IUT Nantes
Pôle Sciences et technologie

Nantes Université

R2.03 - Qualité de développement 1
Automatisation des tests

CM11 - Diagnostic

Jean-Marie Mottu
Nantes Université

Diagnostic : analyse de l'échec des cas de test

- Le diagnostic exploite
  - L'oracle
    - Les vérifications (multiples) effectuées par l'oracle donnent des indications sur la différence entre
      - le comportement attendu et
      - le comportement obtenu
  - La trace
    - Il s'agit de remonter aux sources de l'échec des cas de test
    - Top-down
      - Depuis l'entrée du programme vers la faute causant la divergence
    - Bottom-up
      - Depuis l'erreur (ce que l'oracle a observé)
      - Vers la faute (qu'il faudra corriger)
    - Souvent mixte

Analyse de trace

- Produire une trace est
  - Statique : technique de « model checking »
    - Le code est modélisé puis ce modèle est analysé
  - Dynamique : les tests sont exécutés pour produire la trace

- L'analyse de la trace dynamique est
  - Statique : la trace est stockée pour être analysée à postériori
    - Stockée dans un modèle, dans des logs (plus ou moins structurés)
  - Dynamique : pendant l'exécution on peut observer le comportement du cas de test

Diagnostic
Trace + Oracle

- Trace et oracle vont conjointement permettre d'identifier
  - Les divergences de valeurs
    - Quand une variable n'a pas la valeur attendue
    - Quand elle est utilisée
      - En écriture : quand elle est définie (passage de paramètre, affectation)
      - En lecture : quand elle est utilisée
        - Dans des conditionnelles
        - Dans des affectations
  - Les divergences dans le flot d'exécution
    - Quand l'exécution vient à passer dans des lignes de code où, elle ne devrait pas

Approche top-down

- Exploitation du mode debug d'un IDE
  - Mise en place de point d'arrêt
    - Nécessite d'avoir localisé la source du problème
      - Basé sur les informations renvoyées par l'oracle
        - Quelle assertion a échoué
        - Quelle variable est impliquée
        - Où est manipulée la variable
  - Avancement pas à pas
    - Pas :
      - Par instruction
      - Par appel
    - Après chaque pas, on connait l'état complet du système (variables, attributs)
    - Nécessite de savoir à quoi va ressembler la divergence
      - Difficile quand l'oracle n'a pu l'observer que bien plus tard

Approche Bottom-up
Remontée de trace

- Stocker la trace
- Analyser la trace depuis l'assertion ayant échoué, jusqu'au point de divergence
  - Difficulté de savoir quel est le point de divergence quand la variable impliquée est écrite plusieurs fois

Approche Bottom-up
Recoupement de traces d'exécution

- Exploitation de plusieurs traces pour identifier une zone potentielle où serait la faute
  - Réduit la zone de recherche

- Spectrum-Based Fault Localization (SBFL) techniques
  - Principe
    - identifier pour chaque cas de test les instructions couvertes
    - Combiner ces couvertures pour ordonner les instructions potentiellement fausses
      - Plus une instruction est couverte par des tests échouant et moins elle est couverte par des tests passant, alors plus elle a de potentiel d'être fausse
      - Différents algorithmes de combinaison de couverture

Algorithme de Jones et al. (diapo Le Traon, Baudry)

- Ordonner les instructions de la plus suspecte à la moins suspecte.
- Exemple :

```
pow(x, y:integer) : float
  local i, p : integer
  i := 0;                    {1}
  Result := 1;               {2}
  if y<0 then p := -x;       {3}
  else p := y;               {4}
  while i<p do
    Result := Result * x;    {5}
    i := i + 1;              {6}
  done
  if y<0 then
    Result := 1/Result;      {7}
end
```

- Fonction puissance
- Une faute a été introduite en {3}

  (l'instruction correcte serait p:=-y)

Algorithme de Jones et al. (diapo Le Traon, Baudry)

- Ordonner les instructions de la plus suspecte à la moins suspecte.
- Exemple :

4 cas de test

		1	2	3	4
		x=2 y=4	x=-2 y=0	x=2 y=-4	x=-3 y=-3
pow(x, y:integer) : float					
local i, p : integer					
i := 0;	{1}	1	1	1	1
Result := 1;	{2}	1	1	1	1
if y<0 **then p := -x**;	{3}	0	0	1	1
else p := y;	{4}	1	1	0	0
while i<p **do**					
Result := Result * x;	{5}	1	0	0	1
i := i + 1;	{6}	1	0	0	1
done					
if y<0 **then**					
Result := 1/Result;	{7}	0	0	1	1
end					
		P	**P**	**F**	**P**

Matrice de diagnostic

Algorithme de Jones et al. (diapo Le Traon, Baudry)

		1	2	3	4	Résultats du diagnostic		
		x=2 y=4	x=-2 y=0	x=2 y=-4	x=-3 y=-3	%Passed	%Failed	**Rang**
pow(x, y:integer) : float								
local i, p : integer								
i := 0;	{1}	1	1	1	1	100%	100%	**3**
Result := 1;	{2}	1	1	1	1	100%	100%	**4**
if y<0 **then p := -x**;	{3}	0	0	1	1	33%	100%	**1**
else p := y;	{4}	1	1	0	0	66%	0%	**5**
while i<p **do**								
Result := Result * x;	{5}	1	0	0	1	66%	0%	**6**
i := i + 1;	{6}	1	0	0	1	66%	0%	**7**
done								
if y<0 **then**								
Result := 1/Result;	{7}	0	0	1	1	33%	100%	**2**
end								
		P	**P**	**F**	**P**			

Diagnostic et cas de test

- Les cas de test de nouveau au centre
  - Fournisse explicitement l'oracle
  - Fournisse la trace par la sollicitation du programme avec la donnée de test
- Dépendance entre les données test, les oracles et le diagnostic

- Plus il y aura de cas de test et plus le diagnostic est facile
  - Contraire aux pratiques du test qui minimisent le nombre de cas de test pour atteindre un critère de test donné
  - Itératif
    - Produire des cas de test pour identifier des erreurs
    - Les compléter avec des cas de test pour identifier les fautes

Le cycle de test dynamique