Departamento de Ciência da Computação
Técnicas de Programação 2 - 1/2025

Projeto de Disciplina

Sistema de Gerência de compras e compartilhamento de preços

Descrição

Para o sistema de gerenciamento de compras e compartilhamento de preços, definido na especificação anterior:

Se pode usar uma biblioteca ou outro programa para a localização GPS

Se deve entregar:

Especificação de requisitos:

O projeto vai ser separado por iterações. Para cada iteração o grupo deve fazer o backlog da iteração. O backlog é composto das historietas correspondentes ao sistema.

As historietas devem ser transformadas em estórias de usuário no formato:

Eu como <tipo de usuário> quero <algum objetivo> para <algum motivo>

Ex. Eu como <usuário> quero <fazer uma pesquisa de produto> para <saber os preços dos fornecedores mais próximos>

No documento, toda a estória de usuário deverá ter um numero único ex “EU023”. Este número deverá constar no código escrito de forma que se alguém procurar nos códigos pela string “EU023” encontrará o lugar onde a estória foi implementada e o lugar onde ela foi testada.

Estas estórias de usuário devem ser utilizadas no Scrum e Kanban correspondentes.

Dever ser feita a priorização destas estórias de usuário

Deve ser feito cronograma dizendo quais historietas e requisitos vão ser implementados em cada uma das iterações:

Se deve utilizar as técnicas mostradas em sala de aula.

Para cada iteração deverá ser feito o backlog da iteração e apresentado em aula.

Baseados neste backlog, serão implementados os protótipos:

1) 30/6/2025 – mostrar protótipo 1

2) 14/7/2025 – mostrar protótipo final

Deve ser descrito como será o cumprimento dos seguintes requisitos:

1) Devem ser aplicados neste trabalho todos os conceitos vistos nos trabalhos anteriores. O programa pode ser feito com as seguintes opções:

C ou C++

Java

Python

Javascript

Ruby

Para outra opção de linguagem, o professor deve ser consultado

a) dever ser feito de forma modularizada (ex. makefile, .h e .c).

b) deve usar um padrão de codificação:

ex. https://google.github.io/styleguide/cppguide.html

O código deverá ser devidamente comentado facilitando o entendimento.

Deve ser usado um verificador para o padrão de codificação

Ex .cpplint (https://github.com/cpplint/cpplint).

Utilize o verificador desde o início da codificação pois é mais fácil adaptar o código no início.

c) Testes utilizando um *framework* de teste. Ex. Gtest ou Catch. Como estes testes mostram que o programa segue a especificação. **O desenvolvimento deve ser feito orientado a testes**.

Deverá ser enviado o histórico de versões de código mostrando o desenvolvimento orientado a testes

d) Devem ser feitas revisões no código conforme visto em sala de aula utilizando as checklists vistas no trabalho. Deve ser gerado um laudo (pdf) do que passou na revisão e o que não passou. Deve ser mudado o que for possível para estar dentro do especificado na checklist. O que não for possível mudar por algum motivo forte, deve estar justificado no laudo. Ex. se adequar o código a checklist exigir mudar toda estrutura do programa, não é viável fazer isto.

e) Cada função deve ter assertivas de entrada e de saída como comentários. As assertivas de entrada são tudo o que deve ser verdadeiro para que a função funcione corretamente. Assertivas de saída deve ser tudo o que é garantido pela função. As assertivas de entrada são relacionadas a todos os dados que são utilizados na função. As assertivas de saída são relacionadas a todos os dados que são modificados pela função.

2) Deve ser utilizado um verificador de cobertura

ex. gcov. (http://gcc.gnu.org/onlinedocs/gcc/Gcov.html). O makefile deve ser modificado de forma incluir as flags -ftest-coverage -fprofile-arcs. Depois de rodar o executável rode gcov nomearquivo e deverá ser gerado um arquivo .gcov com anotação.

O verificador de cobertura é utilizado para saber qual percentual do código é coberto pelos testes. Neste caso os testes devem cobrir pelo menos 80% do código por módulo.

Utilize um verificador estático

ex. cppcheck, corrigindo os erros apontados pela ferramenta.

Utilize cppcheck --enable=warning .

para verificar os avisos nos arquivos no diretório corrente (.)

Utilize o verificador estático sempre e desde o início da codificação pois é mais fácil eliminar os problemas logo quando eles aparecem. Devem ser corrigidos apenas problemas no código feito e não em bibliotecas utilizadas (ex. gtest, catch)

3) Deverá ser entregue o histórico do desenvolvimento orientado a testes feitos através do github (https://github.com/)

Um tutorial inicial pode ser encontrado em:

https://guides.github.com/activities/hello-world/

4) O tempo em cada tarefa pode ser facilmente contabilizado utilizando aplicativos ou sites como:

https://toggl.com
clockify.me

5) O projeto deve ser gerenciado o
trello.com ou github

Para usar outro site, consulte o professor.

6) Interface gráfica e frameworks utilizados deverão ser verificada com o professor e deverá seguir os seguintes critérios :

a) open source
b) Licenças : GPL, LGPL, , SSPL, Apache License 2.0, BSD, MIT ou Mozilla

7) O programa e o módulo devem ser depurados. Ex usando o GDB.

(http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html)

(https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf)

8) Se for em C ou C++, utilizar o Valgrind (*valgrind.org/*), utilizar um verificador dinâmico para outra linguagem (se houver)

9) Se for em C ou C++ ou outra linguagem semelhante, utilize diretivas de pré-compilação para evitar o problema de identificador duplicado e melhor gestão do software.

10) Deve ser gerada uma documentação do código usando o programa DoxyGen (http://www.stack.nl/~dimitri/doxygen/): O programa inteiro terá de ser documento usando DoxyGen. Comentários que vão ficar na documentação devem ser do estilo Javadoc.

Para as entregas dos protótipos:

Módulos

O programa deve ser dividido em módulos.

O programa deverá ter um makefile (ou equivalente) adequado colocando os fontes e objetos compilados em diretórios separados como nos trabalhos anteriores.

Observações:

Utilize os princípios de modularidade na criação dos módulos, definidos seus módulos de definição, implementação e organizando suas compilações e ligações (**links**) de forma adequada.

Controle de qualidade das funcionalidades

Deve-se criar um *módulo controlador de teste* (disciplinado) usando o *framework* de teste para testar se as principais funcionalidades e restrições dos módulos.

O desenvolvimento deverá ser orientado a testes.

Parte 2. Escrita

· Um arquivo **LEIAME.TXT** contendo a explicação de como utilizar o(s) programa(s).
· Tantos arquivos **RELATORIO-nome.TXT** quantos forem os membros do grupo. O tema **nome** deve identificar o membro do grupo ao qual se refere o relatório. Estes arquivos devem conter uma tabela de registro de trabalho organizada como a seguir:
Data | Horas Trabalhadas | Tipo Tarefa | Descrição da Tarefa Realizada

Na descrição da tarefa redija uma explicação breve sobre o que o componente do grupo fez. Esta descrição deve estar de acordo com o Tipo Tarefa. Cada Tipo Tarefa identifica uma natureza de atividade que deverá ser discriminada explicitamente, mesmo que, durante uma mesma sessão de trabalho tenham sido realizadas diversas tarefas. Os tipos de tarefa são:

- estudar aulas e laboratórios relacionados
- especificar os módulos
- especificar as funções
- revisar especificações
- projetar
- fazer diagramas
- revisar projetos
- codificar módulo
- Rodar o verificador estático e retirar warnings
- revisar código do módulo
- redigir casos de teste
- revisar casos de teste
- realizar os testes
- instrumentar verificando a cobertura
- documentar com Doxygen
- gerenciar a construção do software

Observações:

- Dica: Preencha esta tabela de atividades ao longo do processo. NÃO DEIXE PARA ÚLTIMA HORA, POIS VOCÊ NÃO SE LEMBRARÁ DO QUE FEZ TAL DIA, TAL HORA. Com relatórios similares a esse você aprende a planejar o seu trabalho.

Diagramas, gráficos e artefatos:

Devem ser entregues os diagramas, gráficos e outros artefatos feitos de acordo com as instruções dadas em sala de aula.

Deve ser enviado um único arquivo compactado (.zip) contendo todos os arquivos necessários (ex. .c .h makefile, estrutura de diretório, informação de como utilizar, etc). Deve constar também os documentos especificados. Deverá haver um arquivo leiame.txt com as informações sobre como o programa deverá ser compilado, como poderá ser executado e quais são os arquivos enviados e o que cada um contem.

Apenas um integrante do grupo deve enviar o trabalho. O nome do trabalho deve ser algo como:
MP_Jose_12345_Proj.zip
Com o primeiro nome e matricula de quem envia pelo grupo.

Cópias de trabalho terão nota zero.

Excelente trabalho!

Deve ser enviada pelo aprender3.unb.br:

Data de Entrega final 13/7/25 até as 23:55