Hints on Test Data Selection: Help for the Practicing Programmer

Richard A. DeMillo
Richard J. Lipton
Frederick G. Sayward

Sumário

- Introdução
- O efeito de acoplamento
- Mutação de programa
- Exemplo de mutação
- Conclusões

Introdução

- Uma das maiores restrições impostas a programadores é o tempo para testes, que muitas vezes é pequeno
  - Dessa forma, o programador precisa de formas rápidas e baratas de realizar testes
  - Metodologias do estado-da-arte são, muitas vezes, extremamente custosas e inviáveis
- Os autores apresentam uma forma mais viável de se realizar testes de software de maneira adequada e em um curto intervalo de tempo

Introdução

- Muitas vezes, companhias de software justificam o curto espaço de tempo reservado para testes
  - “a maior parte dos erros são descobertos nos períodos iniciais, e poucos erros vão sendo descobertos em análises mais duradouras, o que torna uma análise a longo prazo algo inviável”
  - Dessa forma, é consenso reduzir o tempo de testes, favorecendo outros estágios de desenvolvimento

Introdução

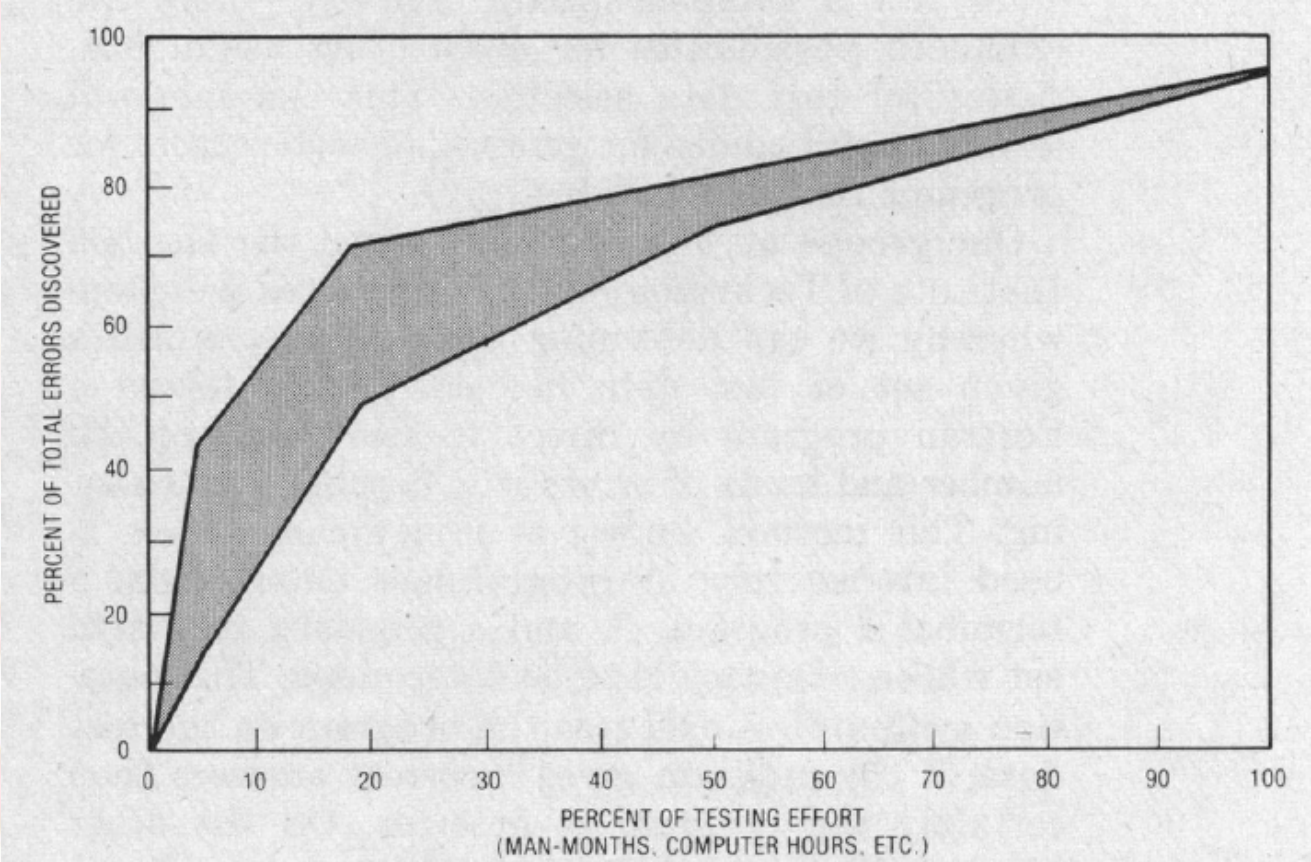

Figure 1. More programming errors are found in the early part of the test cycle then in the final part.

O efeito de acoplamento

- Segundo o artigo, programadores tem uma grande vantagem que é muito pouco explorada: eles criam programas que são *quase* corretos
- Programadores não criam programas aleatória-mente, sempre questionam-se sobre a corretude
- O que eles tem a sua disposição?
  - Idéia aproximada dos tipos de erros mais comuns
  - Habilidade e possibilidade de examinar

O efeito de acoplamento

- Classificação de erros (Goodenough-Gerhart)
  - 1) Falha na satisfação de especificações, devido a erros de implementação
  - 2) Falha na escrita de especificações que corretamente representariam um projeto
  - 3) Falha no entendimento de um requisito
  - 4) Falha na satisfação de um requisito
- Segundo os autores, erros são sempre refletidos em programas como:
  - Falta de caminhos de controle
  - Seleção inapropriada de caminhos
  - Ações inapropriadas ou falta das mesmas

O efeito de acoplamento

- Se programas estiverem próximos de estarem corretos, então os erros devem ser detectáveis como pequenos desvios da funcionalidade pretendida
- E. A Youngs, em sua tese de PhD, analisou 1258 erros simples em programas
- Qual a relação entre os erros simples descobertos por Young e a classificação de erros de Goodenough-Gerhart?
  - O fato de programas estarem *próximos* de sua corretude, mostra que a alta frequência de erros simples é bastante importante

O efeito de acoplamento

- O efeito de acoplamento: dados de teste que distinguem todos programas que diferem de um programa correto apenas por erros simples são tão sensíveis que esses dados também implicitamente distinguem erros mais complexos
  - Em outras palavras, erros complexos estão amarrados, ou acoplados, a erros mais simples
- Segundo o artigo, testes sistêmicos devem ser realizados em busca de erros simples, que apresentarão erros mais complexos, pelo efeito de acoplamento

O efeito de acoplamento

- Análise de caminho
  - O objetivo dos dados de teste é dirigir um programa por todos os seus caminhos de controle
  - Entretanto, apenas passar por todos os caminhos de um programa não é suficiente
  - As sentenças que estabelecem testes lógicos, por exemplo, devem ser analisadas e desmembradas em mais caminhos

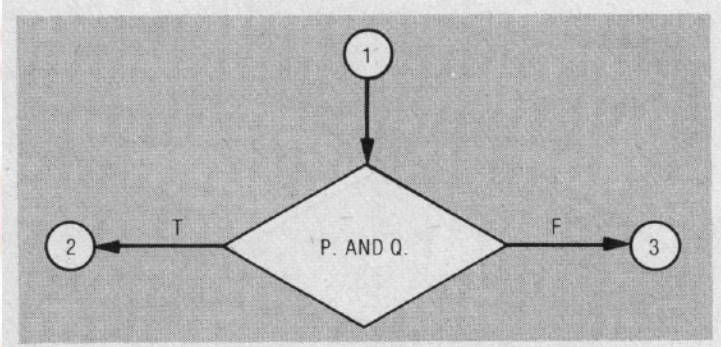

Figure 2. Sample program segment with two paths.

O efeito de acoplamento

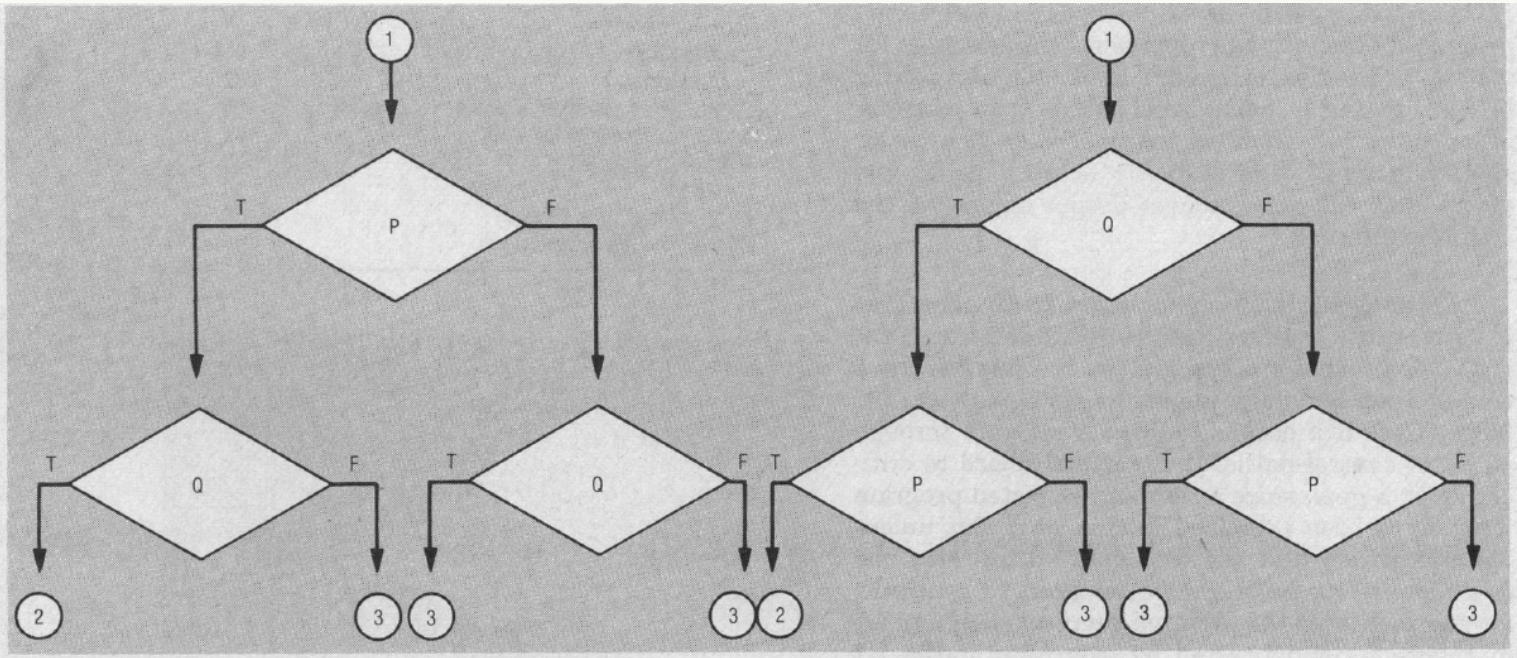

Figure 3. Eight paths may be required for an adequate test.

O efeito de acoplamento

- Segundo o artigo, é mais interessante a escolha de dados de teste que foquem no acoplamento de erros
  - Não apenas dados gerados aleatóriamente, nem dados que foquem apenas na análise de caminhos
- Além dos dados, o que mais pode ser feito?

Mutação de programa

- Sistema onde pode ser determinado o quanto um conjunto de dados (ou testes) é significativo para o teste de um programa
- Medição direta do número e tipos de erro que o sistema descobre em um dado programa
- O método é conhecido como mutação de programa
  - É dado como entrada um programa P
  - E dados de teste T, os quais sua adequação está para ser determinada
- O sistema de mutação inicialmente executa o programa com os dados de entrada
  - Se o sistema reportar erros, certamente o programa contém erros

Mutação de programa

- Senão, ele ainda pode apresentar erros, contudo os dados de teste podem não ser significativos para a detecção dos erros no programa
- No último caso, o sistema de mutação cria diferentes versões do programa de entrada P
  - As mutações de P diferem apenas na presença de erros simples (por exemplo, uma comparação <= é trocada por ==)
- Dessa forma, para os mesmos dados de entrada, podem ocorrer apenas duas situações:
  - A execução de P diverge de suas mutações
  - A execução de P é idêntica em suas mutações

Mutação de programa

- No primeiro caso, a mutação é dita como “morta”, ou seja, o erro introduzido na mutação é distinguido pelos dados de entrada
- No segundo caso, a mutação é dita como “viva”, e uma mutação pode viver por dois motivos
  - Os dados de entrada podem não serem sensíveis o suficiente para diferenciar a mutação de P
  - A mutação e P são programas equivalentes, e nenhum teste irá diferenciá-los (na verdade isso não é um erro)

Mutação de programa

- Testes são ditos “adequados” quando não sobram mutantes, ou sobram mutantes equivalentes a P
- A informação retornada pelo sistema de mutação pode ser utilizada pelo programador
  - O programador observa uma resposta negativa do sistema como uma “pergunta”
  - “para os dados de entrada, não importa se a sentença é uma igualdade ou uma inigualdade, isso está correto?”
  - Dessa forma, o programador utiliza o sistema como guia para melhorar a qualidade dos dados de entrada
    - Dados que expõem os erros do programa

Exemplo de mutação

- Algoritmo MAX

```
      SUBROUTINE MAX (A,N,R)
      INTEGER A(N),I,N,R
    1 R=1
    2 DO 3 I=2,N,1
    3 IF (A(I).GT.A(R))R=I
      RETURN
      END
```

Table 2. Three vectors constitute the initial set of test data.

	A(1)	A(2)	A(3)
data 1	1	2	3
data 2	1	3	2
data 3	3	1	2

Exemplo de mutação

- O conjunto de dados de entrada é adequado?
  - Nenhum dos vetores distingue >= de > na sentença IF
  - Da mesma forma, todos os vetores distinguem erros simples em constantes, com excessão de inicializar o loop DO em “1” ao invés de “2”
  - Todos os erros simples em variáveis são distinguidos da mesma forma, com excessão de erros na sentença IF que substituem “A(I)” por “I” ou por “A(R)”
  - Para esse conjunto, os seguintes mutantes apresentariam o mesmo resultado

Exemplo de mutação

```
   SUBROUTINE MAX (A,N,R)
   INTEGER A(N),I,N,R
1  R=1
2  DO 3 I = 1,N,1
3  IF(A(I).GT.A(R))R=1
   RETURN
   END
```

```
   SUBROUTINE MAX (A,N,R)
   INTEGER A(N),I,N,R
1  R=1
2  DO 3 I=2,N,1
3  IF(I.GT.A(R))R = 1
   RETURN
   END
```

```
   SUBROUTINE MAX (A,N,R)
   INTEGER A(N),I,N,R
1  R=1
2  DO 3 I=2,N,1
3  IF(A(I).GE.A(R))R = 1
   RETURN
   END
```

```
   SUBROUTINE MAX (A,N,R)
   INTEGER A(N),I,N,R
1  R=1
2  DO 3 I=2,N,1
3  IF(A(R).GT.A(R))R = 1
   RETURN
   END
```

	A(1)	A(2)	A(3)
data 4	2	2	1

Exemplo de mutação

- O conjunto de dados não era adequado, por isso foi criado um novo vetor, “data 4”
- Com o novo vetor, a substituiçao de >= por > apresentou resultados errados, então, mutantes resultantes de simples erros relacionais estão “mortos”
- “data 4” também distingue dois erros em “A(I)”
- Sobra apenas um único mutante, que difere na inicialização do laço DO
  - Esse mutante é apenas uma versão menos otimizada do programa original
- Como o novo conjunto de dados deixa “vivos” apenas mutantes equivalentes a MAX, esse é considerado adequado

Conclusões

- A utilização sistemática de dados que distinguem erros de uma determinada classe de erros também ajuda na geração de testes de programas similares
- Para os casos abordados no artigo (programas que manipulam vetores)
  - Incluir casos onde o índice está fora dos limites
  - Incluir casos onde valores são negativos
  - Incluir casos onde valores são repetidos
  - Incluir casos extremos, onde existem apenas um único valor repetido em todo vetor
- No artigo, os autores comentam que nada substitui a experiência adquirida, e esta deve ser utilizada pelos programadores

Conclusões

- Além disso, programadores devem utililizar sua experiência na busca de erros simples, que podem estar ligados a erros mais complexos, por efeito de acoplamento