UNIVERSIDADE FEDERAL DE VIÇOSA - UFV - CAMPUS FLORESTAL

ENGENHARIA DE SOFTWARE II

Análise da Qualidade

Plano Geral de Testes - Equipe 2

Versão 1.0

Vitor Vasconcelos de Melo Pontes - 4255

Henrique Alves Campos - 4231

Florestal - MG

2024

Sumário

1. Introdução

O Plano de Testes reúne todas as informações necessárias para planejar e controlar o esforço de testes durante o desenvolvimento do produto, com o objetivo de assegurar a integridade do código. Neste projeto, serão aplicados Testes de Unidade para verificar o funcionamento correto do código, Testes de Integração/Função para garantir a consistência do código com o banco de dados e as telas do jogo, e Testes de Usabilidade para proporcionar aos usuários uma experiência adequada e alinhada com os requisitos definidos. Os principais responsáveis pela execução dos testes são o Analista de Qualidade, Desenvolvedores Sênior e Júnior e o Designer de Jogo.

2. Abordagem dos Testes

Para implementar e automatizar os testes, utilizaremos o JUnit em conjunto com o GitHub Actions. O JUnit permitirá a criação de casos de teste que simulem interações e funcionalidades do usuário, enquanto o GitHub Actions facilitará a automação contínua desses testes, verificando que o sistema esteja funcionando conforme o planejado em cada etapa do desenvolvimento.2.1 Técnicas e Tipos de Teste

2.1.1 Teste de Unidade

Os testes de unidade têm como principal objetivo verificar se as unidades individuais de código, como classes, métodos ou funções, operam conforme o esperado.

Objetivo da Técnica	Verificar a funcionalidade de cada unidade individual do código de maneira isolada. Isso assegura que cada componente funcione corretamente e cumpra seu propósito.
Técnica:	● Criar os casos de teste utilizando a estrutura de teste JUnit para verificar a funcionalidade das unidades de código. ● Executar os testes de forma automatizada com GitHub Actions, fornecendo entradas válidas e inválidas para verificar se as unidades produzem os resultados esperados. ● produzam resultados esperados.
Estratégias:	● Utilizar as estruturas de teste do JUnit para validar o

	comportamento das unidades de código com diversas entradas. ● Implementar mocks, quando necessário, para isolar as unidades de código e evitar dependências externas, como o banco de dados MySQL.
Ferramentas Necessárias:	● **JUnit**: Estrutura de teste para criação e execução de testes unitários em Java. ● **Mockito ou similar**: Biblioteca de mocking para criar mocks de dependências externas quando necessário.
Critérios de Êxito:	A técnica garante a cobertura de todas as classes e métodos do sistema, validando que cada unidade do código funcione corretamente quando testada individualmente.

2.1.2 Teste de Função

Os testes de função devem se concentrar em todos os requisitos de teste associados a casos de uso, funções e regras de negócio. O objetivo desses testes é garantir a correta aceitação, processamento e recuperação de dados, além da implementação precisa das regras de negócio.

Objetivo da Técnica	Validar a funcionalidade do objeto de teste, incluindo navegação, entrada, processamento e recuperação de dados, a fim de observar e registrar o comportamento esperado.
Técnica:	Executar os recursos, fluxos e funções de cada caso de uso utilizando dados válidos e inválidos, verificando se: ● Os resultados esperados ocorrem ao utilizar dados válidos. ● Mensagens de erro ou aviso apropriadas são exibidas para dados inválidos. ● Cada regra de negócio é aplicada corretamente.
Estratégias:	Configurar GitHub Actions para executar os testes de função de forma automatizada e, sempre que necessário, realizar testes manuais nas interfaces para validação visual e funcional.
Ferramentas Necessárias:	**JUnit**: Para a criação e execução dos testes de função. **GitHub Actions**: Para automação e execução contínua dos testes, conforme alterações no código.
Critérios de	Os resultados obtidos nos testes devem coincidir com os resultados

Êxito:	esperados descritos nos casos de teste, validando a conformidade de cada funcionalidade.

2.1.3 Teste da Interface do Usuário

O Teste da Interface do Usuário (UI) verifica a interação do usuário com o software e assegura que a interface forneça acesso e navegação adequados para as funções do sistema e que funcione conforme o esperado.

Objetivo da Técnica	Observar e registrar a conformidade da interface com os padrões e o comportamento esperado: ● Verificar se a navegação e as funcionalidades da interface atendem aos requisitos e refletem as funções de negócio, incluindo a transição entre telas e o uso adequado de métodos de acesso, como mouse, teclas de tabulação e atalhos. ● Testar a funcionalidade e a disposição dos elementos nas telas, incluindo menus, posição de elementos e foco.
Técnica:	Criar ou ajustar testes para cada tela, a fim de verificar a navegação correta e os estados dos objetos em cada tela.
Estratégias:	Executar testes manuais na interface para validar cada cenário de teste individualmente, registrando os resultados.
Ferramentas Necessárias:	**JUnit**: Para criação de testes automatizados onde aplicável. **GitHub Actions**: Para execução contínua de testes automatizados, além de registros e verificação de conformidade.
Critérios de Êxito:	Os resultados observados devem corresponder aos resultados esperados e descritos nos casos de teste, confirmando a funcionalidade e navegabilidade da interface do usuário.

3. Critérios de Entrada e de Saída

3.1 Critérios de Entrada

O ambiente de testes, abrangendo hardware, software, banco de dados e configurações, deve estar configurado e acessível para utilização. Além disso, todos

os recursos indispensáveis para a realização dos testes, como equipe, ferramentas de teste, dispositivos e dados de teste, devem estar prontos e acessíveis.

3.2 Critérios de Saída

- **Cobertura de Requisitos**: Todos os requisitos de software identificados foram testados e documentados, e os resultados estão alinhados com as expectativas.
- **Execução de Casos de Teste:** Todos os casos de teste programados foram realizados, e os resultados foram registrados e revisados.
- **Critérios de Aceitação:** Os critérios de aceitação definidos para os testes foram cumpridos e validados pelos stakeholders ou pela equipe de qualidade.
- **Defeitos Resolvidos:** Todos os defeitos encontrados durante a execução dos testes foram registrados, acompanhados, solucionados e retestados com êxito.

3.3 Critérios de Suspensão e de Reinício

- **Gravidade de Defeitos:** Caso um número considerável de defeitos críticos ou bloqueadores seja detectado durante a execução dos testes, e esses defeitos impactem severamente a funcionalidade ou a estabilidade do sistema, os testes podem ser interrompidos para permitir a correção imediata.
- **Riscos Inesperados:** Se forem descobertos riscos não antecipados ou eventos imprevistos que possam prejudicar a integridade dos testes, como questões de segurança ou falta de recursos essenciais, os testes podem ser temporariamente suspensos.
- **Recursos Indisponíveis:** Se recursos essenciais para a execução dos testes, como servidores ou hardware específico, se tornarem indisponíveis devido a falhas técnicas ou outros motivos, os testes podem ser interrompidos até que esses recursos estejam disponíveis novamente.
- **Alterações nos Requisitos:** Se mudanças significativas nos requisitos ou nas especificações do software ocorrerem durante a execução dos testes, os testes podem ser suspensos para avaliar o impacto dessas alterações nos casos de teste existentes.

4. Artefatos Produzidos

4.1 Relatório de Cobertura de Teste

Ao final de cada sprint, será elaborado um relatório com a cobertura atual dos testes, sendo ideal que esta permaneça sempre acima de 80%.

4.2 Resultados Detalhados dos Testes

Relatório gerado para os Testes Funcionais e da Interface do Usuário.

4.3 Scripts de Testes Automatizados

A cada sprint, novos testes serão desenvolvidos, e os scripts poderão ser visualizados diretamente no código-fonte na pasta /test.

5. Fluxo de Trabalho de Teste

Em cada sprint, o Desenvolvedor Sênior e os Desenvolvedores Júnior realizarão testes de unidade no código-fonte, com a assistência do analista de qualidade, se necessário. Este analista é responsável por avaliar os testes realizados e criar novos casos de teste. Os testes relacionados à Interface serão conduzidos manualmente assim que a interface estiver integrada ao código-fonte e disponível para testes.