Qualité du logiciel: Méthodes de test

Matthieu Amiguet

2004 – 2005

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Analyse statique de code

- Étudier le programme source *sans exécution*
- Généralement réalisée avant les tests d'exécution
- Deux moyens courants:
  - Listes de défauts typiques
  - Métriques.

CC BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Checklist – exemple

- Référence aux données
  - Variables non initialisées
  - Pointeurs fantômes (*dangling pointers*)
  - Indices des tableaux hors bornes
  - ...
- Calculs
  - Conversion de types
  - Underflow/Overflow
  - Division par zéro
  - Précédence des opérateurs
  - ...

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Checklist – exemple (2)

- Comparaisons
  - Entre types consistants
  - > et < versus >= et <=
  - = versus ==
  - ...
- Contrôle
  - Terminaison des boucles
  - Une itération en trop/en moins (*off-by-one bug*)
  - Code accessible
  - ...

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Métriques

- Moyen de *calculer* un nombre qui mesure la grandeur/la complexité du code
- La plus connue: nombre de lignes de code
- Variantes:
  - Nombre de classes
  - Nombre de méthodes
  - ...
- Ne tiennent pas compte de la complexité!

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Métrique de McCabe

- Proposée par Thomas McCabe
- Aussi appelée *complexité cyclomatique*
- On représente le flux de contrôle du programme sous forme d'un graphe
- Soient
  - a le nombre d'arcs du graphe
  - n le nombre de noeuds
  - e le nombre de points d'entrées
  - s le nombre de points de sortie

Complexité cyclomatique v

$$v = a - n + i + s.$$

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Métrique de McCabe – suite

- Plus la complexité cyclomatique augmente, plus le programme sera susceptible de contenir des erreurs. . .
- . . . et plus il sera difficile à tester
- On considère qu'une complexité de 10 est raisonnable
- La valeur maximale de la complexité cyclomatique peut être un critère de qualité dans le plan qualité
- Attention cependant: Ce n'est pas une mesure absolue de la complexité!
  - Exemple: le "switch".

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Autres métriques

- Il existe beaucoup d'autres métriques du logiciel
  - Henry & Kafura (liaisons inter-modules)
  - Encombrement (Couplage inter-classes)
  - ...
- Aucune ne s'est révélée être un indicateur vraiment fiable...
- Les métriques sont des outils utiles, mais il faut être conscient de leur limites!

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Attitude du testeur

- Le test doit être vu comme un processus “destructif”: le but est de *mettre en défaut* le logiciel
- Un test ne trouvant aucun bug est un échec!
- Éviter l'approche “montrer que ça marche”!
- Pour ces raisons, il est souvent préférable de confier l'activité de test à une équipe séparée de celle du développement.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Limites théoriques

- Prouver que deux programmes calculent la même fonction est en général *indécidable*
- Il n'existe donc pas de test général pour prouver qu'un programme est exempt d'erreurs
- Tout ce qu'on peut faire, c'est augmenter ses chances de trouver les erreurs.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Catégories de test

- On distingue trois catégories de tests
  - Les tests “boîte blanche”, basés sur la structure du code
  - Les tests “boîte noire”, qui testent les fonctionnalités indépendamment de la manière dont elles sont implémentées
  - Les tests “boîte grise”, combinaison des deux approches précédentes.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

La régression

- Lorsqu'une application est formée de plusieurs modules plus ou moins indépendants et testés séquentiellement, il peut apparaître un phénomène nommé *régression*
    1. On teste le module 1, tout va bien
    2. On teste le module 2 dans lequel on découvre un bug
    3. On corrige le bug du module 2
    4. On termine les tests du module 2
    5. On passe aux tests du module 3. . .
    6. . . . et on manque un bug ajouté/mis en évidence dans le module 1 par la modification du module 2!
- Il faut donc chaque fois retester depuis le module 1
- Ces tests sont généralement automatisés.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Les étapes de test

- Le test peut être divisé en différentes étapes
  - tests unitaires (pendant le développement, souvent par les développeurs)
  - tests d'intégration (pendant le développement, parfois par une équipe séparée)
  - tests de validation (chez le fournisseur, par l'équipe de qualification)
  - Tests de validation (chez le client)
  - Tests de suivi d'exploitation.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Arrêter de tester

- Il est important de définir le critère d'arrêt des tests
- Quelques possibilités:
  - Plus aucune erreur détectée (rare. . .)
  - Taux de couverture atteint
  - Durée de l'effort
  - Nombre d'erreurs découvertes
  - Forme de la courbe du nombre d'erreur trouvées en fonction du temps
  - . . .

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Que faire quand tous les modules ne sont pas présents ?

- Lorsqu'on veut tester un module logiciel, il arrive souvent que des modules connexes ne soient pas disponibles
  - On les remplace alors par une simulation
    - Lorsqu'il remplace un module appelant le module en cours, le simulateur s'appelle un *pilote* (ang. *driver*)
    - Lorsqu'il remplace un module appelé, le simulateur s'appelle un *bouchon* (ang. *stub*).

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Tests “boîte noire”

- Visent à évaluer la réaction du logiciel à certaines entrées sans examiner l’implémentation
- Aussi appelé “Black box testing” et “tests fonctionnels”
- La principale difficulté réside dans la sélection d’un ensemble adéquat de valeurs de test.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Partition en classes d'équivalence

- L'espace des données en entrée du programme est généralement beaucoup trop grand pour être testé intégralement (souvent même infini)
- Pour limiter le nombre de tests, on va essayer de partitionner cet espace en *classes d'équivalences* qui devraient avoir le même comportement
- Pour ce faire, il faudra prendre en compte aussi bien les données valides que non-valides
- Exemples
    - 1..99 peut donner trois classes: [1,99], >99, <1
    - (bleu, vert, noir) peut donner quatre classes dont trois valides
- Ce type de partition s'appelle *analyse aux bornes* (boundary analysis).

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Les graphes “cause à effet”

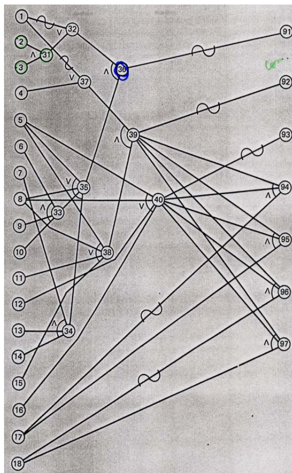
- Servent à systématiser le choix des combinaisons d’entrée

1. Identifier les causes (entrées) et les effets (sorties) du programme à tester
2. Pour chaque effet, identifier les causes qui l’influencent
3. Tracer un graphe “cause à effet” en utilisant
   - des arcs directs
   - des arcs “non”
   - des noeuds “et”
   - des noeuds “ou”

- On pose successivement chaque effet à 1 et on cherche les conditions produisant cet effet.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Exemple



BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Table de test

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	1
2	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1					1	1	1	1	1	1	1				1	1	1
3	0	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1					1	1	1	1	1	1	1				1	1	1
4												1	1	0	0	1	1	1	1	1	1					1	1	1	1	1	1	1				1	1	1
5				0										1				1				1				1				1			1			1		
6	1	1	1	0	1	1	1				1	1			1	1				1				1				1			1			1			1	
7				0				1	1	1			1				1				1				1				1			1			1			1
8				0															1				1				1											
9	1	1	1		1	0	0				0	1			1	1				1				1				1			1			1			1	
10	1	1	1		0	1	0				1	1			1	1				1				1				1			1			1			1	
11												0			0	1				1				1				1			1			1			1	
12																0				1				1				1			1			1			1	
13								1	0	0			1				1				1				1				1			1			1			1
14								0	1	0			1				1				1				1				1			1			1			1
15													0								1				1				1			1			1			1
16																	0				1				1				1			1			1			1
17																		0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
18																		1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
91	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
92	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
93	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
95	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Tests “Boîte blanche”

- Ne s’intéresse plus à l’aspect fonctionnel, “extérieur”, du programme, mais
  - À son flot de contrôle, ou
  - À son flot de données
- Idéalement, on aimerait couvrir tous les chemins possibles
- Mais leur nombre est très grand, parfois infini
- On va donc considérer différents types de *couverture*.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

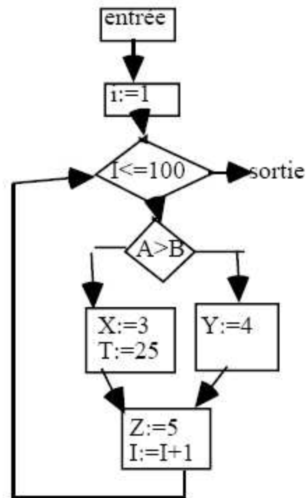
Couvertures naïves

- Toute instruction est exécutée au moins une fois
  - Insuffisant!
  - if x>0 then S endif
  - Couverture complète si x>0, mais le cas x<0 n'est pas testé!
- Toute donnée est utilisée au moins une fois
  - Insuffisant!
  - if cond then a=b+c else a=b-c
  - Couverture complète avec un seul test.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Graphe du flot de contrôle

```
for I := 1 .. 100 loop
  if A>B then
      X:= 3
      T:= 25;
   else
      Y:=4;
   endif;
   Z:=5
end loop;
```



BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Couverture des branches (*branch coverage*)

- Ensemble de test assurant que chaque branche du graphe de contrôle sera parcourue au moins une fois
- Plus complet que la simple exécution de chaque instruction. . .
- . . . Mais toujours insuffisant
  - if a>0 then . . . endif
    if b>0 then . . . endif
  - Couverture des branches avec les données
    - A=1, B=1
    - A=-1, B=-1
  - Mais on a pas testé A=1, B=-1.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Couverture des PLCS

- Portions linéaires de code suivies d'un saut

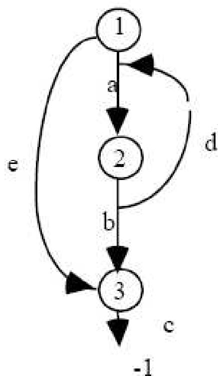
  Angl: *Linear code sequence and jump* (*LCSAJ*)

  - séquence d'instruction entre deux branchements

- La couverture des PLCS est plus complète que celle des branches.

cc BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

PLCS – exemple



PLCS:

- 1 ⟶3
- 1,2,3⟶-1
- 2⟶2
- 1,2⟶2
- 3⟶-1
- 2,3⟶-1.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Autres couvertures basées sur le flot de contrôle

- Couverture des branches essentielles
  - Une branche non-essentielle est exécutée chaque fois qu'une autre branche est exécutée ("passage obligé")
  - Plus "rentable" que les PLCS
- *Structured Path Testing*
  - Travaille sur les chemins ne différant que par le nombre d'itérations
- *Boundary Interior Path Testing*
  - *Boundary path*: chemin sans itération
  - *Interior path*: chemin avec une itération
- ...

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Analyse du flot de données

- On s'intéresse aux différentes utilisations des variables

Définitions

DEF	point de définition d'une variable (x=2)
C-REF	utilisation d'une variable dans un calcul (y=x+2)
P-REF	utilisation d'une variable dans un prédicat (if x==2...)

- On peut alors définir la couverture...
  - des DEF
  - des utilisations (C-REF et P-REF)
  - des P-REF
  - de tous les chemins DEF-REF
  - ...

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Boîte blanche contre boîte noire

- Les tests “boîte blanche” permettent de s’assurer que toutes les parties d’un programme sont testées
- Il faut les remettre à jour à chaque modification de code
- Attention: ne remplacent pas les tests “boîte noire”!
  - if (x+y+z)/3 == x printf ’les 3 nombres sont égaux’
    else printf ’ les 3 nombres sont différents’
  - Couverture avec
    - (x,y,z)=(1,2,3) et
    - x=y=z=2
  - Et pourtant. . .

cc BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Tests “boîte grise”

- Les tests “boîte blanche” demandent une parfaite connaissance du code testé
- Les tests “boîte noire” ne se basent que sur la spécification
- Les situations intermédiaires sont appelés tests “boîte grise”
- Situation courante
  - Le type de test à réaliser est basé sur le code
  - La conception des tests eux-mêmes est basée sur la spécification.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Autres types de tests

- Nous n'avons considéré que les tests les plus généraux
- Suivant les domaines d'application, d'autres types de test peuvent entrer en considération
  - Tests IHM (Interface homme-machine)
  - Tests de configuration
  - Tests d'installation
  - Tests de charge
  - Tests de sécurité.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Tests IHM

- Menus et barres d'outils (fonctionnalité, entrées grisées, raccourcis clavier, . . .)
- Affichage (changement de la taille de la fenêtre, de la résolution d'écran, rafraîchissement, . . .)
- Cohérence de présentation (p.ex. dans les applications web)
- Navigation (effet de la touche de tabulation, . . .)
- . . .

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Tests de configuration

- Les configurations matérielles et logicielles évoluent très vite. . .
- Quelques exemples de problèmes de configuration
    - Environnement 32/64 bits
    - Incompatibilité de bibliothèque
    - Version des pilotes de périphériques
    - Vitesse d'exécution
    - . . .

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

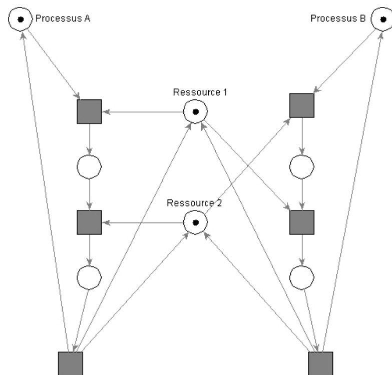
Preuve de programmes

- Dans certains cas “critiques”, on cherche à assurer la validation et la vérification du logiciel par des *preuves*
- Ceci nécessite l’expression de la spécification dans un langage formel approprié
- La spécification peut ensuite être *validée...*
  - au moyen de logiques appropriées (*preuve de programmes*), ou
  - par un parcours exhaustif de l’espace des états possibles (*model checking*)
- Une spécification correcte ne signifie pas que le programme soit correct!

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Spécification formelle

- Divers formalismes sont à disposition
- Exemple: les réseaux de Petri



BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Implementation gap

- Même avec une spécification irréprochable, un programme peut être erroné
- La distance entre la spécification formelle et son implémentation est très difficile à couvrir formellement
  - *Implementation gap*
- Une solution: la logique de Hoare
  - {P} instructions {Q}
  - “Si P est vérifié avant les instructions, Q le sera après”
  - Exemple: {x=1} x++ {x=2}.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch

Références

- Marnie L. Hutcheson, “Software Testing Fundamentals”, Wiley, 2003
- Maurice Rozenberg, “Test logiciel”, Eyrolles, 1998
- `http://www.essi.fr/~hugues/GL/chapitre9.pdf`.

BY-NC-SA distribué sous licence creative common | détails sur www.matthieuamiguet.ch