deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

TQS: Relatório de Controlo de Qualidade

João Vasconcelos [88808], Tiago Mendes [88886], Vasco Ramos [88931]

v2020-05-05

1. Bookmarks do Projeto

Sistematização dos links para os recursos desenvolvidos no projecto:

- Acesso ao(s) projecto(s) de código, bem como GitLab Agile e GitLab CI/CD:
    a. Repositório de Grupo (GitLab, mediante acesso)
- Ambiente de produção:
    a. Sistema em Produção (Spring Boot) ??
    b. REST API (SpringBoot) ??
    c. Documentação da API (Swagger) ??
- Ambiente de Persistência:
    a. Base de Dados (MySQL) ??
- Ambiente SQA :
    a. Análise estática (SonarQube) ??
    b. Monitorização (Nagios XI) ??
- Coordenação da equipa:
    a. Slack (aqui , mediante acesso)

2. Gestão do Projeto

2.1. Equipa e Papéis

Para facilitar a divisão de responsabilidades dentro da equipa, decidimos atribuir cargos específicos a cada elemento:

Cargo	**Descrição**	**Membro/s da equipa**
Product Owner	O product owner representa os interesses dos stakeholders. Tem um conhecimento detalhado sobre o produto e o domínio onde se insere e por isso os restantes elementos da equipa irão falar com ele para clarificar dúvidas sobre futuras funcionalidades do produto. Tem um papel ativo em aceitar os novos incrementos desenvolvidos para a solução final.	João Vasconcelos
DevOps Master	O devops master é responsável pela infraestrutura de desenvolvimento e produção, aplicando as configurações necessárias à mesma. Gere a configuração e a preparação das máquinas de deployment, repositório git, infraestrutura da cloud,operações da base de dados, entre outras responsabilidades.	Tiago Mendes Vasco Ramos
Team Manager	O team manager assegura que existe uma divisão justa de tarefas e que o plano de desenvolvimento é seguido por todos os elementos da equipa. Promove um bom ambiente dentro da equipa e assegura que os requisitos do projeto são entregues dentro dos prazos estabelecidos.	Vasco Ramos

Developer	O developer desenvolve a aplicação/projeto consoante o plano definido pelo team manager.	João Vasconcelos Tiago Mendes Vasco Ramos

2.2. Gestão do Backlog e Atribuição de Trabalho

[Description of agile practices defined in the project for backlog management (user stories oriented) and job assignment, and links to associated resources. cfr. PivotalTracker workflow]

3. Gestão e Qualidade do Código

3.1. Guia de Contribuição (Coding style)

[Definition of coding style adopted. → e.g.: AOS project]

3.2. Métricas de Qualidade de Código

[Description of practices defined in the project for *static code analysis* and associated resources.]

[Which quality gates were defined? What was the r[ationale?]

4. Pipeline de Entrega Continua (CI/CD)

4.1. Development workflow

4.1.1. SCM Workflow

Relativamente ao SCM Workflow, iremos usar **Git Feature Branch Workflow** do Bitbucket. Este subentende que:

- Sempre que um developer queira desenvolver uma nova feature/user story, este deverá fazer o seu desenvolvimento numa nova branch, especificamente criada para o efeito. Esta branch deve ser denominada de forma a que permita rapidamente identificar qual a feature/issue a ser tratada, utilizando o padrão **feature/<feature_name>**.
- Além disso, definimos que sempre que seja necessário corrigir algum bug devemos criar uma nova branch a partir do master, com o seguinte formato: **hotfix/<fix_name>**.
- Na nova branch criada, o developer edita e dá commits das suas implementações. Para além disto, este developer pode também dar push da sua branch para o repositório central, onde esta irá ser armazenado (backup).

- Quando um developer acabar de desenvolver as features associadas à branch que criou, este terá de criar um merge request para que a sua branch seja unida com a master branch. Desta forma, os outros membros da equipa irão receber uma notificação referente a esta situação.
- Os outros developers da equipa dão feedback sobre o código a ser inserido na master branch. Após isto, este código poderá ter de ser reformulado. Assim que o código for aprovado pelos reviewers, a branch onde está a nova feature será merged com a master branch.
- Por fim, define-se também (como acrescento ao flow em que nos baseámos) que todos os merge requests têm de ser aprovados por, pelo menos, um reviewer que não seja a próprio que submeteu o merge request.

Para mais detalhes, pode ser encontrada uma maior descrição do funcionamento deste workflow aqui.

(INCLUIR UMA IMAGEM DO ESTADO FINAL DE APLICAÇÃO DESTE WORKFLOW)

4.1.2. Revisão de Código

De forma a melhorar a qualidade geral do código produzido, é necessário que este seja revisto por diversos developers, de forma a que se encontrarem erros e potenciais situações de riscos. Para que este processo decorra eficazmente, definimos um conjunto de princípios a seguir:

- Uma code review tem como objetivo uma análise minuciosa do código submetido. Tentar entender apenas algumas partes do código poderá, a longo prazo, ter consequências severas, pelo que é normal que uma code review demore uma elevada quantidade de tempo;
    - Caso alguma porção de código não seja percetível, o developer que o escreveu deverá reformular/explicar esta secção, sendo que os comentários não devem ser esquecidos;
    - Não se pode assumir que o código submetido funciona. É necessário fazer build do projeto e correr todos os testes associados ao mesmo. O reviewer poderá até criar novos testes;
    - Caso o código não tenha comentários explicativos, este deve ser documentado corretamente pelo developer que o escreveu;
    - É necessário rever, também, o código “temporário”, uma vez que este se poderá tornar em código para produção;
    - Deve ser realizada uma review, quer aos testes, quer aos *build files* associados a código que está a ser revisto;
    - As reviews de código devem ter em atenção se o code style está de acordo com o definido no início do projeto;
    - A arquitetura de uma solução poderá, também esta, ser revista;
    - Os comentários de uma code review devem ser críticas construtivas;
    - Ao fazer uma code review, as sugestões devem ser feitas de acordo com a seguinte prioridade:
        - Melhorias a nível funcional;
        - Alterações para manter o código clean e fácil de manter;
        - Por fim, sugestões para otimizar o código.

- Acompanhar o estado de uma code review é tão importante como fazer a code review, pelo que cada developer deverá fazer o follow up das reviews que fez.

Relativamente ao processo de code review implementado neste projeto, este tem como suporte as ferramentas disponibilizadas pelo GitLab. Sempre que é feito um merge request, inicia-se um processo de code review do código submetido.

Por fim, para esclarecimento futuro, uma user story é considerada como terminada após completar todo este processo de: Merge Request -> Build e Execução de Testes Automáticos -> Code Review e Validação de Código -> Deployment. Ou seja, uma user story é concluída quando esta é deployed com sucesso, respeitando o flow previamente especificado.

4.2. Ferramentas e Pipelines de CI/CD

[Description of the practices defined in the project for the continuous integration of increments and associated resources. Provide details on the tools setup and config.]

[Description of practices for continuous delivery, likely to be based on *containers*]

4.3. Repositório de Artifacts e Containers [Opcional]

[Description of the practices defined in the project for local management of Maven *artifacts* and associated resources. E.g.: artifactory]

5. Testes de Software

5.1. Estratégia de Teste

[what was the overall test development strategy? E.g.: did you do TDD? Did you choose to use Cucumber and BDD? Did you mix different testing tools, like REST-Assured and Cucumber?...]

[it is not to write here the contents of the tests, but to explain the policies/practices adopted and generate evidence that the test results are being considered in the IC process.]

5.2. Testes Funcionais/Aceitação

[Project policy for writing functional tests (closed box, user perspective) and associated resources.]

5.3. Testes Unitários

[Project policy for writing unit tests (open box, developer perspective) and associated resources.]

5.4. Testes de Sistema e Integração

[Project policy for writing integration tests (open or closed box, developer perspective) and associated resources.]

API testing

5.5. Testes de Desempenho [Opcional]

[Project policy for writing performance tests and associated resources.]