Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: Métodos de Programação
Código da Disciplina: 201600

Métodos de Programação - 201600

Trabalho 3

O objetivo deste trabalho é utilizar o material dado em sala sobre tabelas de decisão para fazer um sistema de backup com um pendrive. Existe uma lista de arquivos que devem ter backup. Quando o programa é executado, ele verifica a data dos arquivos e deve garantir que seja feito o backup de forma adequada. Os detalhes do que deve ser feito está na tabela de decisão dos slides.

Devem ser gerados casos de teste de forma a testar todas as colunas da tabela de decisão.
Deve ser verificado se os arquivos foram transferidos corretamente e se as datas estão adequadas. Para isto podem ser utilizados outro programas e bibliotecas.
Algumas colunas talvez não possam ser testadas devido ao mascaramento ou outras condições. Neste caso, deve deixar indicado.

O desenvolvimento deverá ser feito passo a passo seguindo a metodologia TDD. A cada passo deve-se pensar qual é o objetivo do teste e o significado de passar ou não no teste.

1) O programa deverá ser dividido em módulos e desenvolvido em C/C++ ou Python.
No caso de ser em C/C++, deverá haver um arquivo backup.c (ou .cpp) e um arquivo backup.h (ou .hpp). Deverá haver também um arquivo testa_backup.c (ou .cpp) cujo objetivo é testar o funcionamento da junção de arquivos.
No caso de ser em Python dever ser dividido em backup.py (implementação) e testa_backup.py (teste).

2) Se for em C/C++, utilize o padrão de codificação dado em:
https://google.github.io/styleguide/cppguide.html
quando ele não entrar em conflito com esta especificação. O código dever ser claro e bem comentado. O código deve ser verificado se está de acordo com o estilo usando o cpplint (https://github.com/cpplint/cpplint).

Se for em Python utilize o padrão de codificação dado em:
https://google.github.io/styleguide/pyguide.html
quando ele não entrar em conflito com esta especificação. O código deve ser verificado se está de acordo com o estilo usando o pylint
https://pypi.org/project/pylint/

Utilize o cpplint ou pylint desde o início da codificação pois é mais fácil adaptar o código no início.

3) Faça um documento (.txt ou .pdf) dizendo quais testes você fez a cada passo e o que passar neste teste significa.

4) O desenvolvimento deverá ser feito utilizando um destes frameworks de teste:

C/C++
gtest (https://code.google.com/p/googletest/)
catch (https://github.com/philsquared/Catch/blob/master/docs/tutorial.md)

Python
Robot (https://robotframework.org/)
PyTest (https://docs.pytest.org/en/7.1.x/)

5) Deverá ser entregue o histórico do desenvolvimento orientado a testes feitos através do git (https://git-scm.com/docs/gittutorial)

git config --global user.name "Your Name Comes Here"
git config --global user.email you@yourdomain.example.com
git init
git add *
git commit –m “teste 1”
git log

Compactar o diretório “.git” ou equivalente enviando ele junto.

6) Deve ser utilizado um verificador de cobertura
ex. para C/C++ usar o 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.

Para Python usar o Coverage.py (https://coverage.readthedocs.io/en/6.4.2/)

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.

7) 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 gerar uma documentação mais adequada, rodar
doxygen –g
isto irá gerar um arquivo Doxyfile. Neste arquivo, na linha adequada, colocar:

EXCLUDE = catch.hpp

Isto fará com que o catch.hpp não seja documentado. Uma mudança semelhante deverá ser feita para outro framework se necessário.

Depois de feito isto para documentar basta roda : doxygen

8) Utilize um verificador estático de código

No caso do C/C++
Usar o cppcheck, corrigindo os erros apontados pela ferramenta.
Utilize cppcheck --enable=warning .
para verificar os avisos nos arquivos no diretório corrente (.)

No caso do Python
Usar o mypy

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)

9) Cada função (ou método) deve ter assertivas de entrada e assertivas 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.

Devem ser enviados para a tarefa no aprender3.unb.br um arquivo zip onde estão compactados todos os diretórios e arquivos necessários. Todos os arquivos devem ser enviados compactados em um único arquivo (.zip) e deve ser no formato matricula_primeiro_nome.zip. ex: 06_12345_Jose.zip. Deve ser enviado um arquivo dizendo como o programa deve ser compilado e rodado.

Deve ser enviado o diretório “.git” compactado junto com o “.zip”

Data de entrega:

10/ 8 /22

Pela tarefa na página da disciplina no aprender3.unb.br