10/10/23

## Técnicas de Teste Funcional

* Usam como entrada a especificação do módulo
* Especificação/Contratos:
  + É fundamental gerar casos de teste com valores válidos buscando verificar se o módulo se comporta como especificado
  + No caso de programação por contratos deve-se gerar casos de teste que busquem verificar se a implementação atende as especificações do contrato
* Valor Limit
  + Funciona bem quando programa a ser testado é função de várias variáveis independentes que representam conjuntos que tenham uma relação de ordem
  + Geração de casos de teste:
    - Fixa-se o valor de todas as variáveis menos uma em seus valores nominais
    - A variável escolhida assume os valores: {MIN, MIN+1, NOMINAL,MAX-1,MAX}
    - Exemplo:
    - V1: int pertence [10,20]
    - Casos de teste {10, 11, 15,19,20}
  + Se existir mais de uma variável, deve-se testar as combinações possíveis entre elas
  + Se a variável for uma enumeração deve-se testar todos os valores
* Classes de equivalência
  + Princípio: dividir o domínio de entrada em subconjuntos de maneira que o comportamento de um dos membros do conjunto seja representativo do comportamento de todos os membros do conjunto
  + Gerar pelo menos um caso de teste para cada classe
* Diagramas de estado
  + Se há diagrama de estados disponíveis, gerar casos de teste que garantam cobertura de estados e transições
* Condições de erro
  + Gerar casos de teste que procurem gerar as condições de erro previstas
* Valores inválidos
  + Gerar casos de teste com valores de entrada inválidos
* Usa como entrada o código fonte
* Usado para refinar os casos de teste
  + Gerar o grafo de programa
  + Procurar garantir a cobertura do grafo de programa
    - Verificar se passo por todos os comandos
    - Verificar se exercita cada uma das opções de cada condição
    - Verificar se cada laço itera pelo menos K vezes

## Refatoração

* Uma [pequena] modificação no sistema que não altera o seu comportamento funciona, mas que melhora alguma qualidade não-funcional:
  + Simplicidade
  + Flexibilidade
  + Clareza
  + Desempenho
* Mudança do nome de variáveis
* Mudanças nas interfaces dos objetos
* Pequenas mudanças arquiteturais
* Encapsular código repetido em um novo método
* Generalização de métodos

### Aplicações

1. Melhorar código antigo e/ou feito em ciclos de desenvolvimentos anteriores
2. Desenvolvimento incremental (à la XP)

* Em geral, um passo de Refatoração é tão simples que parece que ele não vai ajudar muito
* Mas quando se juntam 50 passos, bem escolhidos, em sequência, o código melhora radicalmente.

### Passos de Refatoração

* Cada passo é trivial
* Demora alguns segundos ou alguns poucos minutos para ser realizado
* É uma operação sistemática e “óbvia”
* Ter um bom vocabulário de refatorações e saber aplicá-las criteriosamente e sistematicamente

### Refatoração sempre existiu

* Mas não tinha um nome
* Fazia-se de forma ad hoc
* A novidade está em criar um vocabulário comum e em catalogá-las
* Viabilizada/estimulada pela prática de testes unitários

### Quando usar Refatoração

* Sempre há duas possibilidades:

1. Melhorar o código existente
2. Jogar fora e começar do 0

* Avaliar a situação e decidir quando é a hora de optar por um ou por outro

### O Primeiro Passo em Qualquer Refatoração

* Antes de começar a refatoração, verifique se você tem um conjunto sólido de testes para verificar a funcionalidade do código a ser refatorado
* Refatorações podem adicionar erros
* Os testes vão ajudá-lo a detectar erros se eles forem criados

### Formato de Cada Entrada no Catálogo

* Nome da refatoração
* Resumo da situação na qual ela é necessária e o que ela faz
* Motivação para usá-la ( e quando não usá-la)
* Mecânica, i.e., descrição passo a passo
* Exemplos para ilustrar o uso