Tests

Bruno Dufour
Université de Montréal
dufour@iro.umontreal.ca

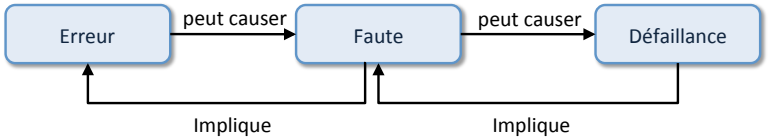
L'importance des tests

- L'erreur est humaine, presque tous les programmes contiennent des erreurs
- Les erreurs sont difficiles à identifier:
  - Grande complexité des logiciels
  - Invisibilité du système développé
  - Pas de principe de continuité
- L'activité de test est essentielle au développement de logiciels de qualité

Définitions

- **Erreur (*error*)**: commise par un développeur
  - Erreur de programmation
  - Erreur de logique
- **Faute** ou **défault (*fault, defect*)**: état interne invalide d'un logiciel après l'activation d'une erreur
- **Défaillance (*failure*)**: manifestation externe d'une faute
  - Observable (ex: le programme dévie de sa spécifications)

Erreurs, Fautes & Défaillances



Développement de logiciels fiables

- Prévention des fautes
  - Méthodes formelles (IFT3911)
  - Réutilisation de code
  - Méthode de développement rigoureuses
- Prévision des fautes
  - Métriques de qualité (IFT3913)
  - Méthodes statistiques
- Identification et correction des fautes
  - Vérification
  - Validation
- Tolérance aux fautes

Tests

- **Test**: exécution d'un programme dans le but de découvrir des erreurs [Myers 1979], non pas « … dans le but de démontrer l'absence d'erreurs »
  - Démontrer l'absence est très difficile ou impossible même pour de petits programmes
  - « Si on tente de démontrer l'absence d'erreurs, on n'en découvre que très peu. Si on tente de démontrer la présence d'erreurs, on en découvre la grande majorité. » [Myers, 1979]
  - Similairement, « … dans le but de démontrer que le programme fait ce qui est demandé » n'est pas suffisant : un programme peut contenir une erreur s'il fait autre chose en plus de ce qui est demandé, par exemple.

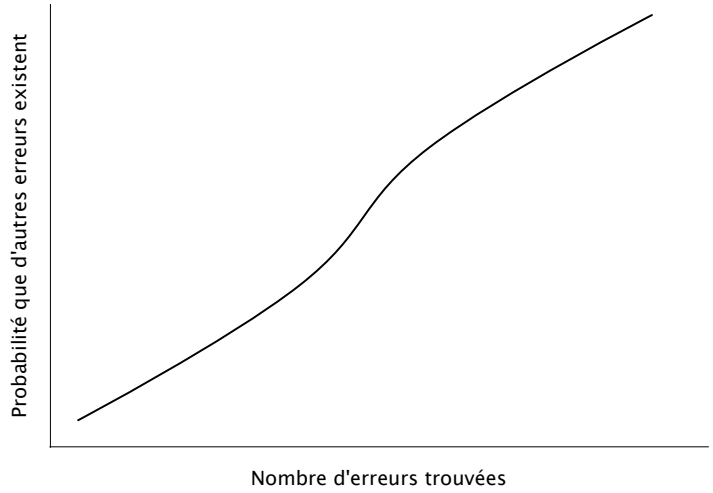
L'importance des tests

- On peut généralement faire l'hypothèse qu'un programme contient des erreurs
- Les tests peuvent être utilisés durant toutes les étapes du développement
  - **Avant** de débuter l'implémentation (TDD)
  - **Durant** le développement
  - **Après** que le logiciel soit complété
- Les tests représentent une partie importante du développement
  - Jusqu'à 33% du budget
  - Jusqu'à 27% du temps de développement

L'art du test

- Bien tester est difficile
  - Tester est un processus destructif
  - Tester une activité très intellectuelle et requiert beaucoup de créativité
- Faire **échouer** le programme est une **réussite** lorsqu'on développe des tests
  - Éventuellement, les tests permettent d'atteindre un niveau de qualité acceptable

Une erreur n'est (presque) jamais seule



Principes de base

- Ne jamais planifier un effort de test sous l'hypothèse tacite qu'aucune erreur ne sera trouvée.
- Un élément nécessaire d'un test est une définition de la sortie ou du résultat
- Un programmeur doit éviter de tester son propre programme
- Les tests doivent être écrits pour les conditions d'entrée qui sont invalides et inattendues, ainsi que pour celles qui sont valides et attendues.
- L'examen d'un programme pour déterminer s'il ne fait pas ce qu'il est censé faire n'est que la moitié de la bataille, l'autre moitié, consiste à déterminer si le programme fait ce qu'il n'est pas censé faire.
- Éviter les tests jetables à moins que le programme soit lui-même vraiment un programme jetable.

L'activité de test

- 2 catégories d'activités :
  - Tests humains : basés sur la lecture du code par un être humain
    - Ex: revues, inspections
  - Tests informatisés : basés sur l'exécution d'un programme par une machine
    - Différents niveaux de granularité
    - Différents niveaux d'automatisation

Inspections et revues

Inspections & revues

- Techniques de test par des humains
  - Une équipe lit ou inspecte le code visuellement
- L'objectif d'une inspection ou revue est l'identification d'erreurs de programmation ou de logique
  - Pas de solutions identifiées en général (test et non débogage)
- Avantages :
  - Des développeurs autres que l'auteur du code sont impliqués
  - Peut être appliqué avant que le code ne soit fonctionnel
  - Permet de réduire le coût de débogage puisque l'emplacement des erreurs est précisément identifié
- Efficacité :
  - Permet de trouver de 30% à 70% des erreurs
  - Est plus efficace que des tests automatisés à détecter certains types d'erreurs

Inspections

- Une inspection de code est un ensemble de procédures et de techniques de détection d'erreurs pour la lecture de code en groupe.
- Une inspection est généralement effectuée en groupe de 4 personnes, et dure entre entre 1 et 2 heures
  - Un modérateur
  - Un programmeur (auteur du code inspecté)
  - Deux autres développeurs

Inspections - Déroulement

- Les participants reçoivent le matériel à l'avance (code, spécification) et se familiarisent avec celui-ci
- Durant l'inspection :
  - L'auteur du code fait la narration de la logique du code, ligne par ligne.
  - Les autres participants posent des questions, et l'équipe tente d'y répondre pour découvrir les erreurs.
  - Le programme est analysé à l'aide d'une liste d'erreurs fréquentes
    - Contient des erreurs qui sont indépendantes du langage utilisé ainsi que des erreurs spécifiques au langage
- À la fin de la session, le programmeur reçoit une liste d'erreurs identifiés. Si plusieurs erreurs sont trouvées, une autre inspection peut être demandée.
  - Cette liste peut servir à améliorer le processus pour les inspections futures

Exemple – List d'erreurs fréquentes

- Erreurs de référence de données
  - Variable non-initialisée
  - Index invalide pour les accès aux tableaux
    - Ne respecte pas les limites, valeur autre qu'un entier
  - Mémoire utilisée n'a pas été préalablement allouée
- Erreurs de déclaration de données
  - Variables non-déclarées
  - Les attributs d'une classe n'ont pas les bonnes propriétés (type, visibilité)
  - Variable ont des noms trop similaires (ex: `volt`, `volts`)
- Erreurs de calcul
  - Calculs impliquant des types mixtes ou incohérents
  - Débordement des valeurs (*overflow*)
  - Division possible par zéro
  - Problème de précédence des opérateurs

Exemple – List d'erreurs fréquentes

- Erreurs de comparaisons
  - Comparaisons entre types mixtes et/ou incohérents
  - Mauvaise comparaison employée (> vs <, >= vs >, == vs equals(), etc.)
  - Dépendance sur le mécanisme évaluation :

```
// x=1, y=0, z=4
if ((y==0 || (x/y)>z) {…}
```

- Erreurs de flot de contrôle
  - Boucle infinie
  - Boucle non-exécutée à cause des conditions initiales
  - Nombre d'itérations qui dépasse ou est inférieur à la limite par 1
  - Décisions non-exhaustives (ex: cas manquants pour `switch`)

Exemple – List d'erreurs fréquentes

- Erreurs d'interface
  - Nombre de paramètres incorrect
  - Incohérence des unités entre arguments et paramètres déclarés
- Erreurs d'entrée / sortie
  - Fichiers non-ouverts avant l'écriture
  - Fichiers non-fermés après l'écriture
  - Exceptions dues aux opérations d'entrée / sortie ne sont pas traitées correctement
  - Fautes d'orthographe / de grammaire dans le texte généré ou affiché
- Autres erreurs
  - Variables non-utilisées
  - Validité des arguments non-testée
  - Fonction absente du programme
  - Présence d'alertes lors de la compilation

Inspection - Avantages

- Permet d'identifier rapidement les erreurs dans les parties du système qui sont les plus à risque
- Permet de recevoir des commentaires sur le style de programmation ou le choix d'algorithmes
- Permet d'apprendre des erreurs des autres programmeurs
- Permet de concentrer l'effort de test automatisé sur les régions problématiques du programme

Revues structurées

- Similaire aux inspections
- Rôles :
  - Modérateur
  - Secrétaire
  - Programmeur (auteur du code en revue)
  - Testeur
- Déroulement :
  - Les participants se préparent avant la revue, comme pour une inspection
  - Le testeur apporte une courte liste de tests qui sont « exécutés » mentalement
    - Ces tests permettent de lancer la discussion
  - Le programmeur reçoit une liste d'erreurs (compilée par le secrétaire), comme pour l'inspection