Qualidade de Software

1. Introdução

A qualidade de software não é apenas um atributo desejável; é um requisito fundamental para o sucesso e sustentabilidade de qualquer aplicação. Software de má qualidade pode resultar em falhas críticas, perdas financeiras e danos à reputação. Como futuros engenheiros informáticos, é essencial dominar os conceitos e práticas que asseguram a criação de software robusto, eficiente e passível de manutenção.

Nesta atividade laboratorial, iremos aprofundar os princípios que regem a qualidade de software, compreendendo como diferentes técnicas contribuem para a deteção precoce de erros e para a garantia de conformidade com os requisitos. Focaremos em técnicas de teste unitário, funcional, de integração e de regressão, entendendo o papel específico de cada um no ciclo de desenvolvimento e manutenção de software.

A automação de testes é uma prática indispensável na indústria atual, permitindo acelerar o processo de desenvolvimento e aumentar a fiabilidade dos testes. Utilizaremos ferramentas como o Selinium, para automatizar testes de interfaces gráficas, e o Jenkins, para implementar pipelines de integração contínua que automatizam builds e execuções de testes. O domínio destas ferramentas é crucial para a eficiência e competitividade no mercado de trabalho.

Além disso, exploraremos métricas de qualidade de código que nos permitem avaliar objetivamente o software desenvolvido. A cobertura de testes fornece-nos uma medida do alcance dos nossos testes sobre o código, indicando potenciais áreas não testadas que podem conter defeitos. Por sua vez, a complexidade ciclomática oferece perceções sobre a complexidade lógica do código, influenciando a sua legibilidade, manutenção e propensão a erros.

Esta atividade laboratorial proporcionará uma experiência prática e integrada, consolidando conhecimentos teóricos através da aplicação em casos reais. Pretende-se que, no final desta atividade, estejam aptos a implementar práticas de qualidade de software que são padrão na indústria, preparando-vos para enfrentar os desafios do desenvolvimento profissional onde a qualidade é um fator crítico de sucesso.

2. Objetivos e competências a desenvolver

Ao concluir esta atividade laboratorial, os estudantes deverão ser capazes de:

- Compreender os conceitos fundamentais de qualidade de software e a sua relevância no desenvolvimento.
- Aplicar técnicas de teste unitário, funcional, de integração e de regressão em projetos de software.
- Utilizar ferramentas de automação de testes, como o Selenium e o Jenkins, para otimizar processos de teste.
- Analisar métricas de qualidade de código, incluindo cobertura de testes e complexidade ciclomática.
- Implementar testes automatizados para interfaces gráficas utilizando o Selenium WebDriver.
- Configurar pipelines de integração contínua no Jenkins para automatizar builds e execuções de testes.
- Fazer refactoring de código com base na análise de métricas para melhorar a sua qualidade e manutenibilidade.
- Documentar eficientemente estratégias de teste, resultados obtidos e ações de melhoria realizadas.

3. Instruções

Nesta atividade laboratorial, irá utilizador a aplicação web em Django para gestão de tópicos de discussão fornecida

na atividade laboratorial 07. A atividade será dividida em 4 etapas:

Etapa 1: Conceitos de Qualidade de Software

A qualidade de software é fundamental para assegurar que uma aplicação cumpre os requisitos esperadas e proporciona uma experiência satisfatória ao utilizador. Trata-se de um conjunto de características que determinam a capacidade do software em cumprir as suas funções, ser fiável, eficiente e fácil de manter. A adoção de standards internacionais permite estabelecer critérios claros e uniformes para avaliar e melhorar a qualidade do software. Dois dos principais standards nesta área são:

- **ISO/IEC 25010** – Este standard define um modelo de qualidade para produtos de software, estruturado em oito características principais (ver Figura 1): (1) *funcionalidade*, capacidade de o software fornecer funções que satisfaçam as necessidades explícitas e implícitas dos utilizadores; (2) *desempenho*, eficiência com que o software utiliza recursos para fornecer resultados atempados; (3) *compatibilidade*, capacidade de o software funcionar em diferentes ambientes e interagir com outros sistemas; (4) *usabilidade*, facilidade com que os utilizadores podem aprender a usar e operar o software; (5) *fiabilidade*, capacidade de o software manter um nível de desempenho especificado sob condições estabelecidas; (6) *segurança*, proteção de informação e dados contra acesso não autorizado; (7) *manutenibilidade*, facilidade com que o software pode ser modificado para corrigir defeitos ou adaptar-se a novas exigências; e (8) *portabilidade*, capacidade de o software ser transferido de um ambiente para outro.

ISO/IEC 25010 – Modelo de Qualidade de Software							
Adequação Funcional	**Fiabilidade**	**Eficiência de Desempenho**	**Usabilidade**	**Manutenibilidade**	**Segurança**	**Compatibilidade**	**Portabilidade**
- Completude funcional - Correção funcional - Adequação funcional	- Maturidade - Disponibilidade - Tolerância a falhas - Recuperabilidade	- Comportamento temporal - Utilização de recursos - Capacidade	- Adequação e reconhecimento - Capacidade de aprendizagem - Operacionalidade - Proteção contra erros do utilizador - Estética da IU - Acessibilidade	- Modularidade - Reutilização - Analisabilidade - Modificabilidade - Testabilidade	- Confidencialidade - Integridade - Não repúdio - Responsabilidade - Autenticidade	- Coexistência - Interoperabilidade	- Adaptabilidade - Instalabilidade - Substituibilidade

Figura 1. Características e subcaracterísticas do ISO/IEC 25010.

- **ISO/IEC 5055** – Este standard fornece uma métrica para avaliar a qualidade interna do código-fonte, forcando em aspetos como: (1) *complexidade*, grau de complicação do código que pode afetar a sua compreensão e manutenção; (2) *escalabilidade*, capacidade do software para lidar com aumentos de carga sem perda significativa de desempenho; (3) *segurança do código*, presença de vulnerabilidades que possam ser exploradas; e (4) *robustez*, capacidade do software resistir a condições inesperadas ou erros sem falhar.

Passos a realizar:

1. Analise detalhadamente os standards ISO/IEC 25010 e ISO/IEC 5055, focando as características de qualidade de software que cada um define e na sua aplicação prática no desenvolvimento.
2. Analise a aplicação web desenvolvida no TP07, comparando as suas funcionalidades e estrutura com os critérios estabelecidos nos standards. Identifique como a aplicação atende ou não a cada característica de qualidade.
3. Documente as suas observações, destacando pontos fortes e áreas que necessitam de melhoria em termos de qualidade de software, conforme definido pelos standards estudados.

Etapa 2: Auditoria manual de segurança

As técnicas de teste são essenciais para garantir que o software funciona conforme o esperado e cumpre os requisitos estabelecidos. Ao aplicar diferentes tipos de testes, é possível identificar e corrigir erros em várias fases do desenvolvimento, melhorando a qualidade e a fiabilidade da aplicação. Os principais tipos de teste a considerar são:

- **Teste unitário** – Foca-se na validação de unidades individuais de código, como funções ou métodos, garantindo que cada componente funciona corretamente de forma isolada.
- **Teste funcional** – Verifica se as funcionalidades do software cumprem os requisitos especificados, avaliando o comportamento do sistema do ponto de vista do utilizador.
- **Teste de integração** – Examina a interação entre diferentes módulos ou componentes, assegurando que funcionem em conjunto sem problemas.
- **Teste de regressão** – Após alterações ou atualizações no código, verifica se as funcionalidades existentes continuam a operar corretamente, identificando possíveis efeitos colaterais.

Passos a realizar:

1. Reveja os requisitos funcionais e não funcionais da aplicação, listando as funcionalidades críticas que devem ser validadas para garantir o correto funcionamento do software.
2. Desenhe testes unitários para as funções e métodos individuais, especificando os casos de teste, dados de entrada, resultados esperados e critérios de aceitação para cada um.
3. Desenhe testes funcionais que validem as funcionalidades "completas" da aplicação do ponto de vista do utilizador, incluindo cenários de uso típicos e casos extremos.
4. Desenhe testes de integração para verificar a interação correta entre os diferentes módulos ou componentes da aplicação, assegurando que a comunicação e dependências entre eles funcionam conforme esperado.
5. Desenhe testes de regressão que possam ser executados após futuras alterações no código, garantindo que as funcionalidades existentes continuam a operar corretamente e que não surgem novos defeitos.

Etapa 3: Automação de testes

A automação de testes é uma prática essencial que permite executar testes de forma eficiente, consistente e repetida, sem a necessidade de intervenção manual constante. Esta prática reduz significativamente o tempo e os recursos necessários para validar o software, aumentando a fiabilidade e a rapidez do ciclo de desenvolvimento. O Selenium e o Jenkins são duas das ferramentas de automação de testes mais utilizadas.

O Selenium (https://www.selenium.dev/documentation/) disponibiliza um conjunto de ferramentas open-source que permite automatizar navegadores web. Com o Selenium WebDriver, é possível simular interações de utilizadores com aplicações web, o que é particularmente útil para testes funcionais e de interface.

O Jenkins (https://www.jenkins.io/doc/book/) é um servidor de automação open-source que facilita a integração contínua (CI) e a entrega contínua (CD) de software. Com o Jenkins, pode automatizar tarefas como a compilação, teste e deployment de aplicações, integrando diversos processos num pipeline unificado.

Passos a realizar:

1. Instale e configure o Selenium WebDriver (https://selenium-python.readthedocs.io/index.html; https://django-selenium.readthedocs.io/en/latest/) no seu ambiente de desenvolvimento, assegurando que tem as dependências necessárias e que pode automatizar interações com a aplicação web.

2. Converta os testes desenhados na Etapa 2 em scripts automatizados (https://docs.djangoproject.com/en/5.1/topics/testing/; https://docs.djangoproject.com/en/5.1/topics/testing/tools/#django.test.LiveServerTestCase) utilizando o Selenium, implementando a simulação das ações do utilizador na interface web e verificando automaticamente os resultados. Utilize o ficheiro `tests.py` fornecido como ponto de partida.
3. Instale e configure o Jenkins (https://www.jenkins.io/download/) preparando-o para executar tarefas de integração contínua e automação de testes no seu projeto.
4. Configure um pipeline no Jenkins que, ao detetar alterações no código, execute automaticamente os testes automatizados, compile a aplicação se aplicável e gira relatórios de resultados. Utilize o `Jenkinsfile` fornecido como ponto de partida.
5. Execute o pipeline configurado no Jenkins, monitorize a execução dos testes automatizados, analise eventuais falhas e tome medidas corretivas para resolver os problemas identificados, garantindo a fiabilidade do processo de automação.

Etapa 4: Métricas de qualidade de código

As métricas de qualidade de código são instrumentos fundamentais para avaliar de forma objetiva a qualidade interna do código. Estas métricas oferecem uma perceção quantificável sobre aspetos críticos do código, permitindo identificar áreas que requerem atenção e orientar esforços de melhoria. Duas das métricas mais relevantes são a cobertura de testes e a análise ciclomática.

- **Cobertura dos testes** – Esta métrica indica a percentagem de código que é executada durante a execução dos testes automatizados. Uma cobertura elevada sugere que o código foi bastante testado, reduzindo a probabilidade de defeitos não detetados. A cobertura pode ser medida em diferentes níveis, como linhas de código, ramos de decisão ou condições lógicas.
- **Complexidade ciclomática** – Mede a complexidade lógica de um programa, baseada no número de caminhos lineares independentes através do código. Uma complexidade elevada pode indicar que o código é difícil de compreender, testar e manter, aumentando o risco de erros. Reduzir a complexidade ciclomática melhora a legibilidade e a qualidade geral do software.

Passos a realizar:

1. Utilize ferramentas apropriadas, como o coverage.py (https://coverage.readthedocs.io/en/7.6.8/; https://docs.djangoproject.com/en/5.1/topics/testing/advanced/), para calcular a cobertura dos testes automatizados, obtendo a percentagem de código que é efetivamente testado pelos testes.
2. Analise o código-fonte da aplicação para determinar a complexidade ciclomática das funções e métodos, utilizando ferramentas como o radon (https://radon.readthedocs.io/en/latest/), identificando áreas com complexidade elevada.
3. Avalie os resultados das métricas obtidas, interpretando-os para identificar partes do código que necessitem de melhoria, como funções excessivamente complexas ou código insuficientemente testado.
4. Documente as métricas de qualidade obtidas, incluindo gráficos ou tabelas que ilustrem a cobertura dos testes e a complexidade ciclomática, e apresente recomendações específicas para melhorar a qualidade do código, aumentar a cobertura dos testes e a complexidade onde necessário.

Bom trabalho!