UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

ISABEL CRISTINA SARTI SPROGIS

Caracterização de Mutantes Resistentes

1. RESUMO

Neste projeto de Iniciação Científica foi realizada a investigação e tentativa de determinar a existência de Mutantes Resistentes dentro da técnica de teste de software baseada em falhas proposta por DeMillo; Buddy; Lipton; Sayward (1978), conhecida como Teste de Mutação.

2. INTRODUÇÃO

O Teste Unitário, também chamado de teste de componentes, é o teste das unidades mínimas do software de forma independente. Na orientação a objetos tais unidades individuais são métodos e objetos das classes. O teste unitário exercita tais unidades através de chamadas para os métodos passando diferentes dados de entrada e comparando o resultado obtido com o que é esperado segundo a especificação do software, validando assim o comportamento dos componentes do software.

O Teste de Mutação (ou Análise de Mutantes) é uma técnica de teste Baseada em Erros - na qual os requisitos do teste provém do conhecimento de principais erros cometidos no desenvolvimento - que necessita de um conjunto de testes unitários para verificar a qualidade do próprio conjunto de casos de teste.

A análise de mutantes proposta por DeMillo et al. (1978) funciona aplicando-se operadores de mutação em um programa. Tais operações inserem uma modificação

sintática no programa, criando para cada modificação uma nova versão do programa original chamada mutante. Cada mutante do programa contém apenas uma única modificação sintática em relação ao programa original.

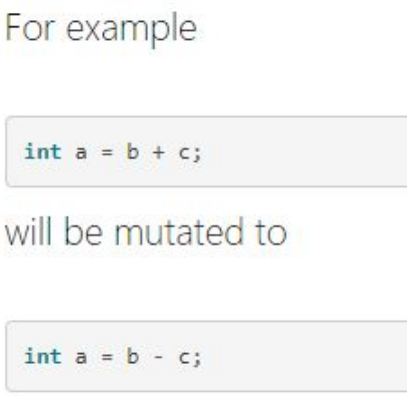

Figura 1: Imagem retirada do site da ferramenta PITEST (https://pitest.org/quickstart/mutators/) que demonstra a aplicação do operador de mutação MATH.

Após submeter novamente os casos de testes no programa com a alteração (dito mutante) espera-se que o resultado do teste nesse mutante seja diferente do programa original, pois a modificação sintática presente no mutante deve produzir uma alteração semântica, que exercitada pelo caso de teste produz um resultado diferente do programa original.

A análise de mutantes pode ser aplicada para avaliar a qualidade de um conjunto de teste. Se executarmos o programa original gerando todos os mutantes e aplicando todo o conjunto de teste, o resultado seria que vários mutantes produziriam resultados diferentes do programa original (com uma mesma entrada) e esses mutantes são considerados mortos

Porém, se nenhum caso de teste consegue mostrar essa diferença para um mutante ele é considerado sobrevivente e temos dois possíveis resultados: Ou o mutante introduzido é equivalente semanticamente mesmo com a variação sintática ou os casos de testes não estão adequados. Há grande esforço gasto para determinar se mutantes são equivalentes, pois eles mantêm a semântica do programa inalterada

atuando como falsos positivos e portanto, não conseguem ser detectados por nenhum conjunto de testes, não importa o quão bom sejam.

3. OBJETIVOS

Pressupõe-se que existam mutantes que denominaremos como resistentes quando retiram-se os equivalentes dos que sobreviveram. Chamados de resistentes pois só são eliminados com dados não triviais criados especificamente para matá-los, o que os tornam bons identificadores de um conjunto de testes de qualidade portanto. O objetivo é encontrar casos de testes melhores com tais dados não triviais a partir desses mutantes como critério.

4. PESQUISA

Para estudar a técnica de testes de mutação foi necessária a escolha de uma ferramenta que pudesse automaticamente propagar as mutações baseadas em conjuntos de operadores diferentes no código de um programa escolhido. Após pesquisar diversas ferramentas foi escolhido o PITEST como ferramenta de mutação por sua facilidade de ver as mutações aplicadas no código e o resultado de sua sobrevivência quando os testes são executados novamente através de relatórios HTML e XML gerados.

E como objeto do estudo escolheu-se utilizar uma biblioteca open source de funções em Java para implementar o Calendário Persa. O diferencial dessa biblioteca é conter 1.231 linhas de código dividida entre 5 classes e um conjunto de 132 testes unitários escritos em JUnit que abrangem todos os métodos de cada classe, o qual tornou o conjunto base de testes unitário do estudo.

A Pesquisa começa por aplicar a ferramenta de mutação na biblioteca e submeter, nas diversas mutações feitas, apenas um único caso de testes unitários. Dessa forma podemos analisar por todo o conjunto de mutações sofridas quais delas

sobrevivem e para quais casos de testes. Cada relatório HTML e XML gerado é salvo com o nome do teste unitário que submeteu-se ao programa pós mutações.

Com esses relatórios pudemos montar uma tabela final contendo 30 mutações com apenas os mutantes que sobreviveram dividida em 7 colunas: (1) Quantas vezes aquela mutação sobreviveu, (2) a classe Java que pertence a mutação, (3) a linha de código na classe respectiva, (3) uma cópia linha de código para referência, (4) a mutação sofrida, (5) o operador de mutação aplicado, (6) Comentários da análise feita e (7) Classificação final como M - Mutante ou E - Equivalente.

Para cada um dos 30 mutante sobrevivente da tabela final foram feitas tentativas de criar testes unitários que exercitem a diferença do programa original com o mutante para demonstrar que tal mutante só pode ser morto apenas com entradas de dados não triviais. Se um novo caso de testes pudesse ser criado com tais dados era então utilizada novamente a ferramenta PITEST para verificar o resultado deste novo caso de teste, a morte deste mutante com esse novo caso, e classificar a mutação como sobrevivente e não equivalente.

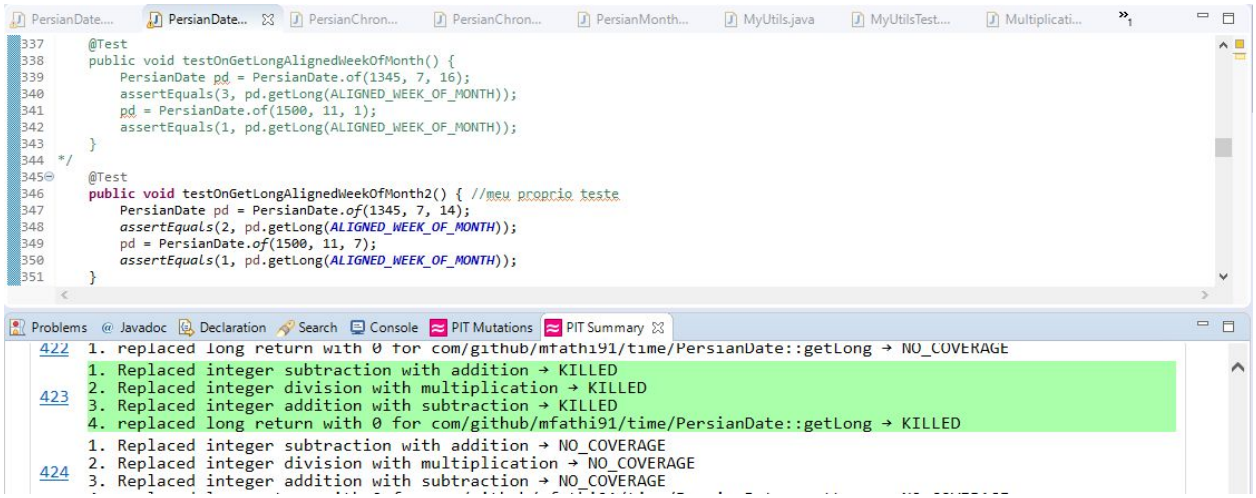

Figura 2: Diferença entre o caso de teste original, o novo caso que utiliza ano igual datas com dias múltiplos de 7 e o relatório HTML gerado com a morte da mutação replaced integer subtraction with addition da linha 423 do método getLong da classe PersianDate.

5. RESULTADO

Levando em conta o processo descrito na pesquisa, das 30 mutações sobreviventes temos:

- 21 mutações foram consideradas equivalentes logicamente ao programa original mesmo após a aplicação do operador de mutação. Ou seja 70% do conjunto de sobreviventes é equivalente ao programa original ou não é possível evidenciar essa diferença.
- 9 mutações foram consideradas mutantes sobreviventes não equivalentes. Apenas 30% dos mutantes não foram mortos por nenhum caso de testes do conjunto original, mas morrem ao conceber novos casos com valores específicos ou asserções melhores.

Entre estes últimos, é possível separá-los em dois grupos: Os que sobrevivem por conta de casos de testes de baixa qualidade e que exercitam pouco ou nada os cenários da unidade pelas entradas; E os que necessitam de uma entrada específica, comumente valores ligados à lógica da linha em questão, como valores limites em um escopo e pontos críticos como valores classificatórios para fazer parte de um conjunto.

A grande adição trazida pelos mutantes encontrados foi poder criar novos casos de testes com mais cenários de dados. Tais dados não triviais são completamente dependentes da semântica em questão e vem da análise de requisitos e entendimento da linguagem de programação. Dessa forma, é muito valioso poder acrescentar ao conjunto casos que matem os mutantes resistentes.

As mutações mais notáveis são criadas pelo operador *changed conditional boundary,* e que apesar de serem apenas três mutações resistentes, chamam atenção ao método heurístico de utilizar valores “críticos” da lógica da programação do cenário que é a Análise de Valores limites ao escrever um código de teste unitário.