Trabalho 2 - Métodos de programação

Ana Paula Martins Tarchetti - 17/0056082

September 25, 2018

Como fazer levantamento de requisitos

1. Deve ser verificado o que o programa deve fazer, qual será a entrada e qual deverá ser a saída.
2. Deve ser verificado os tipos de dados da entrada, se é um inteiro ou uma lista de floats, entre outros.
3. Deve ser verificado qual o tipo de dado a saída deve ser, como por exemplo, quantas casas decimais são necessárias.
4. Qual o tamanho mínimo, médio e máximo da entrada para que não haja problemas de armazenamento, por exemplo se é um grande banco de dados, ou uma atividade de menor porte.
5. Deve se verificado se o programa é sensível à latência, como por exemplo se é uma atividade online de resposta imediata ou se é para calcular informações não imediatas.
6. Deve ser verificado se o software está modularizado, como por exemplo com o uso de Tipos de Dados Abstratos.
7. Deve ser verificado quem usará o programa, por exemplo se é voltado para pessoas da área da computação ou se é voltado para usuários comuns.
8. Deve ser verificado se é possível usar um framework de teste automatizado no programa, como por exemplo a possibilidade do uso de assertivas para testar certa funcionalidade do programa.
9. Deve ser verificado se o programa passa no teste automatizado, como por exemplo, no gTest.
10. Deve ser verificado qual é a porcetagem aceitável da cobertura do teste automatizado, por exemplo algo acima de 80
11. Deve ser verificado se a porcentagem de cobertura do teste automatizado é aceitável, como por exemplo pelo uso do gcov ou gcovr.

12. Deve ser verificado a prioridade das funcionalidades do programa, para saber qual dado deve ser processado primeiro.

13. Deve ser verificado qual o nível de portabilidade do programa, como por exemplo, em quais softwares ele deve funcionar.

14. Deve ser verificado se o programa está comentado para que ele seja legível para outras pessoas que trabalharâo nele.

15. Deve ser verificado qual o nível de segurança dos dados manipulados, por exemplo se são dados públicos ou se necessitam de um nível de privacidade e segurança mais alto.

Como fazer a especificação dos requisitos

1. Descrição geral do sistema, o que ele recebe de dado, o que ele computa e o que ele retorna de dado.

2. Descrição do fluxo de informações, se o programa recebe todas informações no início ou se é um sistema em tempo real.

3. Representação do conteúdo, qual tipo de dado da entrada e especificação do tipo de dado da saída e a precisão necessária.

4. Descrição funcional das partições do programa.

5. Descrição da reação esperada do sistema.

6. Considerações especiais.

7. Prioridade de implementação.

8. Requisitos com relação ao tempo de resposta, se há a necessidade de resposta imediata ou qual o tempo máximo aceitável de computação dos dados.

9. Tratamento de exceções.

10. Narrativas de exemplos de cenários de uso.

11. Análise de perfis de usuários, se é de uso profissional ou pessoal.

12. Estipular se há a necessidade de verificação de usuário.

13. Listar as plataformas para o qual o programa será distribuído.

14. Descrever possíveis modificações futuras do sistema.

15. Listagem dos testes a serem feitos.

Como fazer o design do software

1. Determinar os módulos do programa.
2. Desenvolver uma parte direcionada ao modo de uso do software, por exemplo um setor de ajuda ao usuário.
3. Campo para validação de usuário caso necessário.
4. Criar um protótipo do software antes do produto final.
5. Encapsulamento das implementações das funcionalidades.
6. Estipular configurações padrão
7. Planejar o layout do software, com as respectivas ramificações.
8. Implementar o layout do software planejado anteriormente.
9. Fazer o design gráfico do software.
10. Remover ambiguidades que possam haver no sistema.
11. Analisar a coesão geral do programa.
12. Definir a interdependência entre os módulos.
13. Planejar o design gráfico do software.
14. Planejar os teste a serem feitos.
15. Testar o software e corrigir os erros até que o software esteja aceitável.

Como fazer o código

1. Se o programa trabalha com número e recebe uma letra ele não deve aceitar a letra como entrada, ele deve imprimir uma mensagem de valor inválido e receber uma nova entrada.
2. O programa deve ser escrito de acordo com o padrão especificado e verificado, por exemplo com o cpplint.
3. As funções do programa devem retornar variáveis possíveis de serem testadas por um framework de teste.
4. O programa deve ser dividido um módulos que serão compilados por meio de headers.
5. Deve se fazer o uso de Tipos De Dados Abstratos, por exemplo pilhas e filas e suas operações.

6. Varíaveis contantes devem ser definidas como constantes para evitar possível erros, por exmplo: "define HorasDoDia 24".

7. Nomes de variável devem específicar o que elas representam de forma clara, por exemplo: "int IndexUser;".

8. Deve ser definido um máximo de dados que podem ser lidos para que não haja perda por problemas de armazenamento, por exemplo: "define MaxUsers 5000".

9. Usar comentários apenas quando nem a função, nem a variável e nem a operação especificam o que está acontecendo no código.

10. Eliminar redundâncias, por exemplo: "int a; a=10;" fica "int a=10;".

11. Testar o código enquanto estiver escrevendo, não testar tudo só no final.

12. Troque repetições de declarações por estruturas de repetição a partir da 3ª repetição, por exemplo: "int aux1 = 10; int aux2 = 20; int aux3 = 30 ;int aux4 =40;" por "int aux[4]; int i = 3; while(i–)aux[i] = (i+1)*10;".

13. Debuggar o código enquanto estiver escrevendo e levar em consideração outras porções do código que possam ser alteradas ao corrigir um erro.

14. Evitar uso de ponteiros para não dificultar a depuração do código.

15. Evitar reprocessamento de dados, com o armazenamento temporário de determinadas informações.

Como comentar o código e escrever o código com "design by contract" com assertivas de entrada, saída, invariantes e como comentários de argumentação do código

1. Comentar um cabeçalho com os dados do desenvolvedor e com a especificação geral do código.

2. Não comentar algo que está explicito no código.

3. Não fazer comentários desnecessários ou subjetivos.

4. Comentar enquanto estiver desenvolvendo o código.

5. Comentar especificação de funções.

6. Comentar os parâmetros de entrada nas assertivas de entrada.

7. Comentar as variáveis globais de entrada nas assertivas de entrada.

8. Comentar todos os arquivos de entrada nas assertivas de entrada.
9. Comentar o valor retornado pela função nas assertivas de saída.
10. Comentar todos os parâmetros de saída nas assertivas de saída.
11. Comentar todas as variáveis globais de saída nas assertivas de saída.
12. Comentar todos os arquivos de saída nas assertivas de saída.
13. Comentar as condições a serem satisfeitas pelos dados nas assertivas invariantes.
14. Comentar a especificação do erro tolerado nas assertivas invariantes, quando for o caso.
15. Comentar os estados que constituem a estrutura de dados nas assertivas invariantes.

Como testar o código

1. Dar o nome aos teste, por exeplo: "TesteNumerosNegativos();".
2. Testar entradas com outros tipos de dados e verificar se houve erro.
3. Testar entradas fora do intervalo estipulado e verificar se houve erro, como números negativos, ou extremamente grandes.
4. Testar o tempo de execução das funcionalidades por meio de bibliotecas como "time.h".
5. Executar tanto testes pontuais quanto testes totais.
6. Criar testes com auxilio de estruturas de repetição, para testar vários valores de entrada.
7. Testar se as funcionalidades retornam o valor esperado quando as entradas são válidas.
8. Modularizar os testes.
9. Criar testes baseados nos erros ocorridos na depuração.
10. Testar a precisão das saídas.
11. Realizar testes com caracteres especiais.
12. Testar novamente toda vez que houver refatoração.
13. Testar o caso máximo do intervalo permitido com os valores máximos permitidos dos dados e verificar se há erro de armazenamento, caso haja, diminuir os valores máximos dos intervalos permitidos.

14. Analisar a cobertura do teste, como por exemplo com o gcov e verificar se a porcentagem é aceitável.
15. Analisar quais linhas de código não foram executadas no teste por meio do gcovr e tentar incluí-las no teste.

Como depurar o código

1. Depurar o programa durante a escrita do código.
2. Para cada erro encontrado, corrigi-lo em todas linhas de código relacionadas ao erro.
3. Verificar em qual módulo está o erro, para só depois corrigir.
4. Prestar atenção no uso de ponteiros para encontrar erros difíceis de achar.
5. Isolar o a parte do código onde há o erro e depois corrigir.
6. Tentar reproduzir o erro, para facilitar o processo de encontrar o erro.
7. Dividir as origens de cada erro e corrigir por partes.
8. Análisar os valores armazenados nas variáveis para entender melhor o problema.
9. Utilizar breakpoints para analisar partes do código.
10. Verificar se não há laços infinitos de repetição.
11. Utilizar a função "print" no depurador e não no código.
12. Se uma parte do cógigo está muito confusa, tentar reescrevê-la.
13. Verificar se não há divisão por zero.
14. Testar novemente após a correção de algum erro.
15. Prestar atenção nos erros mais comuns de semântica da linguagem C: esquecer "$" no scanf, "=" em vez de "==" em comparações, esquecer de inicializar variáveis e ";" no lugar errado.