser problème dans une équipe
  - Que faire quand on ne trouve pas de défauts ? Quand s’arrêter de tester ?

Niveau 3

- Au niveau 3, l'équipe utilise les tests pour de **réduire le risque** associé à l'utilisation de logiciels
- Prend en compte les limites des tests (pas d'élimination totale du risque)
- Souligne la notion de risque, catégorise les risques par probabilité et gravité
- Permet à toute l'équipe de se concentrer sur le même but

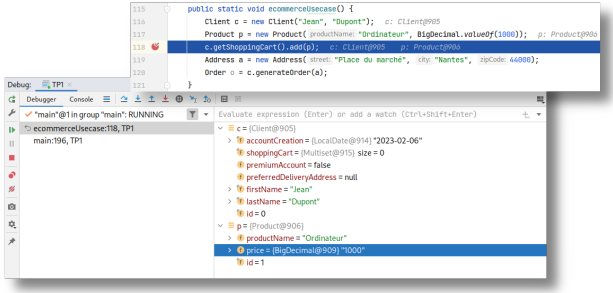
Niveau 4

- Au niveau 4, l'équipe considère que les tests font partie d'une **discipline mentale** permettant de produire des logiciels de **qualité**
- Les tests doivent permettre d'améliorer le processus de développement
  - guider la conception
  - mesurer l'effort d'implémentation restant
  - les testeurs peuvent prendre la direction technique de l'équipe
- Il existe d'autres aspects et techniques permettant d'améliorer la qualité
- Mettre l'accent sur la qualité du logiciel demande des mesures de cette qualité.

Section 5

Les outils de la qualité logicielle

Le débugger



- permet d'arrêter le programme à un endroit choisi puis de l'exécuter pas-à-pas
- permet d'examiner l'état d'exécution
- aide à diagnostiquer une défaillance logicielle : trouver le défaut qui en est la cause
- utile aussi pour se familiariser avec du code nouveau

JUnit

- framework permettant d'automatiser les tests (organisation, exécution, affichage des résultats)
- pas nécessairement limité aux tests unitaires
- fait partie d'une famille de framework de tests (*xUnit*)
  - ce type de framework facilite la mise en place de tests
  - mais pas strictement nécessaire !

Les assertions

```java
static double[] pointwiseDivision(double[] a, double[] b) {
    assert a != null && b != null;
    assert a.length == b.length;

    double[] res = new double[a.length];
    for (int i=0; i < a.length; i++) {
        assert b[i] != 0.0;

        res[i] = a[i] / b[i];
    }
    return res;
}
```

- si la condition est fausse, l'assertion arrête le programme avec une exception
- similaires aux assertions utilisées pour les tests
- il est aussi possible d'utiliser des assertions dans votre code
  - utiles pour documenter les pré-conditions et invariants de votre programme
  - permettent de détecter des erreurs sans attendre une défaillance
  - désactivées par défaut pour ne pas impacter le code en production (pour les exécuter : `java -ea MonProgramme`)

Ressources supplémentaires

- Tutoriel débugger
  - https://blog.jetbrains.com/idea/2020/05/débogueur-basics-in-intellij-idea/ (anglais)
  - https://jetbrains.developpez.com/tutoriel/jetbrains-introduction-debogueur-intellijidea/ (français)
- Tutoriel assertions
  - https://docs.oracle.com/javase/7/docs/technotes/guides/language/assert.html (anglais)
  - https://smeric.developpez.com/java/astuces/assertions/ (français)
- Tutoriel javadoc
  - https://simonandre.developpez.com/tutoriels/java/presentation-javadoc/ (français)