Avaliação Experimental de Metodologias para
Condução de Ensaios de Proficiência de Software

Rodada: Teste de Mutação - Laboratórios Simulados ICMC

Relatório

Nome do Laboratório de Avaliação : **Helvella Lacunosa**
Nome e nUSP dos participantes do laboratório:
Fernando Gonçalves Campos - 12542352
Nilton Palmeira Pacifico Junior - 5889814
Thiago Shimada - 12691032

Disciplina: **SSC0959 - Teste e Inspeção de Software**
Prof responsável: **Marcio Eduardo Delamaro**
Data: **30/05/2024**

1. **Introdução**

   O ensaio de proficiência baseia-se em realizar um teste de mutação de uma biblioteca na linguagem de programação C. Essa biblioteca consiste em uma coleção de funções para um vetor dinâmico, ou seja, não é necessário alocar/desalocar memória para adicionar e remover elementos. Inicialmente foi notado que a biblioteca possui 46 funções, então cada um dos participantes da equipe ficou responsável por realizar os testes de mutação sobre 15 ou 16 funções.

   Para a compreensão do programa, além da leitura do código também foi utilizado o exemplo de teste como base para entender o funcionamento da biblioteca. A execução e interação para a solução foi feita de maneira incremental, em conjunto com a ferramenta NeoProteum. Os testes de mutação foram incrementados à medida que eram vistos mutantes não equivalentes.

   Ao longo da criação de testes, entendemos que algumas funções não possuem mutantes e em alguns casos também é necessário testar um conjunto de funções. Por isso, ao final foram gerados 32 casos de testes que foram capazes de matar todos os mutantes não equivalentes.

2. **Desenvolvimento**

   A ferramenta NeoProteum foi utilizada através da máquina virtual (disponibilizada e configurada seguindo o passo a passo do vídeo no Drive). Os principais comandos utilizados foram:

   - `test-new --type CMAKE --source item\neoproteum-collections-c-main testes`
     - Dentro da máquina virtual, foi criada a pasta “neoproteum-collections-c-main”, para onde foram movidos

todos os arquivos da pasta item. Dessa forma, os comandos se sucederam assim como o vídeo tutorial disponibilizado no Drive.

- `report testes`
- `muta-gen -all -unit 'lib.c' testes`
- `tcase-add -p "0" testes`
    - Este comando foi utilizado para os 32 testes, alterando apenas o número entre as aspas
- `exemuta -exec -p testes`

Alguns comandos extra foram utilizados como:

- `tcase -d testes`
    - Remover todos os testes
- `report --open testes`

Através do relatório html criado pelo NeoProteum, eram vistos os mutantes que continuavam vivos. Os testes foram então expandidos para matar esses mutantes (ou classificá-los como equivalentes).

3. Resultados

Casos de teste: 32

Total de mutantes gerados: 371
Escore de mutação: 100%

Dos 371 mutantes gerados, 19% eram equivalentes.

Pelos testes concluiu-se que o código funciona de acordo com o esperado.

4. Conclusões

Através do ensaio de proficiência foi possível perceber a importância dos testes de mutação. Inicialmente, a maioria dos casos de teste não é capaz de matar todos os mutantes, ou seja, existem cenários onde o teste não captura erros. O teste de mutação revela estas áreas que não são cobertas pelos testes e permite a correção de suas deficiências.

Também é importante ressaltar a grande utilidade da ferramenta NeoProteum, já que ela gera os testes mutantes de maneira automatizada, em geral com poucas anomalias e classifica os tipos de mutantes.

Apenas foi encontrada uma dificuldade, a função de exemplo para criar testes não utiliza a biblioteca da forma que ela aparenta ter sido criada para ser utilizada:

- O exemplo utiliza *void* (unsigned long long)* como um tipo genérico de variável para poder salvar “qualquer” valor, enquanto a biblioteca parece ter sido implementada para utilizar o *void** como ponteiro (o que seria a forma tradicional de utilizá-lo).

- Dado que percebemos essa inconsistência no final do período de testes, não foi possível refazer os testes. Sendo assim, todos os testes foram feitos utilizando *void** como variável.
- Está forma forma de utilizar *void** dificulta a criação de testes para algumas funções:
    - *cc_array_map*: não é possível criar casos de teste para essa função, já que ela não teria nenhum efeito testável, por utilizar valores que deveriam ser ponteiros (para serem modificados) como variáveis. De qualquer forma, não foi necessário criar testes para ela tendo em vista que nenhum mutante modificou ela é simples o bastante para ser possível averiguar sua corretude.
    - *cc_array_reduce*: essa função apresentou um problema causado pela forma esperada da variável de retorno interagir com o código, em uma das chamadas à função de redução é esperado que esta variável se comporte tanto como ponteiro quanto como variável. Como essa função era afetada por mutantes, a solução adotada foi modificar a função de redução para verificar os casos em que a variável de retorno seria passada em dois parâmetros e tratá-la nos dois casos como ponteiro.
    - Algumas das funções de desalocar memória e de destruir o array poderiam causar problemas já que os ponteiros não teriam memória alocada. No entanto, isso não ocorreu durante os testes de mutação, então não foi necessário ter nenhum cuidado especial com elas.