Teste de Mutação

Prof. Otávio Lemos (UNIFESP)
Prof. Fabiano Ferrari (UFSCar)

Teste Baseado em Defeitos

- Vimos Teste Funcional e Teste Estrutural
- Critérios: subdividem domínio de entrada em subdomínios
- Problema: Não existe relação direta entre subdomínio e probabilidade em encontrar defeitos
- Exemplo: dado um caminho requerido, no qual existe um defeito $\rightarrow$ podem existir CTs no subdomínio correspondente que levem e que não levem o programa a falhar.
  - se apenas um CT do segundo grupo for executado, o defeito não será revelado

Teste Baseado em Defeitos

- Técnica de teste baseada em defeitos: utilização dos defeitos típicos do processo de desenvolvimento de software
- Casos de teste exploram esses defeitos
- Ideia: "injetar" defeitos no programa e verificar se casos de teste são capazes de descobri-los

- Error Seeding [1]
- Análise de Mutantes (Teste de Unidade) [2]
- Mutação de Interface (Teste de Integração) [3]
- vários outros relacionados ao teste de mutação desde então...

[1] Mills, H. D. On the statistical validation of computer programs. Technical Report FSC-72-6015, IBM Federal Systems Division, Gaithersburg, MD - USA, 1972.

[2] DeMillo, R. A.; Lipton, R. J.; Sayward, F. G. Hints on test data selection: Help for the practicing programmer. IEEE Computer, v. 11, n. 4, p. 34-43, 1978.

[3] Delamaro, M. E. Mutação de Interface: Um critério de adequação interprocedimental para o teste de integração. Tese de Doutoramento, IFSC/USP, 1997.

Teoria

- Howden [1] – correção (*correctness*): *Um programa P é correto com relação a uma função F se P computa F.*
- Para provar isso, conjunto de CTs deve ser confiável (*reliable*):
  - *Um conjunto de teste T é confiável para um programa P e uma função F, dado que F e P coincidem em T, se e somente se P computa F. Se P não computa F, então T deve conter um CT t tal que $F(t) \neq P(t)$.*
  - DeMillo et al. [2] – é simples mostrar que, para qualquer programa existe um conjunto de CTs confiável; entretanto, não existe procedimento efetivo para gerar CTs confiáveis

[1] Howden, W. E. Weak mutation testing and completeness of test sets. IEEE Transactions on Software Engineering, v. 8, n. 4, p. 371-379, 1982.

[2] DeMillo, R. A.; Lipton, R. J.; Sayward, F. G. Hints on test data selection: Help for the practicing programmer. IEEE Computer, v. 11, n. 4, p. 34-43, 1978.

Teoria

- Para obter definição mais realista, outra abordagem para correção: eliminação
- $\Phi(P)$ – "vizinhança" de P = conjunto de programas alternativos, que depende de P, e do qual P é membro
- Mostrar que todos os elementos de $\Phi(P)$, exceto P e seus equivalentes, são incorretos (pelo menos um CT em T falha)
  - Um conjunto de teste T é adequado para P em relação a Φ se para cada programa $Q \in \Phi$, ou $Q \equiv P$ ou $Q \neq P$ em pelo menos um CT

Teoria

- Se $\Phi(P)$ é muito grande (ou infinito), impraticável
- Assim, escolher um $\Phi(P)$ pequeno, abandonando-se o desejo de provar a correção absoluta do programa – correção *relativa*

Definições

- A ideia básica do critério AM é a **hipótese do programador competente**: um programador competente escreve programas corretos ou próximos do correto.
  - Assumindo a validade desta hipótese: defeitos são introduzidos no programa por meio de pequenos desvios sintáticos que fazem com que a execução do produto leve a um comportamento incorreto.
  - O Teste de Mutação realiza pequenas alterações sintáticas no produto em teste.
  - O testador deve construir um conjunto de testes que mostre que tais modificações criaram produtos incorretos.

Definições

- Uma segunda hipótese explorada pelo Teste de Mutação é o **efeito de acoplamento**.
  - Defeitos complexos são uma composição de defeitos simples.
  - Conjuntos de teste que revelam defeitos simples são também capazes de revelar defeitos complexos.
  - Uma única alteração sintática é aplicada ao programa P em teste, ou seja, cada mutante tem uma única diferença sintática em relação ao programa original.

Definições

Dado um produto que se deseja testar P e um conjunto de teste T para ser avaliado. Os passos de aplicação são:

1. Execução do produto original
   - P é executado com T.
   - Se ocorrer uma falha, o teste termina.
   - Se nenhuma falha ocorrer, P ainda pode conter defeitos que T não foi capaz de revelar.
2. Geração dos mutantes
   - P é submetido a um conjunto de **operadores de mutação** [1] que transformam P em $m_1, m_2, \ldots, m_n$, denominados **mutantes** de P.

[1] Operadores de mutação são regras que representam enganos frequentes ou desvios sintáticos relacionados com uma determinada linguagem de programação.

Definições

4. Execução dos mutantes
   - Os mutantes são executados com o mesmo conjunto de teste T.
   - **Mutantes mortos** - resultados diferentes de P.
   - **Mutantes vivos** - resultados idênticos ao de P.
   - O ideal é ter apenas mutantes mortos, indicando que o conjunto de teste T é adequado para o teste de P em relação ao Teste de Mutação.
5. Análise dos mutantes vivos
   - Mutantes vivos são analisados para identificar possível equivalência em relação a P, ou expor uma fraqueza do conjunto de teste T.

Definições

Análise dos mutantes vivos

- Mutante equivalente
  - Um mutante m é dito equivalente a P se para qualquer dado de entrada $d \in D$, o comportamento de m é igual ao de P: $m(d) = P(d)$.
- Mutante revelador de defeito (*fault-reveling*)
  - Um mutante é dito ser revelador de defeito se para algum caso de teste t, tal que $P(t) \neq m(t)$, conclui-se que $P(t)$ não está de acordo com a especificação, ou seja, a presença de um defeito em P foi revelada.

Definições

- Escore de mutação
  - Medida objetiva para avaliar a adequação de T em relação ao Teste de Mutação.

$$ms(P, T) = \frac{DM(P, T)}{M(P) - EM(P)}$$

Onde

$DM(P, T)$: número de mutantes mortos por T;

$M(P)$: total de mutantes gerados;

$EM(P)$: número de mutantes equivalentes a P.

Definições

- Passos básicos da atividade:
  1. Execução do programa em teste
  2. Geração dos mutantes
  3. Execução dos mutantes
  4. Análise dos mutantes

Aplicação

- Para geração dos mutantes: operadores de mutação
- Exemplo: operador que retira um comando de P
- Um operador pode gerar mais de um mutante
- Mutante de defeito induzido (exemplo acima) e mutantes instrumentados (com funções armadilha) – não modelam algum tipo de defeito; apenas para garantir alguma propriedade nos casos de teste

Exemplo

```
void main() {
int x, y, pow;
float z, ans;
    scanf("%d %d", &x, &y);
    if (y >= 0)
        pow = y;
    else
        pow = -y;
    z = 1;
    while (pow -- > 0)
        z = z*x;
    if (y < 0)
        z = 1 / z;
    ans = z + 1;
    printf("%-5.2f", ans);
}
```

Exemplo

- Operadores [1]:
  - SSDL: eliminação de comandos
  - ORRN: troca de operador relacional
  - STRI: armadilha em condição de comando *if*; e
  - Vsrr: troca de variáveis escalares.

[1] Agrawal, H.; et al.:Design of mutant operators for the C programming language. Technical Report SERC-TR41-P, Software Engineering Research Center, Purdue University, West Lafayette/IN - USA, 1989.

Exemplo

Mutantes para o Operador SSDL

1	scanf("%d %d", &x, &y)

2	if (y >= 0)
	pow = y;
	else
	pow = -y;

3	pow = y
4	pow = -y
5	z = 1
6	while (pow-- > 0) z = z * x;
7	z = z * x
8	if (y < 0) z = 1 / z
9	z = 1 / z
10	ans = z + 1
11	printf("%-5.2f", ans)

Exemplo

Mutantes para o Operador ORRN

12	if (y $>$ 0)
13	if (y $<$ 0)
14	if (y $==$ 0)
15	if (y $<=$ 0)
16	if (y $!=$ 0)
17	while (pow-- $<$ 0)
18	while (pow-- $==$ 0)
19	while (pow-- $>=$ 0)
20	while (pow-- $<=$ 0)
21	while (pow-- $!=$ 0)
22	if (y $>$ 0)
23	if (y $==$ 0)
24	if (y $>=$ 0)
25	if (y $<=$ 0)
26	if (y $!=$ 0)

Exemplo

Mutantes para o Operador STRI

27	if (armadilha_se_verdadeiro(y >= 0))
28	if (armadilha_se_falso(y >= 0))
29	if (armadilha_se_verdadeiro(y < 0))
30	if (armadilha_se_falso(y < 0))

Exemplo

Mutantes para o Operador Vsrr

31	scanf("%d %d", \&y, \&y)	*32*	scanf("%d %d", \&pow, \&y)
33	scanf("%d %d", \&z, \&y)	*34*	scanf("%d %d", \&ans, \&y)
35	scanf("%d %d", \&x, \&x)	*36*	scanf("%d %d", \&x, \&pow;
37	scanf("%d %d", \&x, \&z)	*38*	scanf("%d %d", \&x, \&ans)
39	if (x >= 0)	*40*	if (pow >= 0)
41	if (z >= 0)	*42*	if (ans >= 0)
43	pow = x	*44*	pow = pow
45	pow = z	*46*	pow = ans
47	x = y	*48*	y = y
49	z = y	*50*	ans = y
51	pow = -x	*52*	pow = -pow
53	pow = -z	*54*	pow = -ans
55	x = -y	*56*	y = -y
57	z = -y	*58*	ans = -y
59	x = 1	*60*	y = 1
61	pow = 1	*62*	ans = 1
63	while (x-- > 0)	*64*	while (y-- > 0)
...	*...*	*...*	*...*

Exercício

Avaliar a cobertura do conjunto de teste do programa *Cadeia de Caracteres* (implementado na aula 3).

- Considerar apenas um subconjunto dos operadores de mutação de unidade (conjunto essencial).

Operador	**Descrição**
u-SSDL	Remove um comando do programa.
u-ORRN	Substitui operador relacional.
u-VTWD	Substitui referência a um escalar pelo seu predecessor e sucessor.
u-Ccsr	Substitui referência a um escalar por uma constante.
u-SWDD	Substitui um `while` por um `do-while`.
u-SMTC	Interrompe a execução de um laço após duas execuções.
u-Cccr	Substitui uma constante por outra.
u-VDTR	Força cada referência a um escalar a ser: negativo, positivo e zero.

Desvantagens

- Principal problema é o grande número de mutantes gerados.
  - Mutantes precisam ser compilados e executados.
  - Mutantes vivos precisam ser analisados devido à possível equivalência.
- Requer bom conhecimento da implementação do programa para facilitar a análise de mutantes vivos.

Vantagens

- É fácil de ser estendido para qualquer produto "executável" seja no nível de especificação ou implementação.
- É um dos critérios de teste mais eficazes em detectar defeitos.

Ferramentas

- Existem algumas ferramentas que apóiam a aplicação do Teste de Mutação.
  - Proteum (http://ccsl.icmc.usp.br/pt-br/projects/proteum) para C.
  - MuJava (https://cs.gmu.edu/~offutt/mujava/) para Java.
  - Jester (http://jester.sourceforge.net/) para Java.
  - Pester (http://jester.sourceforge.net/) para Python.
  - Nester (http://nester.sourceforge.net/) para C#
  - Mothra (https://cs.gmu.edu/~offutt/rsrch/mut.html#MOTHRA) para Fortran.
  - Insure++ (http://www.parasoft.com/product/insure/) para C e C++.
  - PIT (http://pitest.org/) para Java.
  - Outras podem ser consultadas em http://www.mutationtest.net/.