CLÁUDIO FILIPE CARVALHO HENRIQUES

MODELOS DE MATURIDADE DE TESTES DE SOFTWARE E O FUTURO UTILIZANDO *MACHINE LEARNING* E INTELIGÊNCIA ARTIFICIAL

SOFTWARE TESTING MATURITY MODELS AND FUTURE USING MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Universidade de Aveiro Departamento de Matemática
2021

CLÁUDIO FILIPE CARVALHO HENRIQUES

MODELOS DE MATURIDADE DE TESTES DE SOFTWARE E O FUTURO UTILIZANDO *MACHINE LEARNING* E INTELIGÊNCIA ARTIFICIAL

SOFTWARE TESTING MATURITY MODELS AND FUTURE USING MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações (Computação e Informática), realizada sob a orientação científica do Doutor João Pedro Antunes Ferreira da Cruz, Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro, e do (co-orientador) Rafael Peixinho, orientador na empresa Bosch Termotecnologia.

o júri / the jury

presidente / president

Prof. Doutor Eugénio Alexandre Miguel Rocha
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor João Paulo Silva Barraca
Professor Auxiliar da Universidade de Aveiro

Prof. Doutor João Pedro Antunes Ferreira da Cruz
Professor Auxiliar da Universidade de Aveiro

agradecimentos / acknowledgements

Agradeço a todos aqueles que me acompanharam ao longo desta fase e que tornaram possível a concretização deste meu objetivo. Ao meu orientador João Pedro Cruz, por ter aceite o meu convite para me orientar e me ter apoiado e confiado no meu trabalho. Aos meus colegas de trabalho Luís Figueiredo e Gil Mesquita, por me permitirem que estivesse envolvido em algumas das suas tarefas diárias e assim conseguir aprender mais e trabalhar com outras ferramentas com as quais não tinha qualquer tipo de experiência, alargando o meu espectro de aprendizagens neste estágio. Ao Tiago Lucas, por ter estado presente durante todo este período de estágio com a sua boa disposição mas acima de tudo com a sua vontade de trabalhar e o seu espírito de ajuda que permitiram o desbloqueio de muitas das dificuldades encontradas. Por fim, mas não menos importante, ao Rafael Peixinho, por se ter tornado não só num orientador mas num mentor que levo para a continuação da minha carreira profissional. Mais do que um tutor ou colega de trabalho, aprendi com ele a saber lidar com algumas situações adversas no mundo empresarial. Mais do que profissionalmente, levo daqui uma amizade.
Quero ainda agradecer à minha família, que sempre me apoiaram:

- Irmã, por me ter dado a conhecer a realidade universitária e assim tornar a minha decisão um pouco mais fácil e segura para o meu futuro. Para além disso, foi nela que vi o exemplo a seguir e a motivação para conseguir concluir esta longa etapa. Obrigado!

- Mano, por me ter acompanhado desde sempre e partilhar comigo a paixão pela programação. Pelas nossas conversas e discussões, pelas aprendizagens e trocas de experiências.
  - Sei que vou ser uma melhor pessoa e profissional por ter o privilégio de partilhar a minha vida contigo todos os dias.

- Margarida, por ter estado presente e ter apoiado as minhas decisões. Por me ter aconselhado e ter sempre uma palavra de conforto para dar. Sei que tenho sempre uma amiga para me dizer o que precisa de ser dito e não apenas para os bons momentos.

- Ana, por ter estado sempre ao meu lado ao longo destes últimos anos. Por, apesar de não estar familiarizada com muitos dos termos técnicos, ouviu sempre as minhas dificuldades no trabalho e esteve sempre lá para meus desabafos, para me aconselhar do melhor caminho a seguir e para ser um melhor profissional todos os dias. Sei que dias difíceis passou, pelo stress e indecisões que trazia do meu dia na empresa, mas espero que os bons momentos compensem os menos bons. Obrigado pela persistência. Sem ela sei que seria mais difícil ter conseguido chegar à realização desde grande objetivo.

- Aos meus pais, por me terem dados as ferramentas necessárias para estar aqui hoje. Espero que se orgulhem tanto de mim como eu tenho de orgulho neles. Espero um dia conseguir ser para os meus filhos o que eles foram para mim!

Palavras Chave

modelo de maturidade, inteligência artificial, *machine learning*, desenvolvimento *web*, *JavaScript*, *NodeJS*, *Jenkins*, *Docker*, qualidade de *software*, testes de *software*

Resumo

O processo de testes de *software* é cada vez mais considerada como uma das etapas mais importantes para garantir o seu bom funcionamento e robustez. Desta forma é importante conseguirmos medir de forma qualitativa a maturidade das equipas de desenvolvimento no que diz respeito ao processo utilizado para desenvolver estes mesmos testes de *software*. Assim, o objetivo deste estágio é encontrar um modelo de maturidade que possa ser aplicado de uma forma prática e que sirva como um método de auto avaliação das diversas equipas de *software* que compõem a divisão de termotecnologia da *Bosch* em Portugal

O futuro foi igualmente merecedor da nossa atenção no decorrer do estágio, onde procuramos ferramentas que utilizem as tecnologias mais avançadas, nomeadamente *machine learning* e *artificial intelligence*, para a definição de casos de testes de *software* que devem ser tidos em conta no momento da sua implementação.

Por fim este relatório apresenta ainda algumas tarefas que não estavam inicialmente previstas com o objetivo de tornar esta experiência curricular ainda mais completa e aumentar o conhecimento adquirido nesta área.

Keywords

maturity model, artificial intelligence, machine learning, web development, JavaScript, NodeJS, Jenkins, Docker, software quality, software tests

Abstract

The software testing process is increasingly considered one of the most important steps to ensure its proper functioning and robustness. Thus, it is important to be able to qualitatively measure the maturity of development teams with respect to the process used to develop these same software tests. Thus, the goal of this internship is to find a maturity model that can be applied in a practical way and that serves as a method of self-assessment of the various software teams that make up the division of thermotechnology Bosch in Portugal.

The future was also worthy of our attention during the internship, where we seek tools that use the most advanced technologies, including machine learning and artificial intelligence, for the definition of software test cases that must be taken into account when implementing them.

Finally, this report also presents some tasks that were not initially foreseen with the objective of making this curricular experience even more complete and increasing the knowledge acquired in this area.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

TT	*Thermotechnology*
CE	*Center of Engineering*
ML	*Machine Learning*
AI	*Artificial Intelligence*
PoC	*Proof of Concept*
IoT	*Internet of Things*
BSH	*Bosch*
KG	*Kommanditgesellschaft* (Parceria Limitada)
GmbH	*Gesellschaft mit beschränkter Haftung* (Sociedade Limitada)
CI	*Continuous Integration*
CD	*Continuous Delivery* ou *Continuous Deploy*

CAPÍTULO 1

Introdução

Atualmente, são cada vez mais as empresas dedicadas à investigação e desenvolvimento de software. Estes sistemas são cada vez mais complexos e ganham, a cada dia, responsabilidades no quotidiano, havendo por isso menos tolerância a falhas inesperadas que alterem a rotina de cada um de nós. Assim, as equipas de desenvolvimento procuram estabelecer procedimentos e normas que permitam lançar os seus produtos para o mercado com mais segurança e fiabilidade. Para tal, podemos considerar que a fase de testes de software é cada vez mais indispensável no ciclo de vida do seu desenvolvimento.
O foco deste trabalho vai ao encontro da fase mencionada anteriormente, onde é proposto avaliar o nível de maturidade no processo de testes de software de diversas equipas de desenvolvimento que compõem a empresa Bosch Termotecnologia.
É ainda igualmente importante pensar no futuro, onde inevitavelmente nos cruzamos com termos como Machine Learning e Inteligência Artificial, e estudarmos como é que estas tecnologias nos podem ajudar a estabelecer normas/regras que otimizem tempo e recursos ao mesmo tempo que mantemos ou até melhoramos o nível fiabilidade do software desenvolvido.

1.1 Visão geral

A realização do estágio curricular teve lugar nas instalações do *Center of Engineering* (CE) da empresa *Bosch Thermotechnology*, em Aveiro (Cacia), integrado numa equipa focada em qualidade de *software*.

1.2 Plano e objetivos

1.2.1 Componentes principais do estágio

O estágio curricular divide-se em duas fases:

- Primeiramente pretende-se avaliar o nível de maturidade no que diz respeito ao processo de testes de *software* das diversas equipas que compõem a Bosch Termotecnologia em Aveiro. Para tal, é necessário conhecer o trabalho desenvolvido por cada equipa, os

recursos utilizados (linguagens de programação, *frameworks*, etc.), os diversos modelos de maturidade de testes de *software* e finalmente construir uma ferramenta que permita que esta avaliação seja realizada por cada equipa, de forma independente, e que forneça informações que ajudem a estabelecer um plano de melhoria nos pontos mais debilitados de todo o processo de testes de *software*.

- Na segunda fase do estágio pretende-se investigar soluções existentes no mercado, interno e externo à Bosch, no que diz respeito à utilização de inteligência artificial e *machine learning* na conceção de cenários de testes com base nos requisitos e casos de uso de uma dada ferramenta de software. Neste sentido é importante estabelecer contactos com os autores/empresas das soluções que vão ao encontro do panorama referenciado e por fim realizar uma prova de conceitos utilizando um dos softwares selecionados anteriormente.

1.2.2 Planificação detalhada do estágio

Os pontos em seguida descritos são um mapeamento com as diferentes etapas representadas na figura 1.1 seguidos de uma breve descrição do objetivo principal de cada uma das diferentes fases deste estágio curricular.

1. **Ferramentas de testes e estratégias de qualidade** (*Software testing tools*) - Recolha de informações relativas ao processo de testes de *software*, tais como:
   - Ferramentas de testes;
   - Linguagens de programação;
   - Estratégias de qualidade.

   Para a realização desta tarefa serão utilizados como recursos os dados disponíveis em aplicações internas, documentação e entrevistas às diversas equipas de desenvolvimento.
2. **Modelos de maturidade de testes de software** (*Software testing maturity model overview*) - Pesquisa e análise dos diversos modelos de maturidade de testes de *software*, tendo como objetivo posterior a escolha de um para implementar na divisão de termotecnologia da Bosch.
3. **Aplicação *web* - Modelo de maturidade de testes de *software*** (*Evaluate software testing maturity model*) - Criação de uma ferramenta, para uso interno, que implemente o modelo escolhido na etapa anterior.
4. **Investigação sobre o estado atual da aplicação do *Machine Learning* e Inteligência Artificial em testes de *software*** (*Software testing tools using ML and AI*) - Consulta de artigos relacionados com a utilização do *Machine Learning* e da Inteligência Artificial no processo de testes de *software* e investigação sobre as atuais soluções no mercado interno e externo da empresa Bosch.
5. **Prova de conceitos** (PoC) - Realização de uma demonstração onde é aplicada uma ferramenta/*software* que utlize o *Machine Learning* e a Inteligência Artificial aplicada ao processo de testes de *software*.

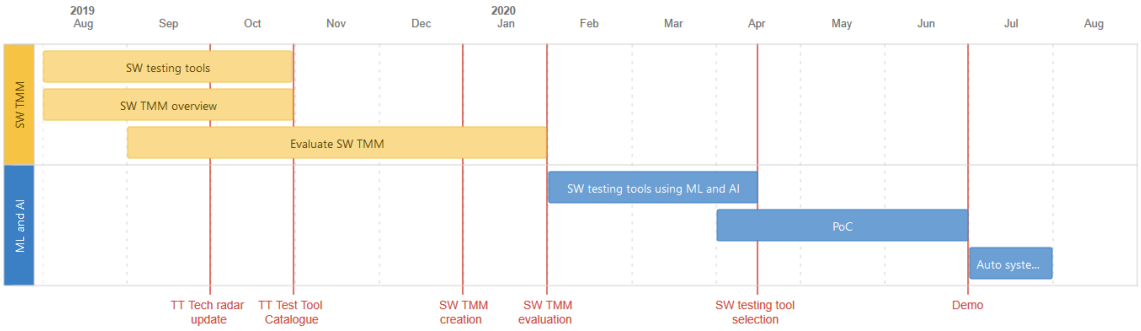

Figura 1.1: Planificação das etapas do estágio

É ainda importante referir que este relatório apresenta tarefas não contempladas no plano inicial, pré-definido pela entidade empresarial, mas que foram surgindo ao longo do percurso. Estas novas tarefas têm como principal objetivo ampliar o espectro de conteúdos académicos e empresariais, contribuindo para o enriquecimento do resultado final do estágio e ainda para a ampliação pessoal de competências e conhecimentos. Assim sendo, é fácil encontrarmos discrepâncias quando comparamos o plano estabelecido previamente com a realização das tarefas ao longo do tempo.

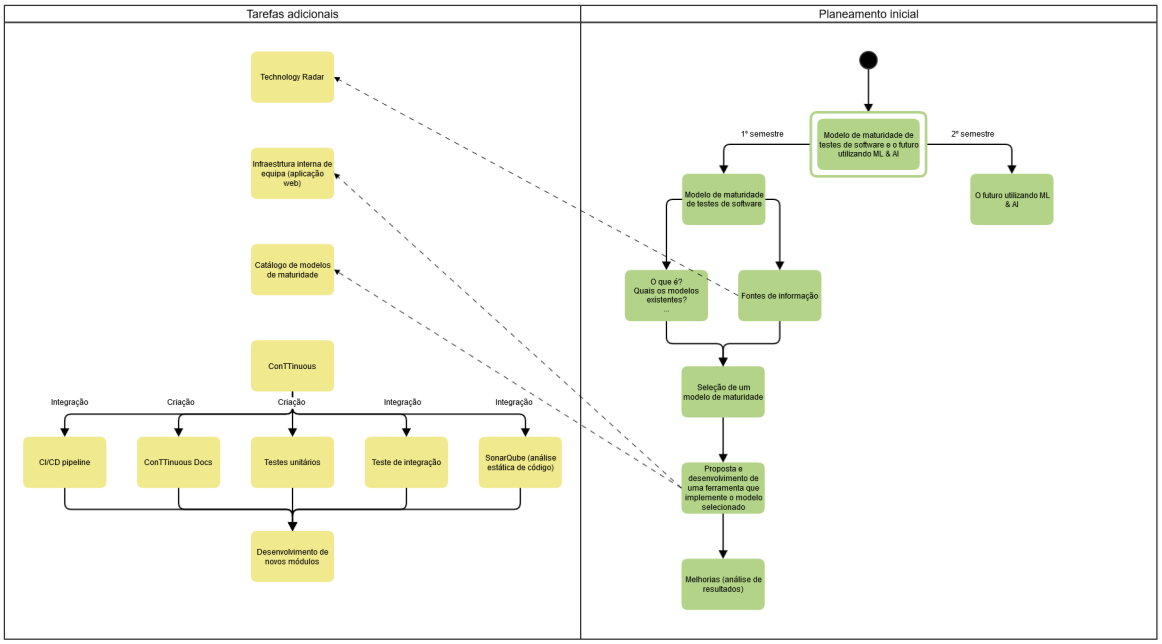

Figura 1.2: Mapeamento das tarefas planeadas e adicionais do estágio

Na figura 1.2 estão representadas a verde as tarefas inicialmente previstas no plano do estágio curricular e a amarelo as tarefas desempenhadas paralelamente, com o objetivo de enriquecer o meu conhecimento a nível técnico e ainda de desenvolver um estágio mais completo. As setas apresentadas neste diagrama representam o mapeamento de algumas das tarefas adicionais com tarefas inicialmente planeadas.

1.3 Estrutura do documento

O presente documento apresenta um capítulo com revisão de conteúdos e conceitos fundamentais para a boa compressão do trabalho desenvolvido ao longo do tempo. Os capítulos subsequentes focam-se cada um deles numa das tarefas desempenhadas e objetivam detalhar o trabalho realizado em cada uma delas. Por fim, é apresentado um capítulo conclusivo onde descrevo as dificuldades encontradas ao longo do estágio e ainda as aprendizagens e vantagens que encontro na sua realização.

- Capítulo 2. Revisão de algumas definições e conceitos importantes para o melhor entendimento de alguns termos descritos durante este documento;
- Capítulo 3: Descrição do trabalho realizado para coletar todos os dados importantes relativamente a estratégias e ferramentas utilizadas pelas equipas no que diz respeito a testes de *software*.
- Capítulo 4. Razões e fundamentos para a escolha de um modelo de maturidade de testes de *software* e sua aplicação.
- Capítulo 5. Descrição do desenvolvimento das diversas aplicações *web* ao longo de todo o período de estágio.
- Capítulo 6. Referências ao trabalho realizado em paralelo no projeto *Jenkins Library*.
- Capítulo 7. Pesquisa e escolha das ferramentas que utilizam *Machine Learning* e Inteligência Artificial aplicadas a testes de *software* e ainda o desenvolvimento de uma prova de conceitos.
- Capítulo 9. Considerações finais.

1.4 Entidade empresarial

1.4.1 Bosch

Fundada em Estugarda em 1886 por Robert Bosch (1861-1942) como uma "Oficina de precisão mecânica e engenharia elétrica", está atualmente presente no mercado mundial.

A estrutura acionista da Robert Bosch GmbH garante a autonomia empresarial do Grupo Bosch, tornando possível o planeamento a longo prazo e a realização de investimentos significativos para salvaguarda do seu futuro. Noventa e dois por cento das ações da Robert Bosch GmbH são detidas pela Fundação Robert Bosch, uma fundação beneficente. A maioria dos direitos de voto é detida pela Robert Bosch Industrieltreuhand KG, uma *trust* industrial a quem está confiada a gestão dos activos empresariais. As restantes acções são detidas pela família Bosch e pela Robert Bosch GmbH.

O Grupo Bosch é líder mundial no fornecimento de tecnologia e serviços. A empresa emprega mais de 390.000 colaboradores em todo o mundo (a 31.12.2016), que contribuíram para gerar uma faturação de 73,1 mil milhões de euros em 2016. As operações do Grupo estão divididas em quatro áreas de negócio: Soluções de Mobilidade, Tecnologia Industrial, Bens de Consumo, e Tecnologia de Energia e Edifícios. Líder em IoT, a Bosch oferece soluções inovadoras para casas e cidades inteligentes, mobilidade e indústria conectada. A empresa

utiliza o seu conhecimento em tecnologia de sensores, *software* e serviços, bem como a sua própria *cloud* IoT para oferecer aos seus clientes soluções conectadas e em diversos domínios a partir de uma única fonte. O objetivo estratégico da Bosch é fornecer inovações para uma vida conectada. Os produtos e serviços do Grupo Bosch são concebidos para cativar e melhorar a qualidade de vida das pessoas através de soluções inovadoras e úteis. Desta forma, a empresa oferece mundialmente "Tecnologia para a Vida". O Grupo Bosch é composto pela Robert Bosch GmbH e cerca de 450 subsidiárias e empresas regionais presentes em aproximadamente 60 países. Incluindo os representantes de vendas e serviços, a rede mundial de desenvolvimento, produção e distribuição da Bosch está presente em quase todos os países. A sua força inovadora é a base para a continuidade do crescimento da empresa. Em cerca de 120 localizações em todo o mundo, a Bosch emprega 59.000 colaboradores em investigação e desenvolvimento.

1.4.2 Unidades de negócio

Como referido no capítulo anterior, a empresa divide-se em 4 principais unidades de negócio, cada uma com as respectivas subdivisões. Estas vêm enumeradas abaixo:

- Soluções de Mobilidade
  - *Powertrain solutions*
  - *Chassis System Control*
  - *Electrical Drives*
  - *Car Multimedia*
  - *Automotive Electronics*
  - *Automotive Afertmarket*
  - *Automotive Steering*
  - *Connected Mobility Solutions*
- Tecnologia Industrial
  - *Drive and Control Technology*
  - *Packaging Technology*
- Bens de Consumo
  - *Power Tools*
  - *Household Appliances*
- Tecnologia de Energia e Construção
  - *Building Technologies*
  - *Thermotechnology*
  - *Bosch Global Service Solutions*

1.4.3 Bosch Portugal

A Bosch é representada em Portugal pela Bosch *Thermotechnology*, em Aveiro, a Bosch *Car Multimedia* Portugal, em Braga, e a Bosch *Security Systems*, em Ovar. Nestas localizações, a empresa desenvolve e fabrica, respectivamente, soluções de água quente, multimédia automóvel e sistemas de segurança e comunicação, 95% dos quais exportadas para os mercados internacionais. A sede nacional do grupo está situada em Lisboa, onde são realizadas atividades de

vendas, *marketing*, contabilidade e comunicação, bem como serviços partilhados de recursos humanos e comunicação para o Grupo Bosch. Além disso, a empresa possui ainda uma subsidiária da BSH Eletrodomésticos, em Lisboa. Com mais de 4.000 colaboradores, a Bosch é um dos maiores empregadores industriais de Portugal e gerou, em 2016, 1,1 mil milhões de euros em vendas internas.

1.4.4 Bosch Termotecnologia

A divisão de termotecnologia fornece produtos de aquecimento com eficiência energética e soluções de água quente (essencialmente a nível residencial), principalmente na Europa. Os produtos da divisão são vendidos sob marcas internacionais e regionais, como Bosch, Buderus, Worcester e Junkers. O portfólio de produtos inclui desde aquecedores, bombas de calor, sistemas solares térmicos e caldeiras de combustível sólido, até plantas de cogeração e caldeiras industriais.

Tendo em vista funcionalidades como a monitorização e diagnóstico remotos, os dispositivos preparados para se ligarem à internet tornam-se cada vez mais importante. É aqui que são desenvolvidas soluções tecnológicas na área da termotecnologia focadas na conectividade, eficiência energética e em casas inteligentes e sustentáveis. Entre as inovações criadas no Centro de Competências estão o esquentador conectado Geos e vários tipos de *software* para plataformas *web* e *mobile* de várias áreas de negócio da Bosch.

Na visita a esta unidade, assim como em Braga, foi possível observar como todas as linhas de montagem estão ligadas. Os dados recolhidos são, depois, analisados para dar importantes *insights* que ajudam a melhorar a eficiência e ajudar os gestores na tomada de decisão. Em Aveiro, a empresa usa ainda cinco robôs autónomos que transportam produtos terminados para o armazém e trazem componentes para as linhas de montagem. "A Indústria 4.0 é já uma realidade nas nossas fábricas", referiu Carlos Ribas, que apontou ainda como os dados são vitais para avaliar a qualidade de tudo o que é produzido pela Bosch.

CAPÍTULO 2

Revisão e conceitos

Neste capítulo são referenciados alguns conteúdos relevantes para um melhor entendimento dos temas abordados nos capítulos que se seguem.

2.1 Ambiente de desenvolvimento

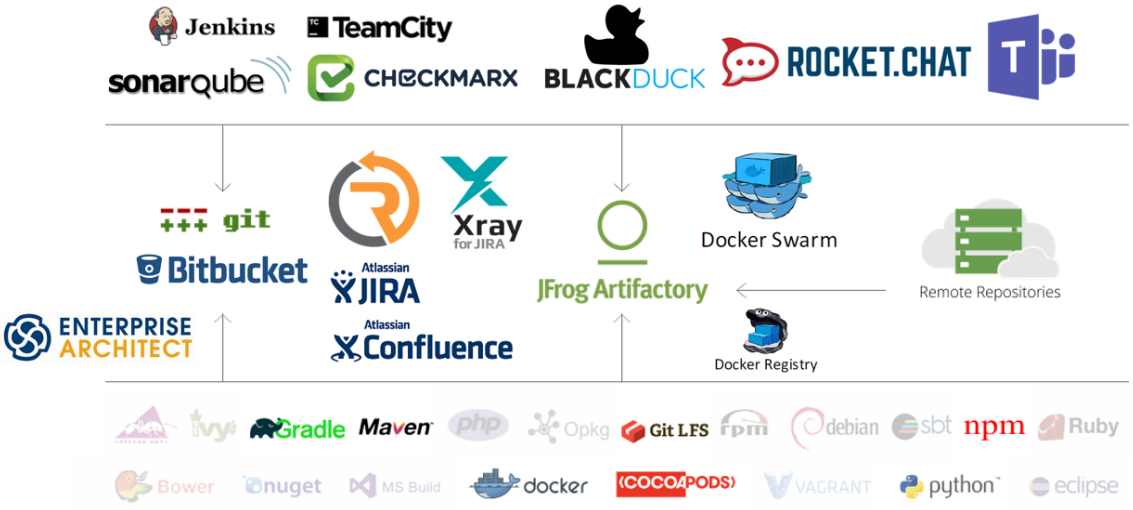

Figura 2.1: Ambiente de desenvolvimento aplicado na divisão de termotecnologia da Bosch

Como se pode reparar na figura 2.1, existe uma vasta gama de ferramentas aplicadas ao desenvolvimento de *software*, tais como ferramentas de controlo de versões, *Continuous Integration*/*Continuous Delivery* ou *Continuous Deploy*, requisitos de engenharia e comunicação. De modo geral, esta figura (2.1) representa o ecosistema de desenvolvimento utilizado pelas diversas equipas de software da Bosch Termotecnologia em Aveiro.

2.1.1 Controlo de versões

O controlo de versões é um sistema que permite preservar as diferentes alterações realizadas a ficheiros ou projetos, tornando possível a navegação para versões anteriores. O mais popular, e também o utilizado ao longo do desenvolvimento deste trabalho, é o ***Git***. Este sistema é aplicado em repositórios, sendo os mais comuns o ***GitHub***, ***GitLab*** e o ***Bitbucket***. Este sistema permite ainda o sincronismo de alterações em ficheiros e projetos entre vários utilizadores.

Para a correta utilização deste sistema são utilizados maioritariamente 4 comandos principais:

- ***git pull***: Para obter a última versão dos ficheiros disponíveis no repositório remoto;
- ***git add***: Para indexar alterações realizadas;
- ***git commit***: Para adicionar as alterações no repositório local e atribuir uma mensagem descritiva das alterações indexadas;
- ***git push***: Para enviar as alterações realizadas no repositório local para o repositório remoto;

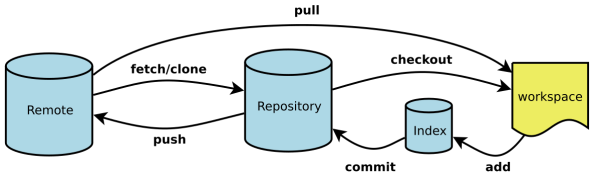

Figura 2.2: Diagrama do fluxo dos comandos git

2.1.2 CI/CD

Do inglês *Continuous Integration, Delivery and Deployment*, o termo *CI/CD* consiste em três conceitos principais;

- Integração contínua
  - ***Git***: Para o controlo de versões;
  - ***Bitbucket***: Repositório remoto;
  - ***Jenkins***: Servidor para a automação do desenvolvimento de software (construção, testes e implementação);
  - ***SonarQube***: Análise estática de código;
  - ***Maven***: Ferramenta para gestão e interpretação de projetos de *software*.
- Entrega contínua
  - ***Docker***: Plataforma que permite criar e executar blocos de software, denominados por contentores. Com este sistema podemos colocar as nossas aplicações num contentor que possui todos os recursos necessários para testar, implementar e publicar mais rapidamente;
  - ***Jfrog Artifactory***: Gestão de repositório binário;
  - ***Kubernetes***: Automação e orquestração das operações de contentores.

2.1.3 Requisitos de engenharia

- ***JIRA***: Ferramenta para a gestão de tarefas e projetos, permitindo associar a cada pessoa as atividades a desempenhar;
- ***Confluence***: Ferramenta *web* ao estilo da *Wikipédia*, que permite a criação de diversas páginas de documentação e partilha entre os diversos utilizadores.

2.1.4 Comunicação

- Microsoft Teams

CAPÍTULO 3

Ferramentas de testes e estratégias de qualidade

3.1 Pré-análise

Como referido no capítulo inicial, com o objetivo de conhecer o trabalho realizado por cada equipa que compõe a divisão de Termotecnologia da Bosch em Aveiro recorreu-se a dados disponíveis em ferramentas internas da empresa (como é o exemplo da aplicação *web Technology Radar* descrita posteriormente), documentação relacionada com a estratégias de qualidade e ainda a entrevistas com cada uma das equipas.

Todo este processo foi planeado de forma a otimizar o tempo disponível. Para tal, foram recolhidas atempadamente as informações relacionadas com os protocolos e metodologias de testes de *software* de cada uma das equipas para que à data da entrevista houvesse apenas uma pequena introdução para inteirar os intervenientes dos objetivos do estágio e em seguida uma validação da informação recolhida.

Na tabela 3.1 podemos ver o planeamento das entrevistas durante os meses de agosto e setembro de 2019.

Equipa	**Data**
Team Backend 1	26 Aug 2019
Team Embedded 1	27 Aug 2019
Team Mobile 1	29 Aug 2019
Team Embedded 2	30 Aug 2019
Team Backend 2	02 Sep 2019
Team Backend 3	03 Sep 2019
Team Embedded 3	04 Sep 2019
Team Embedded 4	05 Sep 2019
Team System Tests 1	06 Sep 2019
Team Mobile 2	10 Sep 2019
Team Backend 4	12 Sep 2019
Team Web App 1	13 Sep 2019
Team Backend 5	16 Sep 2019
Team Embedded 5	17 Sep 2019
Team Fullstack 1	20 Sep 2019
Team Fullstack 2	20 Sep 2019

Tabela 3.1: Agenda das entrevistas introdutórias com cada equipa

3.2 Entrevistas

Nesta etapa foram abordados tópicos relacionados com metodologias, técnicas e níveis de testes de *software*, sendo apresentado um documento a cada equipa para a verificação e validação das informações recolhidas em ferramentas internas e documentação disponibilizadas. Em seguida encontramos os tópicos abordados no documento:

- Metodologias;
  - *White-box*;
  - *Grey-box*;
  - *Black-box*;
- Técnicas;
  - Análise de valor limite;
  - Teste baseado em risco;
  - Análise *short-break*;
  - Verificação de *logs*;
  - Técnicas de diretórios;
- Níveis;
  - Unitários;
  - Integração;
  - Sistema;
  - Aceitação;
- Tipos;
  - *Performance*/*Stress*/Carregamento;
  - Segurança;

  - Interface de usuário;
  - Exploratório;
  - Regressão;
  - Penetração;
  - Caminho;
  - Segurança/Tempo;
- Cobertura de código (Verificação e validação);
- Agendamento de testes;
- Entregas de testes;
- Seguimento de falhas e vulnerabilidades (segurança);
- Revisão de estratégias;
- Ferramentas;
- *Frameworks*.

Estas entrevistas tomaram em média a duração de 30 minutos e contaram cada uma delas com a participação de um ou dois representantes da equipa em questão. Em muitos dos casos foi necessária a validação de algumas informações com a restante equipa de desenvolvimento, situação que em geral demorou 1 semana desde a data da entrevista até obter o *feedback* final.

3.3 Análise dos dados

Com o objetivo de centralizar a informação recolhida, e tendo por base outros exemplos já existentes noutras equipas/divisões da Bosch, foi criado um catálogo de ferramentas de testes de *software*. Esta documentação possibilita assim a acessibilidade por parte de todos sobre quais as tecnologias que estão a ser atualmente utilizadas e quais as mais indicadas para as diferentes tarefas do processo de testes de *software* (ver figura 3.1).

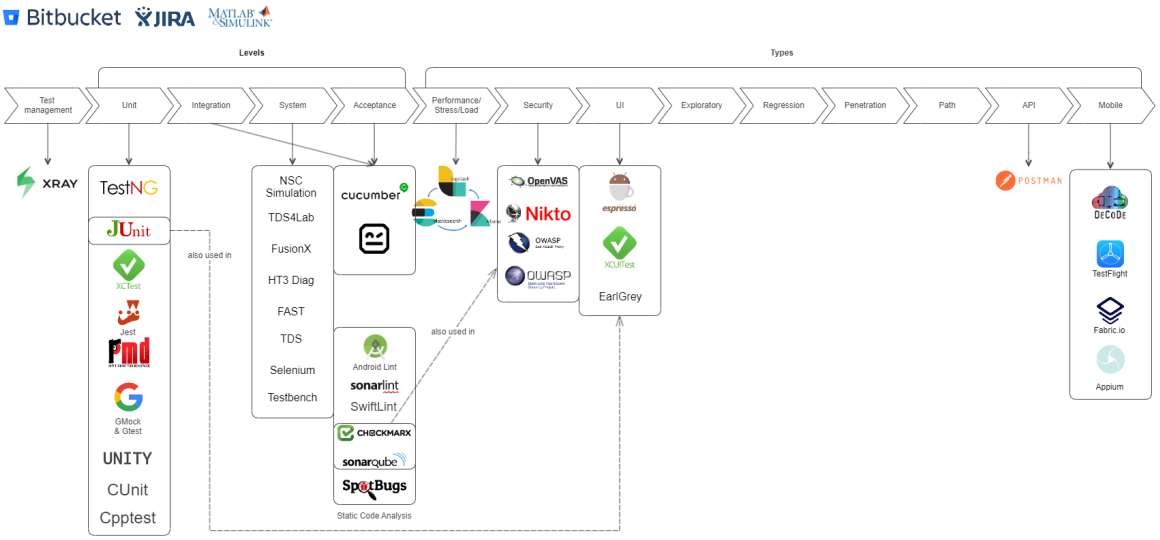

Figura 3.1: Categorização das ferramentas de testes de *software*

Após esta etapa foram detetadas algumas discrepâncias entre os dados apresentados na ferramenta interna *Technology Radar* e os dados validados pelas equipas. Assim, procedeu-se a uma atualização desta ferramenta, descrita mais à frente neste documento.

CAPÍTULO 4

Modelos de maturidade de testes de *software*

Neste capítulo é apresentada uma síntese dos processo de melhoria de testes de software, os vários modelos de maturidade existentes e a forma como os podemos implementar. Por fim, são apresentadas as razões e motivos que nos levaram a escolher o modelo TPI Next.

4.1 O que é um modelo de maturidade?

Um modelo de maturidade é um conjunto critérios definidos para um determinado domínio, organizados por diferentes níveis, de modo a estabelecer a maturidade do processo utilizado no contexto em estudo. Quer isto dizer que um modelo de maturidade pode, por exemplo, ser um questionário organizado em diferentes secções e de acordo com as respostas dadas a cada uma das perguntas é possível obter uma visão geral da maturidade do processo em estudo, sendo no caso deste estágio o processo de testes de *software* [1] [2] [3].

4.2 Porquê melhorar o processo de testes de *software*?

Uma premissa para a melhoria da qualidade de um sistema é a crença de que esta é altamente influenciada pela qualidade do processo usado para o desenvolver. Podemos ainda dizer que os atributos de qualidade internos e externos estão diretamente relacionados com os tipos de testes utilizados para avaliar a qualidade do produto. Assim, somos levados a concluir que a melhoria do processo de testes de *software* permite obter um produto final mais confiável e de maior qualidade.

Outras razões existem para motivar a melhoria do processo de testes de *software*, dentro das quais produzir um *software* de qualidade permite reduzir custos na sua manutenção futura e consequentemente tornar-se mais competitivo a nível de mercado.

4.2.1 Benefícios da melhoria do processo de testes de *software*

Benefícios operacionais:

- Menos tempo de desenvolvimento
- Custos de produção mais baixos
- Melhor planeamento dos custos de teste
- Alinhamento entre os processos de testes internos e os objetivos de valor externo
- Menor probabilidade de falhar as *dead lines* impostas
- Redução de falhas administrativas
- Minimizar os tempos dos ciclos de testes
- Melhor identificação de riscos e gestão
- Desenvolvimento pessoal mais adequado

Benefícios comerciais:

- Mais lucros
- Maior satisfação por parte dos consumidores
- Retorno positivo do investimento
- Redução do tempo das tarefas de testes
- Redução de custos com defeitos
- Melhor reputação interna e externa
- Aumento das oportunidades de negócio
- Redução de custos de suporte

4.3 Como melhorar o processo de testes de *software*?

Naturalmente, existem alguns pontos principais sobre os quais o processo de testes de *software* se baseia com o intuito de o melhorar. Assim, é importante conseguir responder a algumas perguntas que nos ajudam a entender se o processo de testes de *software* que utilizamos é suficientemente bom para o fim ao qual é proposto:

- Visibilidade. Até que ponto é explícito o processo de testes de *software* permitindo, por exemplo, que um novo elemento da equipa de desenvolvimento o consiga entender de forma rápida e intuitiva?
- Aceitabilidade. Até que ponto o processo é, no seu todo, sustentado por uma base teórica que permita ser utilizado sem que seja colocada em causa a sua veracidade?
- Rapidez. Até que ponto o processo é rápido a alcançar o devido objetivo?
- Robustez. Até que ponto o processo consegue lidar com problemas inesperados?
- Confiabilidade. Até que ponto o processo é capaz de corresponder aos objetivos para o qual foi desenhado, isto é, consegue detetar erros antes que estes sejam detetados na fase de produção?
- Adequação. Até que ponto as atividades do processo são úteis?

4.4 Tipos de modelos de maturidade de processos

Existem duas categorias nas quais se enquadram as melhorias de processos: modelos de referência do processo e modelos de referência do conteúdo.

4.4.1 Modelo de referência de processo

Fornece um nível de maturidade com o objetivo de avaliar a capacidade de uma organização em comparação com o modelo e fornecer um conjunto de medidas para melhorar o processo.

Os modelos desta categoria tendem a ser mais rígidos. Com efeito, estes modelos dizem o que fazer para melhorar o processo.

4.4.2 Modelo de referência de conteúdo

Este tipo de modelos fornece avaliações orientadas às oportunidades de negócio, em alguns casos, avaliações comparativas das médias do setor utilizando medições objetivas. Estas avaliações podem ser usadas para estipular um conjunto de medidas que podem ajudar a melhorar o processo.

Os modelos desta categoria permitem que a organização selecione quais as práticas a adaptar de forma a melhorar os processos. Com efeito, estes modelos dizem como podem melhorar o processo.

4.5 Vantagens e desvantagens de usar um modelo já existente

Vantagens:

- Não requer esforço para o planeamento de um novo modelo
- Já implementado por outras organizações - *feedbacks* disponíveis
- Confiança - já foi utilizado antes

Desvantagens:

- Os modelos disponíveis podem não ser apropriados a determinadas organizações
- Dispendioso de implementar

[4], [5]

4.6 Modelos mais utilizados

Com base numa revisão multimodal da literatura nesta área [6], a figura 4.1 apresenta a frequência com que cada modelo é utilizado (em 2015), destacando-se os modelos TMMi [7] e TMM [8], [9] e [10] no topo desta lista.

Na figura 4.2 podemos ver o estado de desenvolvimento de cada modelo. Como se pode observar, também o modelo TPI [12] (um dos mais utilizados) encontra-se numa fase completa à semelhança do que acontece com o modelo TPI Next, mais recente e baseado no TPI. Assim estes foram os 4 modelos considerados para este estudo.

Por sua vez, a figura 4.3 representa a evolução e as relações existentes entre os modelos TPI e TMM e seus sucessores, desde 1985 até 2014.

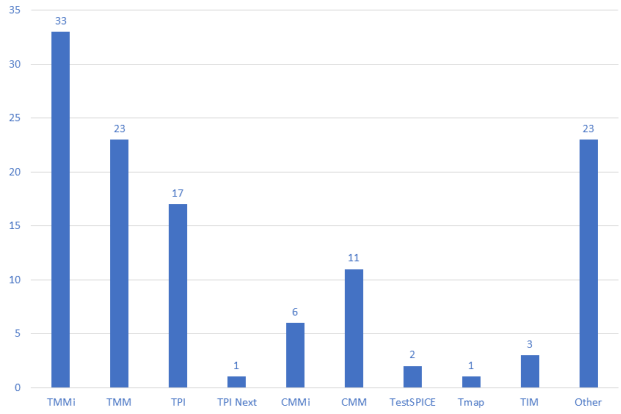

Figura 4.1: Frequência de utilização dos diversos modelos de maturidade. Fonte: [11]

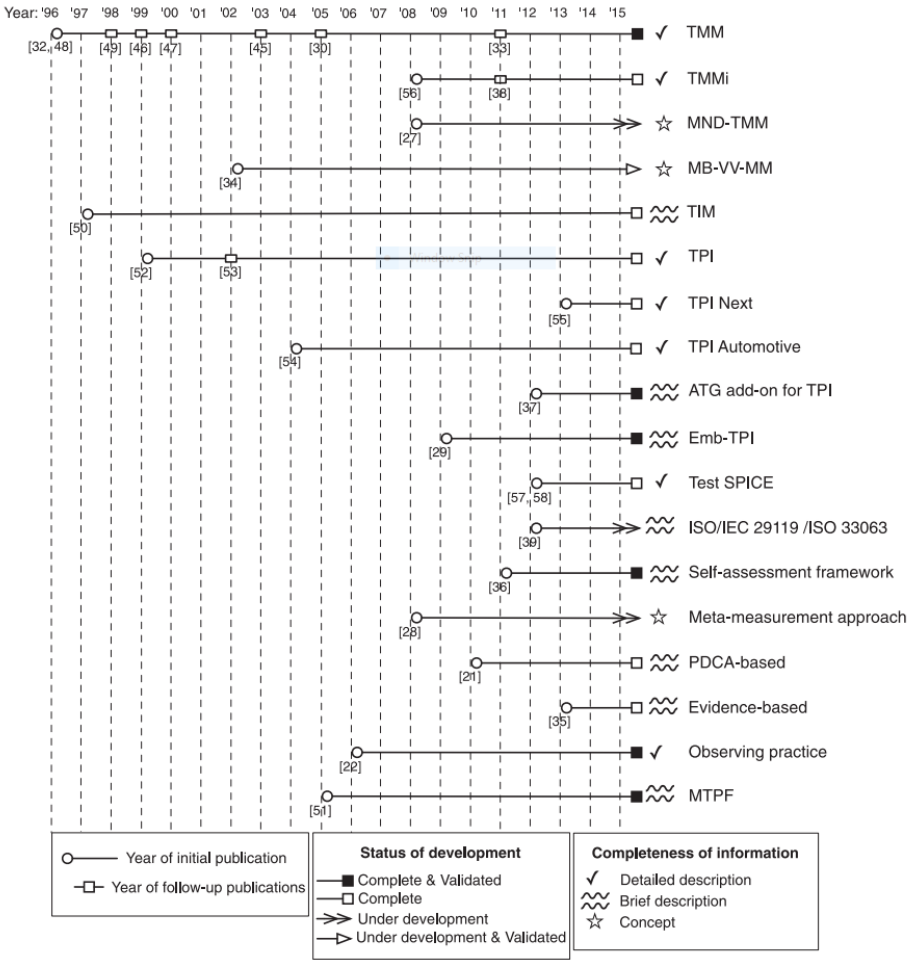

Figura 4.2: Estado de desenvolvimento dos diversos modelos em 2015. Fonte: [13]

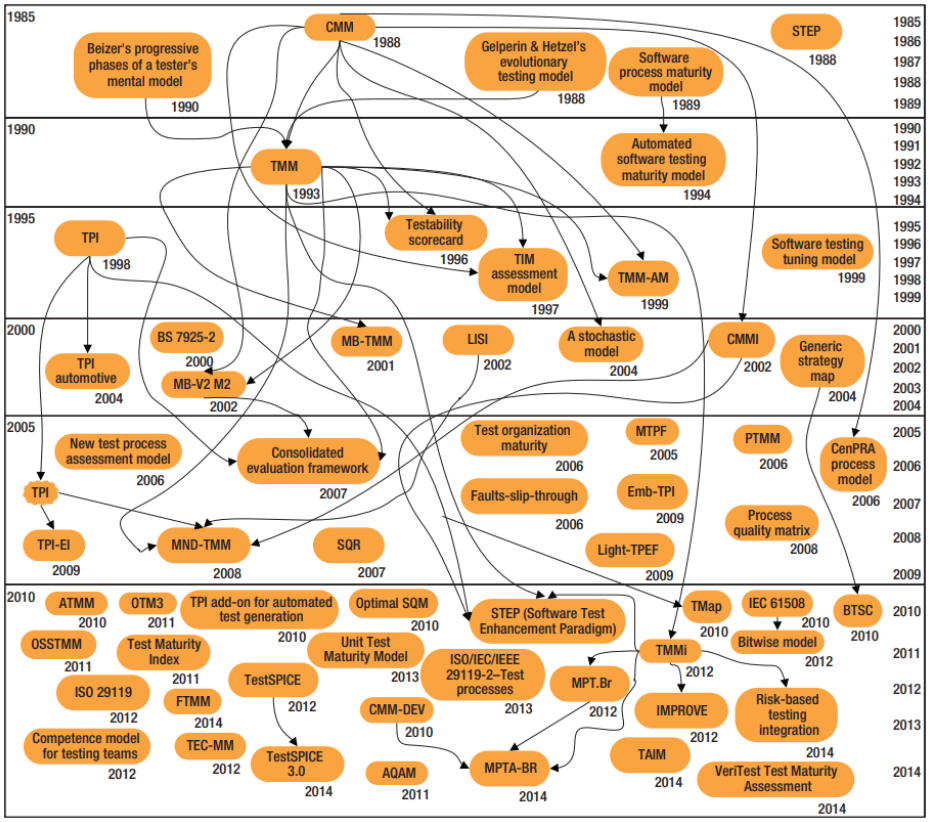

Figura 4.3: Desenvolvimento dos modelos desde 1985. Fonte: [14]

4.7 Seleção do modelo de maturidade

Em primeira instância decidiu-se utilizar um modelo já existente ao invés de criar um novo dadas as suas vantagens relativamente ao tempo de implementação [15].

	TMM	**TPI**
Objetivo	Apoiar a avaliação e os esforços de melhoria dentro de uma organização. Deve ser utilizado por: • Uma equipa de avaliação interna para identificar o estado actual das capacidades; • Uma gestão superior para iniciar um programa de melhoria de testes; • Engenheiros de garantia de qualidade de *software* para desenvolver e implementar planos de melhoria de processos; • Equipas de desenvolvimento para melhorar a eficácia dos testes; • Utilizadores/Clientes para definir o seu papel no processo de teste.	Determinar as áreas fracas e fortes do processo de teste numa organização. Além disso, a maturidade do processo pode ser avaliada e pode apoiar a determinação de actividades de melhoria.
Contém	• Modelo de maturidade; • Modelo de avaliação, contendo um procedimento de avaliação, instrumento/questionário de avaliação, formação de equipas e critérios de selecção.	• Modelo de maturidade; • Matriz de teste de maturidade; • Lista de verificações; • Sugestões de melhoria.
Implementação	Conceptual	Prática
Número de níveis	5	14
Número de áreas chave	13	20
Cobertura de atividades de teste	+	++
Tipo de avaliação	Questionário	Lista de verificação
Elementos de avaliação	• Procedimento de avaliação; • Questionário; • Formação; • Critérios de selecção de equipas.	• Guia de avaliação; • Lista de verificações; • Matriz de teste de maturidade.
Notas	Dissertações difíceis de obter	-

Tabela 4.1: Comparação entre os modelos TMM e TPI

Analisando a tabela 4.1 concluímos que o modelo TPI assume uma abordagem mais prática (facilitando assim a utilização por parte das equipas de desenvolvimento), contém mais áreas chave (permitindo obter com maior especificidade os pontos a melhorar) e ainda uma maior cobertura de atividades de teste.

Mais, sabemos que ambos os modelos (TPI e TMM) respondem afirmativamente aos seguintes critérios de avaliação:

- O objectivo do modelo é melhorar todo o processo de teste?
- O modelo tem uma estrutura de maturidade?
- É gratuito e não requer validação externa?
- Desenvolvimento concluído?
- Mais do que uma breve descrição?
- Disponibilidade de um instrumento?
- Não é específico de um domínio?

Adicionalmente, pretende-se implementar um modelo mais moderno e que forneça uma avaliação do nível de maturidade do processo de testes de *software* (modelos de referência de processos). Assim, consideremos apenas os modelos mais recentes TMMi e TPI Next [16], deixando de parte os seus antecessores TMM e TPI, respectivamente.

TMMi	**TPI Next**
Requer avaliação externa	Avaliação interna
Modelo de referência de processo	Modelo de referência de processo
Mais rígido	Mais rígido
Modelo estático	Modelo contínuo
Fornece medidas para melhorar	Fornece valor de negócio

Tabela 4.2: Comparação entre os modelos TMMi e TPI Next

Um dos requisitos para a seleção do modelo a implementar é que não sejam necessárias auditorias externas e ainda que seja suficientemente flexível de modo a ser transversal às diferentes equipas que compões a divisão de termotecnologia na Bosch uma vez que contamos com um largo espectro de domínios em que as equipas trabalham, desde embedded/hardware, software backend, frontent entre outros [17]. Assim sendo, e analisando a informação disponível na tabela 4.2, excluímos o modelo TMMi.

Contudo existem algumas diferenças entre o TPI Next e o seu antecessor, passando a ter apenas 4 níveis de avaliação ao invés de 14 e reduzindo o número de áreas chave para 16. Apesar de haver o decréscimo nos níveis referidos, a nossa escolha recai no TPI Next [18], não apenas por ser mais recente mas também por ser um modelo já implementado por outras divisões da Bosch.

Neste modelo não existe um nível de maturidade final mas sim uma avaliação para cada uma das áreas chave. Esta avaliação é realizada com base nos valores dos diversos *checkpoints*, divididos em 3 níveis distintos. Para obter um determinado nível é assim necessário cumprir com todos os *checkpoints* pertencentes ao nível em questão e a todos os níveis anteriores. Assim, para conseguirmos obter o nível máximo em cada uma das áreas chave é necessário

cumprir com todos os *checkpoints* existentes nessa área. Caso exista pelo menos um *checkpoint* por cumprir no primeiro nível então será impossível obter qualquer avaliação nessa área, sendo atribuído o nível inicial (4º nível).

As fases que se seguem serão realizadas com o auxílio da construção de uma aplicação *web* que forneça o modelo selecionado. Os detalhes do trabalho realizado para a construção desta ferramenta estão descritos no próximo capítulo.

CAPÍTULO 5

Aplicações web

Neste capítulo estão descritas todas as aplicações web desenvolvidas e/ou suportadas ao longo do meu estágio. A priori, posso afirmar que todas elas partilham a mesma arquitetura e estrutura do deploy[a]. Assim, qualquer característica descrita numa das aplicações web referenciadas em seguida aplicam-se a qualquer uma das outras.
Cada uma das secções seguintes descreve o trabalho realizado numa aplicação web diferente. É importante referenciar que embora o objetivo do estágio curricular seja desenvolver uma aplicação web que implemente o modelo de maturidade, todas as ferramentas desenvolvidas querem contribuir para melhorar a experiência do utilizador, nomeadamente com o desenvolvimento de um catálogo de modelos de maturidade, de modo a centralizar a informação de quais modelos estão disponíveis dentro da rede Bosch.
Aplicações desenvolvidas:

- *Modelo de maturidade de testes de software: aplicação web que implementa o modelo de maturidade selecionado anteriormente e forneça uma avaliação para cada questionário respondido;*
- *Catálogo de modelos de maturidade: aplicação web para listar os modelos de maturidade disponíveis;*
- *Infraestrutura: aplicação web para listar os serviços disponibilizados pela equipa;*
- *Technology Radar: aplicação web que representa a distribuição e utilização de diferentes técnicas/ferramentas/frameworks/linguagens na Bosch Termotecnologia em Aveiro.*

[a]É a fase do ciclo de vida de um software, no contexto de um Sistema de Informação, que corresponde textualmente à passagem do software para a produção

5.1 Modelo de maturidade de testes de *software*

A criação de uma aplicação *web*, que implemente o modelo de maturidade de testes de *software*, surge no âmbito de fornecer às diversas equipas de desenvolvimento um serviço ágil e auto explicativo. Esta etapa tem como objetivo propor uma substituição das atuais ferramentas disponíveis para este efeito, como é o exemplo de folhas de cálculo.

5.1.1 Conceito

Desde o princípio do estágio que um dos principais objetivos é contribuir para a melhoria do dia-a-dia das equipas de desenvolvimento de *software*. Desta forma, a criação desta aplicação *web* vai mais para além da implementação de um modelo de maturidade.

Pretendemos ir ao encontro da solução de alguns dos problemas encontrados nos métodos mais tradicionais que são usados para a avaliação da maturidade de testes de *software*, nomeadamente na fase de interpretação dos resultados e ainda na fase de obtenção de uma evolução comparativa a nível temporal [19], [20].

Em seguida podemos encontrar os protótipos desenvolvidos para esta aplicação [21].

5.1.2 *Mockups* da aplicação *web*

Os protótipos iniciais (*mockups* [1]) foram desenvolvidos para um melhor entendimento do que pretendemos implementar nesta aplicação. Na figura 5.1 podemos encontrar a nossa proposta para a página inicial. No topo desta figura estão representados os botões para importar os questionários previamente guardados e um campo para atribuir um título ao questionário (ex: nome da equipa, ano, ...). Em seguida, no centro da página, encontra-se a lista dos tópicos do questionário, organizados em secções (na figura 5 do apêndice 9.3 encontramos a visualização com uma das secções expandida). Por fim, no rodapé encontram-se os botões para submeter o questionário, e consequentemente obter uma visualização gráfica do resultado, e ainda a opção para o exportar/guardar.

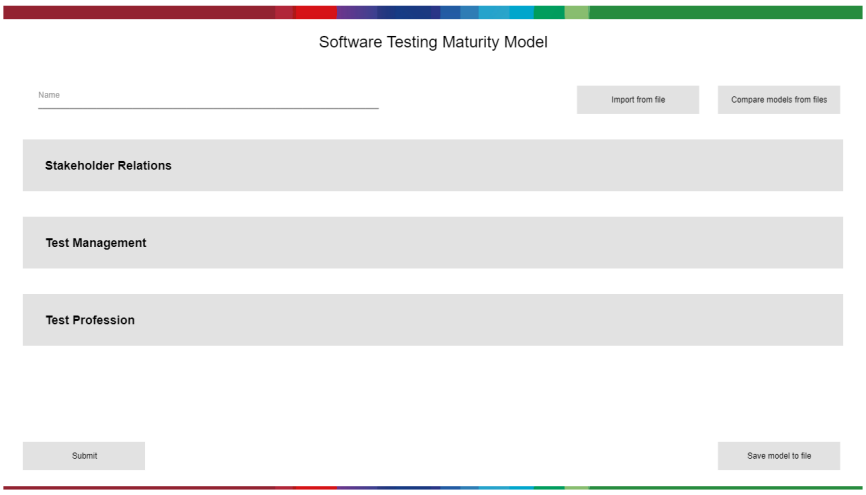

Figura 5.1: Página inicial da aplicação *web* do modelo de maturidade do processo de testes de *software*

Os restantes desenhos podem ser consultados no apêndice 9.3.

5.1.3 Protótipo

Uma vez que esta ferramenta é pensada e criada para ser utilizada pelas equipas de desenvolvimento, foram criados os protótipos iniciais com base no conhecimento e experiências

[1] É um modelo em escala ou de tamanho real de um projeto ou dispositivo, usado para ensino, demonstração, avaliação de design, promoção e outros propósitos.

presentes na nossa equipa, não existindo qualquer tipo de recurso a equipas de *design* gráfico e peritos em experiências de utilizadores. Após uma primeira fase de prototipagem, estes *mockups* foram apresentados às equipas de desenvolvimentos para que pudessem ser validados e recolher novas ideias e correcções de modo a proporcionar uma aplicação final mais adequada às necessidades dos utilizadores. Desta forma, reunimos todos os interessados para apresentarmos os desenhos e após a recolha das opiniões provenientes dos representantes de cada equipa procedemos à sua correção/adaptação.

Na figura 5.2 podemos ver a arquitetura da navegação entre os painéis desta ferramenta. No painel mais à esquerda encontra-se representada a estrutura da página inicial, que é apresentada sempre que pretendemos iniciar o preenchimento de um novo questionário ou ainda para importar ou comparar questionários exportados previamente. Nesta página estão disponíveis as seguintes funcionalidades (representadas pelos quadrados castanhos e cinzentos, na figura):

- Importação de um questionário;
- Comparação entre dois questionários;
- Lista de tópicos do questionário;
- Obter o resultado do questionário;
- Exportar o questionário.

Se optarmos por obter uma visualização do resultado do questionário é alterada a visualização desta página, deixando de estarem disponíveis as opções para importar ou exportar questionários. É ainda substituída a lista com os tópicos do questionário implementado por gráficos que apresentam o resultado/avaliação, incluindo o nível de maturidade em cada área, a prioridade e sugestões de melhorias. Esta estrutura está representada pelo painel azul localizado mais a baixo na figura 5.2.

No restante painel azul, localizado no topo central da figura, encontra-se descrita a estrutura da página da aplicação quando pretendemos comparar dois questionários. Nesta visualização podemos alterar algumas configurações de visualização dos gráficos, visualizar os gráficos com os dados comparativos entre os questionários submetidos e ainda voltar para a página inicial.

Relativamente aos casos de uso, descritos na figura 5.3, o utilizador da aplicação pode:

- **Novo questionário**: iniciar um novo questionário do modelo de maturidade implementado. Após o seu preenchimento, o usuário pode guardar o modelo num ficheiro ou obter o resultado, com gráficos, guias de evolução e ainda dados estatísticos da situação de maturidade.
- **Importar um questionário**: importar um ficheiro previamente gravado e continuar a preencher o questionário ou obter o resultado.
- **Comparar dois questionários**: importar dois ficheiros anteriormente gravados e obtém a evolução que existe entre ambos.

Na figura 5.3, a referência à palavra *submit* não quer traduzir a submissão de um modelo para qualquer tipo de estrutura em *cloud* ou outro tipo de servidores. Neste caso a palavra

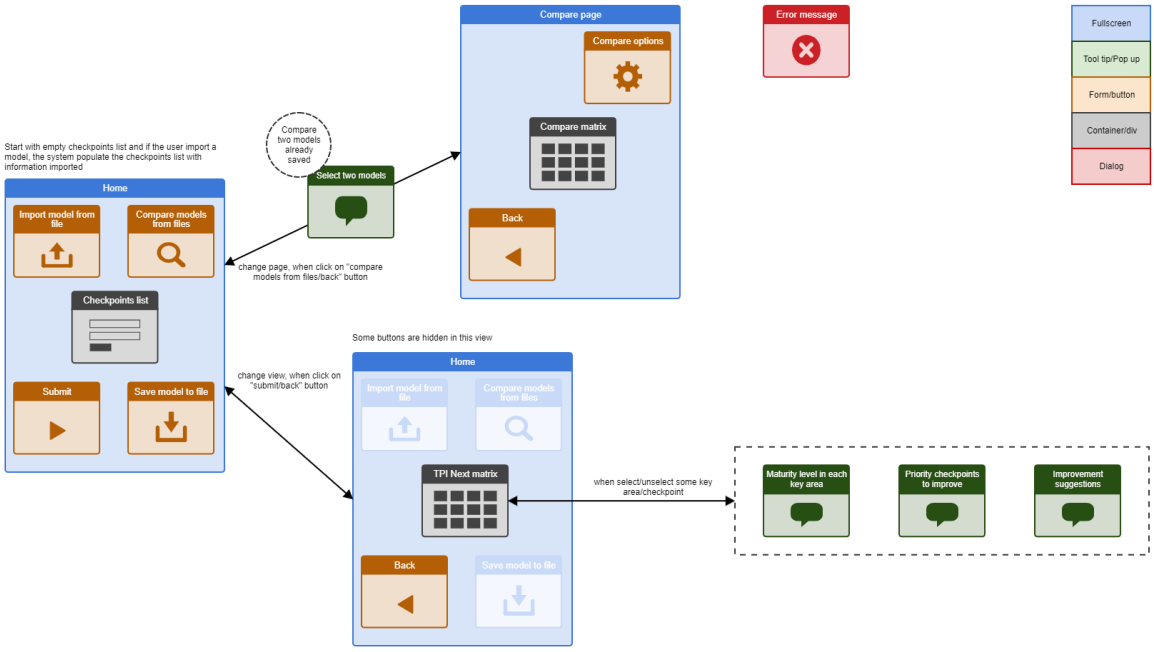

Figura 5.2: Proposta de navegação da aplicação *web* do modelo de maturidade do processo de testes de *software*

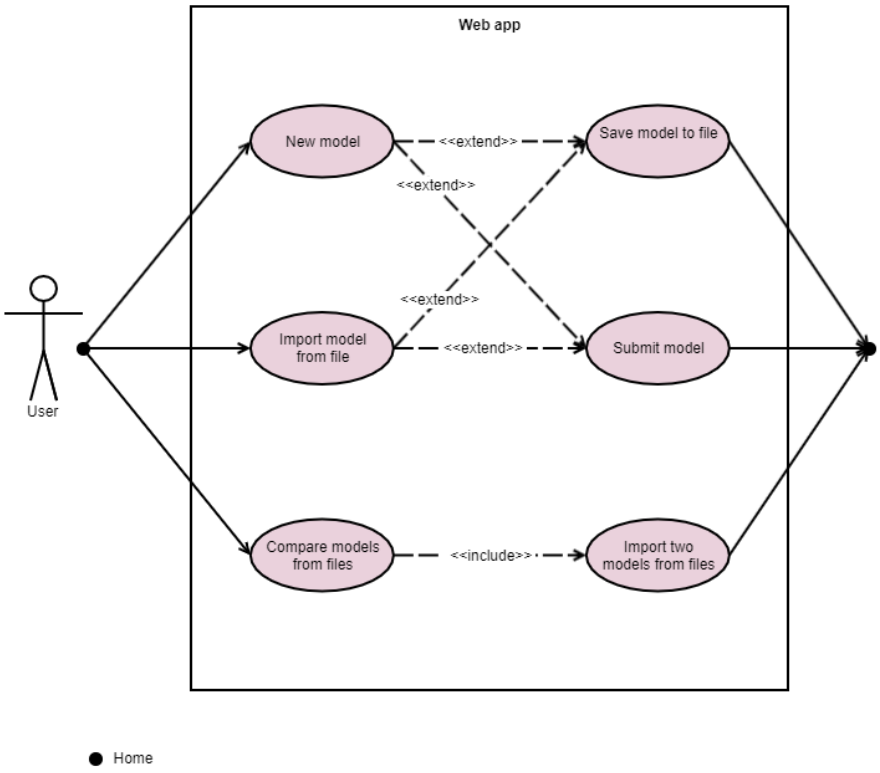

Figura 5.3: Casos de uso da aplicação *web* do modelo de maturidade do processo de testes de *software*

submit significa apenas a fase em que o utilizador informa a aplicação web para fazer uma avaliação das suas respostas, através de *scripts* escritos em *javascript*, e retornar o nível para

cada área do modelo.

Após a validação dos desenhos com o público-alvo seguiu-se a fase de implementação da ferramenta, descrita em seguida.

5.1.4 Implementação

Para a implementação desta ferramenta optou-se por utilizar a *framework* de *JavaScript*, *NodeJS*. Uma vez que não pretendemos obter dados de utilização da ferramenta ou mesmo a obtenção automática dos resultados dos questionários que são preenchidos pelas equipas (sendo esta uma tarefa voluntária e da responsabilidade de cada uma) não temos a necessidade de implementar sistemas de *backend* e/ou bases de dados.

Assim, a implementação desta aplicação incide na seguinte lógica:

- Ler a estrutura do questionário, fornecida num ficheiro .json e que contém, para além da estrutura, a lista de questões;
- Construir a página web, criando os elementos *HTML* necessários de acordo com a estrutura lida;
- Implementar funções em *JavaScript* para interpretar os resultados e criar a visualização gráfica com base nas respostas obtidas;
- Implementar a funcionalidade para exportar o questionário, criando um ficheiro .json;
- Implementar a funcionalidade para importar um questionário previamente exportado;
- Implementar funções em *JavaScript* para interpretar os resultados de dois questionários e criar a visualização gráfica com base da comparação entre ambos.

Uma vez que esta implementação tem como base a estrutura de um questionário num ficheiro .json, facilmente podemos criar outras aplicações para outro tipo de questionários de maturidade que sigam a mesma estrutura, uma vez que a aplicação web é construída dinamicamente.

5.2 Catálogo de modelos de maturidade

À semelhança do trabalho desenvolvido e descrito na secção anterior, existem já outras equipas com algum trabalho realizado relativamente a modelos de maturidade aplicados em outras áreas do ciclo de vida de desenvolvimento de *software*, como é o caso do *Maturity Model Continuous Deployment*. Esta é uma ferramenta para avaliar a maturidade de uma determinada equipa/projeto no que diz respeito à entrega contínua do produto.

5.2.1 Conceito

No decorrer do desenvolvimento da aplicação descrita anteriormente, surgiu a ideia de criar um serviço *web* para catalogar todas as ferramentas desenvolvidas que implementem um modelo de maturidade. Assim, qualquer equipa de desenvolvimento de *software* terá acesso a todos os modelos disponíveis e assim iniciar um processo de avaliação de maturidade nas diversas áreas abrangidas.

5.2.2 *Mockups* da aplicação *web*

Na figura 5.4 está representada a página do catálogo implementado. Nesta página encontram-se listados todos os modelos de maturidade disponíveis, com uma breve descrição e uma ligação para a ferramenta correspondente.

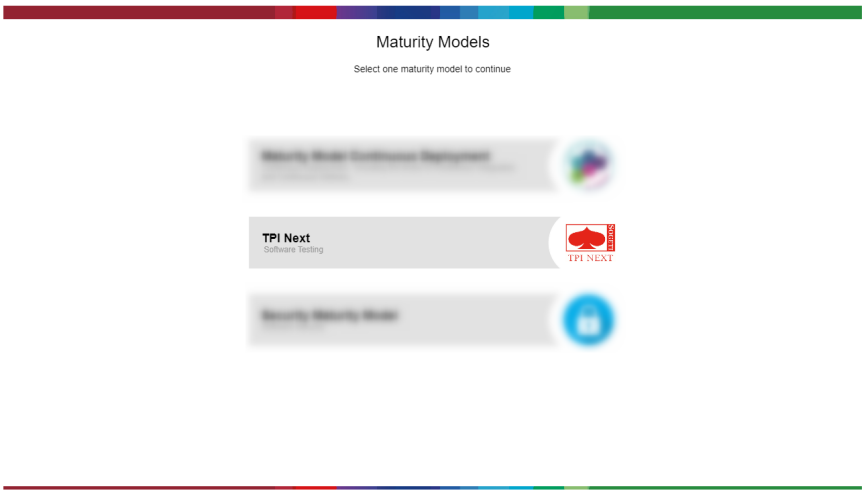

Figura 5.4: Página inicial do catálogo dos modelos de maturidade

5.2.3 Implementação

Um pouco como já acontece com a implementação da ferramenta para o modelo de maturidade de testes de *software*, esta aplicação tem como base ficheiros .json para organizar e apresentar os dados na página *web*. Assim, cada ficheiro .json presente no repositório da aplicação corresponde a um modelo diferente, fornecendo um título, breve descrição, imagem e ainda um endereço *web* para a ferramenta.

Este método permite a adição de novos modelos de maturidade, sem ser necessária a implementação de bases de dados ou de alterações no código fonte.

5.3 Infraestrutura

Com o crescimento do número de aplicações e micro serviços internos surgiu a necessidade de implementação de uma infraestrutura de modo a conseguirmos gerir todos os serviços a partir de um mesmo servidor interno. Esta infraestrutura irá ainda permitir, por parte das equipas de desenvolvimento, o acesso a todos os serviços por nós disponibilizados, existindo uma página web onde estes estão listados.

5.3.1 Conceito

Uma vez que esta infraestrutura tem como principal objetivo centralizar num único serviço toda a gestão e disponibilização das aplicações e micro serviços por nós desenvolvidos, foi decidido criar uma aplicação web onde estivessem listados os recursos disponíveis às equipas de desenvolvimento. Com esta aplicação, todos conseguem saber quais e como podem ser utilizadas estas soluções.

Um exemplo simples de uma aplicação a ser incluída neste servidor é a ferramenta construída para implementar o modelo de maturidade de testes de *software*.

A minha tarefa neste projeto passou por criar a aplicação visual onde estivessem listados os serviços implementados no servidor.

5.3.2 Protótipo

Esta aplicação contém apenas uma página inicial onde constam diversos painéis com um pequeno logótipo, título e descrição, em que cada um representa um serviço distinto.

Existe ainda um categorização dos serviços apresentados em dois grupos:

- Serviços: em que qualquer equipa da Bosch tem acesso ao seu conteúdo (ex: modelo de maturidade de testes de software, Technology Radar, etc.);
- Ferramentas internas: em que o acesso é exclusivo aos membros desta mesma equipa (ex: instância interna de Jenkins, Jira, Grafana).

Figura 5.5: Ideologia da infraestrutura

Na figura 5.5 está representada a ideologia desta aplicação, onde para cada serviço listado existe um redireccionamento para a ferramenta correspondente.

5.3.3 *Mockups* da aplicação web

À semelhança do trabalho realizado para o catálogo de modelos de maturidade, procedemos ao desenvolvimento de um protótipo para a página web a ser desenvolvida. Na figura 5.6 está apresentada esta página inicial onde podemos encontrar a lista de serviços disponíveis para consulta.

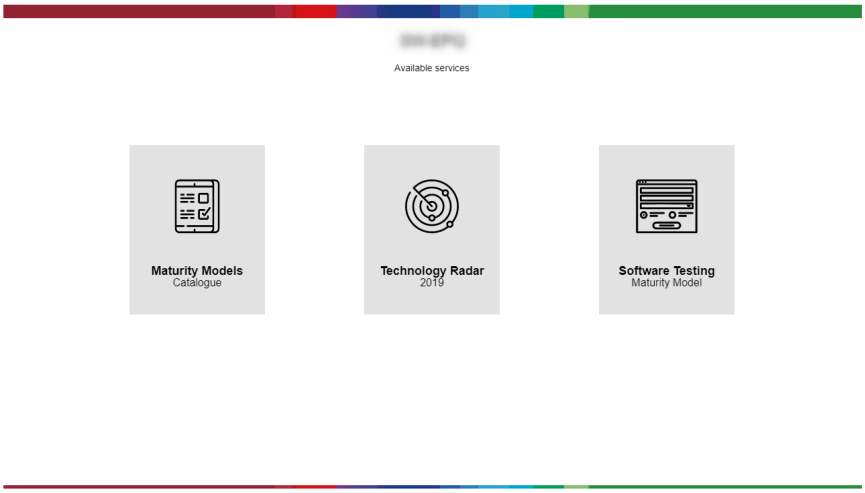

Figura 5.6: Página inicial da infraestrutura

5.3.4 Implementação

Uma vez que já existiam alguns serviços em produção, optámos por unificar a forma como cada um é executado e lançado. Desta forma, podemos ver na imagem 5.7 a estrutura utilizada de forma geral em todos os serviços que estão disponíveis na infraestrutura criada.

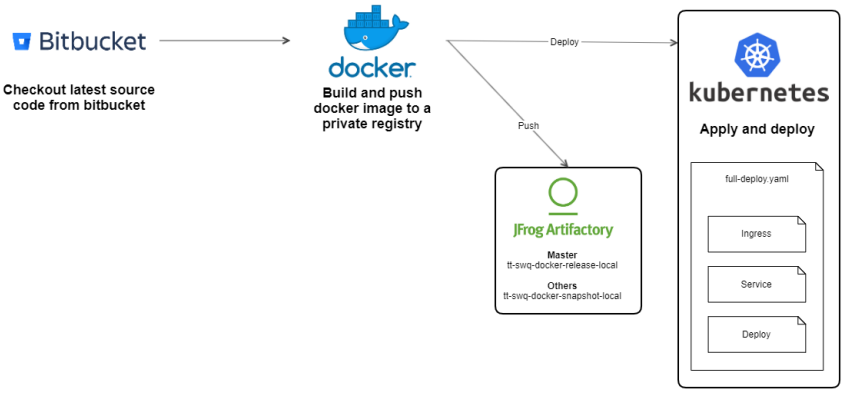

Figura 5.7: *Workflow* do *deploy* de cada serviço na infraestrutura

A própria aplicação *front-end* desta infraestrutura segue a ideologia apresentada e já utilizada nos diferentes serviços da equipa. Esta aplicação web utiliza portanto a *framework NodeJS* e segue as linhas de design da empresa Bosch.

Um pouco à semelhança da aplicação *web* desenvolvida e descrita anteriormente (o modelo de maturidade de testes de software) este serviço é criado dinamicamente com base na lista de ficheiros .json existentes no repositório da aplicação. Cada um destes ficheiros utiliza a mesma estrutura e destinam-se a uma ferramenta distinta que é apresentada na página *web*. Os campos que compõem a estrutura do ficheiro são os seguintes:

- nome da imagem (disponível no repositório da aplicação);
- posição (de modo a conseguir ordenar a ferramenta na aplicação web de uma forma

mais conveniente e/ou lógica);
- título;
- sub-título;
- descrição;
- link (para o serviço da ferramenta);
- link para a página wiki (opcional, no caso de existir documentação associada ao serviço);
- cor;
- tipo de serviço (disponível para todas as equipas de desenvolvimento ou com acesso restrito).

5.4 Technology Radar

Neste secção é descrita a aplicação *web* (interna) *Technology Radar* bem como as alterações realizadas nesta ferramenta que objetivaram não só a atualização da informação que disponibiliza mas também alterações a nível técnico.

5.4.1 O que é o *Technology Radar*?

O *Technology Radar* é um serviço interno que disponibiliza a distribuição de ferramentas, *frameworks*, técnicas e linguagens que estão atualmente a ser utilizadas, categorizando cada uma por nível de popularidade/utilização.

Uma vantagem deste tipo de ferramenta é que permite periodicamente a verificação das tecnologias emergentes e obsoletas e ainda obter uma visão geral do tipo de ferramentas que estão atualmente em uso.

O gráfico que esta ferramenta apresenta está dividido em três estruturas principais:

- **Quadrantes** que representam a forma como agregamos as tecnologias (ferramentas, frameworks, técnicas e linguagens).
- **Anéis** que retratam a sua popularidade (por exemplo: Novato, Intermédio, Avançado) a fim de determinar quando uma determinada tecnologia pertence a um anel específico, a frequência relativa de utilização, no ambiente sob observação, é o principal aspecto a considerar.
- **Pontos** que representam as tecnologias e podem ser vistos como qualquer coisa que desempenhe um papel no desenvolvimento de software.

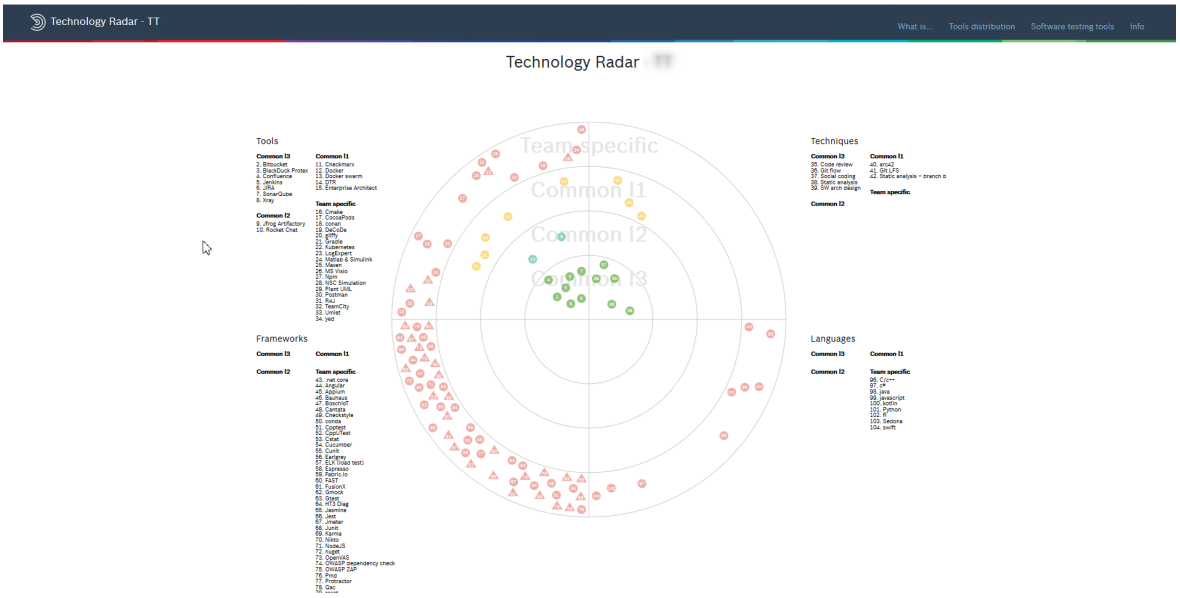

Figura 5.8: Página inicial da aplicação *web* ***Technology Radar***

5.4.2 Atualização de informação disponibilizada

Como referido anteriormente, após a fase de entrevistas com cada uma das equipas detetámos que existiam algumas informações nesta ferramenta que se encontravam desatualizadas. Desta forma procedeu-se à atualização da fonte de dados que alimenta esta aplicação web.

No decorrer desta etapa deparámo-nos com algumas dificuldades no processo de contribuição para a ferramenta, nomeadamente na fase de alteração da informação. Todo o *workflow* desde a contribuição até ao *deploy* da ferramenta tinha margem para melhorias e este foi o ponto inicial para a reformulação da aplicação web.

5.4.3 Reformulação da aplicação web

Alterações implementadas

- Implementação de *NodeJS*
  O ***Technology Radar*** passa a usar a *framework* NodeJS, de modo a simplificar a integração entre módulos em *JavaScript* e *Python* e ainda de forma a unificar o método para deploy, comum a outras aplicações já existentes na empresa.
- Novo visual (*front-end*)
  - Adicionada uma barra de navegação no topo da página com *links* diretos para as diferentes secções;
  - Implementada uma nova estilização, de acordo com as linhas da empresa Bosch;
- Alteração de quase todos ficheiros com formato .csv para o formato .json, de modo a facilitar o seu processamento em *JavaScript*, mantendo apenas no formato original o ficheiro que funciona como fonte de dados, por ser de mais fácil leitura ao olho humano;
- Automação do *deploy* da aplicação quando existe um novo contributo/atualização de informação. Após a aprovação das alterações é automaticamente lançado um *update*

para a *web*, contrariamente ao que acontecia na versão anterior. Para isso, foi criado um *script* em *Python*, que processa as alterações e é executado na *pipeline* da ferramenta.

- Adicionada uma nova secção na aplicação onde é apresentada uma distribuição das ferramentas de testes por níveis e tipos de testes de *software*.
- Alteração das etapas do deploy da aplicação.

Deploy da aplicação web

No caso de haver uma nova contribuição e/ou atualização da fonte de dados, após serem aprovadas, as alterações são automaticamente implementadas.

Para esta tarefa, foram criados scripts python que automatizam o processo de conversão de informação do ficheiro .csv para .json. Estes scripts são executados no início de cada *deploy*.

Nas figuras seguintes podemos encontrar os diagramas que descrevem o *workflow* implementado e ainda as etapas que são executadas ao longo do *deploy*.

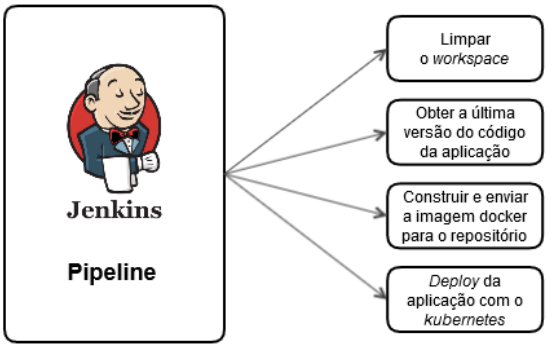

Figura 5.9: Estrutura da *pipeline Jenkins*

Na figura 5.9 estão representadas as várias etapas executadas no serviço *Jenkins*. É importante referir que estas etapas são comuns a qualquer um dos serviços descritos neste capítulo.

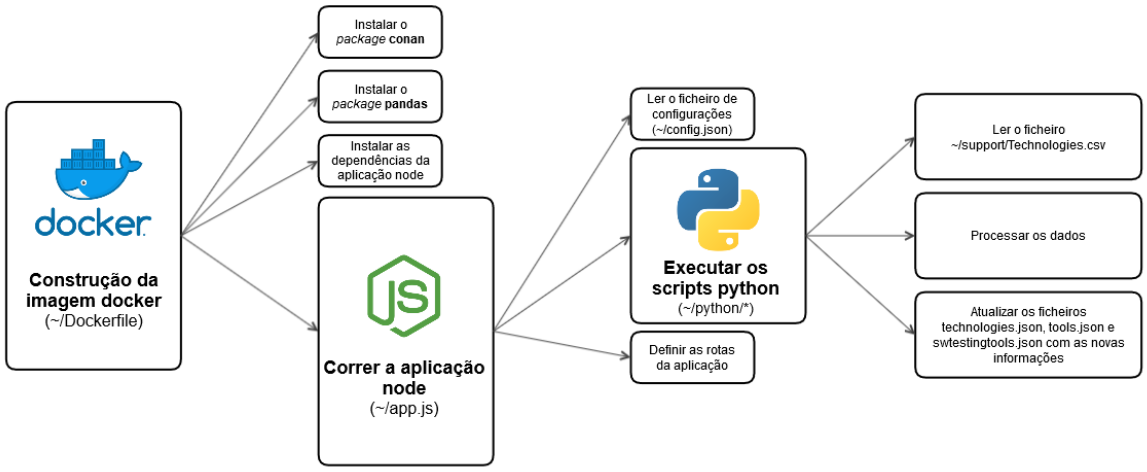

Figura 5.10: Construção da image *docker*

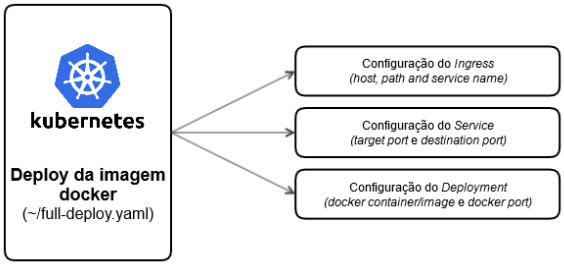

Figura 5.11: Deploy da aplicação com o *kubernetes*

Na figura 5.11 estão representadas as etapas para a construção e o *deploy* da imagem do serviço com o *Kubernetes*. À semelhança das etapas executadas no serviço *Jenkins*, também estas são comuns aos diversos serviços implementados na infraestrutura mencionada anteriormente.

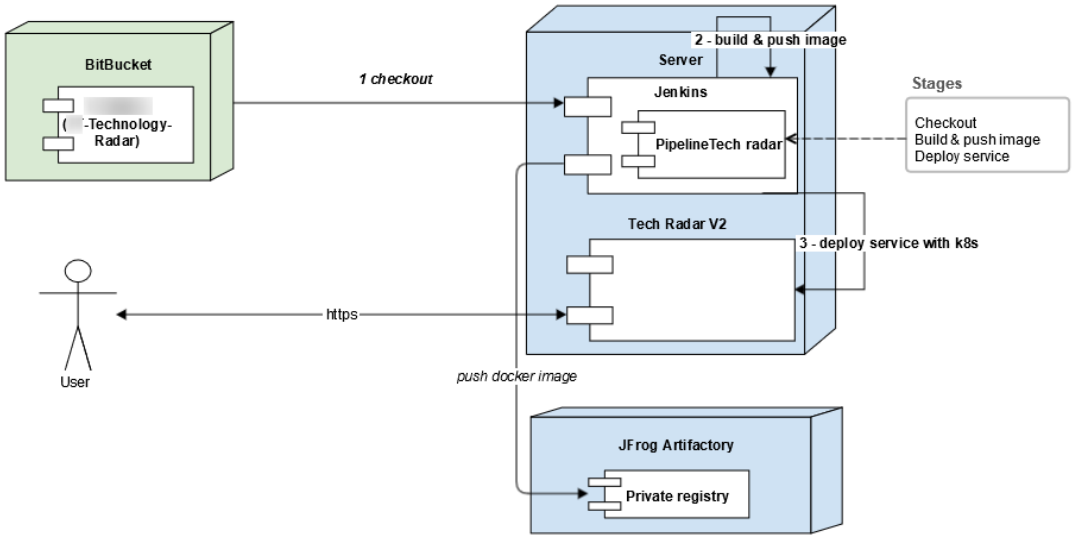

Figura 5.12: Arquitetura do *deploy*

CAPÍTULO 6

Projeto Jenkins Library

O Jenkins Library é um projeto criado com o objetivo de fornecer métodos comuns nas diversas pipelines das equipas de desenvolvimento. Embora este seja um projeto já em produção existe ainda margem de melhorias. Ao longo do tempo de estágio na empresa fui integrado neste mesmo projeto com o objetivo de adquirir mais conhecimentos e assim enriquecer o meu percurso a nível da aprendizagem e experiência. Também do lado da empresa encontrei sempre o apoio para que a minha colaboração neste projeto fosse possível.

6.1 Documentação

Num primeiro contacto com este projeto foi proposta a geração de documentação de forma automática, substituindo a tradicional página *wiki* disponível como recurso interno da empresa. Para esta tarefa foram desempenhadas as seguintes etapas:

- Adicionar as dependências *Maven* necessárias para a geração da documentação;
- Criar um novo repositório para arquivar as diferentes versões da documentação (vamos denominar por **repositório da documentação**);
- Criar uma aplicação *web* para apresentar a documentação gerada, nas diferentes versões disponíveis;

6.1.1 Aplicação *web*

Esta aplicação tem como objetivo apresentar as diferentes versões da documentação gerada automaticamente para o projeto *Jenkins Library*. Desta forma decidiu-se implementar com recurso à *framework NodeJS* uma pequena interface onde constam as diferentes versões disponíveis com um *link* direto para uma documentação, ao estilo de *JavaDocs* (neste caso é denominado por *GroovyDocs*).

Uma vez que esta aplicação está implementada na infraestrutura descrita no capítulo anterior, a estrutura da *pipeline Jenkins* que é executada para lançar a ferramenta e a estrutura da conteinerização com o *Kubernetes* seguem o mesmo esqueleto anteriormente descrito (figuras 5.9 e 5.11).

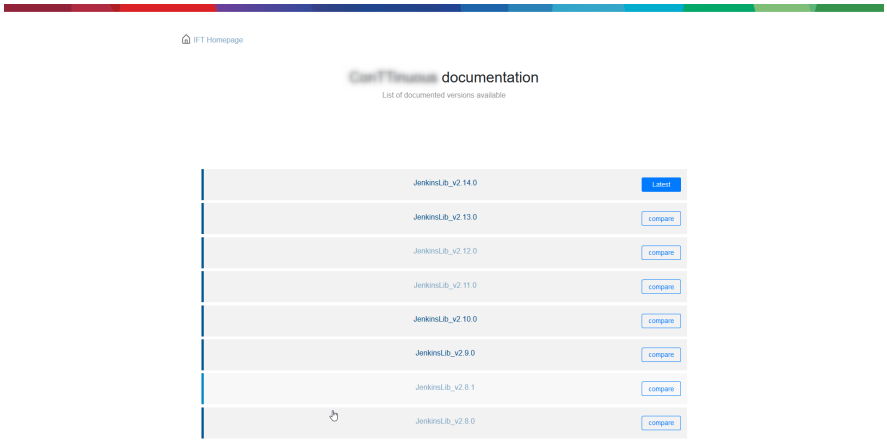

Figura 6.1: Página principal da aplicação *web* para documentação do projeto *Jenkins Library*

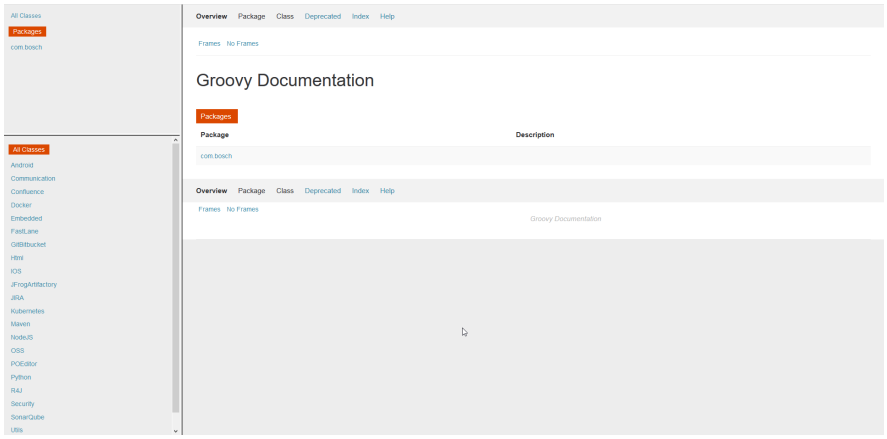

Figura 6.2: Página de documentação gerada automaticamente com *GroovyDocs*

6.2 Integração com o *Jenkins*

Uma vez que este projeto começou a ter novas *releases* de uma forma mais constante, surgiu a necessidade de implementar uma *pipeline* que conseguisse automatizar algumas das principais tarefas. Desta forma, tive a oportunidade de aprender e contribuir para a implementação de uma *pipeline* de integração contínua com as seguintes tarefas:

- Obter a última versão do código fonte do projeto, a partir de um repositório do *Bitbucket* (vamos denominar por **repositório fonte**);
- Fazer *build* do projeto;
- Correr uma pipeline externa que funciona como validação para testes de integração;
- Gerar a documentação automaticamente (ao estilo de JavaDocs);
- Obter a lista de métodos que perderam compatibilidade na nova versão;
- Publicar a documentação gerada na etapa anterior no **repositório da documentação**;
- Gerar uma página *wiki* com documentação relevante para a utilização da biblioteca (ex: exemplos práticos de implementação, notas da *release*, *link* para a documentação do código);
- Criar uma *tag* no **repositório fonte** de modo a manter o código versionado (a *tag* é criada com base na versão obtida no momento da *build* do projeto).

Uma vez que o objetivo é a automação, optámos ainda por implementar uma *pipeline* no **repositório da documentação**, que é executada sempre que uma nova versão fica disponível:

- Obter a última versão do código fonte, a partir do **repositório da documentação**;
- Construir a aplicação *web*, baseada em *NodeJS*;
- Fazer o *deploy* da aplicação com recurso ao *Kubernetes*.

6.3 Implementação de testes unitários

A implementação de testes unitários [22] é uma das áreas onde tenho menos conhecimentos adquiridos ao longo do meu percurso académico. Este estágio curricular vai ao encontro desta minha lacuna de forma a terminar esta fase académica com um leque de experiências e conhecimentos mais amplo e completo. As tarefas desempenhadas foram as seguintes:

- Adicionar as dependências *Maven* necessárias para a implementação dos testes unitários;
- Criar um ficheiro de testes para cada classe do projeto;
- Integrar os testes unitários na *pipeline* de integração contínua;

6.3.1 *Spock framework*

Para esta tarefa foi utilizada a *framework Spock* para a implementação dos testes unitários. Na tabela 6.1 podemos encontrar um exemplo de um dos testes implementado com esta *framework*.

```
def "checkout By Branch where gitCredentials=#gitCredentials and gitRepo=#gitRepo
    and gitBranch=#gitBranch"() {
    setup:
    jenkinsFileMock = new JenkinsFileMock()
    jenkinsFileMock.clearStaticData()
    def gitBranch = 'gitBranch'
    def gitCredentials = 'gitCredentials'
    def gitRepo = 'gitRepo'
    Map git = [branch: "${gitBranch}", credentialsId: "${gitCredentials}", url:
        "${gitRepo}"]
    jenkinsFileMock.addGitMock(git, 'success', 0)
    gitBitBucket = new GitBitbucket(jenkinsFileMock)
    when:
    gitBitBucket.checkoutByBranch(gitBranch, gitCredentials, gitRepo)
    then:
    jenkinsFileMock.gitOutputs.size() == 1
}
```

Tabela 6.1: Exemplo de teste unitário para o método checkoutByBranch() da classe GitBitbucket

Neste exemplo (tabela 6.1) está a ser implementado uma função para testar o método checkoutByBranch() da classe GitBitbucket. Este método tem como objetivo fazer o **checkout** do código fonte, isto é, enviar para o serviço *Jenkins* todo o código disponível. Neste caso, e uma vez que é um teste unitário, apenas temos interesse em verificar se a estrutura do método está em conformidade. Assim estamos a simular qualquer comando específico de *plugins* utilizados, fazendo apenas a verificação que os comandos são executados.

Os argumentos passados para testar o método são:

```
/**
 * Pre-Requisites:
 * <a target="_blank" href="https://plugins.jenkins.io/git"> Git Plugin for
    Jenkins</a> and Git installed on Jenkins machine.
 *
 * @param gitBranch - Branch to checkout.
 * @param gitCredentials - Git credentials ID, defined and stored on Jenkins
    credentials.
 * @param gitRepo - Repository to checkout from.
 * @return
 *
 * This method performs the 'git checkout' by branch.
 */
def checkoutByBranch(String gitBranch, String gitCredentials, String gitRepo) {
   stepJenkinsFile.git branch: "${gitBranch}", credentialsId: "${gitCredentials}",
       url: "${gitRepo}"
}
```

Tabela 6.2: Código do método checkoutByBranch() da classe GitBitbucket

- gitBranch: o nome da *branch* no repositório;
- gitCredentials: as credencias para que seja possível obter o código;
- gitRepo: o nome do repositório.

De modo a autenticar este teste, está a ser realizada uma validação final para verificar que o comando *git* para obter o código é executado. Como ou porquê é que esta é uma validação suficiente? Se olharmos para a estrutura do código que está a ser testado, disponível na 6.2, verificamos que este comando é o único código do método. Assim conseguimos validar que apenas um comando é executado (jenkinsFileMock.gitOutputs.size() == 1) e que é exatamente o comando que será executado em produção se passarmos os mesmos parâmetros, pois apenas realizamos a simulação do comando com os parâmetros selecionados (qualquer tentativa para correr outro comando distinto será interpretada como errada e o código retorna uma falha).

Ao todo, foram implementados mais de 200 testes unitários em 22 classes de implementação, totalizando a cobertura de mais de 4.700 linhas de código distribuídas por 197 métodos diferentes.

Após esta fase, a *pipeline* passou a ter as seguintes etapas (a negrito a nova adição):

- Obter a última versão do código fonte do projeto, a partir do repositório fonte;
- Fazer *build* do projeto;
- **Correr todos os testes unitários**;
- Correr uma pipeline externa que funciona como validação para testes de integração;
- Gerar a documentação automaticamente (ao estilo de JavaDocs);
- Obter a lista de métodos que perderam compatibilidade na nova versão;
- Publicar a documentação gerada na etapa anterior no repositório da documentação;
- Gerar uma página *wiki* com documentação relevante para a utilização da biblioteca;
- Criar uma *tag* no repositório fonte de modo a manter o código versionado.

6.4 Integração com o *SonarQube*

Após a implementação dos testes unitários seguimos para a fase seguinte: obter as métricas da análise estática de código e de cobertura dos testes implementados. Com este objetivo desempenhei as seguintes tarefas:

- Preparação do ambiente *SonarQube* para a integração com o projeto *Jenkins Library*;
- Incluir na *pipeline* do projeto uma nova etapa para a análise estática do código e obtenção das métricas e ainda uma validação das métricas obtidas com os parâmetros definidos pela empresa como os mínimos necessários (*quality gate*).

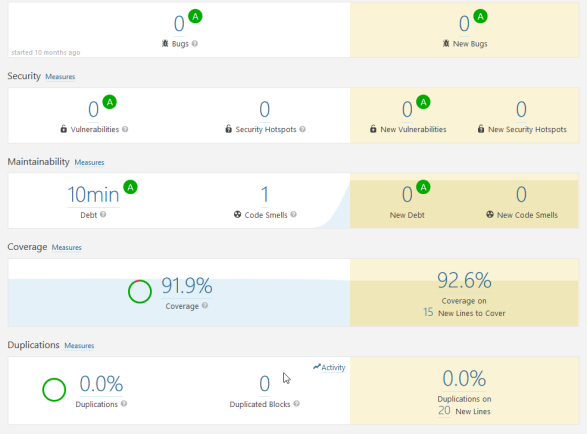

Figura 6.3: *Visualização do resultado obtido no SonarQube*

Na figura 6.3 encontra-se um exemplo do resultado obtido na ferramenta *SonarQube* após a realização da análise do código estático e dos testes unitários.

Uma vez mais a *pipeline* deste projeto foi atualizada para que a validação das métricas do *SonarQube* fossem realizadas de forma automática.

- Obter a última versão do código fonte do projeto, a partir do repositório fonte;
- Fazer *build* do projeto;
- Correr todos os testes unitários;
- **Analisar o código estático**;
- **Validar as métricas obtidas na análise realizada na etapa anterior**;
- Correr uma pipeline externa que funciona como validação para testes de integração;
- Gerar a documentação automaticamente (ao estilo de JavaDocs);
- Obter a lista de métodos que perderam compatibilidade na nova versão;
- Publicar a documentação gerada na etapa anterior no repositório da documentação;
- Gerar uma página *wiki* com documentação relevante para a utilização da biblioteca;
- Criar uma *tag* no repositório fonte de modo a manter o código versionado.

6.5 Integração com o *Grafana dashboards*

A criação de *dashboards* com o *Grafana* surgiu no âmbito de entender como este serviço pode ser utilizado no futuro. Assim, os *dashboards* implementados e descritos em seguida

servem o propósito de experimentação e não têm para já um caso de uso específico de onde e como podem ser utilizados no nosso dia a dia.

Para a execução desta tarefa foi necessário implementar na nossa infraestrutura duas novas bases de dados:

- *Prometheus*;
- *InfluxDB*.

Foi ainda necessário implementar a aplicação *Grafana* para que esta fosse utilizada por parte de todos os membros da equipa.

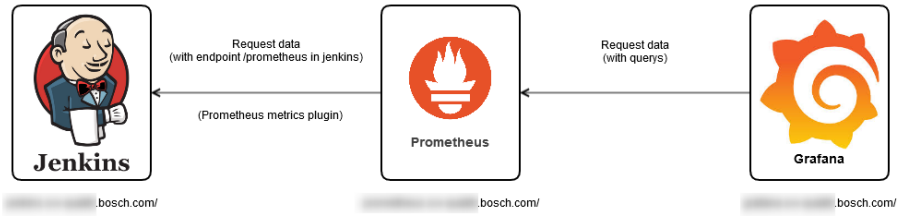

Figura 6.4: *Dashboard* com métricas do *Jenkins*

A figura 6.4 representa o fluxo implementado para obter e enviar os dados referentes ao estado do *Jenkins* até chegar ao *Grafana*. No primeiro momento, o *Jenkins* disponibiliza as métricas através de um *plugin* instalado nesta instância. Este *plugin* cria um novo *endpoint* denominado por */prometheus* que é utilizado pela base de dados *Prometheus* para obter os dados. Por sua vez, o *Grafana* faz pedidos à base de dados, por meio de *querys*, para os representar graficamente.

Figura 6.5: Arquitetura do fluxo de dados para *dashboard* com o estado de cada *build* no *Jenkins*

Na figura 6.5 está ilustrado o fluxo dos dados referentes ao *dashboard* implementado para fornecer informações relativas ao estado de cada *build* presente na instância do *Jenkins*. O *Jenkins* começa por enviar os dados para a base de dados *influxdb* quando alguma *build* termina, através do *plugin Auto status*. Uma vez disponíveis os dados para consulta, o *Grafana* faz o pedido à base de dados para representar graficamente a informação obtida (ver figura 6.6).

Por fim, implementámos um *dashboard* para representar graficamente os dados obtidos do *SonarQube*. Nesta situação, ilustrada na figura 6.7, não é possível enviar diretamente os dados do *SonarQube* para uma base de dados, uma vez que não temos direitos administrativos para

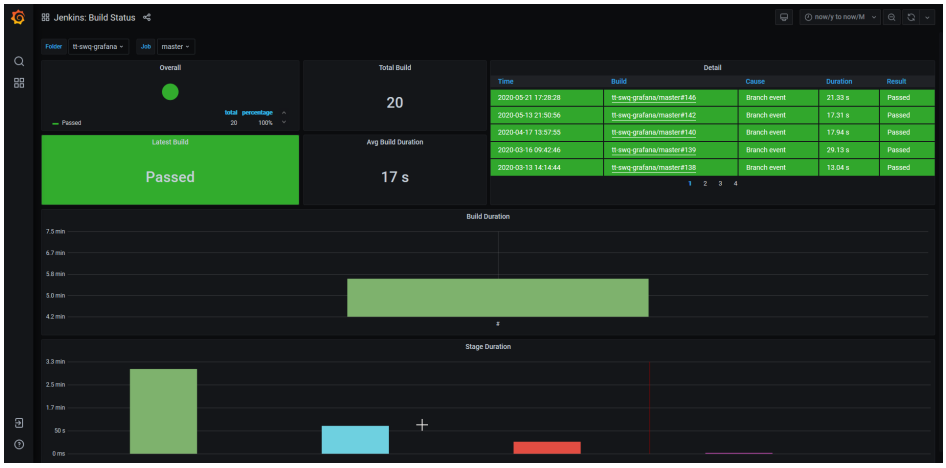

Figura 6.6: *Dashboard* com o estado de cada *build* no *Jenkins*

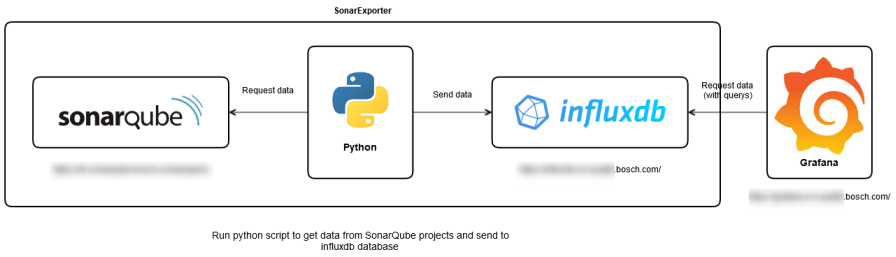

Figura 6.7: Arquitetura do fluxo de dados para *dashboard* com as métricas do *SonarQube*

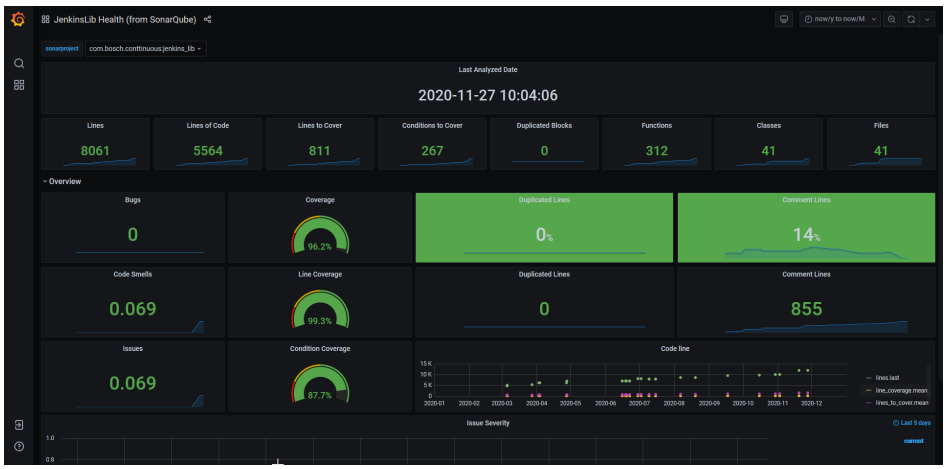

Figura 6.8: *Dashboard* com as métricas do *SonarQube*

a instalação de um *plugin* que o permitisse. Assim, optou-se por utilizar um *script python* que estabelecesse a conexão entre o *SonarQube* e a base de dados *influxdb*.

Com o objetivo de ter sempre os dados atualizados, este script *python* foi inserido na nossa pipeline do projeto, após a etapa de validação das métricas obtidas na análise realizada ao código no *SonarQube*. Finalizada esta etapa, o *Grafana* consegue obter os dados e representar graficamente, à semelhança do que acontece nos casos descritos anteriormente (ver figura 6.8).

CAPÍTULO 7

Aprendizagem computacional e inteligência artificial aplicadas à geração de casos de testes

Nesta segunda fase do estágio pretende-se investigar sobre algumas ferramentas que utilizam o machine learning e inteligência artificial no domínio de testes de software. Em concreto, pretende-se estudar as opções disponíveis para a geração de casos de testes baseado em requisitos do sistema.
É importante referenciar que o objetivo deste capítulo, e consequentemente desta fase do estágio, é encontrar uma ferramenta que seja adequada à maioria dos casos de uso das equipas de desenvolvimento. Assim, o nosso foco está no estudo e investigação das soluções já existentes e que fornecem uma solução mais concordante do nosso ponto de vista e não a investigação da implementação de cada uma das ferramentas ou do desenvolvimento de uma ferramenta que utilize o Machine Learning (ML) e Artificial Intelligence (AI).

7.1 Pesquisa de ferramentas

Inicialmente pretende-se encontrar empresas/softwares que utilizam o ML e AI aplicados aos testes de *software*. Neste sentido decidiu-se dividir esta pesquisa em dois domínios distintos: softwares desenvolvidos e disponíveis dentro da empresa Bosch e por outro lado empresas fora da Bosch que forneçam este tipo de ferramentas.

7.1.1 Soluções *inside* Bosch

No que diz respeito à geração de casos de testes recorrendo a ML e AI, existe já algum trabalho desenvolvido por outras equipas da Bosch. As ferramentas descritas na tabela 7.1 são algumas das mais desenvolvidas nesta domínio e foram-me apresentadas pela empresa para seguir com esta avaliação.

Ferramenta	Casos de uso
Validação de Requisitos 1	• Análise de requisitos • Requisitos para a concepção de cenários de teste (apenas positivos)
Validação de Requisitos 2	• Análise de requisitos • Requisitos para a concepção de cenários de teste (apenas positivos)
Geração de Casos de Teste 1	• Requisitos para a concepção de cenários de teste (inclui positivos e negativos)
Análise de Requisitos 1	• Análise de requisitos

Tabela 7.1: Lista de projetos/ferramentas que utlizam ML e AI para a geração de casos de teste (*inside* Bosch)

Uma vez que não conseguimos abordar todas as soluções já conhecidas, optámos por numa fase inicial contactar apenas as equipas que fornecem as ferramentas que mais se adequam aos nossos casos de uso. Assim, iniciámos esta tarefa de estudo e investigação estabelecendo ligações com as equipas responsáveis pelos projetos *Validação de Requisitos 1* e *Validação de Requisitos 2* por fornecerem soluções tanto para a análise de requisitos como para a geração de cenários de teste, ainda que apenas para os casos positivos.

Ferramenta ***Validação de Requisitos 1***

O *Validação de Requisitos 1* é uma ferramenta numa fase de desenvolvimento mais avançada do que a descrita anteriormente. Esta ferramenta utiliza o processamento de linguagem natural como componente principal de ML e AI e ainda utiliza grafos de conhecimento e modelos de linguagem. Ao longo desta fase de avaliação da ferramenta, participei em palestras relativas a esta solução e orientadas pelos gestores do projeto.

Ferramenta ***Validação de Requisitos 2***

Tendo em conta que esta ferramenta está disponível em código aberto e acompanhada de alguma documentação, não tivemos a necessidade de agendar uma conversa inicial para obter uma visão geral da ferramenta e prosseguir com uma pequena implementação.

Numa primeira análise, podemos ver que este *software* consiste em duas fases distintas:

- Validação dos requisitos;
- Geração de casos de teste.

Para ambos os ramos é necessária a formalização da lista de requisitos iniciais, recorrendo a uma interface web.

Por sua vez, para a formalização de cada um dos requisitos, contamos com uma lista de padrões aplicáveis a distintos cenários (é nesta fase onde a inteligência artificial é aplicada). Algumas das opções estão listadas a seguir:

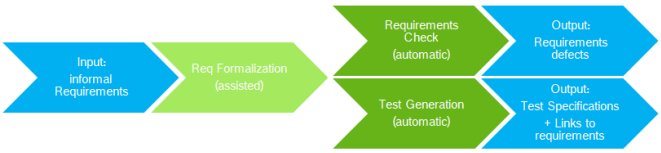

Figura 7.1: *Funcionamento geral da ferramenta Validação de Requisitos 2*

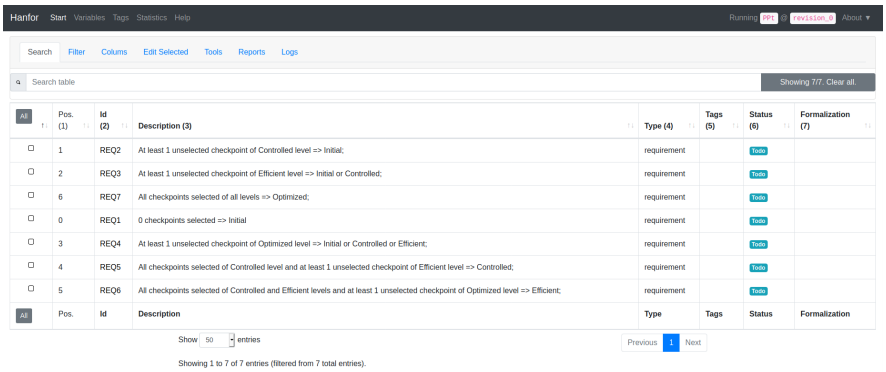

Figura 7.2: *Interface web para a formalização dos requisitos do sistema a ser testado*

- It is never the case that EXPR holds
- It is always the case that EXPR holds
- It is always the case that if EXPR holds, then EXPR holds as well
- Transition to states in which EXPR holds occur at most twice
- It is always the case that ORDER
- It is always the case that REALTIME

(EXPR, ORDER e REALTIME representam as variáveis dos nossos requisitos.)

Após a formalização de todos os nossos requisitos, podemos validar que não existem requisitos contraditórios com outros distintos (primeiro ramo).

Caso tenhamos todos os requisitos em conformidade, podemos então passar para o segundo ramo, onde obtemos os casos de testes para cobrir os nossos requisitos iniciais.

Para ambas as situações são utilizadas aproximações baseadas em *Model Checking* e *Theorem proving*.

7.1.2 Soluções *outside* Bosch

Neste domínio, a empresa apresentou-me 4 propostas principais:

- EggPlant
- Testsigma
- TeamScale

- Specmate.io

Por uma questão de tempo, o estudo e investigação sobre os *softwares* que estas empresas disponibilizam decorreu de forma sequencial. Assim iniciámos este processo reunindo com as empresas EggPlant e Testsigma.

EggPlant

Na primeira conversa com a EggPlant contactámos com 3 representantes da empresa britânica onde nos foi apresentado uma pequena demonstração do *software* por eles desenvolvido.

Para conseguirmos ter uma melhor percepção de como o *software* pode ser utilizado optámos por recorrer à ferramenta por nós desenvolvida no âmbito desta tese: a aplicação web do modelo de maturidade de testes de *software*. Contudo, e uma vez que não temos qualquer tipo de licenças de utilização do *software* da EggPlant, esta demonstração com recurso ao nosso modelo de maturidade ficou condicionada por questões técnicas de ligações a serviços externos à Bosch. Assim, não conseguimos ter uma avaliação deste *software* aplicado à nossa ferramenta web.

Testsigma

Relativamente ao *software* da empresa Testsigma, e após uma reunião inicial com um dos representantes da empresa, acordámos que a avaliação seria realizada de uma forma distinta. Mais uma vez não temos qualquer tipo de licença para a utilização do *software* e os problemas técnicos para conseguirmos utilizar um serviço externo à Bosch mantêm-se. Assim optámos por enviar para a empresa Testsigma uma lista de requisitos do nosso sistema acompanhadas de imagens ilustrativas do *workflow* de cada requisito (ver os exemplos nas imagens 7.3 e 7.4).

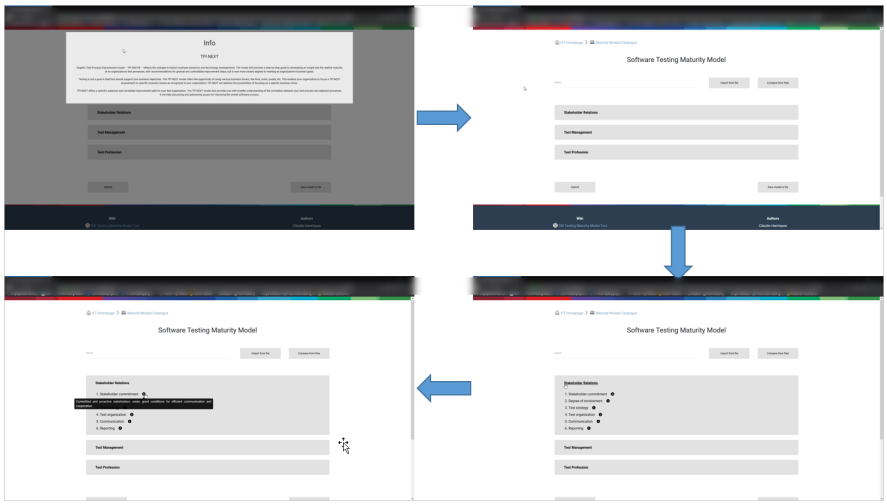

Figura 7.3: *Workflow* para validar o texto de ajuda para um determinado ponto do questionário

O resultado obtido pela empresa encontra-se representado na figura 7.5 e ainda nas figuras disponíveis no apêndice 9.3

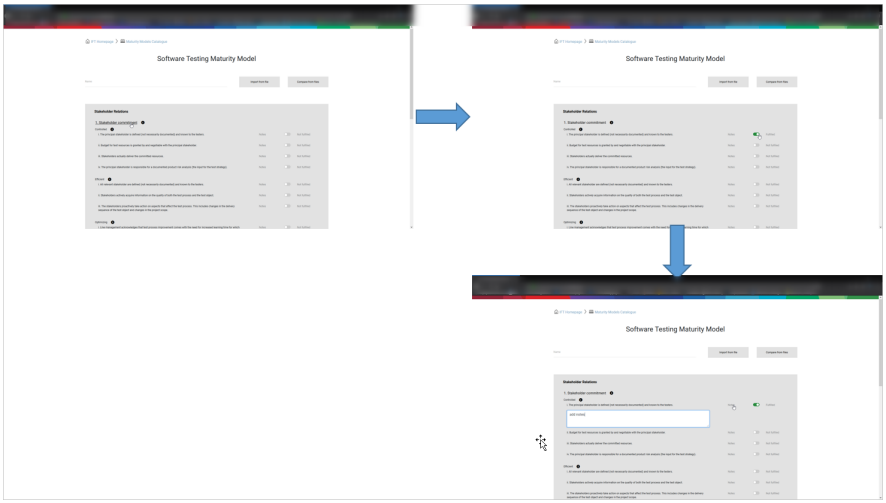

Figura 7.4: *Workflow* para validar a funcionalidade de alterar o estado de um ponto do questionário e adicionar um comentário

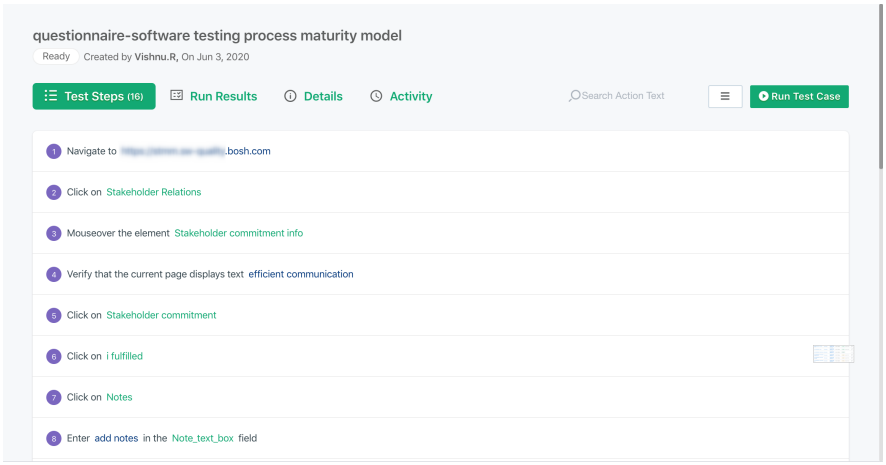

Figura 7.5: Caso de teste para validação das funcionalidades do preenchimento do questionário

7.2 Escolha da ferramenta

Uma das complicações mais evidentes no contacto com empresas externas é a comunicação e a dificuldade na utilização de alguns *software* externos à Bosch. Este dado deve-se às politicas de segurança da empresa e pouco podemos fazer para facilitar a troca de informação com o exterior. Assim, optámos por escolher uma das duas ferramentas descritas anteriormente e desenvolvidas por equipas da Bosch. Embora a ferramente *Validação de Requisitos 1* esteja numa fase de desenvolvimento mais avançada quando comparada com a *Validação de Requisitos 2*, existem alguns pontos positivos nesta última que nos motivaram a optar por ela:

- Código aberto;
- Lista de requisitos informal, ao invés de um texto/documento;
- O output obtido é mais adequado para as equipas de desenvolvimento;
- Documentação mais acessível;
- Comunicação entre equipas mais fluida.

7.3 Prova de conceitos

A prova de conceitos, para demonstrar como se pode aplicar a ferramenta *Validação de Requisitos 2*, tem como objetivo obter os casos de teste necessários de implementar para validar o nível de maturidade em cada área chave dependendo dos valores dos *checkpoints* que compõem cada nível. Assim, inicialmente definiu-se os seguintes requisitos informais:

- Se $Controlled == false$ então $Maturity == 0$
- Se $Controlled == true$ e $Efficient == false$ então $Maturity == 1$
- Se $Controlled == true$ e $Efficient == true$ e $Optimized == false$ então $Maturity == 2$
- Se $Controlled == true$ e $Efficient == true$ e $Optimized == true$ então $Maturity == 3$

À lista de requisitos apresentada anteriormente, foi aplicado individualmente um dos padrões que a ferramenta disponibiliza para a formalização do requisito (ilustrado na figura 7.6).

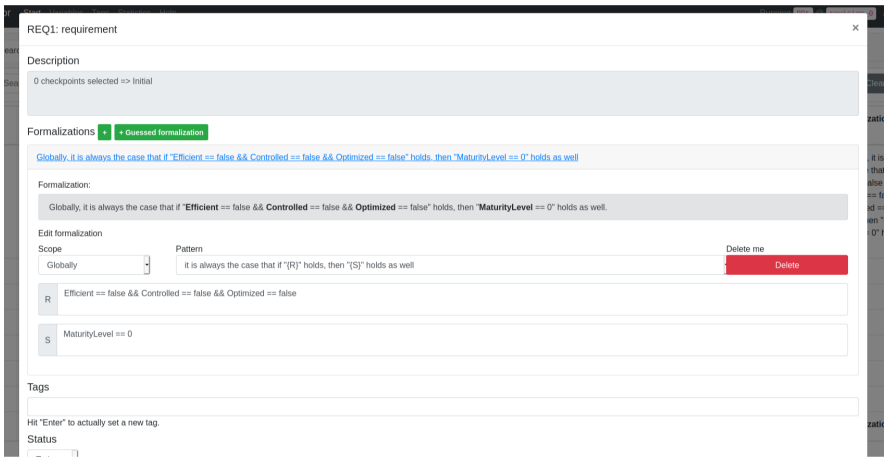

Figura 7.6: Formalização de um requisito informal

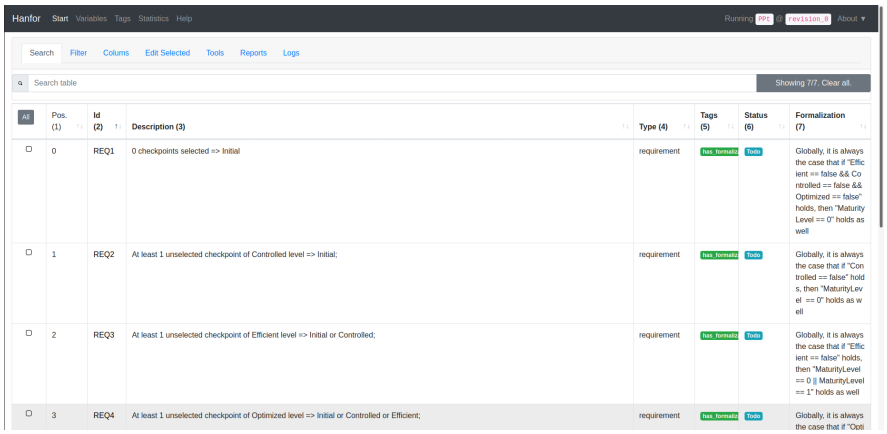

Figura 7.7: Lista de requisitos formalizados

Requisito informal	*Padrão aplicado*	*Requisito informal obtido*
Se $Controlled == false$ então $Maturity == 0$	É sempre o caso que se 'EXPR' se verifica, então 'EXPR' também se verifica	Globalmente, é sempre o caso que se $'Controlled == false'$ se verifica, então $'Maturity == 0'$ também se verifica

Tabela 7.2: Exemplo de formalização de um requisito informal

Um exemplo de um requisito formalizado, após ser aplicado o padrão adequado está descrito na tabela 7.2. Para este caso específico, foi necessário definir 2 varáveis em duas expressões:

- Expressão 1: 'Controlled == false'
- Expressão 2: 'Maturity == 0'

- Variável 1: Controlled (*true* ou *false*)
- Variável 2: Maturity (0, 1, 2 ou 3)

O caso de teste gerado para dar cobertura a este requisito depende agora das variáveis definidas na formalização do requisito, obtendo o seguinte resultado:

***Inputs*:**

Controlled := false

***Output* esperado:**

Maturity := 0

Contudo, é agora necessário definir quando as variáveis Efficient, Controlled e Optimized tomam os valores *true* ou *false*. Para tal, foram adicionados mais 6 requisitos informais. Assumindo que o nível Controlled tem 4 *checkpoints*, o nível Efficient tem 3 *checkpoints* e por fim o nível Optimized tem também 3 *checkpoints*, obtemos as seguintes definições:

- Se $Checkpoint\_C\_1 == true$ e $Checkpoint\_C\_2 == true$ e $Checkpoint\_C\_3 == true$ e $Checkpoint\_C\_4 == true$ então $Controlled == true$
- Se $Checkpoint\_C\_1 == false$ ou $Checkpoint\_C\_2 == false$ ou $Checkpoint\_C\_3 == false$ ou $Checkpoint\_C\_4 == false$ então $Controlled == false$
- Se $Controlled == true$ $Checkpoint\_E\_1 == true$ $Checkpoint\_E\_2 == true$ $Checkpoint\_E\_3 == true$ então $Efficient == true$
- Se $Controlled == false$ ou $Checkpoint\_E\_1 == false$ ou $Checkpoint\_E\_2 == false$ ou $Checkpoint\_E\_3 == false$ então $Efficient == false$
- Se $Controlled == true$ e $Efficient == true$ e $Checkpoint\_O\_1 == true$ e $Checkpoint\_O\_2 == true$ e $Checkpoint\_O\_3 == true$ então $Optimized == true$
- Se $Controlled == false$ ou $Efficient == false$ ou $Checkpoint\_O\_1 == false$ ou $Checkpoint\_O\_2 == false$ ou $Checkpoint\_O\_3 == false$ então $Optimized == false$

A formalização destes requisitos é dada por:

- Globalmente, é sempre o caso que se $'Checkpoint\_C\_1 == true$ e $Checkpoint\_C\_2 == true$ e $Checkpoint\_C\_3 == true$ e $Checkpoint\_C\_4 == true'$ se verifica, então $'Controlled == true'$ também se verifica
- Globalmente, é sempre o caso que se $'Checkpoint\_C\_1 == false$ ou $Checkpoint\_C\_2 == false$ ou $Checkpoint\_C\_3 == false$ ou $Checkpoint\_C\_4 == false'$ se verifica, então $'Controlled == false'$ também se verifica
- Globalmente, é sempre o caso que se $'Controlled == true$ e $Checkpoint\_E\_1 == true$ e $Checkpoint\_E\_2 == true$ e $Checkpoint\_E\_3 == true'$ se verifica, então $'Efficient == true'$ também se verifica
- Globalmente, é sempre o caso que se $'Controlled == false$ ou $Checkpoint\_E\_1 == false$ ou $Checkpoint\_E\_2 == false$ ou $Checkpoint\_E\_3 == false'$ se verifica, então $'Efficient == false'$ também se verifica
- Globalmente, é sempre o caso que se $'Controlled == true$ e $Efficient == true$ e $Checkpoint\_O\_1 == true$ e $Checkpoint\_O\_2 == true$ e $Checkpoint\_O\_3 == true'$ se verifica, então $'Optimized == true'$ também se verifica
- Globalmente, é sempre o caso que se $'Controlled == false$ ou $Efficient == false$ ou $Checkpoint\_O\_1 == false$ ou $Checkpoint\_O\_2 == false$ ou $Checkpoint\_O\_3 == false'$ se verifica, então $'Optimized == false'$ também se verifica

Com a adição de todas estas definições, os casos de teste gerados passam a ter a seguinte estrutura:

***Inputs*:**

Checkpoint_E_1 := true, Checkpoint_E_2 := true, Checkpoint_E_3 := true, Checkpoint_C_3 := true, Checkpoint_C_4 := true, Checkpoint_C_1 := true, Checkpoint_C_2 := true, Checkpoint_O_1 := true, Checkpoint_O_2 := true, Checkpoint_O_3 := false

***Output* esperado:**

Maturity := 2

CAPÍTULO 8

Processamento de linguagem natural

Neste capítulo descrevo o trabalho desenvolvido para a classificação de textos, recorrendo a algoritmos de processamento de linguagem natural. Esta tarefa surgiu no momento em que foi necessário organizar os inúmeros artigos e textos científicos.

8.1 Estrutura da *pipeline*

A primeira fase desta *pipeline* é obter os textos no formato .csv para que possa ser aplicado um algoritmo para classificar cada um deles. Terminada a classificação, pretende-se organizar os artigos em *clusters* e processar os textos de modo a obter factos sobre os rótulos/etiquetas obtidas na fase anterior. No fim desta sequência, o resultado final para cada um dos *clusters* é uma lista de palavras chave e alguns factos encontrados nos textos.

Cada caixa preta apresentada na figura 8.1 representa a aplicação de um algoritmo de *machine learning*. Para cada caixa é fornecido um ou mais *input(s)* retornando um ficheiro .csv com os resultados obtidos.

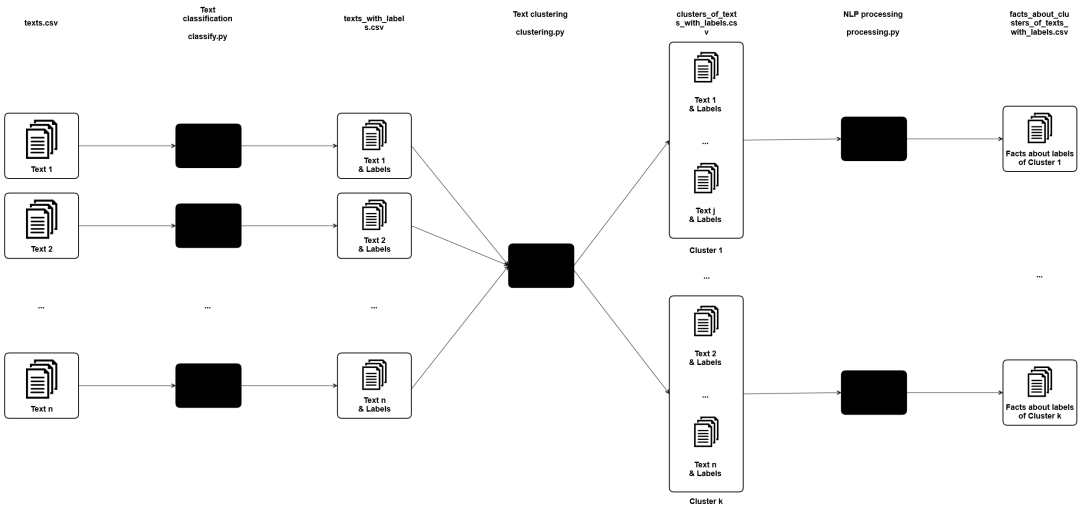

Figura 8.1: Sequência implementada para classificar e obter factos sobre textos

8.2 Implementação

Este sistema foi implementado recorrendo à construção de uma *pipeline* no *Jenkins*, criando um *container Docker* para a criação do ambiente com *python* e correr os algoritmos descritos anteriormente.

8.3 Melhorias futuras

É um facto que guardar os artigos provenientes de *websites* e/ou ficheiros em .pdf não é uma tarefa fácil de executar manualmente. Assim, pretende-se futuramente melhorar este sistema para conseguir obter automaticamente o conteúdo de um dado artigo.

Uma outra melhoria é o aprimoramento dos algoritmos de classificação, organização em *clusters* e processamento dos textos e ainda o melhoramento do resultado final obtido, eliminando factos repetitivos e a adição da referência para cada um deles.

CAPÍTULO 9

Conclusão

9.1 Aprendizagens

Esta foi uma das fases mais gratificantes do meu percurso académico. Sinto que consegui evoluir como pessoa e sobretudo como profissional, estando agora mais preparado para entrar no novo ciclo que se avizinha. Do meu ponto de vista, o estágio em contexto empresarial é a melhor opção que podemos tomar se queremos abraçar desde logo uma carreira profissional e hoje posso ter a certeza disso mesmo.

Para além de todas as tecnologias e projetos onde tive a oportunidade de trabalhar, consegui estabelecer contacto e trocas de experiências com outras equipas de desenvolvimento que me acompanharam todos os dias. Consegui perceber um pouco do funcionamento geral deste tipo de equipas e de empresas.

9.2 Dificuldades

Uma das maiores dificuldades ao longo desta longa etapa foi sem dúvida o panorama vivido em praticamente metade do estágio. A situação pandemia do país obrigou a algumas alterações no dia-a-dia de cada um de nós e da vida empresarial. Em concreto, o facto de desde meados de março estar a trabalhar a partir de casa, quase a totalidade do tempo, trouxe algumas perturbações na comunicação e na partilha de algumas informações mais casuais mas que no meu ponto de vista se tornam importantes para o bom funcionamento de uma equipa. Contudo, e com as pessoas certas, penso que este foi um obstáculo minimizado à medida que o tempo foi passando e consegui-se aprender a trabalhar de uma forma um pouco distinta do nosso antigo habitual.

Um outro fator que dificultou, em parte, o desenvolvimento de algumas tarefas do estágio foi a comunicação com empresas externas e a troca de informações. Contudo, é totalmente

compreensível as regras impostas pela empresa de modo a salvaguardar informações sensíveis e ainda o bom funcionamento de todos os nossos serviços internos.

No que toca a habilidades pessoais, mais conhecidas como *soft skills*, uma das minhas dificuldades prenderam-se com a língua inglesa. Não sendo este um dos meus pontos fortes mas ao mesmo tempo necessário para o meu dia-a-dia, para além da dificuldade sentida na comunicação com outras equipas ou colegas de trabalho senti ainda alguma evolução da minha parte ao longo do tempo.

9.3 Considerações finais

Relativamente ao relatório de estágio, espero conseguir com este documento passar todas as minhas aprendizagens e tarefas desempenhadas. Por ser um estágio em contexto empresarial alguns pormenores técnicos e de implementações dos sistemas estão omissos. Contudo, penso conseguir relatar na sua generalidade os tópicos abordados ao longo dos 12 meses.

Referências

[1] Microsoft. (2007). Maturity model for test organization, URL: https://docs.microsoft.com/pt-pt/archive/blogs/testing123/maturity-model-for-test-organization (acedido em 07/08/2019).

[2] S. Menon. (2017). Using the Testing Maturity Model to improve Quality Assurance Process, URL: https://303software.com/mobile-app-development/using-the-testing-maturity-model-to-improve-quality-assurance-process/ (acedido em 07/08/2019).

[3] A. Bertolino, «The (Im)Maturity Level of Software Testing», *SIGSOFT Softw. Eng. Notes*, vol. 29, n.º 5, pp. 1–4, set. de 2004, ISSN: 0163-5948. DOI: 10.1145/1022494.1022540. URL: https://doi.org/10.1145/1022494.1022540.

[4] I. Burnstein, T. Suwanassart e R. Carlson, «Developing a Testing Maturity Model for software test process evaluation and improvement», em *Proceedings International Test Conference 1996. Test and Design Validity*, 1996, pp. 581–589. DOI: 10.1109/TEST.1996.557106.

[5] M. Kohlegger, R. Maier e S. Thalmann, *Understanding maturity models. Results of a structured content analysis*. na, 2009.

[6] K. Hrabovská, B. Rossi e T. Pitner, «Software testing process models benefits & drawbacks: a systematic literature review», *arXiv preprint arXiv:1901.01450*, 2019.

[7] A. P. Furtado, S. Meira e M. A. W. Gomes, «Towards a Maturity Model in Software Testing Automation», em *ICSEA 2014*, 2014.

[8] I. Burnstein, A. Homyen, R. Grom e C. Carlson, «A model to assess testing process maturity», *Crosstalk*, vol. 11, n.º 11, pp. 26–30, 1998.

[9] I. Burnstein, T. Suwannasart e C. Carlson, «Developing a Testing Maturity Model : Part I», 1996.

[10] I. Burnstein e T. Suwannasart, «Developing a Testing Maturity Model, Part II», 1996.

[11] V. Garousi, M. Felderer e T. Hacaloglu, «Software test maturity assessment and test process improvement: A multivocal literature review», *Information and Software Technology*, vol. 85, jan. de 2017. DOI: 10.1016/j.infsof.2017.01.001.

[12] J. Andersin e J. Andersin, *UNIVERSITY OF HELSINKI TPI – a model for Test Process Improvement*, 2004.

[13] W. Afzal, S. Alone, K. Glocksien e R. Torkar, «Software test process improvement approaches: A systematic literature review and an industrial case study», *Journal of Systems and Software*, vol. 111, pp. 1–33, 2016, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.08.048. URL: http://www.sciencedirect.com/science/article/pii/S0164121215001910.

[14] V. Garousi, M. Felderer e T. Hacaloğlu, «What We Know about Software Test Maturity and Test Process Improvement», *IEEE Software*, vol. 35, n.º 1, pp. 84–92, 2018. DOI: 10.1109/MS.2017.4541043.

[15] A. Mikhailau. (2018). QA process maturity: Models and capabilities, URL: https://www.scnsoft.com/blog/qa-process-maturity-models-and-capabilities (acedido em 26/08/2019).

[16] M. A. T. Laksono, E. K. Budiardjo e A. Ferdinansyah, «Assessment of Test Maturity Model: A Comparative Study for Process Improvement», sér. ICSIM 2019, Bali, Indonesia: Association for Computing Machinery, 2019, pp. 110–118, ISBN: 9781450366427. DOI: 10.1145/3305160.3305203. URL: https://doi.org/10.1145/3305160.3305203.

[17] E. Jung, «A Test Process Improvement Model for Embedded Software Developments», em *2009 Ninth International Conference on Quality Software*, 2009, pp. 432–437. DOI: 10.1109/QSIC.2009.64.

[18] S. Software. (2017). Using the test process improvement models. Case study based on TPI Next model, Anton Muzhailo, URL: https://de.slideshare.net/Sigmasoftware/using-the-test-process-improvement-models-case-study-based-on-tpi-next-model-anton-muzhailo (acedido em 14/08/2019).

[19] Y. Wang, M. Mäntylä, S. Eldh, J. Markkula, K. Wiklund, T. Kairi, P. Raulamo-Jurvanen e A. Haukinen, «A Self-assessment Instrument for Assessing Test Automation Maturity», *Proceedings of the Evaluation and Assessment on Software Engineering*, abr. de 2019. DOI: 10.1145/3319008.3319020. URL: http://dx.doi.org/10.1145/3319008.3319020.

[20] A. F. Araujo, C. L. Rodrigues, A. Vincenzi, C. G. Camilo e A. F. Silva, «A Framework for Maturity Assessment in Software Testing for Small and Medium-Sized Enterprises», 2013.

[21] Tricentis. (2019). Continuous Testing Maturity Model, URL: https://www.tricentis.com/wp-content/uploads/2019/01/Continuous-Testing-Maturity-Model.pdf (acedido em 08/08/2019).

[22] D. Abdullah e I. Burnstein, *Practical Software Testing*, 2003.

Apêndice A

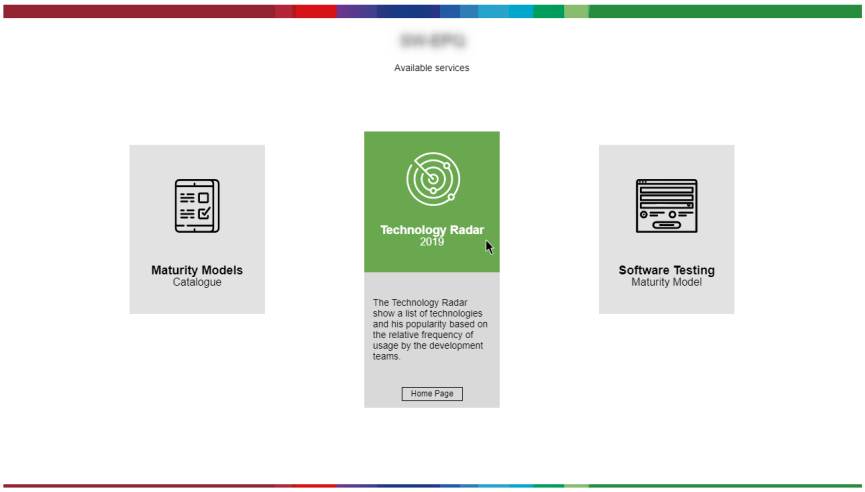

Figura 1: Página inicial da aplicação da infraestrutura

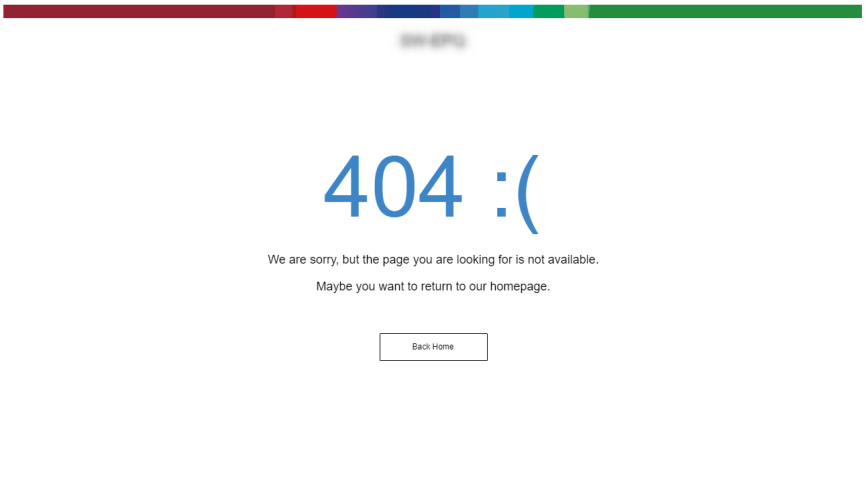

Figura 2: Página de erro da infraestrutura

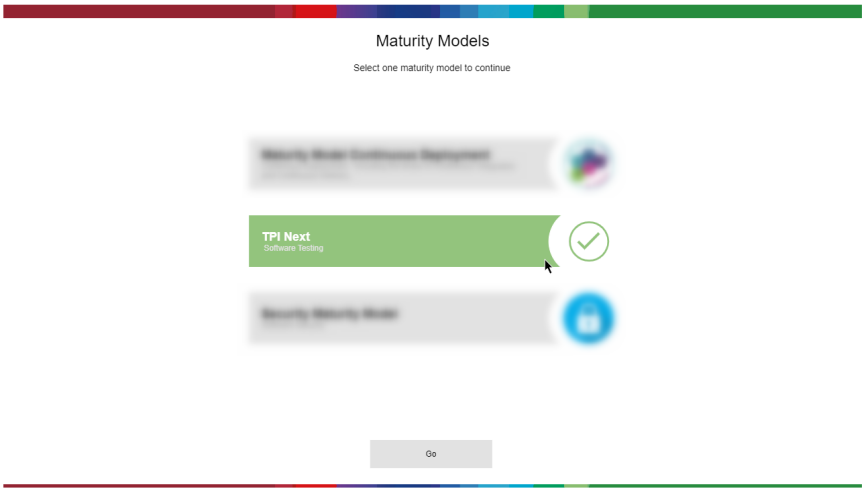

Figura 3: Página inicial do catálogo dos modelos de maturidade com 1 modelo selecionado

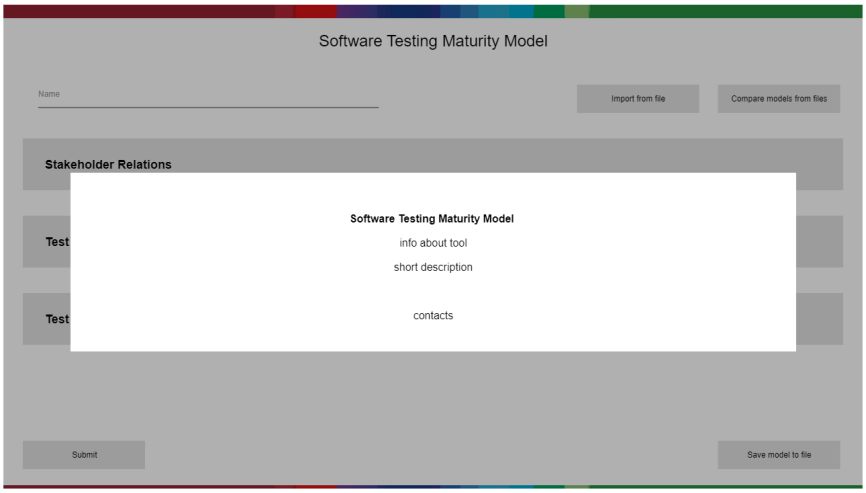

Figura 4: Página inicial do modelo de maturidade de testes de software com mensagem inicial

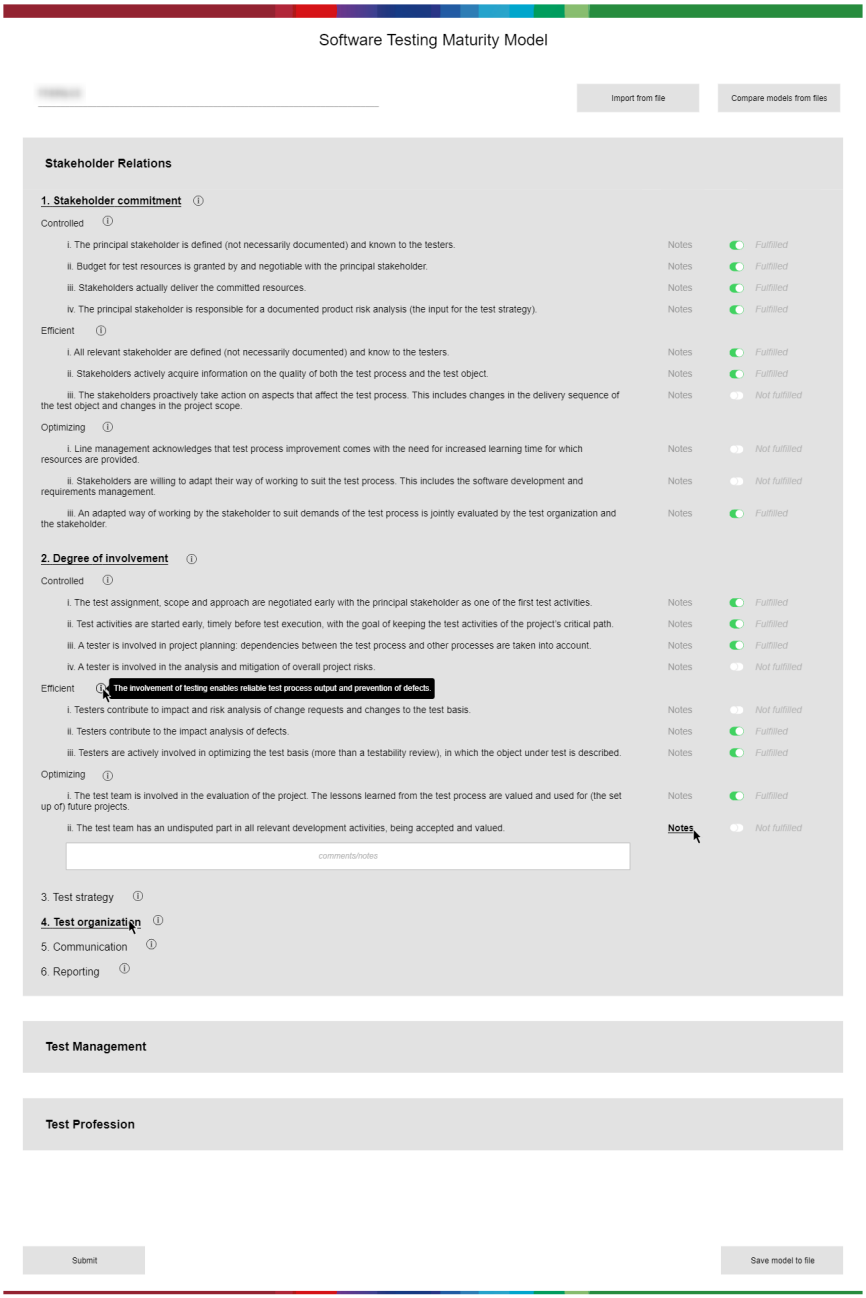

Figura 5: Página inicial do modelo de maturidade de testes de software com as áreas chave expandidas

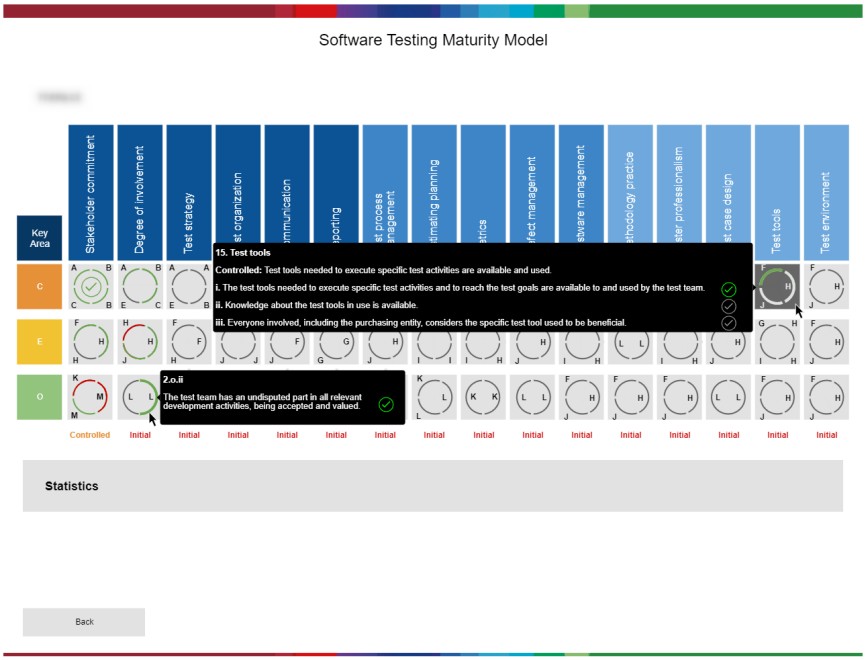

Figura 6: Página de análise do modelo de maturidade de testes de software com elementos de ajuda

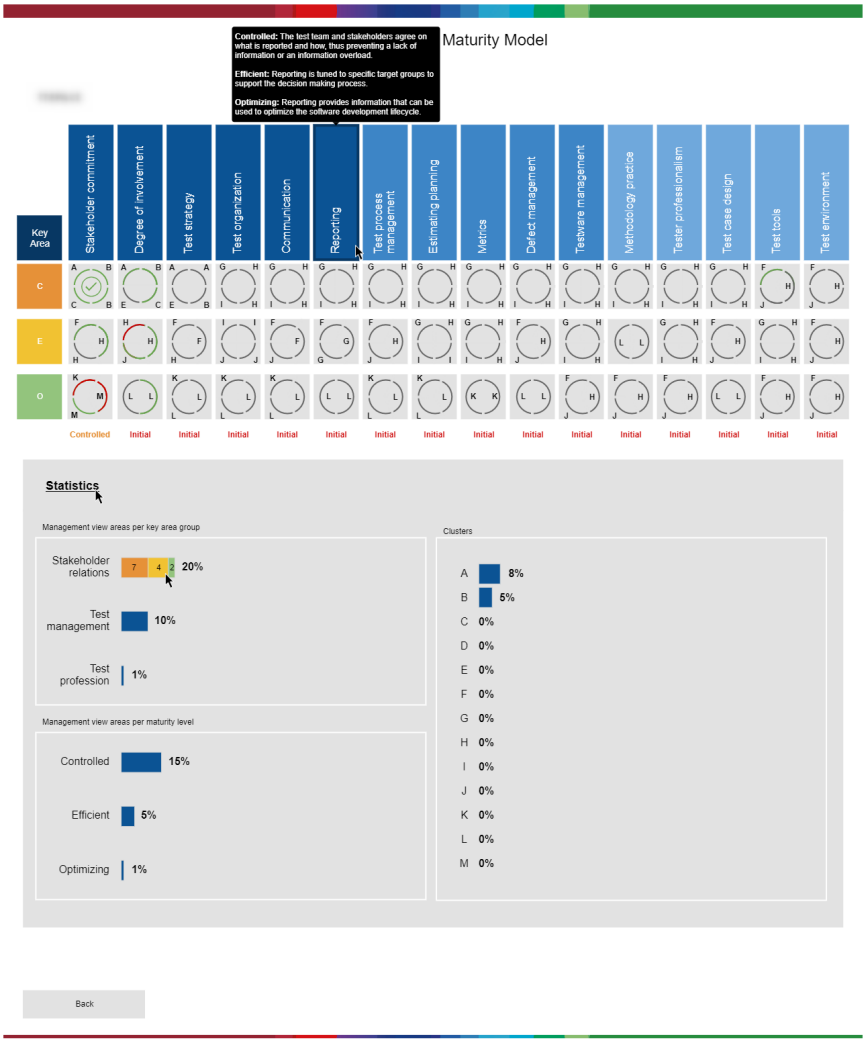

Figura 7: Página de análise do modelo de maturidade de testes de software com área de estatísticas expandida

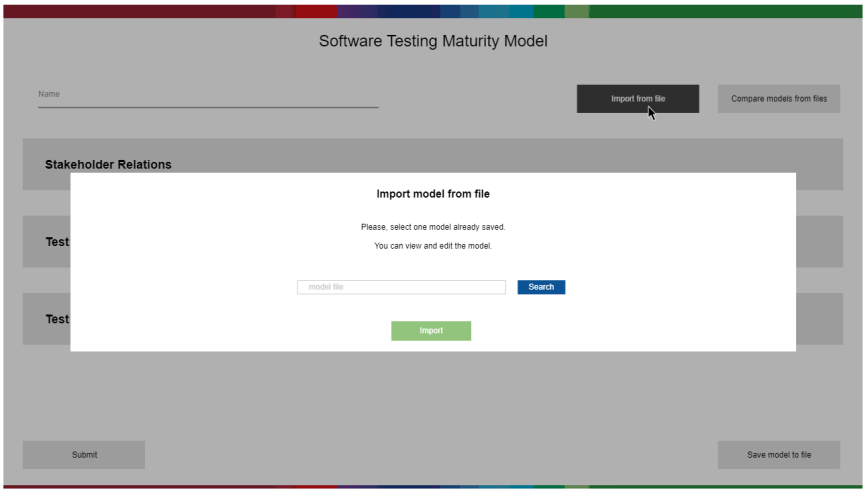

Figura 8: Página inicial do modelo de maturidade de testes de software em modo de importação de um ficheiro

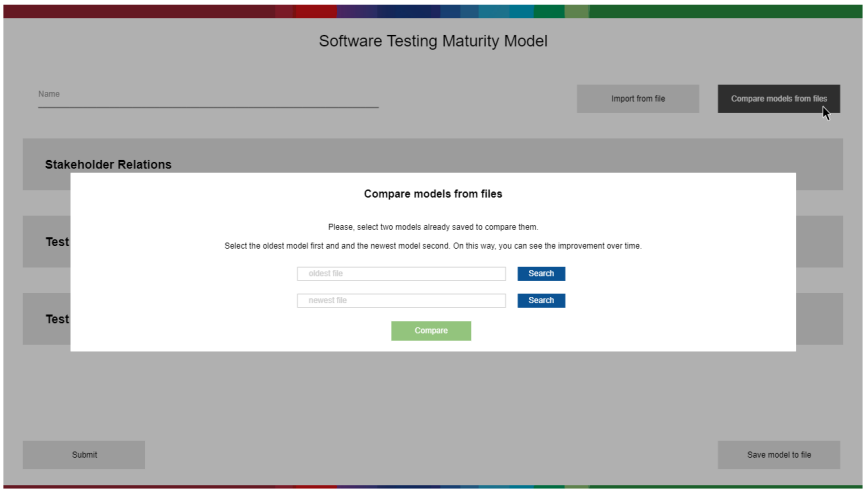

Figura 9: Página inicial do modelo de maturidade de testes de software em modo de comparação de dois ficheiros

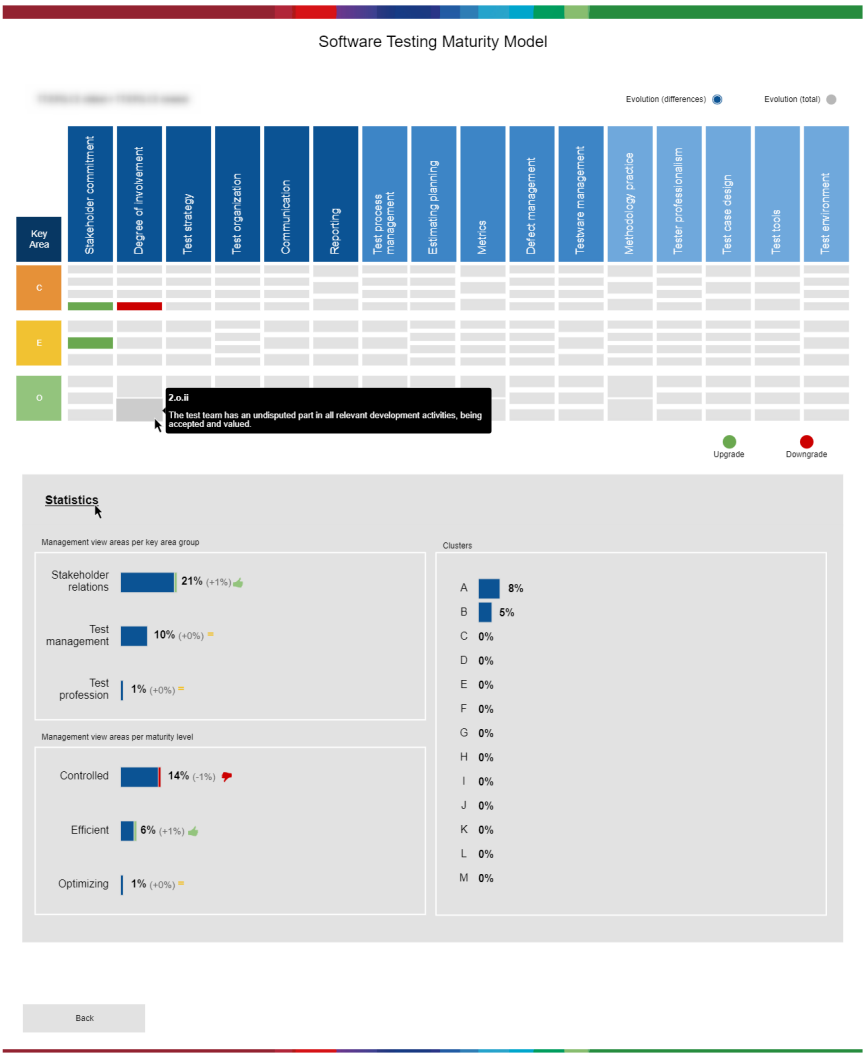

Figura 10: Página de análise do modelo de maturidade de testes de software para comparação entre dois ficheiros

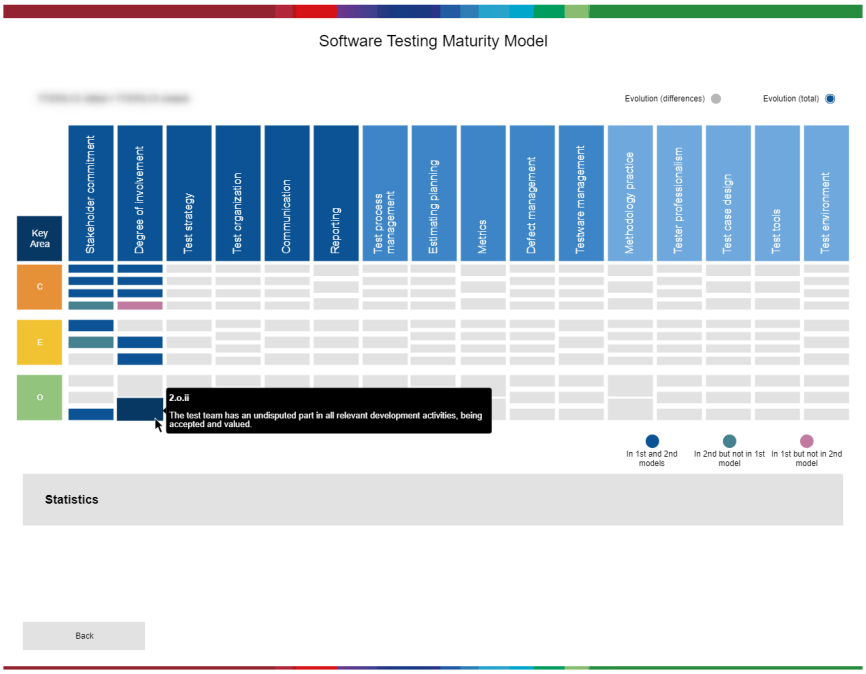

Figura 11: Página de análise do modelo de maturidade de testes de software para comparação entre dois ficheiros (2)

Apêndice B

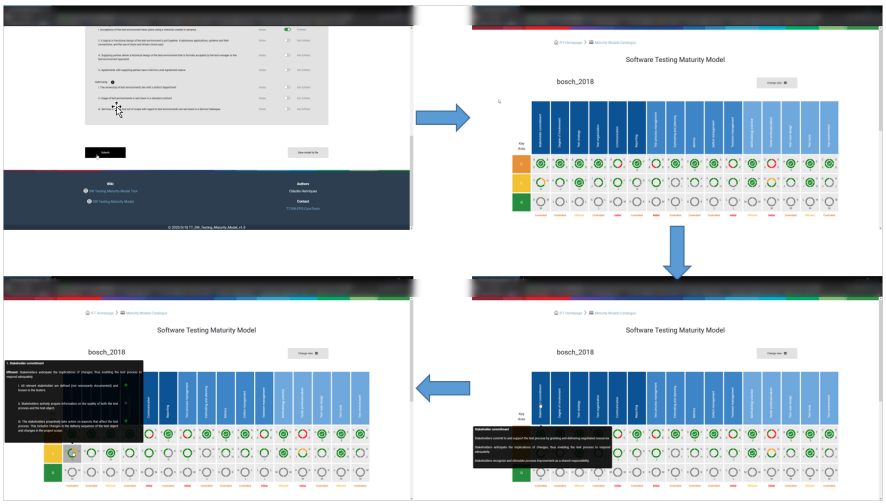

Figura 12: Submeter e visualizar do resultado do questionário

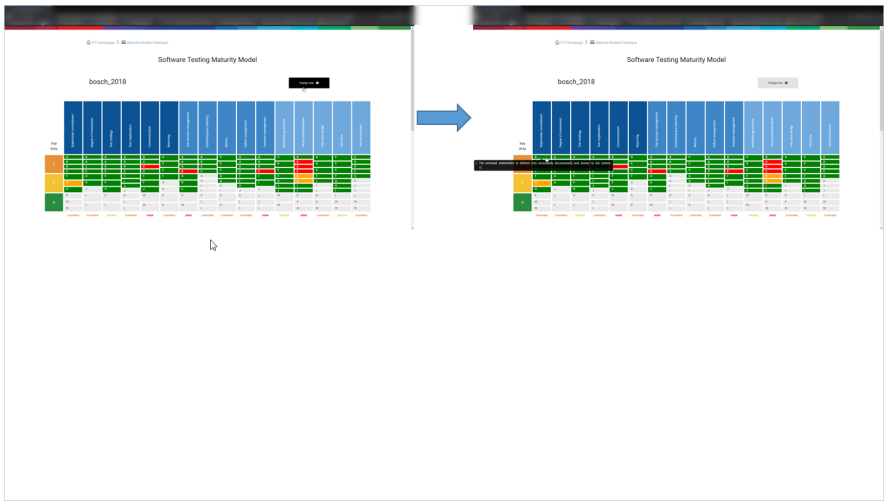

Figura 13: Alterar a vista do resultado do questionário

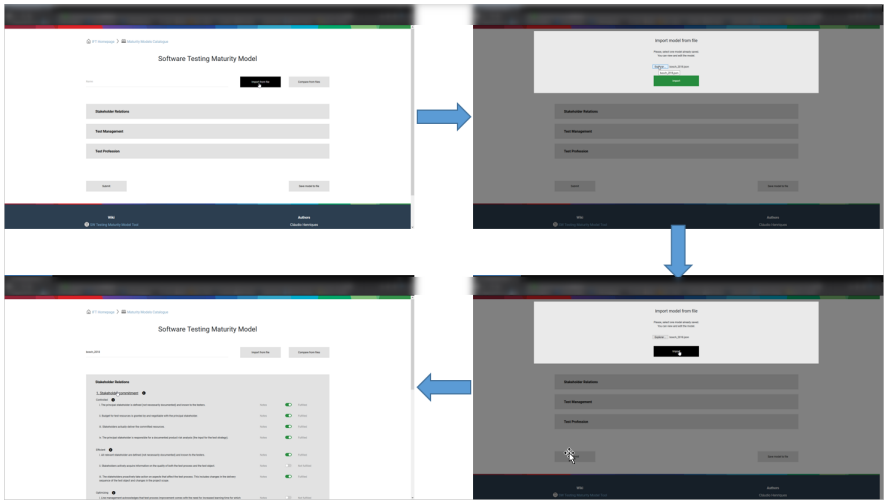

Figura 14: Importar um questionário

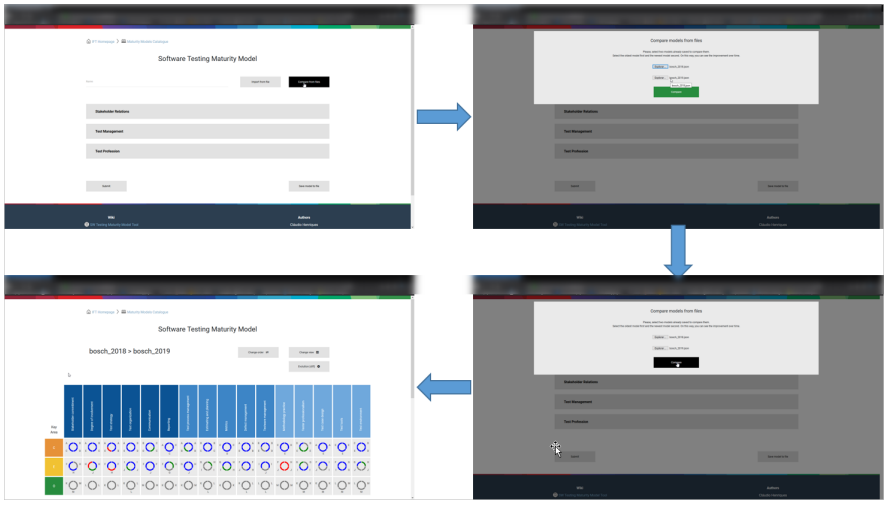

Figura 15: Importar dois questionários

Figura 16: Visualizar a comparação entre dois questionários

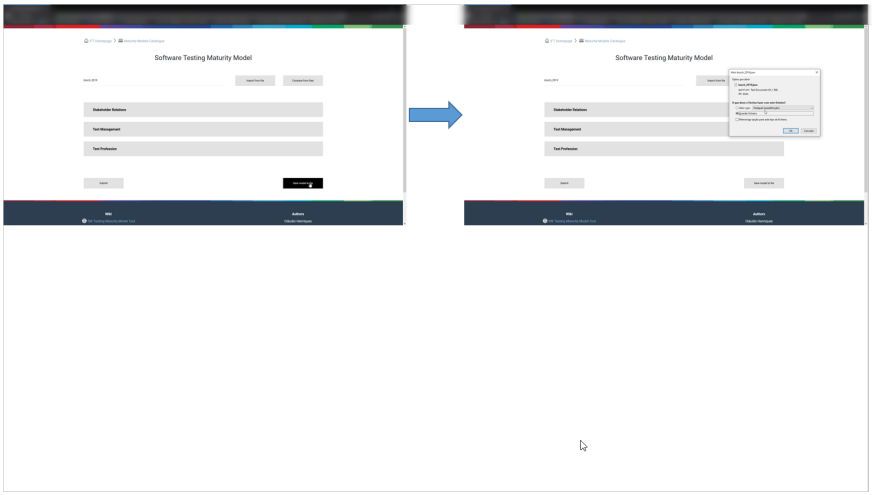

Figura 17: Exportar um questionário

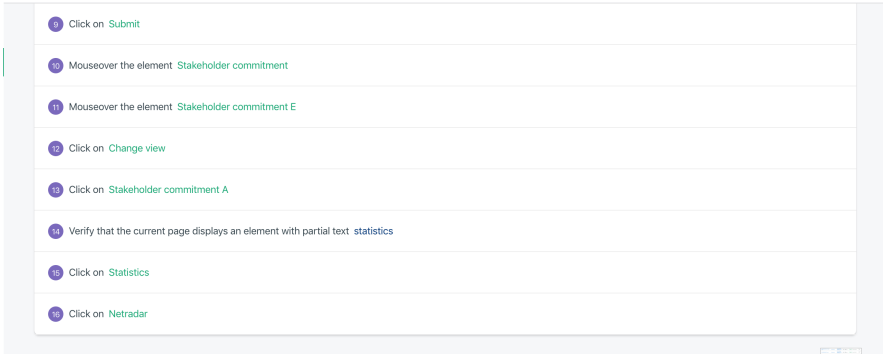

Figura 18: Caso de teste para validação das funcionalidades da submição do questionário

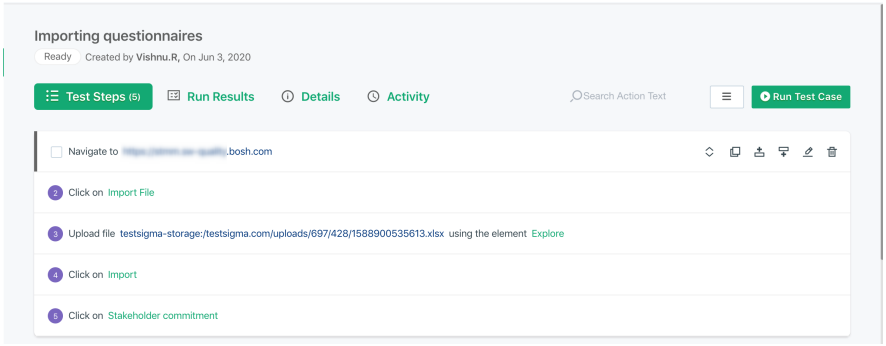

Figura 19: Caso de teste para validação da importação do questionário

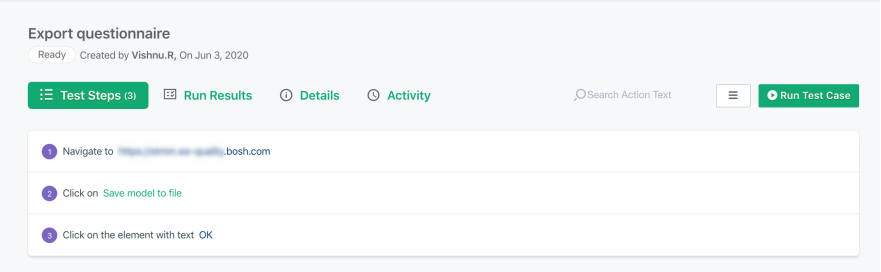

Figura 20: Caso de teste para validação da exportação do questionário

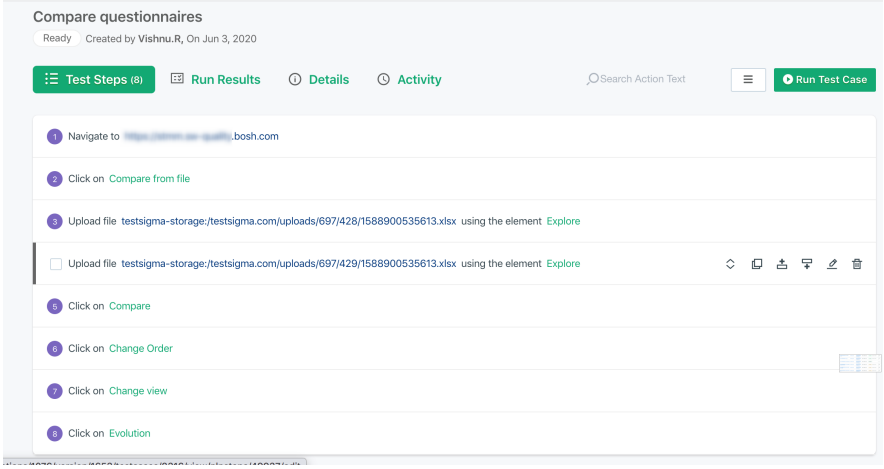

Figura 21: Caso de teste para validação da comparação entre dois questionários