INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Guia para aumentar a testabilidade do software

JOÃO CLÁUDIO DA ROCHA MARTINS
Junho de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Guia para aumentar a testabilidade do software

João Cláudio da Rocha Martins

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Alberto Sampaio

Coorientador: Isabel Sampaio

Porto, junho 2022

“Só fazemos melhor aquilo que repetidamente insistimos em melhorar. A busca da excelência não deve ser um objetivo, e sim um hábito.” (Aristóteles)

Resumo

A testabilidade tem sido estudada ao longo do tempo, havendo aspetos que a influenciam e métricas que permitem aferir o seu grau.

O presente trabalho foi desenvolvido com o objetivo de aumentar a cobertura e qualidade do software, potenciando para isso a testabilidade do mesmo. Este trabalho foi desenvolvido no âmbito da unidade curricular de Tese/Dissertação/Estágio, do Mestrado em Engenharia Informática (TMDEI), do Instituto Superior de Engenharia do Porto (ISEP).

Neste trabalho, realizou-se um estudo sobre o que influencia a testabilidade e de que forma esta pode ser avaliada e potenciada. Com base na informação recolhida, e tendo em conta o contexto da empresa promotora do estudo, Sysnovare, desenvolveu-se um guia, que contempla um conjunto de regras e sugestões, que visam permitir desenvolver código mais testável. O guia foi aplicado num projeto piloto, por uma equipa de desenvolvedores de software, tendo-se verificado, no final do período de avaliação da solução, melhorias que vão ao encontro dos objetivos pretendidos. A solução desenvolvida enquadra-se numa primeira fase de melhoria da qualidade do software desenvolvido na organização promotora do projeto, esperando-se que exista um trabalho futuro, que permita a continuação deste tipo de melhorias.

Palavras-chave: Testabilidade; Boas Práticas; Cobertura; Esforço para Testagem; Código Testável; Guia.

Abstract

Testability has been studied over time. There are some aspects that influence it and metrics that allow measuring its degree.

The present work was developed with the objective of increasing the coverage and quality of the software, enhancing its testability. This work was developed within the scope of the Thesis/Dissertation/Internship course, of the master's in computer engineering (TMDEI), of the Instituto Superior de Engenharia do Porto (ISEP).

In this work, a study was conducted on what influences testability and how it can be evaluated and enhanced. Based on the information collected and considering the context of the company promoting the study, Sysnovare, a guide was developed, which includes a set of rules and suggestions, which aim to allow the development of more testable code. The guide was applied in a pilot project, by a team of software developers, having verified, at the end of the solution evaluation period, improvements that meet the intended objectives. The developed solution is part of a first phase of improving the quality of the software developed in the organization promoting the project, and it is expected that there will be a future work that allows the continuation of this type of improvement.

Keywords: Testability; Good habits; Coverage; Testing Effort; Testable Code; Guide.

Agradecimentos

Primeiramente, agradeço ao Instituto Superior de Engenharia do Porto (ISEP) e a todos os docentes com os quais contactei, pelo conhecimento transmitido, não só no decurso do Mestrado em Engenharia Informática (MEI), como também ao longo da Licenciatura em Engenharia Informática (LEI). Gostaria ainda de agradecer em particular ao meu orientador, professor Alberto Sampaio, e à minha coorientadora, professora Isabel Sampaio, por todo o acompanhamento e orientação aquando da realização desta dissertação, estando certo de que o seu contributo promoveu em alta medida a qualidade da mesma.

De seguida, quero agradecer à Sysnovare pela oportunidade e pelas condições de trabalho proporcionadas, agradecendo, em especial, o supervisor que me acompanhou neste percurso, António Cunha, por toda a disponibilidade, apoio e ensinamentos transmitidos.

Seguidamente, agradeço a todos os meus colegas pelo seu contributo ao longo do meu percurso académico e profissional.

Por fim, mas não menos importante, não posso deixar de agradecer aos meus pais e à minha namorada, por todo o suporte e apoio que me dedicaram e proporcionaram, não só no decurso desta dissertação, mas em todo o meu percurso académico, profissional e pessoal.

A todos, e para todos, o meu mais sincero obrigado.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

API	*Application Programming Interface*
BD	Base de Dados
BPMN	*Business Process Model and Notation*
FTP	*File Transfer Protocol*
IDE	*Integrated Development Environment*
QEF	*Quantitative Evaluation Framework*
NPM	*Node Package Manager*
PL/SQL	*Procedural Language/Structured Query Language*
PNG	*Portable Network Graphics*
REST	*Representational State Transfer*
SMTP	*Simple Mail Transfer Protocol*
SOAP	*Simple Object Access Protocol*
SVG	*Scalable Vector Graphics*
TDD	*Test Driven Development*

1 Introdução

Neste capítulo é feita uma contextualização do tema do projeto, sendo analisado não só o problema que lhe deu origem, bem como os seus objetivos e o planeamento para o mesmo. Adicionalmente, é ainda descrita a metodologia de investigação seguida e a estrutura deste mesmo documento.

1.1 Contexto

A "**Sysnovare – Innovative Solutions, SA**" é uma empresa tecnológica, fundada em 2009, que fornece soluções digitais nas áreas de "Gestão Integrada de Contraordenações", "Gestão de Recursos Humanos", "Gestão académica", "Gestão de Despesas" e "Assinaturas Digitais" (Sysnovare, 2015b). A sua missão consiste em responder às necessidades dos seus clientes, mantendo uma relação de confiança com os mesmos. Desta forma, a empresa prima pela entrega de soluções de alta qualidade, fácil integração e eficiência (Sysnovare, 2015a).

A Sysnovare tem um conjunto de produtos já construídos, que vão sendo evoluídos ao longo do tempo, tendo em conta, por exemplo, as necessidades que vão sendo apresentadas pelos seus clientes. Os produtos evoluem de forma incremental, dando origem a novas versões ao longo do tempo.

Relativamente ao desenvolvimento de software, privilegiam-se metodologias ágeis, de modo a potenciar um valor incremental e iterativo, aumentando assim a escalabilidade dos produtos construídos (Srivastava et al., 2017). Sendo assim, para cada versão (cada *Sprint* corresponde a uma nova versão) são definidos uma série de objetivos a alcançar, sendo estes objetivos desenvolvidos e testados.

No que diz respeito ao número de colaboradores, a Sysnovare conta, à data, com cerca de trinta colaboradores, sendo cerca de metade destes colaboradores desenvolvedores de software.

1.2 Problema

O problema a solucionar diz respeito à **baixa testabilidade do software desenvolvido**.

O facto de os produtos desenvolvidos pela Sysnovare terem uma cobertura de testes inferior ao desejado dificulta a deteção de defeitos. Alguns destes defeitos são apenas detetados quando os produtos já se encontram em ambiente de produção, ou seja, a serem utilizados pelo utilizador final. Obviamente que esta é uma situação a evitar, tendo em conta que a Sysnovare pretende entregar os seus produtos com a maior qualidade possível, mantendo assim a satisfação dos seus clientes.

É ainda de salientar que o aparecimento deste tipo de defeitos pode interferir com o planeamento efetuado. Isto pode acontecer devido à necessidade de realocar recursos humanos, para resolver alguns problemas, havendo por vezes objetivos que acabam por não ser alcançados numa determinada versão, por causa do aparecimento de defeitos.

Devido à situação em cima relatada, pretendeu-se apurar a origem dos erros, percebendo-se que grande parte deles estavam afetos a componentes do software não cobertas pelos testes automáticos. Assim, quando confrontados com esta situação, os desenvolvedores da Sysnovare colocaram em questão o nível de testabilidade do software, tendo em conta o esforço elevado que assumiram sentir para o testar.

Com base nesta informação, surgiu a necessidade de entender como aumentar a testabilidade do software desenvolvido na Sysnovare.

1.3 Objetivos

Ambiciona-se aumentar a testabilidade do software desenvolvido na Sysnovare, de modo a otimizar o tempo consumido com a realização de testes automáticos, tornando o processo de testagem mais rápido. Assim, espera-se alcançar níveis de cobertura superiores aos registados anteriormente, contribuindo para uma maior confiabilidade do software, que culminará com um menor número de defeitos a serem detetados em momentos tardios.

Apesar da empresa desenvolver software recorrendo a três grandes tecnologias – PL/SQL, *NodeJs* e *AngularJs*-, o objetivo deste projeto é maximizar a testabilidade do código inerente à lógica de negócio, ou seja, o código desenvolvido do lado do servidor, em *NodeJs*. Esta decisão foi tomada pela empresa e assenta em dois fatores:

- a lógica de negócio é tratada no software desenvolvido em *NodeJs* e PL/SQL;
- o código desenvolvido em PL/SQL encontra-se atualmente a ser migrado para a tecnologia de *NodeJs*, sendo que se visa que no futuro apenas a última tecnologia referida seja responsável por este aspeto, não sendo objetivo da empresa aumentar a testabilidade do código já desenvolvido em PL/SQL.

Com o aumento da testabilidade ambicionam-se outros resultados:

1. Cobertura: ambiciona-se que a cobertura atinga níveis superiores (95%), de modo que a maior parte do software esteja devidamente testado;
2. Equipa de Desenvolvimento: pretende-se diminuir o esforço sentido pela equipa de desenvolvimento aquando da testagem do software;
3. Dependências: espera-se que o software não dependa de fatores externos para ser testado.

Estes objetivos foram definidos pela empresa e são apresentados com maior detalhe no Anexo A.

Para se conseguir atingir estes objetivos, deve-se elaborar um documento, identificado como Guia, que oriente os desenvolvedores nalgumas práticas que devem ser cumpridas e que auxilie na tomada de decisão. Assim, pretende-se que a equipa de desenvolvimento siga o referido guia, de modo a tornar o software mais testável. Existem alguns requisitos (colocados pela empresa) que o guia deve cumprir para potenciar o atingimento da solução desejada, quer em termos de estrutura, quer no que diz respeito ao enquadramento do mesmo no contexto da empresa. Estes requisitos encontram-se apresentados na Tabela 1.

Tabela 1 – Lista de requisitos do projeto.

Tópico	Requisitos
Guia	O guia deve apresentar regras que devem ser (sempre que possível) seguidas para aumentar a testabilidade.
	O guia deve apoiar a tomada decisão do desenvolvedor nalgumas questões ambíguas.
	O guia deve ser autoexplicativo.
	O guia deve sugerir a utilização de ferramentas que potenciem a melhoria da qualidade de código, nomeadamente da testabilidade.
	O guia deve ter em conta as dificuldades reportadas pela equipa de desenvolvimento.
	O guia deve estar acessível a qualquer membro da equipa de desenvolvimento.
	O guia deve ser editável, de modo a potenciar a sua evolução.
	O guia deve ser integrado nas práticas atuais de desenvolvimento de software da empresa.

1.4 Planeamento

Para atingir uma solução que permitisse resolver o problema em mãos, seguiram-se uma série de fases. Note-se que o planeamento não era estanque, sendo natural que algumas das fases fossem revisitadas ao longo da dissertação.

Inicialmente, averiguou-se o contexto da empresa no momento inicial da dissertação, de modo a perceber qual era o seu panorama, no que diz respeito à tecnologia em análise, práticas e processos presentes, níveis de cobertura e opiniões dos desenvolvedores.

De seguida, analisaram-se trabalhos realizados num âmbito semelhante ao atual, de modo a entender que conclusões já foram tiradas, trabalhando com essas informações, desenvolvendo um trabalho que acrescente o máximo de valor. Aqui foram também aferidas que tecnologias poderiam auxiliar na persecução do objetivo delineado.

Tendo como base a informação recolhida, fez-se uma análise que visou entender o que poderia ser aplicável no contexto do problema.

Foi também realizada uma Análise de Valor ao trabalho, de modo a entender a necessidade do mesmo, percebendo ainda o seu contributo.

Seguidamente, com base na análise prévia, construi-se um Guia escrito, que auxilia os desenvolvedores a tomarem decisões que potenciam a testabilidade do seu código e dos seus sistemas.

O guia desenvolvido foi usado por uma equipa de desenvolvimento, durante um período de tempo definido.

Tendo terminado esse período, foram recolhidos e analisados os dados resultantes da utilização do guia, de modo que fosse possível avaliar a adequação do mesmo, com base numa série de dimensões e fatores - mensuráveis – definidos previamente.

O planeamento descrito encontra-se na Figura 1.

Figura 1 – Cronograma do projeto.

1.5 Metodologia de investigação

De modo a validar e solidificar a pesquisa efetuada neste documento, aumentando a sua relevância, definiu-se um método de investigação a seguir na mesma, sendo este o *Design Science Research* (Pacheco Lacerda et al., 2013).

O *Design Science Research* (DSR) é uma abordagem metódica vocacionada para a criação de artefactos que visam solucionar determinados problemas, produzindo conhecimento científico através da construção de soluções inovadoras, destinadas a resolver problemas reais, fazendo, ainda, uma contribuição científica prescritiva (Dresch et al., 2015). Com a aplicação do DSR, visa-se obter uma solução, integrada num determinado domínio de um problema, que deve ser devidamente avaliada, tendo em consideração aspetos como valor e utilidade. Tem sido apontada como uma abordagem de pesquisa adequada a contextos onde existe colaboração com organizações, pretendendo-se testar ideias em conceitos reais, obtendo soluções satisfatórias para um determinado problema – não visando necessariamente a obtenção de uma "solução ideal".

Assim, o DSR visa a melhoria de processos/soluções ou até o apuramento de novos processos/soluções, estando enquadrado num determinado problema – e eventualmente

organização (Dresch et al., 2015). Desta forma, o DSR tem em consideração a relevância do estudado para o contexto onde se insere, privilegiando sempre o máximo de rigor, de modo a permitir que o estudado e apresentado seja de alta qualidade e confiabilidade. Na Figura 2, está apresentada a relação entre o DSR, o ambiente onde este se insere e o conhecimento que suporta o estudo em questão, sendo evidenciadas a necessidade de ter em consideração aspetos como relevância e rigor.

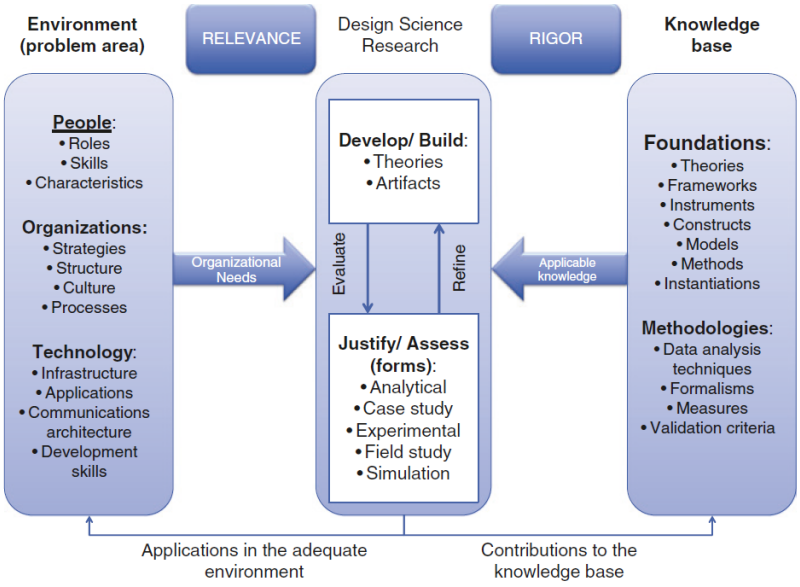

Figura 2 – Relevância e rigor no enquadramento da aplicação do DSR (Dresch et al., 2015).

Para suportar a pesquisa efetuada no DSR, devem-se seguir sete critérios (Dresch et al., 2015; Pacheco Lacerda et al., 2013):

1. Da pesquisa realizada devem resultar artefactos viáveis;
2. O objetivo da pesquisa é desenvolver soluções para resolver problemas específicos e relevantes para as organizações;
3. Devem-se demonstrar a qualidade e eficácia dos artefactos resultantes através de metodologias de avaliação;
4. A pesquisa efetuada deve trazer informações claras e verificáveis na área onde se inserem, sendo fundamental a existência de uma base clara para as mesmas;
5. A pesquisa deve ser rigorosa, tanto na construção da solução, como na sua avaliação;
6. A procura por um determinado artefacto carece da utilização dos meios disponíveis, satisfazendo as necessidades do ambiente do problema;
7. A pesquisa efetuada deve ser apresentada tanto para um público mais orientado para a área de tecnologia, como para um público associado à área de gestão.

É ainda de salientar que se conduziu a aplicação do DSR, segundo Dresch et al (2015), que apresenta uma sequência de passos padronizada. A referida sequência conta com a possibilidade de revisitar passos anteriores, caso exista a necessidade de suprimir aspetos que

possam não ter sido alcançados com o decurso normal dos vários passos, sendo uma aplicação mais flexível e que abrange um conjunto maior de possibilidades de sequências de ações.

Assim, realizaram-se os seguintes passos (Dresch et al., 2015):

- Identificou-se o problema;
- Analisou-se o problema, de modo a haver um maior conhecimento e consciencialização sobre o mesmo;
- Realizou-se uma revisão sistemática da literatura, analisando temáticas relacionadas com o problema em foco;
- Com base no estudo previamente efetuado, percebeu-se o que a solução final poderia incluir;
- Seguiu-se o desenho da solução;
- Iniciou-se a construção da solução;
- Aplicou-se a solução – este passo não se encontra especificado no método seguido, mas tornou-se essencial para potenciar a avaliação da solução;
- Avaliou-se a solução;
- Clarificou-se o que contribuiu positivamente e negativamente para a qualidade da solução;
- Realizaram-se algumas conclusões sobre os resultados atingidos, tendo em consideração as decisões que foram sendo tomadas ao longo do estudo;
- Extrapolaram-se distintas aplicações do trabalho realizado, para diferentes contextos, ambientes e problemas;
- Apresentou-se e debateu-se o resultado atingido com as partes interessadas, nomeadamente equipa de desenvolvimento da Sysnovare e membros da gestão da Sysnovare.

1.6 Estrutura do Documento

No primeiro capítulo, são enquadrados o projeto e a empresa promotora do mesmo, sendo ainda descrito o problema, os objetivos, a abordagem e o planeamento do projeto.

No segundo capítulo, é feita uma recolha do estado da arte no que diz respeito a testes de software, testabilidade do software e contexto da empresa.

No terceiro capítulo, é feita uma análise de valor do projeto, de modo a entender o valor do mesmo para as várias partes interessadas.

No quarto capítulo é apresentado o desenho da solução, sendo explicada a forma como esta será construída e ainda referidas algumas alternativas às decisões tomadas.

No quinto capítulo descreve-se a solução desenvolvida, explicando-se a forma como esta foi entregue e quais os seus principais conteúdos.

No sexto capítulo, avalia-se a solução, de modo a entender o que se pode concluir acerca da mesma.

Por fim, no sétimo e último capítulo, realizam-se algumas conclusões sobre o trabalho realizado nesta dissertação, criticando-se o efetuado e extrapolando-se o eventual trabalho futuro associado a este projeto.

2 Estado da Arte

No presente capítulo enquadram-se, primeiramente, alguns conceitos relacionados com testes de software. Seguidamente, apresenta-se o contexto da empresa (no início do projeto) e analisa-se a temática em foco - a testabilidade-, no que diz respeito aos fatores que a influenciam, métricas relacionadas e ferramentas relacionadas.

2.1 Testes de software

A testagem do software permite validar se um determinado software faz aquilo para o qual foi construído, sendo desta forma possível prevenir o aparecimento de erros, reduzir os custos de desenvolvimento e melhorar o desempenho (IBM, n.d.). Estes tipos de testes podem ser manuais ou automáticos.

Tanto os testes manuais, como os automáticos, envolvem **casos de teste**. Um caso de teste é definido através de uma especificação de parâmetros de entrada, execução de condições e verificação de resultados, permitindo averiguar se uma determinada funcionalidade apresenta o comportamento esperado (Hue et al., 2018; Sharma R. M., 2014).

Os testes manuais são a técnica mais antiga de testagem de software (Sharma R. M., 2014). Nesta técnica, o testador executa os casos de teste manualmente, sem suporte de software que agilize o processo, de modo a encontrar potenciais defeitos ou validar um determinado código.

Os testes automáticos são suportados por software e permitem uma execução simultânea (e repetitiva) de vários casos de teste pré-definidos, sendo a intervenção humana reduzida ao mínimo (Sharma R. M., 2014).

De acordo com estudos (Dobles et al., 2019; Sharma R. M., 2014), os testes automáticos apresentam algumas vantagens, quando comparados com os testes manuais. De seguida são enumerados as consideradas mais relevantes (Dobles et al., 2019; Sharma R. M., 2014):

- Os testes automáticos permitem uma monitorização contínua do software já testado, tendo em conta que são repetíveis.
- Os testes automáticos são mais confiáveis que os testes manuais (porque há minimização do erro humano).
- Os testes automáticos requerem um menor número de recursos humanos para execução dos casos de teste, o que diminui o seu custo.
- Os testes automáticos fornecem informação detalhada acerca da cobertura do software em análise.
- Os testes automáticos acarretam menos custos. Apesar do custo inicial associado aos testes automáticos ser superior – essencialmente devido ao tempo empregado na

construção de *scripts* de testagem e na configuração de diversos aspetos -, verifica-se que, a longo prazo, este curso é amortizado, devido à possibilidade de execuções múltiplas e repetidas destes testes.

- Os testes automáticos são mais eficazes na deteção de defeitos.

Existem vários tipos de testes automáticos, como por exemplo testes unitários e testes de integração. Os testes unitários visam apurar a integridade de um excerto de um software específico, tipicamente uma função ou um módulo (Lewis, 2009). Por outro lado, os testes de integração propõem testar uma combinação de "componentes" de um software, de modo a apurar o seu correto funcionamento. Estes "componentes" podem variar desde módulos, a diferentes servidores, tratando-se de um tipo de testagem apropriada a sistemas distribuídos e cliente/servidor.

2.2 Testabilidade do software

A testabilidade do software consiste no esforço necessário para testar esse mesmo software (Engineering Standards Committee of the IEEE Computer Society, 2010).

Nesta secção, visa-se fazer um levantamento do enquadramento da testabilidade na atualidade, dos critérios principais que a influenciam, das métricas que permitem aferir o seu grau e ainda das ferramentas que poderão auxiliar ao seu crescimento/monitorização.

2.2.1 Estudos sobre Testabilidade

A testabilidade de software tem sido estudada, tanto por investigadores, como por profissionais (Garousi et al., 2019). Estes estudos incidem, maioritariamente, nas abordagens adequadas à medição e melhoria da testabilidade, havendo, ainda, estudos sobre outros aspetos, como por exemplo a testabilidade de requisitos e o desenho para testabilidade.

Na área de testes, o número de estudos sobre testabilidade é inferior ao de outros tópicos, como, por exemplo, *Search-Based Software Testing* (SBST) ou *Mutation Testing* (Garousi et al., 2019). De seguida, na Figura 3, podemos ver um gráfico ilustrativo desta mesma discrepância.

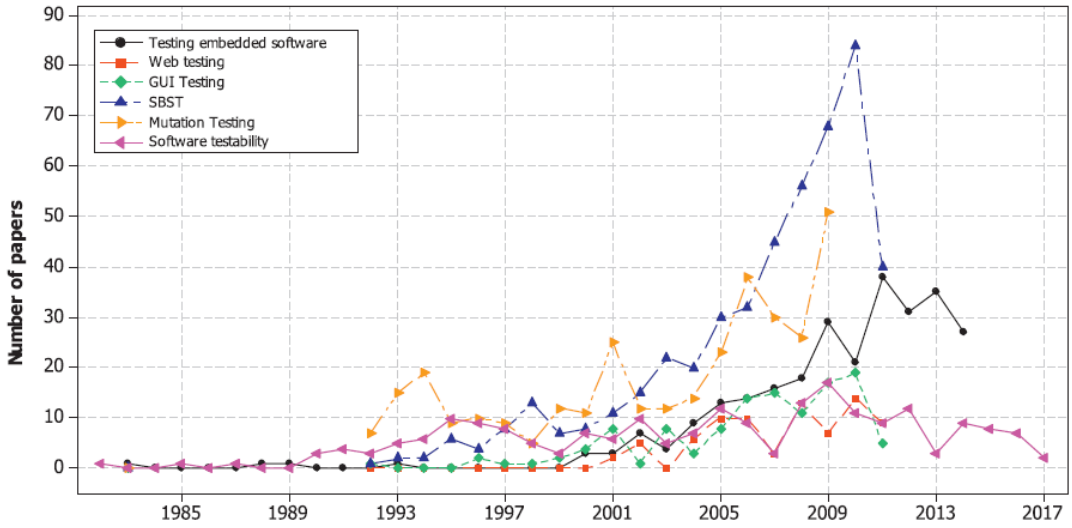

Figura 3 – Evolução do número de estudos sobre testabilidade do software, em comparação com outras áreas (Garousi et al., 2019).

Como é possível constatar no gráfico, segundo este estudo – que compara o número de artigos de seis áreas relacionadas com a testagem -, entre a última década e meia do século XX e a primeira década e meia do século XXI (sensivelmente), a testabilidade, a par de *Web Testing* e a *Gui Testing,* é uma das três áreas com o menor número de estudos (Garousi et al., 2019). Este fator pode ser explicado pelo facto de a testabilidade ser um conceito mais extenso e abstrato, quando comparando com as restantes áreas apresentadas no estudo.

O facto de a testabilidade ter um número inferior de estudos, quando comparada com outras áreas, não invalida que os estudos existentes na área tenham uma diversidade relativamente alta (Balogun et al., 2020). Alguns destes estudos são direcionados a contextos específicos, como paradigmas de programação, analisando por exemplo métricas que podem ser incorporadas em softwares de análise automática de testabilidade de projetos assentes em programação orientada a objetos. Outros estudos optam por construir modelos que calculam a testabilidade, tendo em conta vários aspetos, como a observabilidade, controlabilidade, interação entre classes ou dependências entre as mesmas (Bruntink & van Deursen, 2006a). Existem ainda estudos que visam prever a testabilidade tendo em conta modelos *Unified Modeling Language* (UML) ou o ciclo de desenvolvimento do software. Finalmente, existem estudos que visam aferir a testabilidade tendo em conta uma série de métricas.

De modo a analisar a evolução da testabilidade ao longo dos anos, pesquisaram-se as bibliotecas digitais *Google Scholar*, *ACM Digital Library*, *Digital Bibliography & Library Project (DBLP)* e *Science Direct* quanto ao número de estudos sobre o tema. A pesquisa foi feita através de *Systemic mapping review* (SMR) (Ann Kitchenham et al., 2015; Sampaio, 2015). Como critério de inclusão/exclusão definiu-se que se pretendiam documentos para um intervalo de anos especificado, entre dois mil e dezassete e dois mil e vinte e um. Utilizaram-se como expressões de pesquisa "testabilidade de software" e "*software testability*". Os resultados obtidos estão apresentados na Figura 4.

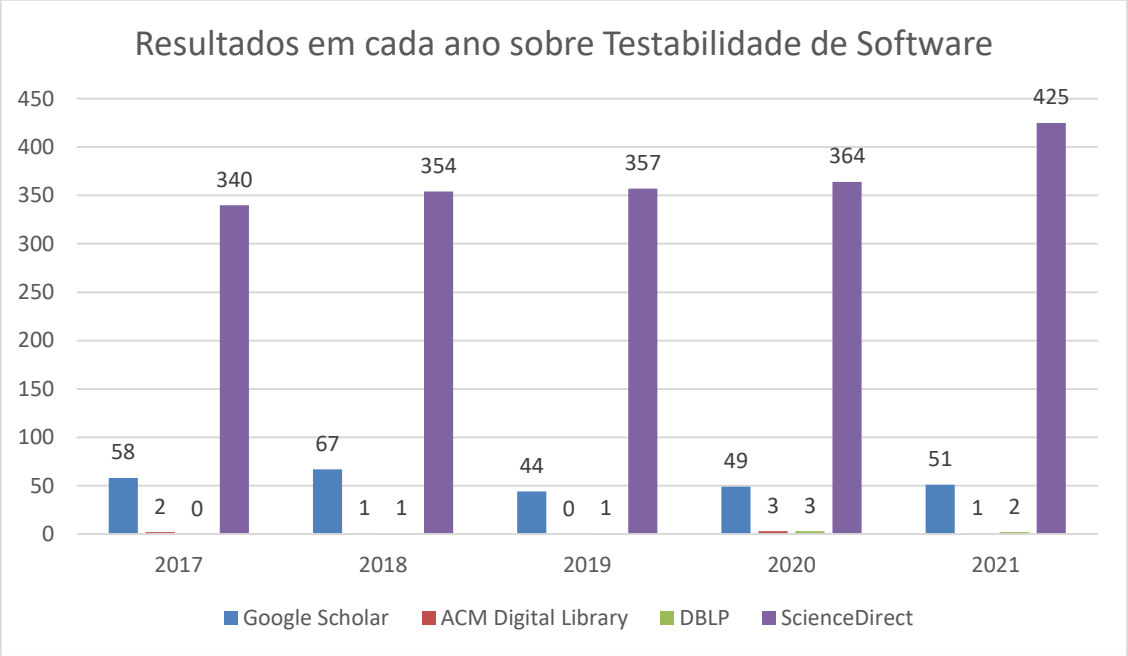

Figura 4 - Número de Resultados apresentados pelo *Google Scholar* para cada ano na pesquisa efetuada sobre testabilidade de software. Resultados obtidos em 2022-01-28.

Como se pode verificar na Figura 4, no que concerne ao *Google Scholar*, dos últimos cinco anos, 2018 foi o ano com um maior número de resultados apresentados, havendo um significativo decréscimo em 2019. Não obstante, o número de resultados tem vindo a aumentar gradualmente nos últimos três anos. Relativamente aos resultados apresentados na ACM Digital Library e na DBLP não parece haver nenhuma tendência, sendo ainda de salientar que o número de resultados apresentados foi manifestamente inferior. Por fim, relativamente aos dados apresentados pela Science Direct, nota-se um crescimento gradual do número de resultados desde 2017 até 2021 – com um aumento mais considerável nos últimos dois anos -, sendo o número de resultados manifestamente superior aos apresentados pelas restantes bibliotecas.

2.2.2 O que Influencia a Testabilidade

São vários os fatores que podem impactar a testabilidade, como por exemplo alguns atributos de qualidade particulares (Garousi et al., 2019). Ao longo desta secção serão explicados os considerados mais relevantes, tendo em conta a revisão de literatura efetuada. Pretende-se que alguns destes fatores influenciem o desenvolvimento da solução. Note-se que alguns fatores vão sendo apresentados mais frequentemente que outros, como verificável na Figura 5.

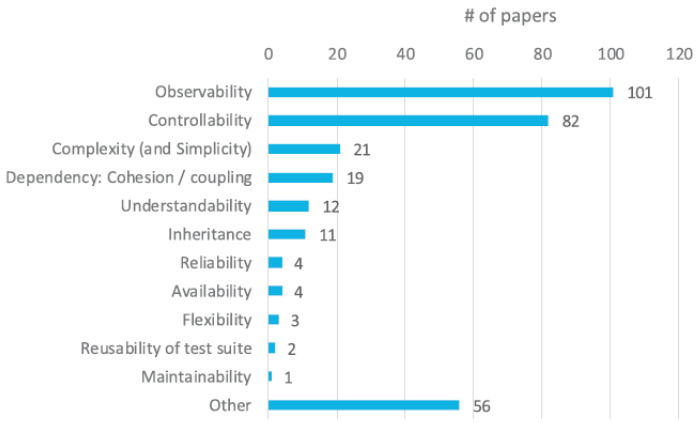

Figura 5 – Histograma com fatores que influenciam a testabilidade, segundo o reportado num conjunto de estudos (Garousi et al., 2019).

Através da análise do gráfico da Figura 5 é notável a tendência de referenciação da observabilidade e da controlabilidade como fatores que influenciam a testabilidade. Adicionalmente, outros fatores vão sendo reportados em menor abundância, como por exemplo a coesão, a complexidade e a compreensibilidade.

Alguns dos fatores encontram-se diretamente relacionados, como por exemplo a controlabilidade, que é implicada pela isolabilidade e pela capacidade de implantação, como pode ser visto na Figura 6 (Tarlinder, 2017).

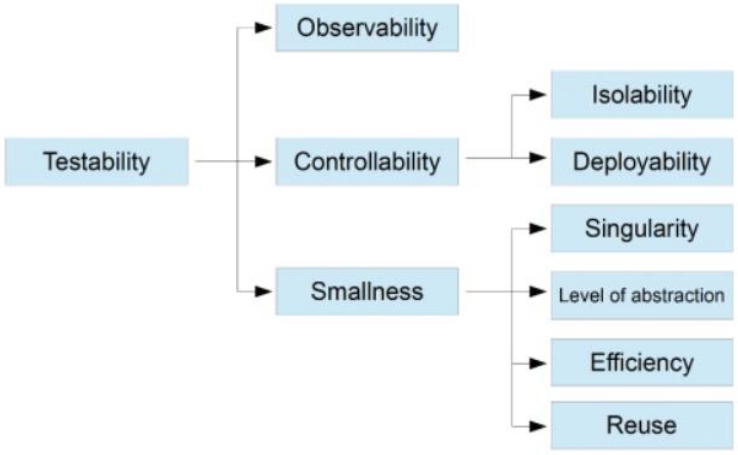

Figura 6 - Exemplo de correlação de fatores que implicam a testabilidade (Tarlinder, 2017).

2.2.2.1 Débito Técnico

O débito técnico recai no custo associado à necessidade de modificação de código já existente, devido a este ter sido construído segundo uma abordagem menos completa, mas inicialmente mais rápida (Allman, 2012). Assim, a decisão tomada pelos desenvolvedores tem implicações em ações futuras de manutenção, sendo que decisões de menor qualidade podem causar atividades de modificação de maior custo, podendo este aspeto afetar negativamente a testabilidade (Kruchten et al., 2012).

2.2.2.2 Legibilidade

A legibilidade é um dos fatores anunciados em vários trabalhos que tratam o tema da testabilidade, e resulta essencialmente da clareza inerente ao código de um determinado software (Boswell et al., 2012). A legibilidade de um determinado excerto de código aumenta consoante o grau de compreensão do mesmo. Da maior compreensão do código (maior legibilidade) advém uma maior facilidade de testagem, tornando, inerentemente, o software mais testável. A falta de comentários elucidativos e nomenclaturas coerentes, referidos no questionário inicialmente aplicado à equipa de desenvolvimento, podem contribuir para uma menor legibilidade do software.

2.2.2.3 Observabilidade

A observabilidade é frequentemente enunciada como um fator fulcral para a testabilidade, consistindo na capacidade de acompanhar, medir e estimar o estado interno de um sistema, tendo em conta os seus dados de saída – *outputs* (Bruntink & van Deursen, 2006b; Filho et al., 2020; Garousi et al., 2019; Roy S. Freedman, 1991). Segundo apurado, uma maior observabilidade facilita a testabilidade.

2.2.2.4 Controlabilidade

A controlabilidade é também apresentada como essencial para a testabilidade, residindo na capacidade de controlar o estado interno de um sistema, sendo para isso importante conseguir acompanhar o seu desenrolar, resultando aqui uma alta relação com a observabilidade, previamente descrita (Bruntink & van Deursen, 2006b). Uma maior controlabilidade, como acontece com a observabilidade, contribui para um aumento da testabilidade (Garousi et al., 2019). A controlabilidade e a observabilidade são os dois fatores mais frequentemente apontados como principais influenciadores da testabilidade (Filho et al., 2020; Garousi et al., 2019; Gill, 2021).

Como será referido na secção referente à análise interna, no momento de início da dissertação, não existia nenhuma ferramenta de depuração de código (ver 2.3.3). Este tipo de ferramentas permite ter uma visão mais pormenorizada do comportamento do software, podendo assim aumentar a observabilidade e a controlabilidade, aumentando concomitantemente a testabilidade (Silva, 2011).

2.2.2.5 Compreensibilidade

A compreensibilidade diz respeito a quanto um artefacto de software é autoexplicativo (Gao & Shih, 2005). Pode estar refletida na documentação de um determinado requisito, nos comentários inerentes a um excerto de código, ou mesmo na documentação relativa ao software em desenvolvimento – seja este uma API ou uma biblioteca. Quanto maior for a compreensibilidade maior será a testabilidade.

2.2.2.6 Disponibilidade

A disponibilidade diz respeito à capacidade de resposta que um sistema apresenta quando é feito algum requerimento (Garousi et al., 2019). A especificação da necessidade de uma alta disponibilidade para determinados componentes específicos de um produto pode dificultar a testabilidade (González et al., 2009). Se um componente tem requisitos de alta disponibilidade

em tempo de execução, a realização de testes (sobre este atributo de qualidade) pode afetar essa mesma disponibilidade, tornando-se difícil perceber que testes devem ser realizados, de modo que o sistema em execução não seja afetado notoriamente. Por outro lado, segundo o aferido, a disponibilidade da documentação relativa a um determinado componente de software (como por exemplo Especificação de Requisitos ou Manual de Utilizador) é importante para potenciar a testabilidade do mesmo (Gao & Shih, 2005).

2.2.2.7 Flexibilidade

A flexibilidade reside na capacidade de adaptação de um projeto para fornecer capacidades relacionadas com as presentemente existentes (O'Keeffe & Ócinnéide, 2006). Uma alta flexibilidade promove uma maior testabilidade (Abdullah et al., 2015; Goel & Gupta, 2012).

2.2.2.8 Isolabilidade

A isolabilidade prende-se no grau no qual cada componente pode ser isolado, podendo ser testado separadamente, não apresentando assim dependências para com outros componentes (Ingeno, 2018; Richards, 2015). Quanto maior for a isolabilidade de um componente, maior será a sua testabilidade.

2.2.2.9 Acoplamento

O acoplamento consiste no grau de dependência entre diferentes classes, ou seja, a dependência que determinadas classes mantêm com outras, para que possam funcionar (Robert C. Martin, 2009). Quanto maior a dependência entre classes, maior o seu acoplamento e menor a sua isolabilidade. Quanto maior é o acoplamento verificado, menor é a testabilidade de um software, pelo que se deve tentar construir sistemas com baixo acoplamento, que são, consequentemente, mais testáveis (Khan & Mustafa, 2009). Para diminuir o acoplamento, podem-se explorar diversas técnicas, como o *Dependency Injection (DI)*, aumentando, desta forma, a testabilidade (Ingeno, 2018). Existem alguns padrões que potenciam a DI, como o *Constructor Injection*, *Property Injection*, *Method Injection* e *Service Locator*.

2.2.2.10 Herança

A herança permite a substituição dinâmica de um objeto (programação orientada a objetos) por uma versão derivada do mesmo, promovendo a reutilização e o desenvolvimento incremental (Canal et al., 2001). Segundo o apurado, a herança prejudica a testabilidade, pelo que, quanto maior for a dependência hierárquica verificada devido à Herança, mais difícil será testar o software (Mouchawrab et al., 2005).

2.2.2.11 Redundância

A existência de código repetitivo e casos de teste redundantes não beneficia a testabilidade de um componente, pelo que devem ser evitados (Chowdhary, 2009; Garousi et al., 2019).

2.2.2.12 Padrões de Software

Padrões de software são soluções testadas e apresentadas para resolver problemas comuns e repetitivos (Bafandeh Mayvan et al., 2017). A utilização adequada de padrões de software é enunciada como benéfica para o aumento da testabilidade (Chowdhary, 2009). Contudo, deve-se ter em consideração que os padrões apresentados não resolvem todos os problemas,

podendo uma adoção desmedida/inconsciente dos mesmos potenciar outro tipo de problemas afetos à testabilidade ou outros atributos de qualidade do software (Baudry et al., 2003). Assim, a utilização dos diferentes padrões de software deve seguir um conjunto de regras claro, que permita aos desenvolvedores entender, de forma inequívoca, quando e de que forma os devem aplicar.

2.2.2.13 Simplicidade

A simplicidade traduz-se em vários aspetos, alguns deles já mencionados, como por exemplo o baixo acoplamento (Chowdhary, 2009). Pode ser definida como o grau de divisão dos vários artefactos de um software, devendo estes estar devidamente segregados, com base nas suas responsabilidades (Garousi et al., 2019). Uma alta simplicidade atrai uma alta testabilidade.

2.2.2.14 Complexidade

Os termos de complexidade e simplicidade são, em certa medida, opostos. A complexidade consiste no número de partes que compõe um software e as suas inter-relações (Garousi et al., 2019). Uma menor complexidade poderá contribuir para uma maior testabilidade.

2.2.2.15 Tempo de Execução de Testes

Quer sejam realizados testes manuais, quer sejam executados testes automáticos, o tempo inerente à execução dos mesmos, caso excessivo, prejudica a testabilidade (Garousi et al., 2019).

2.2.2.16 Automatização

A automatização é o nível do qual um processo ou ação pode ser automatizada (Ingeno, 2018). Das várias tarefas que podem ser automatizadas, deve-se aqui salientar a automatização dos testes - recorrendo a testes automáticos - ou a automatização de processos de construção e entrega da solução. A automatização deve ser trabalhada, de modo a potenciar a testabilidade (Bruntink & van Deursen, 2004).

2.2.2.17 Ferramentas

As ferramentas utilizadas para a realização de testes podem influenciar a testabilidade sentida na execução dos mesmos (Bruntink & van Deursen, 2004). Assim, devem-se privilegiar ferramentas de simples utilização, que diminuam a carga de responsabilidades do desenvolvedor. Adicionalmente, a definição de casos de teste na presença de Interfaces Gráficas é um exemplo de como reduzir esforço.

2.2.2.18 Documentação

A documentação de um projeto e de todos os artefactos a si anexos é um fator importante para clarificar o mesmo, a nível do seu propósito, estrutura, ciclo de vida, necessidades, comportamentos, entre outros, havendo várias razões que justificam o acompanhamento de um produto com um conjunto de documentação associada (Bruntink & van Deursen, 2004). No contexto da testabilidade, a documentação é importante para auxiliar no desenvolvimento de código claro e na posterior construção e concretização de testes adequados.

2.2.2.19 Desenvolvedor/Testador

O conhecimento que o desenvolvedor e/ou testador apresenta, quer a nível técnico, quer a nível de entendimento do problema e dos seus requisitos, pode influenciar a testabilidade

(Kedemo, 2015). Adicionalmente, o estado mental e físico do desenvolvedor/testador pode influenciar, quer negativamente, quer positivamente, a testabilidade. Por vezes, os desenvolvedores não partem para a testagem por entenderem que esta tem um custo elevado ou por sentirem que devem direcionar os seus esforços a outras ações (Ghafari et al., 2019).

2.2.2.20 Produto

O contexto do produto pode afetar também a testabilidade (Kedemo, 2015). Como contexto do produto entendam-se aspetos como paradigmas de desenvolvimento, recursos (tempo, dinheiro, ferramentas), relações (entre equipas, clientes e terceiros) e até eventuais riscos relacionados com o próprio projeto, equipa, empresa ou carreiras. Adicionalmente, o produto em si, bem como os seus aspetos anexos, como a sua visão, objetivos e requisitos, podem influenciar a própria testabilidade.

2.2.2.21 Requisitos

Alguns dos fatores já apontados podem estar relacionados diretamente com requisitos funcionais ou não funcionais, sendo este um aspeto também impactante no que respeita à testabilidade de um software (Garousi et al., 2019). A exigência de determinados aspetos para um software pode facilitar/dificultar a sua testagem, como já referido anteriormente, quando se explicou o conceito de disponibilidade, tendo-se percebido que a sua necessidade pode comportar alguma dificuldade/indefinição, aquando da testagem do software.

2.2.2.22 Dependências Externas

As dependências externas são aqui entendidas como todos os fatores externos a um componente, que influenciam diretamente a sua testagem — internet, configurações, integrações com componentes externos, entre outros. Assim, cada componente é tanto mais testável quanto menor for o número de dependências externas (Chowdhary, 2009). Idealmente, cada componente deveria poder ser testado individualmente, sem nenhuma necessidade de configurações específicas nem intervenientes externos.

2.2.2.23 Ambiente e Processo

O ambiente sobre o qual os testes são executados pode também impactar a testabilidade do software (Garousi et al., 2019). O ambiente incorpora vários fatores como ferramentas utilizadas, dependências internas e externas, detalhes do produto, requisitos e processo no qual este se encontra inserido.

Como será explicado em 2.3.3, na empresa, os testes são realizados depois da codificação do respetivo software, seguindo uma abordagem de *Test Last Development*. No entanto, existem técnicas, como o *Test Driven Development*, que potenciam a testagem do software (Spirlandeli et al., 2019). Assim sendo, a adoção de TDD – técnica de programação onde os testes de software são escritos antes do código fronte - potencia a testabilidade do software, tendo em conta que os desenvolvedores constroem o seu código com base na necessidade de cumprir com os testes previamente delineados, obrigando a uma atividade contínua de retrospeção e de autoavaliação do software, garantindo ainda que todos os requisitos inicialmente colocados para uma determinada funcionalidade são cumpridos (Aniche & Gerosa, 2015; Ethan Trostler, n.d.; Gencel & Demirors, 2007; Ghafari et al., 2019). Entende-se então, com base na informação

aqui apresentada, que o processo seguido para o desenvolvimento e testagem do software pode ter influências em aspetos como a sua cobertura e testabilidade.

2.2.2.24 Desenho e Arquitetura da Solução

A forma como cada componente se encontra desenhada, quer no que diz respeito à sua arquitetura, quer no que concerne ao seu fluxo de interações, influencia em grande medida a testabilidade (Garousi et al., 2019). Relativamente a este aspeto, devem ser tidos em conta fatores já identificados, como a observabilidade e controlabilidade, aquando do desenho e arquitetura de cada solução - a solução deve ter uma arquitetura simples e adequada, que suporte a sua testagem.

2.2.2.25 Casos de teste

Os casos de tese, referidos em 2.1, podem também eles impactar a testabilidade do software, no que diz respeito à sua definição e validação (Garousi et al., 2019).

2.2.2.26 Critérios de Testagem

Um dos fatores que pode ter impacto no esforço associado à testagem de um software está relacionado com os critérios definidos, como por exemplo o nível de cobertura mínimo delineado ou os componentes que são incluídos no processo de testagem/planeamento (Bruntink & van Deursen, 2004).

2.2.3 Métricas para Testabilidade

São várias as métricas apresentadas em diversos estudos para aferir a testabilidade, havendo trabalhos que utilizam, para este efeito, alguns dos aspetos descritos na secção anterior, como, por exemplo, a observabilidade (Filho et al., 2020; Sharma & Saha, 2018).

Estas métricas visam correlacionar determinados aspetos previamente definidos, de modo a designar qual o nível de testabilidade de um software, sendo comum algumas métricas permitirem conjeturar quantitativamente o esforço previsto para a testagem (Schrammel, 2020).

Da mesma forma, é possível correlacionar as várias métricas, sendo que umas nutrem mais proximidade entre si do que outras, como se pode verificar na Figura 7 (Terragni et al., 2020).

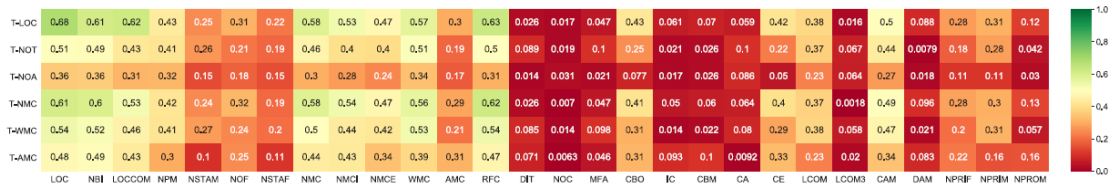

Figura 7 - Coeficiente de correlação de classificação de Spearman (valores absolutos) - as métricas de esforço de teste são normalizadas pela cobertura de linha (Terragni et al., 2020).

Na Figura 7, são especificados valores absolutos, entre zero e um, de correlação entre métricas de aferição de testabilidade, onde os valores mais próximos de um indicam uma maior correlação.

Nesta secção visa-se realizar um levantamento de algumas destas métricas, de modo que se possa perceber que métricas se podem utilizar para avaliar o projeto piloto que no qual se aplica a solução desta dissertação.

2.2.3.1 Lines Of Code (LOC)

A LOC conta o número de linhas de código existentes para uma determinada classe (M. Badri & Toure, 2012; Bruntink & van Deursen, 2004).

2.2.3.2 Source Lines of Code (SLOC)

A SLOC é derivada da LOC e conta o número de linhas de código fonte (M. Badri et al., 2016).

2.2.3.3 Test Lines of Code (TLOC)

A TLOC é uma métrica frequentemente utilizada para aferir a testabilidade e é sustentada pelo levantamento do número de linhas inerente à construção de testes (Sharma & Saha, 2018).

2.2.3.4 TNbLOC

Na aplicação da TNbLOC é aferido o número de linhas de código de uma classe de teste, de modo que este possa ser comparado com o número de linhas de código da respetiva classe a ser testada (M. Badri et al., 2016; Kim et al., 2010; Sharma & Saha, 2018). Esta métrica permite conjeturar o esforço associado à testagem de um determinado software, sendo possível fazer uma previsão do custo necessário para testar um determinado componente, tendo como base a informação previamente recolhida.

2.2.3.5 TNbOfAssert

Na aplicação da TnbOfAssert é analisado o número de instâncias de verificação da integridade e estado numa classe de testagem (L. Badri et al., 2011; Sharma & Saha, 2018). Estas verificações são tipicamente realizadas através de um método "*assert*" e permitem comparar o comportamento esperado de uma classe com o comportamento efetivamente verificado. Esta métrica permite perspetivar o esforço necessário para validar uma determinada classe.

2.2.3.6 TWMPC

A TWMPC é a soma da complexidade dos métodos de uma classe de teste, onde cada método é pesado com a sua complexidade ciclomática (L. Badri et al., 2011; Sharma & Saha, 2018).

2.2.3.7 Depth of Inheritance Tree (DIT)

Esta métrica é aplicável a situações de programação orientada a objetos, medindo o comprimento entre um determinado nó e o "nó raiz", numa hierarquia de classes, onde existe Herança (M. Badri & Toure, 2012; Bruntink & van Deursen, 2004; Ingeno, 2018; Rabee Shaheen et al., 2010) .

2.2.3.8 Number of Children (NOC)

Esta métrica calcula o número de descendentes de uma classe, pelo que quanto maior for o NOC de uma classe, maior será o número de filhos e responsabilidades associadas (M. Badri & Toure, 2012; Singhani & Suri, 2015).

2.2.3.9 Weighted Method Per Class (WMC)

A WMC permite calcular a complexidade de uma classe, recorrendo à soma da complexidade ciclomática dos métodos que a constituem, sendo um indicador do esforço inerente à manutenção de uma classe (M. Badri & Toure, 2012; Singhani & Suri, 2015). A complexidade ciclomática é uma medida que considera os caminhos independentes que o algoritmo pode seguir, permitindo a previsão de dificuldades associadas a processos como os de manutenção e testagem do software (Ebert & Cain, 2016).

2.2.3.10 Average Method Complexity (AMC)

Métrica que permite calcular a complexidade de uma classe, realizando uma média da complexidade ciclomática - previamente referida em 2.2.3.9 - dos métodos que a constituem, tendo um comportamento similar à WMC, mas fornecendo uma informação independente da envergadura de uma classe (Terragni et al., 2020).

2.2.3.11 Coupling between Objects (CBO)

A CBO permite dar conta do acoplamento de uma determinada classe (M. Badri & Toure, 2012; Bruntink & van Deursen, 2006a; Singhani & Suri, 2015). Um alto valor de CBO indicará um alto acoplamento, sendo, portanto, um indício para uma baixa testabilidade, como já referido anteriormente (ver 2.2.2.9).

2.2.3.12 Lack of Cohesion Metrics (LCOM)

Permite comparar o número de pares de métodos/funções sem similaridade com o número de pares de métodos/funções com semelhança, realizando uma subtração entre o primeiro e o segundo dos valores enunciados (M. Badri & Toure, 2012; Rabee Shaheen et al., 2010; Singhani & Suri, 2015). Tipicamente, um valor de LCOM elevado aponta para uma baixa coesão.

2.2.3.13 Fan Out (FOUT)

O FOUT pode ser apontado como o número de dependências de saída de um componente/classe para outros componentes/classes (Balogun et al., 2020; Bruntink & van Deursen, 2004, 2006a; Rabee Shaheen et al., 2010; Tarlinder, 2017). Pode ser vista como uma versão simplificada do CBO, tendo em conta que apenas se averigua o número de componentes/classes usadas por uma componente/classe e não quais componentes/classes a usam.

2.2.3.14 Response For a Class (RFC)

Averigua o número de métodos implementados numa classe, sendo apontado como uma métrica que analisa a complexidade, onde um alto RFC normalmente resulta de um desenho complexo (M. Badri & Toure, 2012; Bruntink & van Deursen, 2004, 2006a; Filho et al., 2020; Garousi et al., 2019; Mouchawrab et al., 2005; Singhani & Suri, 2015).

2.2.3.15 Rate of Component Observability (RCO)

Representa a percentagem de atributos legíveis entre todos os campos de um determinado componente, havendo um intervalo de valores definido para caracterizar se o perpetuado é apropriado ou não (Rabee Shaheen et al., 2010). Como o nome indica, esta medida está altamente relacionada com a Observabilidade.

2.2.3.16 Estimativas de Testabilidade

Segundo o apurado, existem várias equações para estimar a testabilidade, recorrendo, por exemplo, a uma combinação ponderada de outras métricas (Singhani & Suri, 2015).

2.2.4 Ferramentas para Análise de Código

Nesta secção serão referidas algumas ferramentas de análise estática, que têm a capacidade de analisar código em diversos aspetos, como atributos de qualidade, erros, entre outros. Este tipo de tecnologias pode ser eventualmente útil para auxiliar no aumento da testabilidade, tendo em conta que poderão transmitir *feedback* aos desenvolvedores, com maior enfâse no momento de codificação. O estudo deste tipo de ferramentas é importante, tendo em consideração que grande parte dos desenvolvedores questionados demostrou sentir a sua "necessidade", assim como será explicado em 2.3.6.

Para este estudo, procuraram-se por ferramentas de análise estática de código, tendo-se ainda procurado as ferramentas mais sugeridas para análise de código *JavaScript*, sendo esta a linguagem de programação utilizada em *NodeJs* pela empresa. Do estudo efetuado, resultou o seguinte conjunto de ferramentas: *SonarQube*, *ESLint*, *Codacy*, *ReSharper* e *Klockwork* (Crivello, n.d.; Pradeesh, 2021).

Note-se que foram encontradas outras ferramentas, para além das apresentadas seguidamente. No entanto, no estudo efetuado, não se considerou que estas apresentassem aspetos diferenciadores que justificassem a sua inclusão.

2.2.4.1 SonarQube

O *SonarQube* é uma ferramenta - de análise de código - capacitada de várias funcionalidades, como por exemplo análise de erros e vulnerabilidades (SonarQube, 2022). Pode ser configurada para assegurar o cumprimento de determinados aspetos que sejam requeridos e possibilita a integração com outras ferramentas, sendo, por vezes, parte integrante do ciclo de vida e monitorização de vários softwares. A testabilidade pode ser analisada recorrendo a esta ferramenta, sendo que a mesma é apurada para averiguar a manutenibilidade, por exemplo.

2.2.4.2 ESLint

O ESLint permite uma análise estática do código, permitindo encontrar problemas enquanto o programador se encontra a construí-lo, tendo em conta que pode ser integrado na maioria dos editores de texto, podendo ainda ser executado como parte de uma pipeline de integração contínua (ESLint, 2022a). É configurável e preparado para a linguagem de *JavaScript*, sendo, portanto, uma solução para produtos desenvolvidos em *NodeJs* (ESLint, 2022b, 2022c). Tendo

em conta a sua configurabilidade e análise de código em momentos de desenvolvimento, pode ser considerada uma boa opção para alertar os desenvolvedores de situações que afetem negativamente a testabilidade.

2.2.4.3 Codacy

O *Codacy* é uma das ferramentas mais populares no mercado (Crivello, n.d.; Pradeesh, 2021). Permite a realização de análises estáticas, sendo altamente escalável e integrável com outras tecnologias (Codacy, 2022).

2.2.4.4 ReSharper

O *ReSharper* é apontado pela *JetBrains* como a extensão indicada para análise de código no IDE *Visual Studio*, para desenvolvedores *dotNet* (JetBrains, 2022b). Não obstante, apresenta suporte para *JavaScript* e aplica mais de 2200 verificações ao código, apresentando sugestões em momento de codificação (JetBrains, 2022a). Permite ainda geração de código e sugere alterações ao mesmo.

2.2.4.5 Klocwork

O *Klocwork* permite analisar código para várias linguagens de programação, onde o *JavaScript* está incluído (Perforce, 2022). Tem extensões para vários IDE e analisa, por defeito, diversos aspetos do código – como segurança e qualidade -, sendo ainda capacitado de configuração adicional. É apontado como direcionado aos desenvolvedores, apresentando uma série de características que apoiam os mesmos – como *feedback* detalhado, possibilidade de configuração e análise arquitetural.

2.2.4.6 Comparação de Tecnologias

Aqui serão comparadas as cinco tecnologias previamente referidas, analisando uma série de fatores considerados pela equipa de desenvolvimento como importantes. Estes fatores foram recolhidos através de uma entrevista com os membros da equipa de desenvolvimento. Na Tabela 2 é feita uma comparação entre as diferentes tecnologias, onde o sinal "✓" significa que a ferramenta cumpre com o fator dessa linha, o sinal "X" indica que a ferramenta não cumpre com o fator dessa linha e o sinal "?" apresenta ferramentas onde não se encontrou informação acerca do fator em análise.

Tabela 2 – Comparação entre Ferramentas de Análise de Qualidade de Código.

Fator	SonarQube	ESLint	Codacy	ReSharper	Klocwork
Permite Analisar o Código	✓	✓	✓	✓	✓
Tem Regras de Análise Pré-Definidas	✓	✓	✓	✓	✓
Permite Adicionar Regras	✓	✓	✓	✗	✓
Apresenta sugestões em momento de codificação	✗	✓	✓	✓	✓

Permite explicar sugestões	✓	✓	✓	✓	✓
É independente de IDE	✓	✓	✓	✗	✓
Permite Gerar Relatórios de Análise de Código	✓	✓	✓	✗	✓
Permite análise de código JavaScript	✓	✓	✓	✓	✓
Facilmente modificável para cada versão de um projeto	✗	✓	✗	✗	✗

Como é possível constatar, dentro das várias ferramentas apresentadas, alguns fatores são sempre tidos em consideração, como por exemplo a existência de regras pré-definidas e a apresentação de sugestões. Existem, no entanto, outros fatores que acabam por ser mais restritos, como a dependência de um determinado IDE ou a capacidade da ferramenta ter um comportamento diferente para uma determinada versão de um projeto. Este último aspeto pode ser importante para estabelecer diferentes critérios para diferentes versões de um software, sendo isso de fácil concretização com a edição de um ficheiro de configuração ESLint que esteja num repositório de gestão de versões, em conjunto com o próprio repositório, por exemplo.

2.2.5 Conclusões sobre testabilidade

A testabilidade é afetada por um conjunto relativamente extenso de aspetos, muitos deles não imediatamente mensuráveis, como por exemplo a observabilidade e outros muito dificilmente mensuráveis, como é o caso do estado físico, mental e cognitivo de cada desenvolvedor. Alguns destes aspetos são referidos mais vezes (nos estudos analisados) do que outros, havendo relações diretas e indiretas entre vários. Para a elaboração da solução, privilegiar-se-á a análise dos fatores diretamente relacionados com o apurado em 2.3, mas tentar-se-á ter em consideração os restantes, de modo a perseguir uma solução mais completa. Adicionalmente, poder-se-á ter ainda em consideração ferramentas que potenciem estes mesmos fatores.

As métricas apresentadas permitem averiguar a testabilidade, analisando, para tal, vários aspetos. Conforme visto anteriormente, existem vários influenciadores de testabilidade, sendo, portanto, normal que possam existir várias métricas, especialmente porque alguns dos fatores podem ser analisados individualmente, recorrendo às suas métricas associadas. Posto isto, algumas das métricas referidas, e eventualmente outras não referidas, podem ser combinadas, de modo a estimar a testabilidade, recorrendo para tal a algumas equações já estabelecidas. Das métricas apresentadas, selecionaram-se algumas que permitem analisar o conjunto de fatores mais vezes referenciados pela equipa de desenvolvimento (ver 2.3.6). Assim, utilizar-se-

á na avaliação da solução: a SLOC e a TLOC, para comparar o número de linhas de código afeto à testagem com o número de linhas do código fonte respetivo, de modo a validar a quantidade de código necessária para testar um determinado software, quando comparado com o código fonte do mesmo; e a WMC, de modo a avaliar a complexidade média das classes do projeto piloto, tentando perceber qual é efetivamente a complexidade de excertos do software.

No que diz respeito a tecnologias de avaliação/análise de código, segundo o estudo realizado, poder-se-á dizer que o mercado apresenta várias soluções. Dentro destas soluções, existem ferramentas direcionadas a contextos específicos e ferramentas com um maior grau de alcance. Do que foi apurado, a testabilidade não é tida como fator fundamental na análise de código, sendo, por vezes, utilizada como fonte de avaliação de outras medidas, como, por exemplo, a manutenibilidade. É de realçar a existência de ferramentas que permitem a configuração de regras de análise, que poderão ser anexadas às já existentes, o que poderá ser de alta utilidade no contexto desta dissertação – podem ser construídas regras direcionadas à análise de testabilidade. De todas as ferramentas analisadas, a ESLint é a que cumpre com um maior número dos requisitos considerados.

2.3 Análise da Situação Interna

No que respeita ao contexto da Sysnovare, torna-se importante: introduzir a tecnologia sobre a qual se pretende aumentar a testabilidade; aferir as práticas/cuidados existentes no que concerne ao desenvolvimento de software; explicar o processo de desenvolvimento, testagem e entrega do software; analisar os tipos de testes utilizados e a cobertura associada; e aferir, por fim, a opinião da equipa de desenvolvimento. As informações apresentadas nesta secção dizem respeito ao verificado na empresa no momento correspondente ao início do projeto.

2.3.1 Tecnologia

O *NodeJs* é um *runtime Javascript* desenvolvido com base no "*Google Chrome V8*" e no seu *ECMAScript* (linguagem de programação baseada em *scripts*), pelo que a sintaxe usada é muito semelhante ao *JavaScript front-end* (tendo em conta que este baseia-se noutra implementação de *ECMAScript*) (Hunter, 2020; Mardan, 2018). Este software visa potenciar a construção de aplicações altamente escaláveis. Tem como seu objetivo facilitar uma construção iterativa e incremental, onde existe oportunidade de reutilização de código, utilizando por exemplo as bibliotecas da tecnologia, que podem ser publicadas/acedidas no repositório da *NPM*, ou em repositórios particulares.

Esta tecnologia é utilizada pela Sysnovare para o desenvolvimento do lado do servidor, sendo a escolha privilegiada para o tratamento da lógica de negócio.

2.3.2 Práticas de Desenvolvimento

De modo a entender o panorama verificado na empresa, torna-se relevante fazer um levantamento das estratégias delineadas e seguidas. Relativamente ao software desenvolvido em *NodeJs*, existe especificação de algumas práticas que devem ser seguidas pela equipa de desenvolvimento.

No que concerne à arquitetura, não existia, inicialmente, qualquer padronização. No entanto, desde 2020, foi adotada pela empresa uma arquitetura que privilegia o domínio de cada problema, passando a ter em consideração conceitos como Agregados e os seus constituintes – como Entidades e Objetos de Valor -, garantindo, assim, que os produtos refletem as necessidades de cada problema, indo ao encontro da especificidade de cada negócio. Adicionalmente, cada agregado é constituído por várias camadas – controladores, serviços, modelos, mapeadores, repositórios, entre outros – para potenciar a segregação responsabilidades (Evans & Fowler, 2014). Na Figura 8 é ilustrada a arquitetura, de forma resumida.

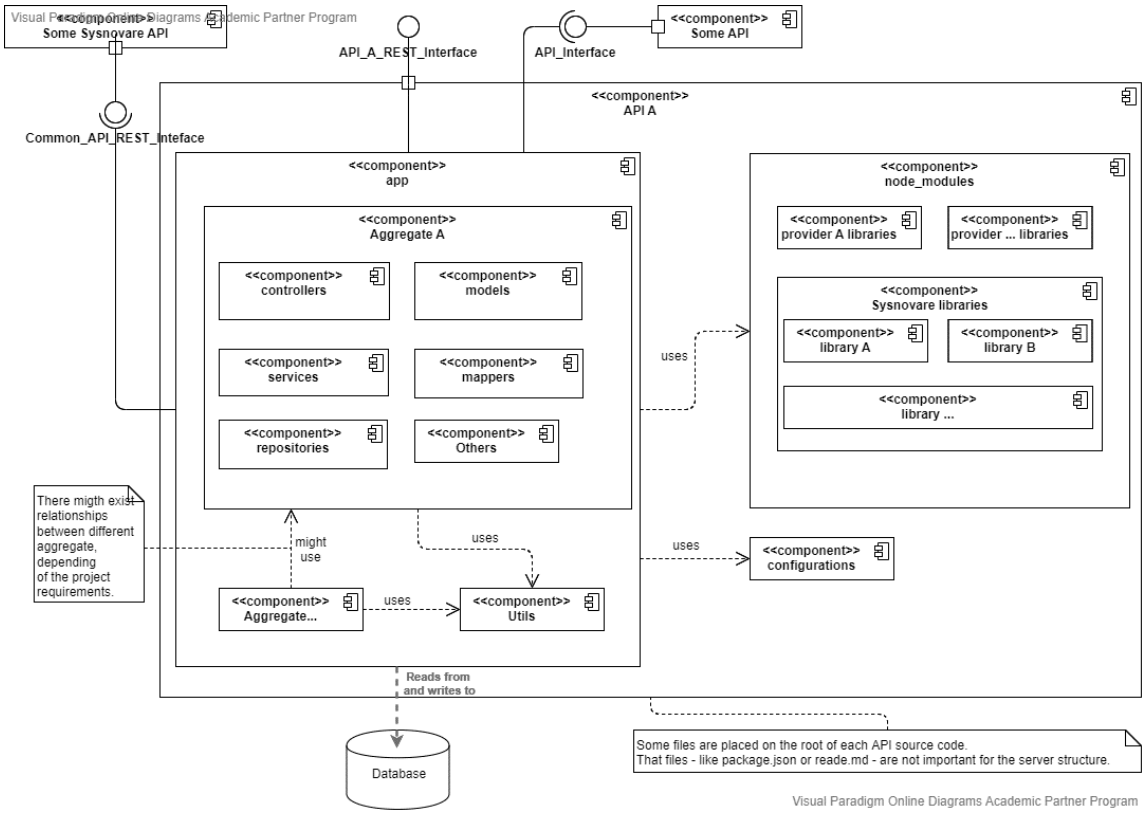

Figura 8 – Diagrama de Componentes, seguindo o nível 3 do modelo C4 (Vázquez-Ingelmo et al., 2020).

De modo a potenciar a extensibilidade do software, a Sysnovare promove não só a construção de bibliotecas que possam ser incorporadas em diversas aplicações, como também aplicações que possam ser integradas, através de pedidos REST, por exemplo, como se pode ver na figura anterior.

A arquitetura adotada tenta aproximar-se do negócio, tendo-se em consideração aspetos como *Bounded Contexts*, na medida em que diferentes contextos do mesmo problema se encontram separados, quer em diferentes Agregados, quer em diferentes aplicações, caso isso seja justificativo (Evans & Fowler, 2014). Este tipo de separação é ponderado com critério e tem sempre em consideração a especificidade de cada problema.

É ainda de referir que não existe um paradigma de programação que seja adotado exclusivamente, sendo de enunciar a aplicação de alguns em particular, nomeadamente: Programação Imperativa, Programação Funcional; e Programação Orientada a Objetos (Maurizio Gabbrielli & Simone Martini, 2010). A intercalação entre estes paradigmas é pouco coerente, sendo que são utilizados para a resolução do mesmo tipo de problemas em diferentes contextos, podendo haver aqui um problema de definição e padronização de qual paradigma deve ser adotado em cada situação.

Deve-se salientar ainda a adoção de alguns padrões de software, como exemplo o *Front Controller Pattern*, o *Repository Pattern*, o *Mapper Pattern* e o *Data Trasnfer Object*, embora, segundo o apurado, não exista uma grande consciencialização por parte da maior parte dos desenvolvedores acerca destes (e outros) padrões de software (Buchmann et al., 2007; Fowler, 2011).

2.3.3 Processo de Desenvolvimento, Testagem e Entrega

Os produtos desenvolvidos pela Sysnovare são construídos e entregues através do seguimento de uma série de passos. Cada versão de um produto inicia com análise e especificação de requisitos, seguida da sua implementação, testagem e entrega. Todo o software é evoluído recorrendo à tecnologia de controlo de versões - Git. É ainda utilizado um software de automatização de processos - Jenkins - para um grupo pequeno de projetos, sendo que, num panorama final, todos os produtos da Sysnovare acabam por ser entregues sem recurso a este software.

Tendo em conta o tema da dissertação, de todos os processos existentes é importante analisar com um maior critério o processo de testagem, que envolve **testes manuais** e **testes automáticos**.

O software desenvolvido pela empresa é tipicamente testado em três fases: na primeira fase, cabe à equipa desenvolvimento testar o software, utilizando, para isso, tanto testes automáticos, como testes manuais; na segunda fase, cabe à equipa de consultoria testar o funcionamento do sistema através de testes manuais; na última fase, é testado pelos clientes, num ambiente próximo do produtivo – designado como ambiente de qualidade -, o funcionamento do sistema, através de testes manuais. Note-se que esta fase não se verifica em todos os clientes.

É de realçar que, para os testes automáticos, é tipicamente criado um esquema (*schema*) de base de dados, para cada componente a ser testado individualmente (apenas aplicável em

componentes que estabeleçam comunicação com base de dados). Este esquema de base de dados é preparado com a estrutura necessária para suportar os testes a serem realizados. De modo a garantir a integridade dos testes, cada execução dos testes automáticos inicia com uma limpeza dos potenciais dados remanescentes de uma prévia execução de testes, seguido por um conjunto de operações de plantação de dados. Isto permite que cada execução de testes execute sobre um ambiente controlado, onde é desde início assegurada a integridade dos dados, facilitando assim a execução repetitiva dos mesmos.

Todas as fases previamente referidas realizam-se depois do código em análise ser desenvolvido, pelo que se trata de uma abordagem *Test Last Development* (Janzen & Saiedian, 2006). Na Figura 9, pode-se ter uma ideia do processo de desenvolvimento, testagem e entrega do software da Sysnovare, sendo que, para isso, se incluíram todas as fases previamente referidas, de modo a potenciar uma melhor ideia da integração dos processos de testagem naquilo que é o fluxo de desenvolvimento seguido pela empresa.

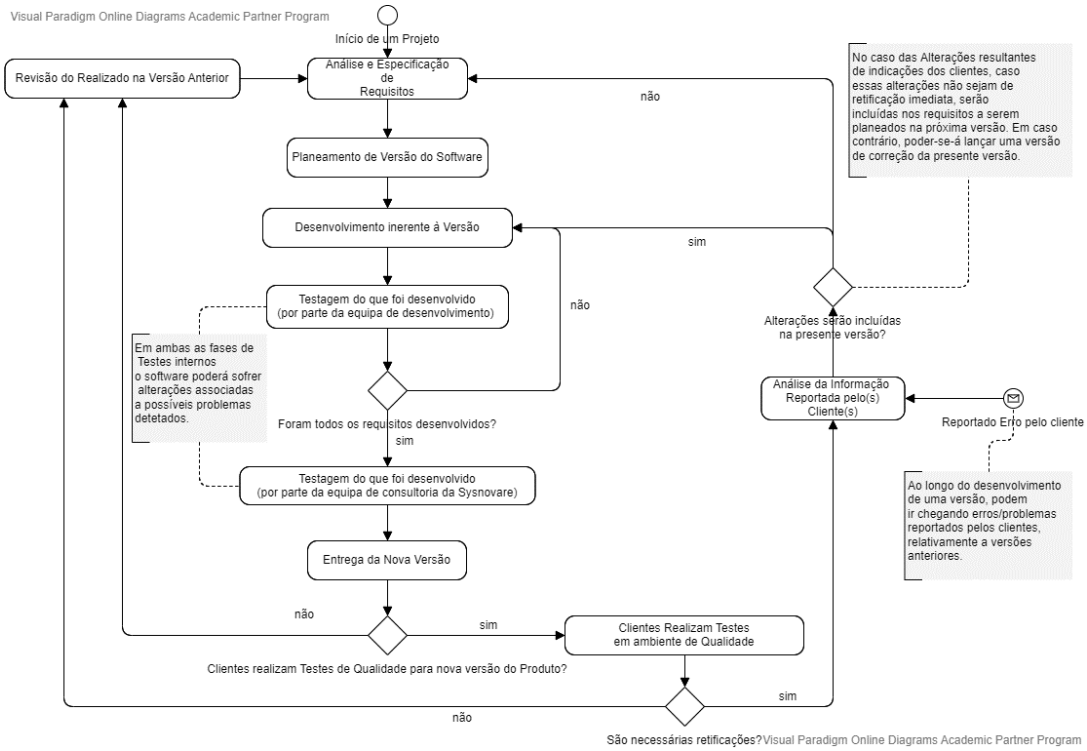

Figura 9 – Diagrama BPMN ilustrativo do processo de planeamento, desenvolvimento, testagem e entrega do *software* da Sysnovare.

Como é possível verificar, cada versão de um produto inicia com a análise e especificação de requisitos, seguida da implementação e consequente testagem dos mesmos. Conforme referido anteriormente, os testes são executados após o desenvolvimento. É ainda de realçar que, em cada versão, podem ser incluídas correções de erros (de versões prévias), reportados pelos clientes, acerca de software que já se encontra em produção – muitas vezes estes erros impossibilitam a finalização do que foi inicialmente planeado para uma dada versão, pelo que,

caso não sejam problemas com gravidade aferida como elevada, serão potencialmente incorporados numa versão seguinte.

Deve-se realçar que, para uma versão ser entregue a um cliente, não é obrigatório que os níveis de cobertura cumpram os objetivos idealizados, sendo apenas recomendado. Este tipo de controlo poderia ser realizado, de modo a garantir que o software entregue aos clientes seguia os requisitos de qualidade colocados pela Sysnovare.

Relativamente às práticas de desenvolvimento e testagem do software, é de evidenciar que a estratégia de depuração seguida se baseia na escrita de informações para ficheiros de registo ("l*og*"), não havendo, no momento inicial da dissertação, uma ferramenta de depuração que possa ser utilizada quer durante o desenvolvimento quer durante a realização de testes manuais.

2.3.4 Testes Automáticos

Tendo em conta as vantagens dos testes automáticos face aos testes manuais (apontadas em 2.1), a empresa decidiu que os testes automáticos deveriam ser privilegiados face aos testes manuais, aquando da realização de testes por parte da equipa de desenvolvimento. Desta forma, de seguida, analisar-se-ão os tipos de testes automáticos presentemente realizados na Sysnovare, no que concerne ao software desenvolvido em *NodeJs*, sendo este o alvo do presente estudo.

São realizados dois tipos de testes automáticos: testes unitários e testes de integração. Segundo o chefe da equipa de desenvolvimento, os desenvolvedores devem privilegiar os testes de integração – em detrimento dos testes unitários -, sendo, assim, este tipo de testes os mais frequentemente utilizados para validar casos de teste. Esta indicação deriva da perceção do chefe da equipa de desenvolvimento, que considera a realização dos testes de integração suficiente na maioria das situações, devido à sua capacidade de "testar o software como um todo", contemplando a integração de vários componentes do sistema. Não obstante, e em situações onde os testes de integração não permitam cobrir com rigor determinadas partes de um sistema, realizam-se, a título excecional e complementar, testes unitários.

Tanto os testes de integração como os testes unitários são realizados recorrendo a uma biblioteca, de seu nome "*Lab*", disponibilizada pela *Framework Hapi* (Nguyen et al., 2015). Através desta biblioteca é possível perceber, entre outros aspetos, qual a cobertura atual do código em análise e qual o código que não se encontra devidamente testado.

2.3.5 Níveis de Cobertura no Início da Dissertação

Tendo em conta que o projeto visa aumentar a cobertura do software desenvolvido do lado do servidor - em *NodeJs* -, torna-se relevante, no contexto do problema em análise, escrutinar quais os níveis de cobertura, do software desenvolvido pela empresa, no momento de início da dissertação. Desta forma, recolheram-se, num momento incipiente, os níveis de cobertura nos

diferentes componentes, desenvolvidos em *NodeJS*, pela Sysnovare, sendo que estes componentes englobam tanto servidores, como bibliotecas. Esta informação, que permite ter uma ideia mais concreta da situação no momento de arranque da dissertação, encontra-se representada na Tabela 3, onde é apresentada a cobertura para cada um dos vinte e seis componentes analisados e a data na qual o levantamento de cobertura foi efetuado.

Tabela 3 – Níveis de cobertura de código através de testes automáticos no início da dissertação.

Nome (único) do Componente	Percentagem de Cobertura	Data de Aferição de Cobertura
fiscalizacao-api	76.59%	2022-01-15
fiscalizacao-xlsx-export	0%	2022-01-15
gic-api	30.28%	2022-01-15
gic-autos-ws	92.18%	2022-01-15
noladge-api	87.39%	2022-01-15
noladge-xlsx-export	70.19%	2022-01-15
rhsuite-chatbot	71.19%	2022-01-15
rhsuite-external-services	74.71%	2022-01-15
rhsuite_api	71.62%	2022-01-15
rhsuite_xlsx_export	97.73%	2022-01-15
sci-api	77.36%	2022-01-15
sys-hapi-auth	92.95%	2022-01-15
sys-hapi-autopay	71.18%	2022-01-15
sys-hapi-errors	84.60%	2022-01-15
sys-hapi-formularios	80.02%	2022-01-15
sys-hapi-gessi	83.03%	2022-01-15
sys-hapi-guias	87.50%	2022-01-15
sys-hapi-notifs	88.76%	2022-01-15
sys-hapi-rgpd	92.23%	2022-01-15
sys-hapi-utils	98.92%	2022-01-15
sys-node-db	82.23%	2022-01-15
sys-node-mail	90.87%	2022-01-15
sys-node-utils	71.43%	2022-01-15
sys-scheduler	87.69%	2022-01-15
sys-xlsx-export	0%	2022-01-15
sys_docx_templates	89.53%	2022-01-15
Média Total	**75.01%**	2022-04-04

Foi apontado na identificação do problema, que a deteção do mesmo tinha surgido no aparecimento de erros afetos a partes do código sem cobertura, considerando-se que os níveis de cobertura seriam baixos, tendo em conta o ambicionado (ver 1.3). Com efeito, tendo em consideração os dados aqui levantados, confirma-se que as percentagens de cobertura são efetivamente insuficientes, para aquilo que é o objetivo da empresa (os níveis de cobertura diagnosticados são em regra - fora duas exceções – inferiores aos 95% delineados nos objetivos de percentagem de cobertura – ver Anexo A). Deve-se ainda salientar que a cobertura média

ronda os 75%, o que é também inferior ao apontado como objetivo, em concordância com o anteriormente aferido.

De modo a entender melhor a baixa cobertura nalgumas das aplicações, analisou-se ainda o código não coberto. Desta análise, resultaram alguns aspetos a destacar: o código que comunica com serviços externos nunca era testado; existia um défice na cobertura dos dados armazenados em cache e nas situações onde é aplicada Herança.

2.3.6 Opinião da Equipa de Desenvolvimento

Como já foi referido anteriormente, a equipa de desenvolvimento tem um papel ativo na testagem do software, participando tanto na realização de testes manuais como de testes automáticos. Assim, de modo a entender melhor as suas opiniões e dificuldades, inquiriu-se a equipa de desenvolvimento, tendo-se elaborado um questionário para esse efeito.

Este questionário é constituído essencialmente por perguntas de resposta aberta, tendo em conta que o objetivo aqui não é limitar a opinião dos questionados a um conjunto de alternativas fixo, mas sim, possibilitar o surgimento de opiniões diversificadas, que possam contribuir para a construção de uma solução, que vá ao encontro das necessidades da equipa, acabando efetivamente por ajudar a mesma a desenvolver software mais testável. Assim, pretende-se aferir quais são as maiores dificuldades sentidas aquando da testagem do software e quais são os aspetos que, no entendimento dos desenvolvedores, podem beneficiar a testabilidade (ver Anexo B).

Das opiniões recolhidas (9 desenvolvedores responderam ao questionário), são apresentadas, na Tabela 4, as consideradas mais relevantes (obviamente que não existem respostas exatamente iguais, mas tentou-se fundir respostas análogas). Para cada resposta é apresentado o número de vezes que essa resposta (ou resposta análoga) foi referida.

Tabela 4 – Respostas mais relevantes para cada pergunta do questionário de resposta aberta.

Pergunta	Resposta	Nº Vezes
Que dificuldades sente/sentiu ao testar o código por si desenvolvido?	"Sinto dificuldades em nomear os testes, de acordo com a funcionalidade que testo e as verificações efetuadas".	4
	"Por vezes tenho dificuldade em perceber que testes devo realizar para testar uma funcionalidade".	4
Que dificuldades sente/sentiu ao testar o código desenvolvido por outrem?	"Tenho dificuldade em perceber que testes devo efetuar… Por vezes tenho que ir ler o código para perceber quais são os fluxos possíveis para uma funcionalidade, o que me cobra muito mais tempo".	2
	"Tenho dificuldade em testar código com alta complexidade, porque torna-se mais difícil percebê-lo."	7
	"Não consigo perceber rapidamente o código desenvolvido, porque vários programadores são	7

	bastante distintos na sua forma de programar, sendo que por vezes não existem comentários elucidativos do que esta a ser realizado".	
Em termos gerais, que dificuldades sente/sentiu ao testar software, nomeadamente através de Testes Automáticos?	"A maior dificuldade que sinto é na quantidade de código necessária para a preparação da base de dados para realizar testes".	7
	"Conseguir testar todas os fluxos possíveis para uma determinada funcionalidade".	2
	"Testar Casos de Uso que necessitem de integrações com outros componentes/sistemas externos."	5
	"Sinto falta de documentação que especifique com maior critério e rigor as várias funcionalidades e o próprio software. "	5
Considere a Testabilidade como o grau de facilidade com que testa o software, sendo que uma maior Testabilidade significa que o software é mais facilmente testável. Na sua opinião, o que poderá potenciar uma maior Testabilidade no software desenvolvido na Sysnovare?	"Deveria haver um plano de testes bem definido para cada Caso de Uso."	3
	"Todos os testes deveriam seguir nomenclaturas que identifiquem corretamente e inequivocamente o que está a ser testado…Isso facilitaria o trabalho futuro, em momentos de manutenção de código, onde algumas funcionalidades modifiquem o seu comportamento, obrigando ao reajuste dos testes".	4
	"Deviam existir mais comentários no código que explicassem, de forma breve e elucidativa, o que é feito em cada método ou bloco de código".	7
	"Deveria existir alguma ferramenta que desse algum tipo de feedback, especialmente em momento de desenvolvimento, de eventuais excertos de código que possam ser melhorados, tendo em conta um conjunto de regras."	8
	"Penso que deveria existir documentação do software, nomeadamente dos vários casos de uso, como das integrações correntes para cada aplicação. Penso que isto aumentaria a testabilidade do código, porque seria possível ter uma perceção visual da estrutura do software."	5
	"Deveria ser mais rápido preparar o ambiente para a testagem."	7

Como é possível ver na tabela acima apresentada, existem vários aspetos que são apontados pelos desenvolvedores, desde aspetos relacionados com nomenclaturas e comentários, a fatores inerentes a dependências externas no momento da testagem e necessidades de configuração. Foi ainda dado alguma ênfase à insuficiente documentação do software.

É de salientar que a dificuldade inerente à preparação do ambiente de testagem, os problemas relacionados com a diversidade de estilos de programação seguidos por diferentes desenvolvedores e a complexidade inerente a algumas componentes/métodos são as dificuldades mais reportadas pelos desenvolvedores neste questionário.

Por fim, deve-se ter em conta que grande parte dos desenvolvedores sugere a adoção de ferramentas que os auxiliem, em momento de codificação, a não desenvolver código de má qualidade.

A uniformidade de opiniões/sugestões apresentadas pelos desenvolvedores pode ser explicada por alguns debates que foram realizados sobre o tema, antes do preenchimento do questionário.

Toda esta informação será tida em consideração para a construção do guia.

2.3.7 Conclusões sobre Análise Interna

Do estudo que se realizou sobre o ambiente interno da empresa, nomeadamente no que diz respeito a práticas, percebeu-se que existe algum cuidado em desenvolver software escalável, de simples integração e continuidade, sendo isso visível, por exemplo, na preocupação da segregação do software por áreas de negócio e responsabilidades, ou até na adoção de alguns padrões de desenvolvimento de software.

Relativamente aos processos de desenvolvimento, testagem e entrega do software, nota-se uma relativa dependência de atividades manuais, que requerem uma atenção e colaboração direta de recursos humanos, nomeadamente na realização de testes manuais e na entrega dos produtos aos clientes.

Relativamente aos níveis de cobertura aferidos por testes automáticos, verificou-se aquilo que se alegava, ou seja, que estes estão abaixo do pretendido, sendo que a média de cobertura ronda os 75%, quando deveria passar para um mínimo de 95%.

Por fim, pode-se ainda dizer que a equipa de desenvolvimento foi relativamente concordante na sua opinião, havendo um conjunto de temáticas que foram referidas por diferentes desenvolvedores, sendo, portanto, essas matérias importantes para a construção de uma solução que vá ao encontro das necessidades da equipa.

2.4 Conclusões

Nos estudos realizados sobre testabilidade, são vários os fatores apresentados como influenciadores, tendo-se verificado uma maior recorrência de alguns deles, como a controlabilidade e observabilidade. Ainda dos estudos realizados resultam métricas que permitem apurar a testabilidade, estando por vezes estas métricas diretamente associadas a alguns dos fatores que a influenciam. Tendo em conta os fatores analisados e as métricas recolhidas, na construção da solução haverá um maior foco nos fatores associados aos pontos mais referenciados pela equipa de desenvolvimento no questionário, recorrendo-se, no término da construção e aplicação da solução, a métricas associadas a este conjunto, para validar parte da solução desenvolvida.

Foram encontradas várias tecnologias, no que diz respeito à análise de código. Para as tecnologias encontradas foram colocadas um conjunto de questões consideradas importantes para a equipa de desenvolvimento, sendo a ESLint a ferramenta que cumpriu positivamente com um maior número de questões.

A Sysnovare apresenta um processo de desenvolvimento e entrega onde é denotada uma forte componente manual, havendo, no entanto, alguns processos automatizados através de testes automáticos e da utilização de uma ferramenta de desenvolvimento contínuo.

O software desenvolvido segue algumas regras pré-definidas, no entanto denotou-se alguma incoerência nos processos de tomada de decisão entre aquilo que são as diferentes opções existentes para o mesmo tipo de problema.

Quando questionada, a equipa de desenvolvimento sobre a temática da testabilidade, denotou-se alguma recorrência das opiniões apresentadas pelos seus membros. Assim, deu-se maior ênfase: à incoerência/inconsistência inerente às diferentes "formas de programação" de cada desenvolvedor; ao alto custo sentido para preparar um ambiente de testagem de software; à excessiva complexidade de excertos de código; e ao potencial contributo que ferramentas de análise de código poderão dar ao aumento da testabilidade.

Na Figura 10 são apresentadas as relações entre diversos aspetos referidos no estado na arte, tendo em vista o desenho e construção da solução (guia).

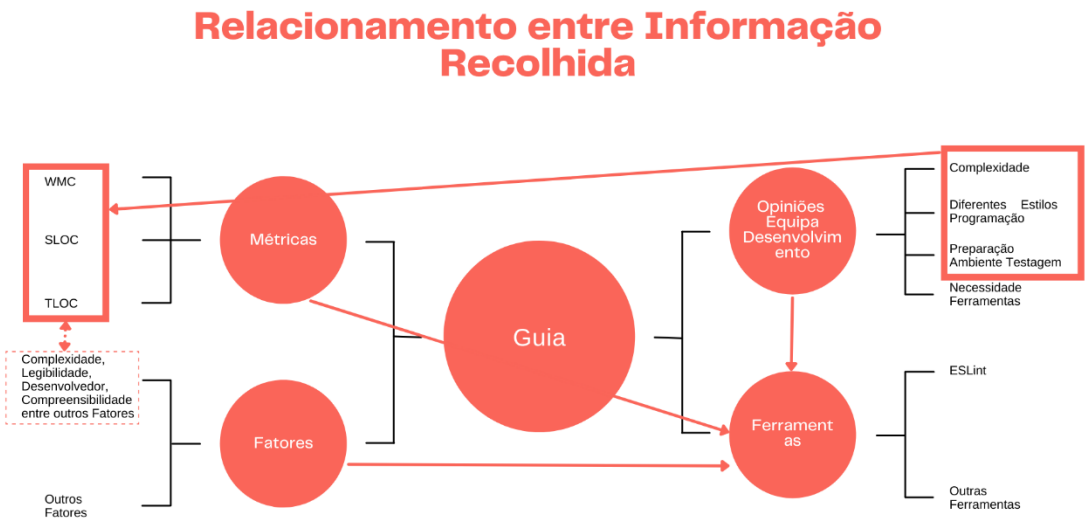

Figura 10 – Relação e ilações obtidas da informação recolhida no Estado da Arte.

No presente trabalho, pretende-se potenciar o aumento da testabilidade, tendo em conta as opiniões da equipa de desenvolvimento e os fatores associados à testabilidade, recorrendo a

ferramentas que possibilitem a sua monitorização contínua – ferramentas estas escolhidas tendo em consideração as necessidades apresentadas pela equipa de desenvolvimento. Para avaliar a solução, será utilizado um conjunto de métricas associadas a algumas das dificuldades apresentadas pelos desenvolvedores. As métricas e os fatores terão um papel preponderante na configuração das eventuais ferramentas a utilizar. De momento, equaciona-se utilizar a ESLint como ferramenta de monitorização e análise estática do código dos desenvolvedores, mas futuramente poderão ser incluídas outras ferramentas.

3 Análise de Valor

Nesta secção é apresentada a Análise de Valor da dissertação. Para tal, recorreu-se ao *New Concept Development Model*; realizou-se uma averiguação do valor para o cliente, valor percecionado e proposta de valor; analisou-se o modelo de negócio; e concretizou-se uma análise funcional.

3.1 New Concept Development Model

O *New Concept Development Model* divide-se em três partes, sendo elas o motor, os elementos-chave e os fatores ambientais, que influenciam os anteriores (P. A. Koen et al., 2014). O motor representa o cerne do modelo, centrando-se na visão, estratégia, cultura, recursos, equipas e colaboração (P. Koen et al., 2001). Os elementos-chave são cinco e são abaixo explicados e analisados, nas seguintes secções (Vacek, 2006). Como fatores ambientais são considerados os fatores externos – como a gestão de conhecimento, a competição, as tendências do mercado e as tecnologias - que influenciam as duas partes do modelo previamente referidas.

Na Figura 11 são esquematizados os elementos-chave e as suas relações.

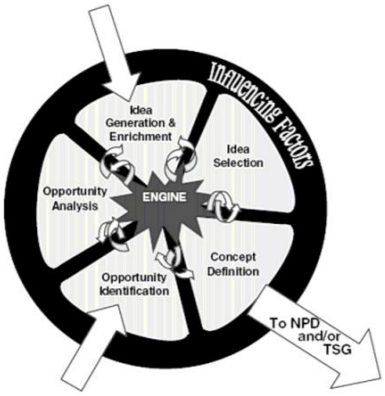

Figura 11 - New Concept Development Model (Vacek, 2006).

Note-se que entre os diferentes elementos-chave existem setas, que os correlacionam bidireccionalmente, indicando que não existe necessariamente uma forma linear de percorrer todos os pontos (Vacek, 2006). Existem ainda duas setas vindas do exterior, para o elemento de "Geração e Enriquecimento de Ideias" e para o elemento de "Identificação da Oportunidade", significando que estes são os pontos de partida possíveis para iniciar a análise. Por fim, é de realçar a seta que parte da "Definição de Conceito" para o exterior, sendo assim identificado o último passo do processo.

3.1.1 Identificação da Oportunidade

A Identificação da Oportunidade consiste na análise de oportunidades e ameaças inerentes ao negócio, sendo contribuintes para esta fase aspetos como tendências culturais, recursos, economia, entre outros (P. Koen et al., 2001; Vacek, 2006).

A oportunidade surge da necessidade de melhoria da qualidade do software entregue aos clientes da empresa, sendo este mais confiável com uma cobertura superior ao inicialmente verificado (ver Tabela 3). Este aspeto é de alta relevância, tendo em conta que a Sysnovare tenciona entregar aos seus clientes produtos de alta qualidade, proporcionando-lhes a melhor experiência possível, demostrando assim uma imagem de profissionalismo. A possibilidade de apresentar produtos mais confiáveis e com uma margem de erro reduzida amplia a imagem da empresa e aumenta a competitividade da mesma, no mercado em que se insere, potenciando a manutenção de clientes já adquiridos e a angariação de novos clientes, semeando assim uma trilha de sucesso ao longo do tempo.

Como explicado no estado da arte, os testes automáticos potenciam a deteção antecipada de erros. Sabendo que na Sysnovare a percentagem de cobertura de testes automáticos fica aquém do pretendido, e sabendo que uma maior testabilidade potenciará o atingimento de níveis superiores de cobertura. Assim, identificou-se a oportunidade de aumentar a testabilidade do software, de modo a potenciar o alcance de níveis superiores de cobertura sobre o mesmo. Ambiciona-se ainda tirar partido de ferramentas que já existam no mercado,

de modo a potenciar o cumprimento de vários fatores importantes para a qualidade, nos quais se insere a testabilidade, previamente referida.

3.1.2 Análise da Oportunidade

Na análise da oportunidade, devem ser estudados o mercado e a probabilidade de sucesso da(s) oportunidade(s) identificada(s), recorrendo ao planeamento e avaliação das vantagens, consoante o enquadramento da empresa e dos seus recursos (P. Koen et al., 2001; Vacek, 2006).

De acordo com o referido anteriormente (ver 1.1), a Sysnovare é uma empresa tecnológica que desenvolve e vende software para diversas áreas. Apesar de servir várias áreas, a equipa de desenvolvimento da Sysnovare tem pouco mais de meia dúzia de constituintes, por isso torna-se importante construir software altamente testável, facilitando, assim, o trabalho dos desenvolvedores, otimizando o tempo entregue à construção e validação de testes automáticos.

Presentemente, existem várias empresas que adotam técnicas e padrões que potenciam a testagem e uma maior cobertura do software, como por exemplo *Test Driven Development* (TDD), sendo frequentemente relatados resultados positivos com a sua adoção (Spirlandeli et al., 2019). O aumento da testabilidade poderá colocar a empresa numa primeira linha de disputa com empresas que sigam este tipo de ideias, notando-se ainda que este tipo de ideias será tido em consideração, no presente estudo, podendo eventualmente contribuir para o aumento da própria testabilidade.

3.1.3 Geração e Enriquecimento de Ideias

Tendo a oportunidade devidamente identificada e analisada, pode-se partir para a Geração e Enriquecimento de Ideias, transformando a(s) oportunidade(s) identificada(s) em ideias concretas (P. Koen et al., 2001; Vacek, 2006).

Em **primeiro** lugar surge a **ideia** de construir um documento, que sumarize uma série de práticas e regras a serem seguidas, aquando do desenvolvimento do software, de modo que este seja o mais testável possível. Este documento deveria estar ao alcance de cada desenvolvedor e ter em consideração vários aspetos recolhidos aquando de um estudo sobre testabilidade, auxiliando ainda na tomada de decisão, em algumas situações que possam suscitar dúvidas ao desenvolvedor. Apesar de carecer de uma componente de desenvolvimento associada, podem estar presentes neste documento não só a sugestão de ferramentas existentes para este efeito, como também as configurações personalizadas que potenciem a testabilidade.

Como **segunda ideia**, poder-se-ia ir um pouco além do descrito na primeira ideia, desenvolvendo uma aplicação personalizada que auxiliasse nos aspetos previamente referidos. Esta aplicação apresentaria então uma série de indicações ao desenvolvedor e seria capaz de sugerir soluções consoante um conjunto de parâmetros previamente definidos, auxiliando, de forma dinâmica, em vários processos de tomada de decisão.

Como **terceira ideia**, existe a possibilidade de construir uma aplicação que analise, de forma automática, um determinado software, indicando o seu grau de testabilidade e o que deveria/poderia ser alterado no mesmo, de modo a aumentar este grau.

3.1.4 Seleção da Ideia

Nesta secção serão selecionadas, do conjunto de ideias geradas, quais delas serão concretizadas (P. Koen et al., 2001; Vacek, 2006). O investimento na(s) ideia(s) selecionada(s) deve ser justificado, sendo importante garantir que essa(s) ideia(s) garante(m) retorno.

3.1.4.1 Análise hierárquica (AHP)

O *Analytic Hierarchy Process* (AHP) é um método para tomada de decisão criterioso, que permite a priorização de alternativas em situações com critérios conflituosos (Buchanan & Gardiner, 2003; Ulkhaq et al., 2018). Este método estrutura diferentes níveis hierárquicos, onde o primeiro nível diz respeito ao propósito do problema, o segundo contem os critérios ambicionados e o terceiro lista o conjunto de soluções possíveis. Tendo em conta o problema em mãos e as ideias previamente geradas, na Figura 12 encontra-se o respetivo diagrama AHP.

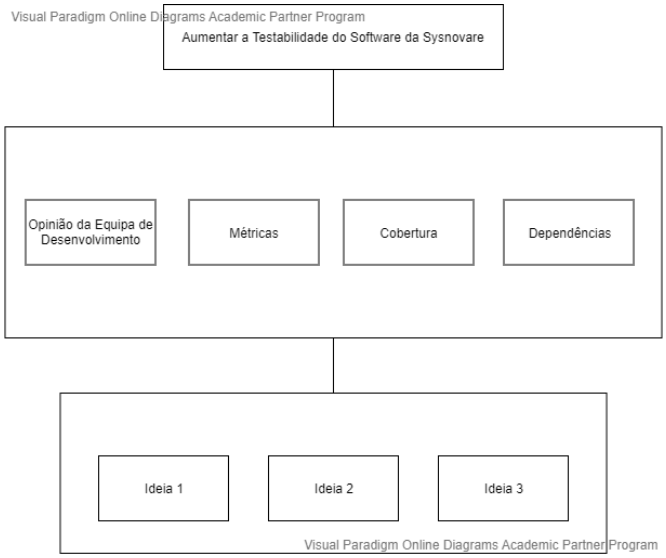

Figura 12 - Árvore Hierárquica de Decisão.

Como critérios utilizaram-se parte dos objetivos apontados para esta dissertação, incluindo-se ainda um critério relacionado com métricas que permitam aferir a testabilidade.

De modo a entender qual ideia será a mais adequada, torna-se necessário atribuir uma importância a cada critério. Para tal, considerou-se a seguinte escala (Saaty & Katz, 1990).

Tabela 5 – Escala de níveis de Importância (Saaty & Katz, 1990).

Nível de Importância	**Explicação**
1	Os dois critérios em comparação apresentam a mesma importância.
3	Um dos critérios tem ligeiramente mais importância que o outro.
5	Um dos critérios tem fortemente mais importância que o outro.
7	Um dos critérios tem muito mais importância que o outro.
9	Um dos critérios é de absoluta importância, quando comparado com o outro.
2,4,6,8	Valores intermédios entre os transatos níveis de importância.

Tendo em consideração a escala enunciada na Tabela 5, analisar-se-ão os critérios definidos, comparando a importância relativa de cada um e construindo assim uma tabela de avaliação AHP, analisável na Tabela 6.

Tabela 6 – Tabela de Avaliação AHP.

Critérios	Opinião da Equipa de Desenvolvimento	Cobertura	Métricas	Dependências
Opinião da Equipa de Desenvolvimento	1	1/2	7	3
Cobertura	2	1	9	5
Métricas	1/7	1/9	1	1/5
Dependências	1/3	1/5	5	1
Total	3.48	1.81	22	9.2

Por fim, poder-se-ão normalizar os dados obtidos, dividindo cada valor pelo total da sua coluna, obtendo ainda a prioridade relativa, de acordo com a média aritmética de cada linha, como se pode verificar na Tabela 7.

Tabela 7 – Tabela de Avaliação AHP normalizada com informação de prioridade relativa.

Critérios	Opinião da Equipa de Desenvolvimento	Cobertura	Métricas	Dependências	Prioridade Relativa
Opinião da Equipa de Desenvolvimento	0.29	0.28	0.32	0.33	0.31
Cobertura	0.57	0.55	0.41	0.54	0.52
Métricas	0.04	0.06	0.05	0.02	0.04
Dependências	0.10	0.11	0.23	0.11	0.14
Total ≈	1.00	1.00	1.00	1.00	1.00

Analisando as tabelas acima apresentadas, percebe-se que a cobertura é o critério com maior peso, seguido da Opinião da Equipa de Desenvolvimento. Por outro lado, as métricas, não são vistas como um fator de elevada importância, em comparação com os restantes critérios.

A ideia selecionada é a primeira apresentada na Geração e Enriquecimento de Ideias. Selecionou-se esta ideia por se entender que é a única das três existentes que cultiva o conhecimento dos desenvolvedores, obrigando os mesmos a alguma atividade de introspeção e pensamento crítico, apesar de terem em seu auxílio um documento com uma série de indicações. Adicionalmente, como referido na apresentação da ideia, podem ser sugeridas ferramentas que auxiliem os desenvolvedores nalguns aspetos, tirando assim partido do que já existe no mercado. Considerou-se, ainda, que as primeiras duas ideias contribuem mais que a terceira para o atingimento do critério de cobertura, o que corrobora a escolha da primeira ideia.

Com a concretização da ideia selecionada espera-se que a equipa de desenvolvimento possa desenvolver um conhecimento uniforme e equilibrado sobre o tema de testabilidade, criando um ambiente integrado, onde a aprendizagem da equipa e dos seus constituintes é valorizada, esperando, desta forma, que a cobertura passe a ter os valores desejados. Em contrapartida esta ideia será, potencialmente, olhando para todas as ideias apresentadas, a que obrigará a um maior esforço inicial, por parte da equipa de desenvolvimento, no que concerne ao entendimento da estrutura do documento e à sua análise.

3.1.5 Definição do Conceito

Na Definição do Conceito deve ser apresentada a ideia selecionada e as suas vantagens (Vacek, 2006).

Construir um documento, guia, que auxilie os desenvolvedores na sua tomada de decisão quando do desenvolvimento, de modo a aumentar a testabilidade do software, de modo que o mesmo atinga valores superiores de cobertura e tenha uma maior qualidade.

3.2 Valor

O Valor é definido consoante necessidades, critérios, interesses, benefícios, atitudes e preferências, podendo variar consoante distintas perceções (Nicola et al., 2012). Estas variações entre perceções podem ser verificadas entre o desenvolvedor do produto (bens ou serviços) e o cliente, podendo-se dizer, portanto, que o valor anexa uma quanta parte de cariz subjetivo.

No contexto desta dissertação, o valor reside no aumento da qualidade dos produtos desenvolvidos pela Sysnovare, sendo que com uma maior testabilidade e o superior nível de cobertura alcançado, poder-se-á prevenir, com maior sucesso, a ocorrência de erros de software em ambientes de produção, ou seja, quando estes estão a ser utilizados pelo cliente. Com o aumento da qualidade dos produtos disponibilizados será possível aumentar a confiança

por parte dos clientes, manter um alto nível de competitividade no mercado, elevando a reputação da empresa e angariando novos clientes.

3.2.1 Valor para o Cliente

O Valor para o Cliente consiste, como o nome indica, na perceção que o cliente tem sobre um determinado produto/serviço (Nicola et al., 2012). Esta perceção pode também variar de cliente para cliente, consoante as vivências e os contextos de cada um, sendo que o valor é influenciado por vários aspetos, como as necessidades e preferências, tal como já referido (Graf & Maas, 2014). Pode-se considerar que o valor para o cliente é positivo se este entender que teve mais benefícios que custos com a aquisição de determinado produto/serviço.

No que diz respeito ao trabalho em análise, pode-se dizer que os clientes poderão ter uma melhor experiência aquando da utilização dos produtos da Sysnovare, havendo uma minimização de erros com o aumento da cobertura do código, não havendo processos indevidamente interrompidos.

3.2.2 Valor percecionado

O Valor Percecionado relaciona-se com o valor esperado por parte do cliente e o valor sentido, ou seja, reside no *tradeoff* entre as espectativas inicialmente desenvolvidas – antes de adquirir um determinado produto/serviço -, e o que é recebido – depois da aquisição do produto/serviço (Graf & Maas, 2014). Assim, o como acontece no Valor para o Cliente, o Valor Percecionado pode variar de cliente para cliente.

No presente contexto, é importante avaliar os sacrifícios e os benefícios - para o cliente - inerentes a este estudo. Em termos de benefícios, espera-se que os clientes beneficiem de aplicações de maior qualidade e confiabilidade. No que diz respeito a sacrifícios, podem resultar custos acrescidos, resultantes de um maior custo associado à testagem dos produtos – idealmente, caso o nível de Testabilidade aumente consideravelmente, este custo será de reduzido (ou nulo) impacto.

3.2.3 Proposta de Valor

A Proposta de Valor está diretamente relacionada com a forma como uma empresa apresenta um determinado produto/serviço a um conjunto de potenciais clientes, sendo este um aspeto diferenciador no que diz respeito à disputa com a concorrência (Osterwalder & Pigneur, 2003). Assim, na Proposta de Valor, devem ser evidenciados os pontos fortes do produto/serviço, esclarecendo todas as questões de potencial entendimento dúbio, garantindo, desta forma, um melhor entendimento por parte do cliente sobre o proposto.

Esta dissertação tem como objetivo resolver o problema presente relacionado com a deteção retardada de erros no software, devido à baixa cobertura do mesmo. Para cumprir este objetivo,

espera-se apurar de que forma o software apresenta maior testabilidade, construindo um guia que mapeie estes fatores. Este guia deverá ser seguido pela equipa de desenvolvimento, facilitando assim a testagem e consequente cobertura do software, o que poderá, potencialmente, diminuir a quantidade de erros detetados tardiamente, aumentando a qualidade dos produtos produzidos pela Sysnovare. Como referido anteriormente, o aumento da qualidade dos produtos da Sysnovare poderá trazer benefícios para a mesma, tanto internamente – tendo em conta que haverá menos alterações inesperadas aos planos inicialmente delineados -, como externamente - sendo que haverá uma maior confiança dos clientes e um maior reconhecimento no mercado, havendo assim uma melhoria da reputação da empresa.

No que diz respeito aos clientes, que são os consumidores dos produtos da Sysnovare, poderá haver um aumento do grau de satisfação e confiança, relacionados com o aumento da qualidade do software, previamente referida, o que poderá garantir a sua fidelidade.

3.3 Modelo de Negócio (Canvas)

O Modelo de Negócio ilustra a forma como uma organização cria, entrega e recolhe valor (Osterwalder et al., 2014). A sua construção pode assentar em nove blocos/secções que cobrem o cerne do negócio - clientes, oferta, viabilidade e infraestrutura – e constitui uma base para delineação e implementação de estratégias. O Modelo de Negócio afeto ao presente projeto encontra-se apresentado na Figura 13.

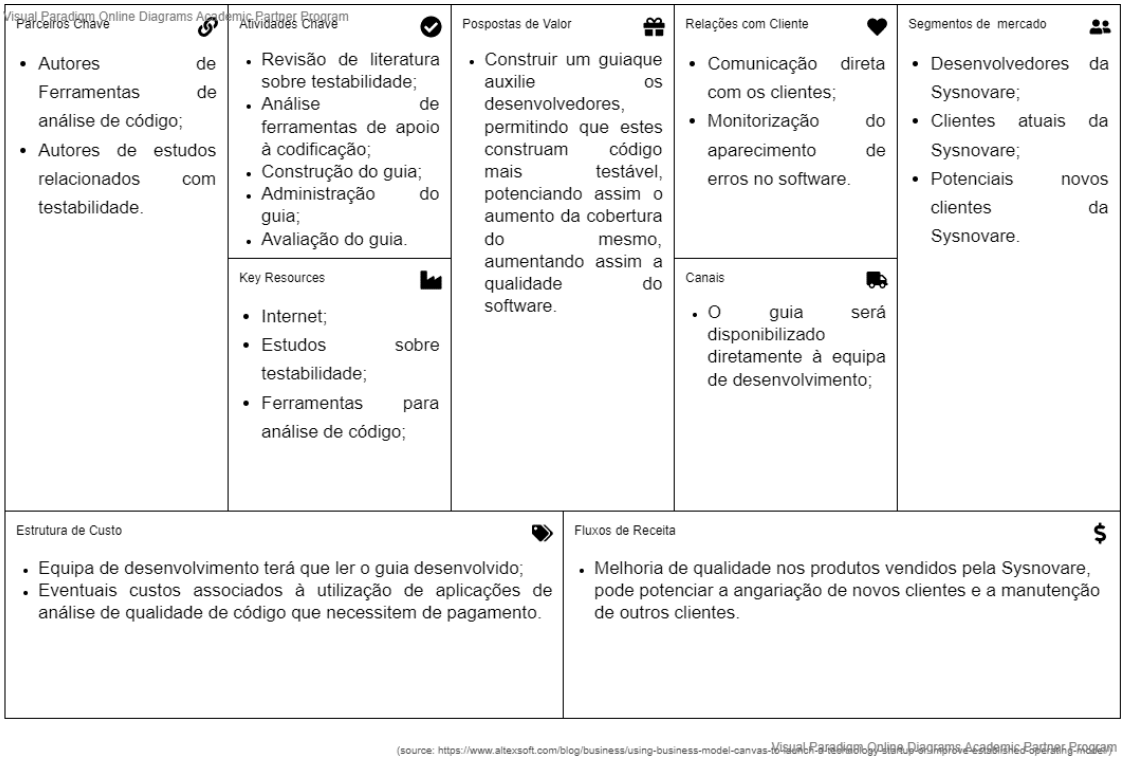

Figura 13 – Modelo de negócio Canvas.

Relativamente ao apresentado na figura, é de salientar que o projeto em questão visa uma melhoria interna da qualidade do software da Sysnovare, sendo que essa melhoria se refletirá nos produtos e serviços prestados, tendo-se por isso incluindo nos segmentos do mercado os clientes da empresa. Da mesma forma, este projeto não terá fluxos de receita diretos e não será distribuído aos clientes da Sysnovare, mas sim à equipa de desenvolvimento que aplicará será responsável por tirar partido do mesmo.

3.4 Análise Funcional (FAST)

A criação de um diagrama *Function Analysis System Technique* (FAST) é importante para clarificar os objetivos de um projeto, especificando as ações/funções integrantes do mesmo, bem como as suas relações lógicas, respondendo a perguntas pertinentes como "porquê", "como", "quem" e "quando" (Rains, 2018). Concomitantemente, o diagrama deve apresentar o objetivo do projeto e como este será alcançado, recorrendo a duas funções limite *Higher Order Function* – "como" – e *Assumed Function* – "porque" -, que serão interligadas através de um conjunto intermédio de funções com a responsabilidade de explicar e mapear o decurso do processo.

De seguida, na Figura 14, é apresentado o diagrama FAST da presente dissertação.

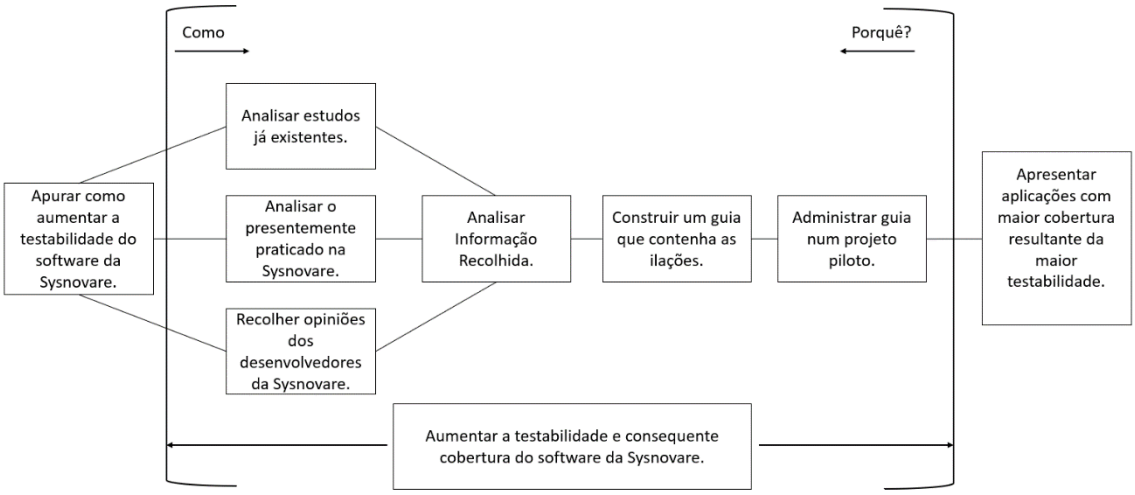

Figura 14 – Diagrama FAST da dissertação.

Como pode ser analisado no diagrama acima apresentado, existe o objetivo de aumentar a testabilidade e a cobertura do software da Sysnovare. Como tal, será necessário entender como é que isso pode ser feito, tendo em conta o contexto da empresa e um conjunto de informações recolhidas externamente. A maior parte destas funções serão realizadas pelo autor que desenvolve esta dissertação, havendo apenas duas funções onde existe colaboração direta da equipa de desenvolvimento: "Recolher opiniões dos desenvolvedores da Sysnovare" e "Administrar o Guia num Projeto piloto". Podemos ainda verificar que do lado esquerdo existe a função de ordem superior "Apurar como aumentar a testabilidade do software da Sysnovare",

sendo esta seguida pelas funções básicas e secundárias. Por fim existe a função assumida – "Apresentar aplicações com maior cobertura resultante da maior testabilidade". O objetivo do projeto encontra-se representado na parte inferior do diagrama – "Aumentar a testabilidade e consequente cobertura do software da Sysnovare".

4 Análise e Desenho

A solução baseia-se na construção de um guia escrito que exponha várias regras a serem seguidas, auxilie na tomada de decisão em situações de dúvida e apresente ferramentas que possibilitem o aumento da testabilidade do software.

Espera-se que o guia reúna um conjunto de alternativas e informações relevantes, resultantes de todos os estudos e análises realizados, permitindo o aumento da qualidade do código desenvolvido pela Sysnovare, fundamentalmente nos aspetos afetos à testabilidade.

4.1 Análise de requisitos

Como já explicado anteriormente, a solução baseia-se na elaboração de um guia escrito que potencia o desenvolvimento de software com uma alta testabilidade. Para cumprir este objetivo, foram colocados um conjunto de requisitos que o guia a deve cumprir (ver Tabela 1). Estes requisitos são analisados em seguida:

1. "O guia deve apresentar regras que devem ser seguidas para aumentar a testabilidade" – o documento a desenvolver deve ter no seu conteúdo um conjunto de regras, devidamente declaradas como regras, que devem ser seguidas pela equipa de desenvolvimento;
2. "O guia deve apoiar a decisão do desenvolvedor nalgumas questões ambíguas" – o guia a desenvolver deverá contemplar alguma orientação no que diz respeito a situações onde a tomada de decisão do desenvolvedor não seja necessariamente clara;
3. "O guia deve ser autoexplicativo" – a consulta do guia deverá ser suficiente para perceber o nele perpetuado, não havendo necessidade de consultar informações externas;
4. "O guia deve sugerir a utilização de ferramentas que potenciem a melhoria da qualidade de código, nomeadamente da testabilidade" – a solução apresentada deve sugerir criteriosamente um conjunto de ferramentas que poderão facilitar o trabalho do desenvolvedor no que diz respeito ao desenvolvimento de código com alta testabilidade.
5. "O guia deve ter em conta as dificuldades reportadas pela equipa de desenvolvimento" – a solução necessita de contemplar as informações recolhidas da interação realizada com os desenvolvedores, de modo a poder solucionar os seus problemas.
6. "O guia deve ser acessível a qualquer membro da equipa de desenvolvimento" – todos os membros da equipa de desenvolvimento da Sysnovare devem ter acesso contínuo ao guia desenvolvido, podendo consultá-lo quando bem entenderem;

7. “O guia deve ser editável, de modo a potenciar a sua evolução” – a plataforma e formato escolhidos para disponibilizar o guia devem permitir que este seja atualizado ao longo do tempo, estando essas atualizações disponíveis para os membros da equipa de desenvolvimento;
8. “O guia deve ser integrado nas práticas atuais da empresa” – a disponibilização do guia deve estar enquadrada nas práticas atuais da empresa, não se pretendendo que este seja entregue num(a) formato/ferramenta que seja novo(a) para o contexto tecnológico atualmente verificado, de modo a potenciar a sua utilização e aderência por parte da equipa que o irá consumir.

Adicionalmente, foram considerados outros aspetos, não como requisitos da solução, mas como objetivos que se visam alcançar através da aplicação desta mesma solução, nomeadamente:

1. Cobertura: pretende-se que o software da Sysnovare aumente os níveis de cobertura, sendo para tal verificados os vários casos de uso e derivações dos mesmos num determinado software;
2. Dependências: ambiciona-se que o software seja independente e que, portanto, não altere o seu comportamento devido a fatores externos, aquando do processo de testagem automática;
3. Equipa de desenvolvimento: ambiciona-se que a equipa de desenvolvimento sinta melhorias no processo de codificação e testagem do software, sentindo um menor esforço associado a este tipo de processos, principalmente no ultimamente referido.

O guia deverá ter em consideração todos os requisitos e objetivos colocados pela organização, estando a sua construção e desenho diretamente relacionados com os mesmos.

4.2 Construção do guia

O guia irá ser desenvolvido gradualmente, na medida em que diferentes componentes do mesmo serão adicionadas/retificadas, consoante a informação recolhida da pesquisa. Assim sendo, é espectável que vá sofrendo uma série de modificações ao longo do projeto, ficando cada vez mais completo e capaz. Adicionalmente, espera-se que o guia possa contemplar alterações futuras, de modo a acompanhar o ambiente interno e externo onde se encontra inserido.

Para a construção do guia serão tidas em consideração as informações apuradas no estado da arte, nomeadamente no que diz respeito: ao que influencia a testabilidade; às ferramentas de análise de qualidade de código e às opiniões dos desenvolvedores da equipa de desenvolvimento da Sysnovare.

A construção da solução deve iniciar-se pelos pontos aferidos como de maior importância, no estado da arte, sendo apenas depois continuado pelos restantes fatores. Para todos estes aspetos deve ser apresentada uma solução adequada, resultante de uma pesquisa sobre o tema respetivo.

De forma alternativa, o guia poderia ser construído tendo apenas como pilar o conhecimento primário do autor e as opiniões apresentadas pela equipa de desenvolvimento no primeiro questionário. No entanto, considerou-se esta abordagem muito limitada e insuficiente para solucionar o problema em mãos, tendo em consideração que não contemplaria informações resultantes de estudos similares, podendo seguir um caminho tendencioso, influenciado apenas pelas vivências pessoais do autor e dos membros da equipa de desenvolvimento da Sysnovare.

4.2.1 Arquitetura

Apesar de não se tratar de uma solução que implique desenvolvimento de software, pode-se na mesma explicar não só a sua estrutura, bem como a sua potencial interação com o utilizador.

Depois de se ter debatido com o chefe da equipa de desenvolvimento, decidiu-se que o guia tem uma componente inicial, onde apresenta os três grandes componentes que o constituem. Esta estrutura está apresentada na Figura 15.

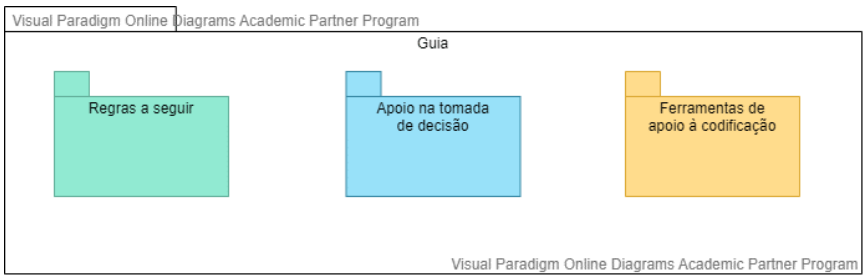

Figura 15 – Vista de desenvolvimento de nível 1, segundo o modelo C4 (Vázquez-Ingelmo et al., 2020).

Da mesma forma que o guia é composto por um conjunto de componentes, cada componente que o constitui é formada por um corpo de secções, que visam tratar, de forma coerente e inequívoca, temas de potencial proveito para o contexto em análise. Assim, cabe ao leitor, depois de decidir que componente pretende analisar, selecionar esse componente, de modo que seja possível consultar a informação sobre o mesmo, como se pode analisar na Figura 16.

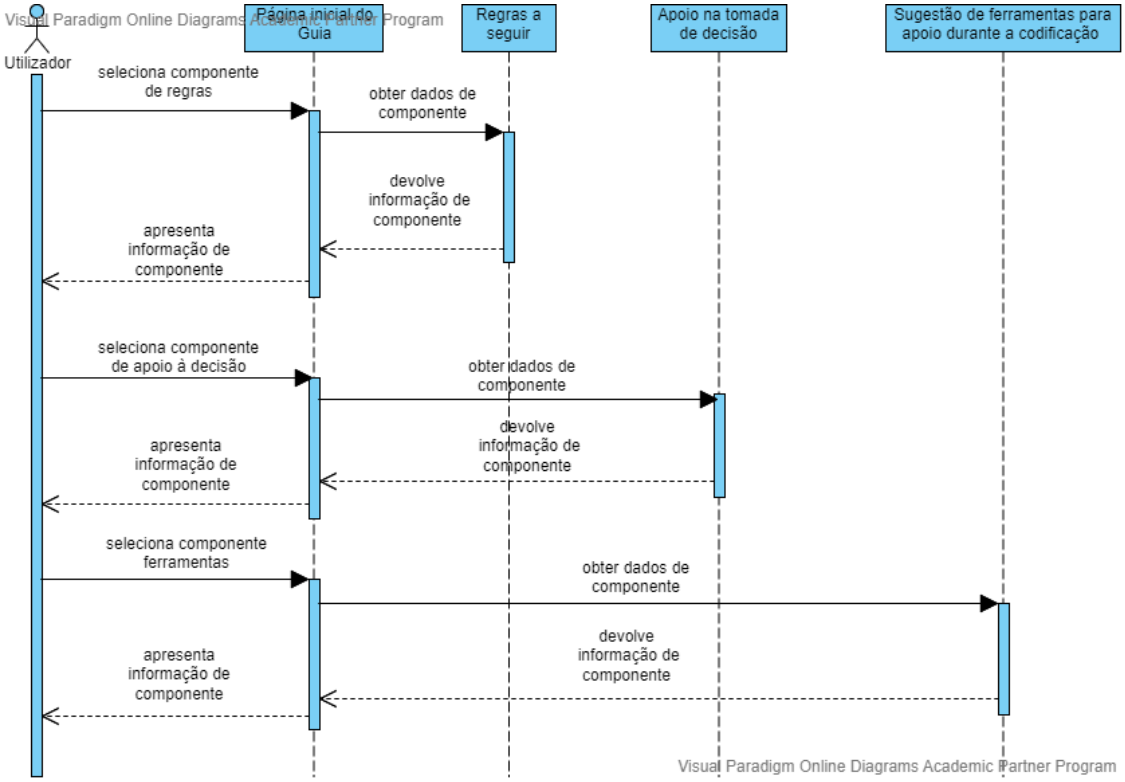

Figura 16 – Vista de processo exemplificativa da interação do utilizador com o guia.

Na Figura 17, está apresentada, de forma simplificada, e a título ilustrativo, a estrutura do componente com o título “Regras a seguir”, de modo a ser possível entender o tipo de estrutura destes componentes.

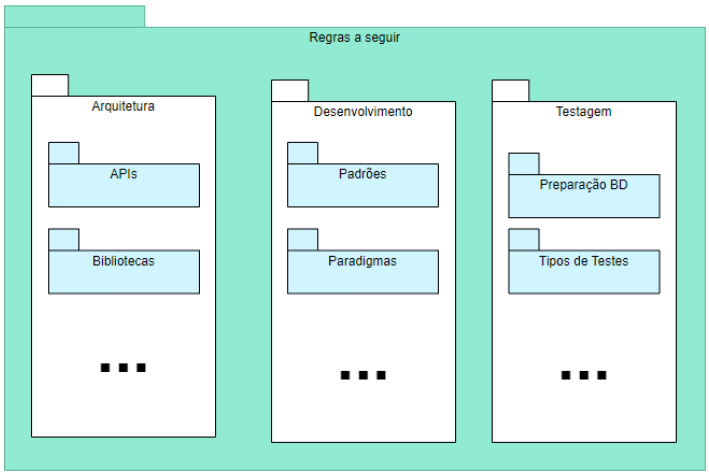

Figura 17 – Vista de desenvolvimento de nível 2 e 3, segundo o modelo C4, para a componente de “Regras a seguir” (Vázquez-Ingelmo et al., 2020).

Como se pode verificar na figura anterior, cada secção de cada componente é também ela composta por vários tipos de informação. Deve-se notabilizar que as secções existentes e os seus constituintes podem sofrer alterações ao longo do tempo.

Cada uma das três secções terá várias subsecções onde serão tratados diversos aspetos afetos ao tema em questão.

Na primeira secção, serão apontadas várias regras que devem ser sempre seguidas com o objetivo de aumentar a testabilidade. Idealmente, estas regras contribuirão sempre para a testabilidade, independentemente do contexto de cada problema ou projeto. Esta secção é a considerada mais relevante para o contexto do projeto em questão, por parte da empresa promotora do mesmo.

Na segunda secção, serão sugeridas soluções para problemas específicos que possam surgir. Esta secção difere na primeira na medida em que cada solução apresentada surge com base num determinado contexto, tendo em consideração um conjunto de fatores que a influenciam. Esta secção poderá, potencialmente, num trabalho futuro, ser aprimorada com a construção de uma ferramenta de apoio à tomada de decisão, como apontado na geração e enriquecimento de ideias, aquando da análise de valor (ver segunda ideia em 3.1.3).

Na terceira secção, serão apresentadas ferramentas que podem/devem ser utilizadas para monitorizar a qualidade do código desenvolvido e para auxiliar noutros aspetos afetos à codificação. Estas ferramentas agirão como complemento às informações apresentadas nas duas primeiras secções, sendo que certas ferramentas podem monitorizar a qualidade de código em momento de desenvolvimento, de modo que o desenvolvedor não cometa alguns erros de forma inadvertida. Ambiciona-se que existam configurações construídas para algumas das ferramentas apontadas, de modo a potenciar o controlo dos fatores que influenciam a testabilidade.

Como alternativa à estrutura apresentada, poder-se-ia seguir uma estrutura baseada nos vários fatores apurados como influenciadores de testabilidade, apresentando regras e sugestões de forma individualizada para cada fator. No entanto, considerou-se que esta solução poderia originar alguma repetição de informação, na medida em que existem fatores com alto grau de relação, havendo, por isso, regras transversais a esses mesmos fatores, que podem ser apresentadas numa sequência lógica de ideias. Considerou-se ainda mais simples para o desenvolvedor adquirir um conjunto inicial de informação não variável, analisando para tal a primeira e terceira secções do guia, recorrendo depois, de forma mais recorrente, à segunda secção, para tratar aspetos que careçam de sugestões apresentadas pelo guia – seria mais trabalhoso identificar estes dois tipos distintos de informação caso se seguisse uma estrutura baseada em fatores.

4.2.2 Conteúdos

Nesta secção serão explicados os principais conteúdos contemplados na solução. Apesar de se perceber que são vários os aspetos que poderão influenciar a testabilidade, tentar-se-á responder aos mais realçados pela equipa de desenvolvimento, relacionando esta informação com a recolhida na análise de que fatores impactam na testabilidade.

4.2.2.1 Uniformização de decisões

Foram vários os aspetos apresentados pela equipa de desenvolvimento como prejudiciais para a testabilidade, como a falta de coerência na forma de programação de alguns desenvolvedores, e a baixa legibilidade do software. Adicionalmente, na recolha de fatores que influenciam a testabilidade, percebeu-se que existem soluções que poderão potenciar um aumento da testabilidade.

Tendo em conta a informação previamente apresentada, nesta secção são explicados os padrões e sugestões que foram colocados no guia, com o objetivo de uniformizar as soluções seguidas para o mesmo tipo de problema, auxiliando ainda a tomada de decisão nalgumas situações.

4.2.2.2 Ferramentas de apoio à codificação

As ferramentas de apoio à codificação acabaram por se verificar como um aspeto potencialmente relevante para a obtenção de uma maior testabilidade. Assim, e em acordo com o chefe da equipa de desenvolvimento, serão apresentadas algumas ferramentas como de utilização obrigatória, enquanto outras serão declaradas como de utilização opcional. Todas as ferramentas apresentadas serão devidamente introduzidas e acompanhadas de uma explicação de como podem ser configuradas e utilizadas.

4.3 Disponibilização e Implantação

Nesta secção será explicado como é que o guia será entregue e como será enquadrado no contexto atual da empresa, percebendo de que forma é que os membros da equipa de desenvolvimento o vão consultar e aplicar.

4.3.1 Implantação

O guia desenvolvido será incorporado na Wiki da Sysnovare. Uma Wiki consiste num conjunto expansível de páginas WEB interligadas e um sistema de hipertexto, para armazenar e modificar dados (Choy & Ng, 2007). A Wiki da Sysnovare tem acesso restrito, sendo que cada utilizador, com permissões, poderá consultar e até editar as várias páginas existentes. A Wiki da Sysnovare é utilizada para armazenar informação relacionada com vários aspetos, sendo um deles o desenvolvimento de software, onde o guia se irá encontrar. De seguida, na Figura 18, poder-se-á ver o guia integrado no contexto da Wiki da Sysnovare, segundo o acima descrito.

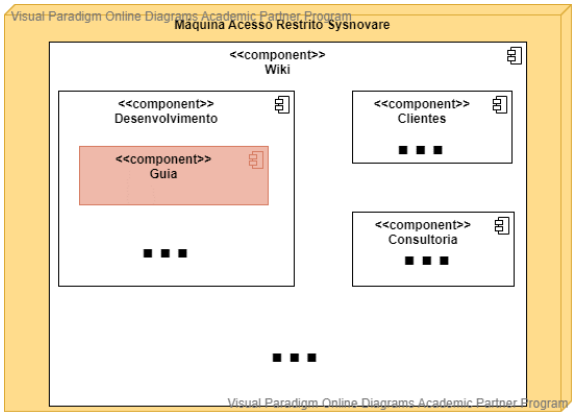

Figura 18 – Vista de implantação de nível 1 com abstração de vista lógica da Wiki de nível 1, com granularidades superiores incorporadas (Vázquez-Ingelmo et al., 2020).

Cada desenvolvedor, acederá, numa perspetiva de utilizador, à Wiki da Sysnovare, mais concretamente à componente onde se encontra o guia, para consultar a informação do mesmo. Assim, podemos ver a Wiki como o sistema com o qual o utilizador contacta, não havendo nenhuma comunicação necessária entre este sistema e eventuais sistemas externos, como se pode verificar na Figura 19.

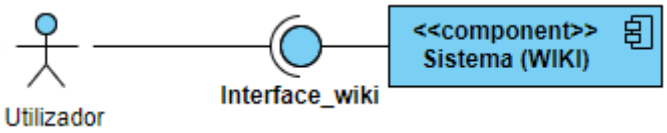

Figura 19 – Vista lógica de nível 1, onde se pode perspetivar a utilização do sistema por parte do utilizador (Vázquez-Ingelmo et al., 2020).

Alternativamente, poder-se-ia entregar este guia num ficheiro, sem recorrer a nenhum tipo de plataforma. No entanto, esta solução apresenta vários problemas, tendo em conta que: não segue o padronizado pela empresa, no que diz respeito à disponibilização e gestão de informação; obrigaria a um cuidado adicional associado à distribuição do ficheiro; seria mais problemático adicionar nova informação ao Guia, sendo que cada desenvolvedor teria que repetidamente readquirir a versão mais recente do mesmo.

Outra alternativa seria colocar esta informação numa outra plataforma, mas preferiu-se seguir o padronizado pela empresa, de modo a facilitar a integração desta informação na já existente, simplificando assim o acesso à mesma.

4.3.2 Disponibilização

De modo a contextualizar a equipa de desenvolvimento do guia realizado, garantindo que este seja empregue da melhor forma possível nos vários projetos, realizar-se-ão um conjunto de apresentações, devidamente delineadas, que permitirão expor a estrutura do guia e explicar alguns conceitos do mesmo. As apresentações realizadas foram escolhidas em conjunto com o

chefe da equipa de desenvolvimento da Sysnovare, pretendendo-se apresentar os conceitos que poderiam representar uma maior novidade para a maioria dos constituintes da equipa de desenvolvimento.

De modo a possibilitar a integração do guia desenvolvido nos trabalhos da equipa de desenvolvimento, deve-se realizar uma reunião com os diferentes gestores de projetos da empresa, de modo a ajustar o planeamento de cada projeto no período inicial de aplicação do guia, de modo a facilitar a sua inclusão. Este planeamento deverá ter um impacto a longo prazo nos diferentes projetos, aumentando a sua testabilidade.

Posto isto, o guia deve ser inicialmente analisado por cada desenvolvedor - de modo que se cumpram as regras apresentadas na primeira secção sejam cumpridas. Adicionalmente, pode-se reconsultar sistematicamente o guia, para que este possa auxiliar em situações de necessidade na tomada de decisão.

Futuramente, poderão ser adicionadas outras informações ao guia, resultantes de estudos ou experiências inseridas no contexto da empresa, permitindo assim que a informação seja incremental e adequada às necessidades. Prevê-se que neste tipo de situações existam apresentações realizadas à equipa de desenvolvimento, como se verificou neste projeto.

Na Figura 20 é representado o processo que um desenvolvedor tomará, aquando do desenvolvimento de software, recorrendo à consulta do guia.

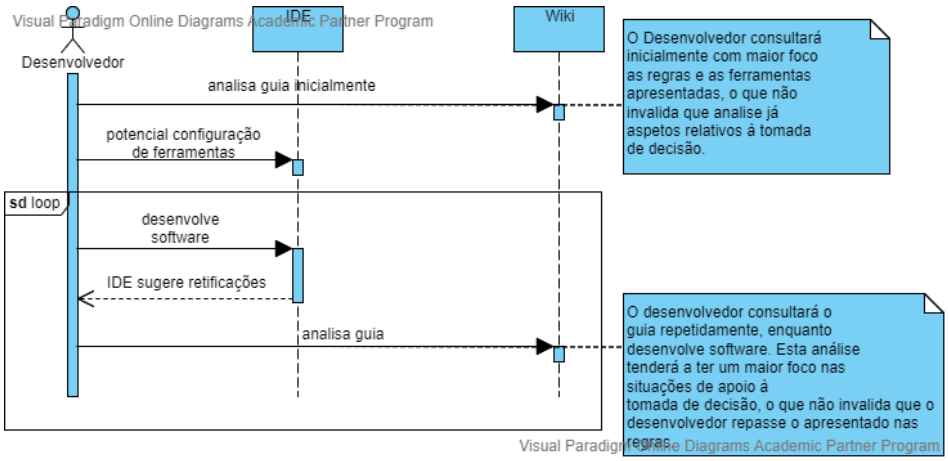

Figura 20 – Vista de processo para enquadrar utilização do guia no processo de desenvolvimento para um desenvolvedor.

Como se pode verificar, o seguimento do guia, por parte de cada desenvolvedor será “paralelo” ao desenvolvimento de software, podendo existir uma consulta contínua do guia enquanto o desenvolvedor constrói o código. Adicionalmente, a configuração das ferramentas sugeridas permitirá um feedback contínuo para com o desenvolvedor, possibilitando que o seu código siga uma série de regras.

No momento do desenvolvimento da dissertação não se pretende descartar código que não cumpra com os requisitos mínimos delineados. No entanto, este tipo de restrição é ponderado pela empresa, esperando-se que exista no futuro um maior controlo do código que não cumpra com as normas estabelecidas para a testabilidade (ou eventualmente outros aspetos), recorrendo a ferramentas de desenvolvimento e integração contínua, integradas com ferramentas de análise de código - (ESLint, 2022a; SonarQube, 2022).

Adicionalmente, de modo a garantir que novos desenvolvedores absorvam também esta informação, o guia – assim como o conteúdo das apresentações realizadas -, será integrado no conjunto de documentação a ser analisada aquando do período de formação de novos membros – cada desenvolvedor da Sysnovare passa por um período de formação de duas semanas (pelo menos), quando inicia o seu trabalho.

4.4 Conclusões

Comporão o guia um conjunto de três secções – "Regras a Seguir", "Apoio na tomada de decisão" e "Ferramentas de apoio à codificação". Cada uma das referidas secções será composta por um conjunto de informação diretamente relacionada com o estudo levado a cabo no estado da arte. Para se potenciar a utilização do guia na empresa: definiu-se a existência de sessões de introdução ao guia e seus conteúdos; delineou-se a realização de uma reunião com os diferentes gestores de projetos; e decidiu-se que o guia será disponibilizado na Wiki da Sysnovare, mais concretamente na componente afeta ao desenvolvimento de software.

5 Solução

Neste capítulo, será explicada a solução implementada para o problema em análise. Para tal, explicar-se-á como se implantou e disponibilizou a solução, analisando ainda, de forma superficial, os seus conteúdos.

5.1 Implantação e Disponibilização

Como anteriormente delineado, o guia foi implantado na Wiki da Sysnovare, na secção respetiva à documentação para a equipa de desenvolvimento, como se pode verificar na Figura 21.

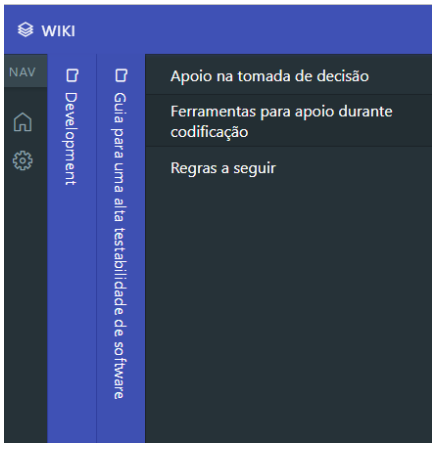

Figura 21 – Enquadramento do guia na Wiki da Sysnovare.

Cada componente do guia seguiu a estrutura previamente delineada aquando do desenho da solução, pelo que, a título ilustrativo, a secção de “Regras a Seguir” é constituída por uma secção de “Arquitetura”, uma secção de “Desenvolvimento” e outra de “Testagem” – como de pode verificar na Figura 22-, de acordo com o anteriormente explicado.

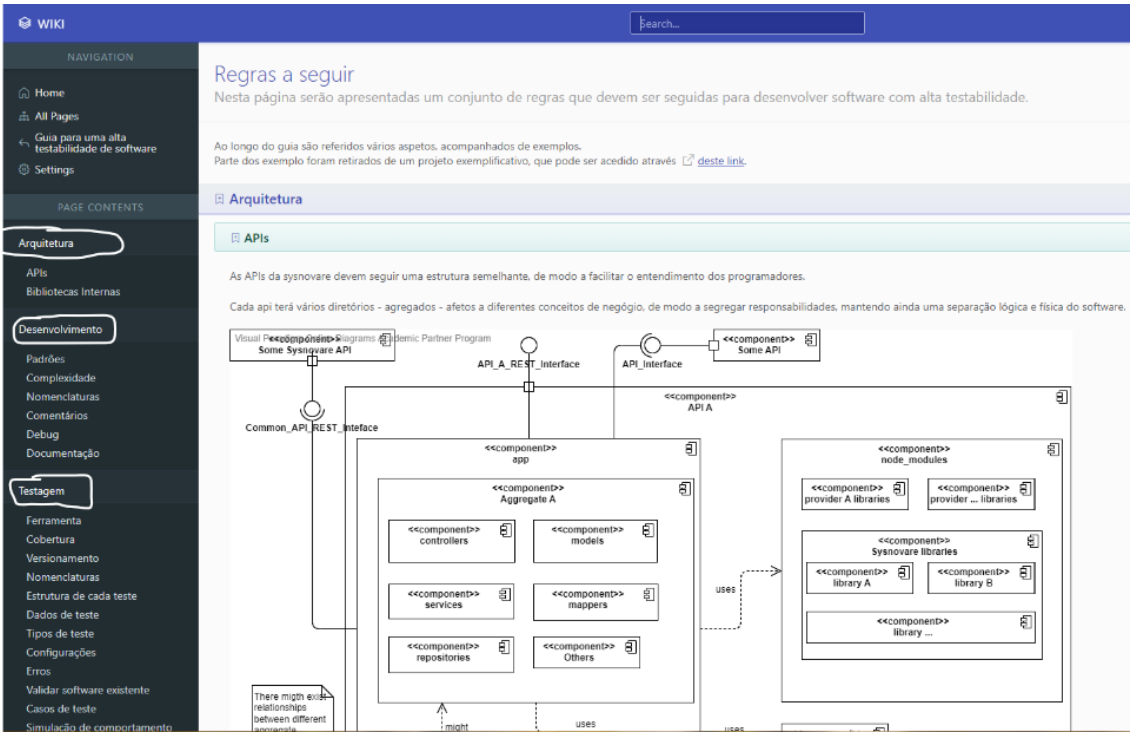

Figura 22 – Excerto de componente de “Regras a seguir”, na Wiki da Sysnovare.

De modo a facilitar a compreensão e adoção do guia foi realizada uma reunião com os vários gestores de projeto e chefe da equipa de desenvolvimento, de modo que o planeamento dos trabalhos dos vários projetos da empresa tivessem em conta um período de estudo/habituação ao guia.

Para além desta reunião, foram realizadas sessões de formação para explicar conceitos de potencial novidade para alguns membros da equipa de desenvolvimento, de modo a aumentar o conhecimento dos desenvolvedores, permitindo que estes pudessem tomar um maior proveito do guia e dos conceitos neles presentes. Ficou à responsabilidade dos membros da equipa de desenvolvimento estudar o guia e colocar eventuais questões que pudessem surgir. O conteúdo destas apresentações foi colocado no serviço de armazenamento de arquivos utilizado pela empresa, sendo acessível por todos os endereços de email do domínio da empresa, de modo que novos membros possam também ter acesso a esta informação.

5.2 Conteúdos

Nesta secção explicar-se-á, sucintamente, o conteúdo da solução desenvolvida e algumas das decisões tomadas aquando da construção da mesma. Para estar de acordo com o explicado anteriormente, agruparam-se em diferentes secções alguns temas, sendo nessas mesmas secções apresentado o que se fez nesse âmbito.

Note-se que a organização da secção de Conteúdos não reflete a disposição dos mesmos na solução – na Wiki tentou-se colocar as informações com um encadeamento lógico e de fácil perceção para os desenvolvedores.

Todas as decisões tomadas e informação colocada no guia foram devidamente apresentadas e debatidas com o chefe da equipa de desenvolvimento, de modo que não fosse construída uma solução que não se enquadrasse com o espectável no contexto da empresa.

Deve-se notar que para a construção do guia analisaram-se diversas soluções, não sendo possível, no tempo destinado à construção da solução, incluir todas no guia. Assim, incluíram-se no guia certos componentes que visam resolver grande parte dos défices encontrados relativos ao problema em análise nesta dissertação, havendo, obviamente, espaço no futuro para que a informação agora apresentada seja evoluída. Adicionalmente, procurou-se colocar informação que pudesse ser facilmente integrada naquilo que é o conhecimento presente dos desenvolvedores.

É ainda de salientar que na construção do guia colocaram-se, sempre que possível, exemplos práticos da aplicação das regras/sugestões apresentadas, de modo a facilitar o entendimento das mesmas, servindo também de referência para a sua aplicação. Adicionalmente, criaram-se ainda aplicações de demostração, com uma envergadura reduzida, para enquadrar melhor alguns dos conceitos explicados. Estas demonstrações são acessíveis a partir do guia.

A solução trata um grande conjunto de aspetos, satisfatoriamente, apesar de, eventualmente, não apresentar sempre uma solução ideal para todos eles. Isto vai de encontro ao pretendido pela empresa: não se pretende uma solução ideal para um conjunto reduzido de particularidades que aumentem a testabilidade, mas sim uma solução satisfatória para um conjunto mais alargado de aspetos que permitam melhorar a testabilidade do software

Nem todas as informações colocadas no guia serão aqui referidas. Privilegiaram-se as que se entendem ter uma maior ênfase.

5.2.1 Uniformização de decisões

Como explicado no desenho da solução, foram especificados um conjunto de padrões/abordagens a serem seguidos aquando do desenvolvimento do software, de modo a uniformizar o código desenvolvido, para a maximizar a testabilidade. Alguns destes padrões encontram-se já validados e usados na indústria, enquanto outras abordagens visam suprimir especificamente alguns défices detetados no ambiente da empresa. Note-se que, na construção da solução, foi tido em consideração o facto de os padrões não resolverem todos os problemas, devendo-se sempre ter cuidado na forma como estes são aplicados, como explicado anteriormente (ver 2.2.2.12).

Alguns dos conteúdos presentes nesta secção não estão diretamente relacionados a padrões ou princípios específicos. No entanto, visam potenciar a compreensão de todo o software desenvolvido, por parte de todos os desenvolvedores (tendo em consideração as dificuldades

relatadas no primeiro questionário aplicado à equipa de desenvolvimento), o que poderá potenciar a testabilidade.

5.2.1.1 Herança

Depois de um estudo ao software desenvolvido na Sysnovare, verificou-se que existem alguns casos, ainda que pontuais, onde é aplicada a Herança.

A herança, como explicado no estado da arte, permite a substituição dinâmica de um objeto por uma derivação do mesmo, podendo impactar negativamente a testabilidade em situações onde a hierarquia de dependência entre classes é elevada (Canal et al., 2001; Mouchawrab et al., 2005).

Tendo em conta a informação recolhida, a utilização de herança é desencorajada no guia, não sendo, apesar disso, proibida. Assim, de modo a evitar a utilização excessiva/incorreta da herança, definiram-se algumas regras. Algumas dessas regras são apresentadas em seguida, à medida que são explicados princípios e abordagens.

5.2.1.2 Open-Closed Principle (OCP)

Este princípio integra os princípios SOLID e defende que deve ser possível estender o comportamento de uma classe sem a modificar (Fenton, 2018; Ingeno, 2018; Robert C. Martin, 2009). Com este princípio, é possível garantir que novas codificações assentes em codificações já existentes não afetam o código anteriormente desenvolvido.

A adoção deste princípio é sugerida para situações onde não se pretenda de facto alterar as funcionalidades já existentes, com a adição de uma nova funcionalidade elas relacionada.

5.2.1.3 Liskov Substitution Principle (LSP)

O princípio da substituição de Liskov afirma que qualquer classe deve ser substituível pela sua classe pai (superclasse), sem disso resultem consequências inesperadas (Fenton, 2018; Ingeno, 2018; Robert C. Martin, 2009). Definiu-se que este princípio deve ser seguido à regra para situações onde seja praticada a herança, de modo a evitar erros inesperados no sistema.

5.2.1.4 Interface Segregation Principle (ISP)

Este princípio afirma que uma entidade nunca deve ser forçada a implementar uma interface que contenha elementos nunca por si utilizados, devendo o software ser dividido em várias interfaces mais pequenas e coesas, de modo a cobrir apenas as funcionalidades necessárias (Fenton, 2018; Ingeno, 2018; Robert C. Martin, 2009). Este princípio foi apresentado no guia como uma regra a seguir, no que diz respeito à construção e implementação de interfaces, de modo a evitar implementações indesejadas e ambíguas de software, o que poderá afetar negativamente a testabilidade.

5.2.1.5 Dependency Inversion Principle (DIP)

Este princípio defende que o código de alto nível não deve depender de interfaces de baixo nível, usando para isso abstrações, de modo a diminuir o acoplamento do código (Fenton, 2018; Ingeno, 2018; Robert C. Martin, 2009). O facto de haver um potenciamento de um baixo acoplamento, como verificado no estado da arte, é benéfico para uma maior testabilidade. O

DIP estende os conceitos de OCP e LSP, sendo que ao depender apenas de abstrações, o código fica mais desvinculado de detalhes de implementação.

Com este princípio, a camada de alto nível interage apenas com a interface de abstração da implementação desejada, sendo essa interface a responsável por definir qual classe realizará a respetiva implementação, como se pode verificar na Figura 23 (Tarlinder, 2017).

Figura 23 – Implementação de uma funcionalidade de obtenção de mensagem sem o princípio DIP (do lado esquerdo) e com o princípio DIP (do lado direito) (Tarlinder, 2017).

Este princípio é apresentado como uma sugestão aos desenvolvedores, sendo, de certa forma, apresentado como de aplicação "obrigatória" para casos onde exista mais do que uma implementação possível para uma mesma funcionalidade, como se acabará por explicar de seguida, na secção acerca de injeção de dependências.

5.2.1.6 Single Responsibility Principle (SRP)

Este padrão integra os princípios SOLID e afirma que uma classe ou módulo deve ter um, e apenas um, motivo para mudar, dando uma visão mais profunda daquilo que são as responsabilidades que devem ser entregues a cada classe e até a cada método (Fenton, 2018; Ingeno, 2018; Robert C. Martin, 2009).

Para o cumprimento deste padrão, analisou-se o software desenvolvido, de modo a perceber aquilo que são as ações predominantes no software, de modo a criar uma estrutura que possa ser seguida em grande parte das situações. Esta estrutura e seus constituintes têm já em conta as responsabilidades de cada componente e pode sofrer alterações no futuro, caso os requisitos do software desenvolvido pela empresa assim o obriguem.

Tendo como base a arquitetura já utilizada em grande parte das soluções da Sysnovare, referida no estado da arte, tentaram-se definir uma série de regras relativamente à segregação de responsabilidades, tendo-se analisado com detalhe o software desenvolvido na empresa e as suas necessidades. Assim, cada agregado deve ser constituído pelas seguintes componentes (Buchmann et al., 2007; Fowler, 2011; Terragni et al., 2020):

- **Controladores**: componentes onde são declaradas as diversas rotas e especificados os serviços a si afetos, sendo o ponto de acesso externo para o agregado em questão (padrão ***Front Controller***);
- **Serviços**: componentes onde são geridos os fluxos inerentes às várias funcionalidades de negócio, sendo o ponto de acesso interno para o agregado em questão;

- **Modelos**: componentes onde são especificadas as entidades e objetos de valor, bem como a sua instanciação e alteração de estado. São estas componentes as únicas que podem requerer à componente que comunica com bases de dados a alteração de dados, passando-lhe os dados devidamente já construídos (padrão ***Builder***);
- **Mapeadores**: componentes com a responsabilidade de converter uma estrutura de dados inicial numa outra estrutura de dados final (padrão ***Mapper***);
- **Esquemas**: componentes onde são validadas as estruturas e conteúdo de objetos e parâmetros;
- **Repositórios**: componentes responsáveis pela comunicação direta com as bases de dados (padrão ***Repository***);
- **Serviços externos**: componentes responsáveis por comunicar com sistemas externos, como por exemplo outras APIs.

As **responsabilidades de cada componente não devem ser violadas**, não sendo espectável que certos componentes tomem ações que vão para além das suas responsabilidades. Da mesma forma, as comunicações internas entre componentes deve seguir um fluxo pré-definido, de modo a evitar dependências circulares e de modo a garantir coerência no processo de codificação.

Esta explicação foi apresentada no guia por escrito, em conjunto com um diagrama de componentes ilustrativo do referido. O referido diagrama é apresentado na Figura 24.

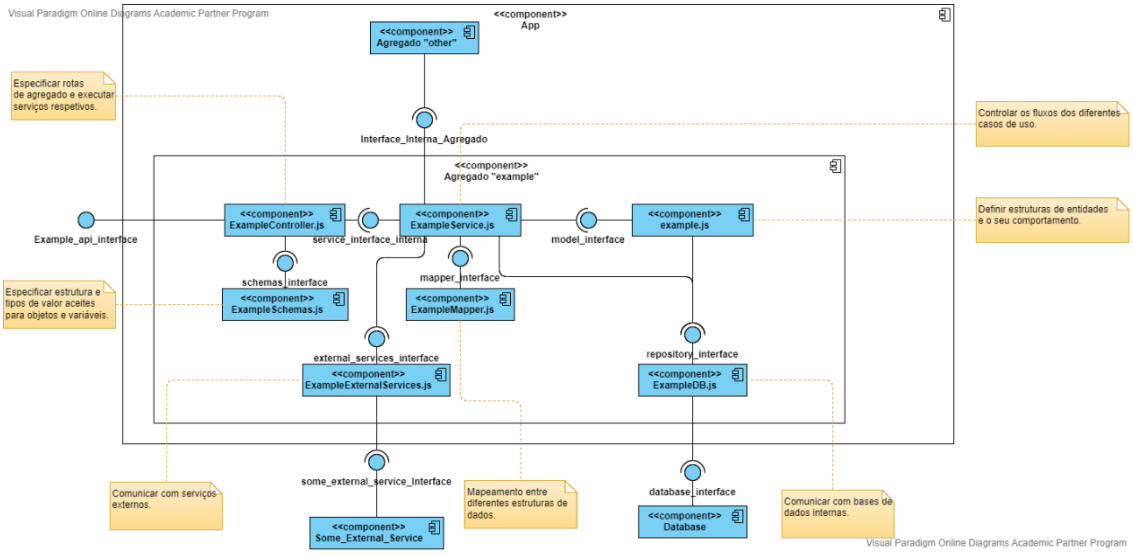

Figura 24 – Diagrama de componentes ilustrativo das diferentes componentes de um agregado e suas responsabilidades.

Tendo em conta esta informação, e permitindo-se que certas classes expandam o comportamento de classes terceiras, definiu-se como regra que uma classe que expanda outra classe não pode ser estendida por uma terceira classe, garantindo assim uma árvore hierárquica reduzida, que não impacte negativamente a testabilidade do software, garantindo que a utilização de herança, caso exista, não seja desmedida. Neste tipo de situações, sugere-se aos

desenvolvedores que repensem o desenho previamente delineado, de modo que este possa estar enquadrado com os requisitos e regras do software.

É salientado no guia que o conjunto de componentes apresentados pode ser alargado, caso se verifique essa necessidade, tendo em conta a especificidade de negócio e enquadramento do software desenvolvido. Estas alterações poderão também ser refletidas futuramente no guia, ao longo do tempo.

5.2.1.7 Dependency Injection

A injeção de dependências foi reconhecida, no estado da arte, como um padrão que potencia a testabilidade, devido à diminuição do acoplamento entre os componentes do software. Este padrão é muito importante para o cumprimento do SRP porque permite passar informação entre objetos, de modo que apenas o objeto com a respetiva responsabilidade acabe por tratar devidamente essa mesma informação (Robert C. Martin, 2009).

Ao contrário do *TypeScript*, o *JavaScript* não é o ideal para a especificação do tipo das estruturas de dados, podendo a mesma variável assumir múltiplas formas, desde um número, a um campo de texto ou objeto (Fenton, 2018).

Sendo assim, a injeção de dependências deve ser ponderada e realizada de forma controlada. De modo a aumentar a testabilidade, sugere-se que os dados injetados estejam sempre coerentes com o negócio, sendo que cada elemento injetado deve ser representativo de algo no contexto do fluxo onde está inserido. Da mesma forma, deve-se delegar a informação aos componentes que sabem o seu significado e que a sabem tratar corretamente, de modo que seja minimizada a possibilidade de erros.

Existem casos onde a mesma funcionalidade tem diferentes implementações para modelos distintos de dados. Depois de debater com o chefe da equipa de desenvolvimento, foi decidido que neste tipo de situações todas as implementações dessas funcionalidades devem seguir estruturas iguais, de modo a que seja possível injetar, eventualmente de forma parametrizada, a implementação desejada, indicando, por exemplo, qual o modelo desejado, seguindo alguns princípios já referidos, como o DIP, e até as soluções defendidas por outros padrões, como por exemplo o *Factory* - permite criar determinados objetos sem expor a sua lógica de criação, sendo o pedido da sua criação realizado através de uma interface comum (Fenton, 2018; Fowler, 2011).

Da mesma forma, definiu-se que todos os processos de alteração de dados na base de dados devem começar com a criação de uma instância do modelo respetivo, injetando os dados necessários para o mesmo. Com a informação injetada recebida no construtor da classe ou em métodos de alteração de informação, esta classe faz o seu tratamento de dados, sabendo que tipos de dados aceita e o que fazer com eles, garantindo que as alterações à base de dados respeitam a lógica de negócio.

5.2.1.8 Singleton Pattern

Com este padrão pode-se garantir que um determinado objeto tem uma instanciação única no sistema, sendo possível ter um maior controlo (controlabilidade) sobre o seu comportamento e estado, garantindo que não existem inconsistências de dados, sendo que o objeto apresenta um caráter global a toda a aplicação (Ethan Trostler, n.d.; Harmes & Diaz, 2008).

Depois de um estudo ao software desenvolvido na Sysnovare, verificou-se que existem alguns casos onde se armazenam dados em *cache*. Foi definido que o *Singleton Pattern* deve ser utilizado, em regra, para a manutenção de dados armazenados em cache, garantindo que estes estão sempre atualizados. Esta abordagem poderá contribuir para combater o défice verificado no estado da arte relativamente à cobertura do código afeto a dados armazenados em cache.

5.2.1.9 Observer Pattern

Com este padrão é possível que certos componentes monitorizem outros, sendo averiguado o seu comportamento e alterações, de modo que seja possível despoletar medidas que vão ao encontro das necessidades de negócio para aquele acontecimento em específico (Harmes & Diaz, 2008). Este padrão pode-se verificar a vários níveis, sendo que os referidos componentes podem variar desde a objetos até uma API inteira, por exemplo, sendo inclusivamente uma opção seguida em arquiteturas de microsserviços (Li et al., 2019). Adotando este padrão pode-se aumentar a simplicidade de alguns aspetos do software, diminuindo o acoplamento entre classes (Chowdhary, 2009).

Este padrão é apresentado como sugestão para situações onde se pretendam abstrair comportamentos humanos do sistema, permitindo manter os fluxos mais coesos, controlados e tipicamente mais rápidos. A diminuição da interferência humana pode ainda facilitar os processos de testagem.

Sugere-se este padrão para situações onde existam alterações de dados armazenados em cache, de modo que estes estejam coerentes com a realidade. Este enquadramento é muito relevante porque pode solucionar alguns problemas presentemente verificados na empresa neste aspeto, quando existe cache de dados afeto a mais do que uma biblioteca interna (foi verificado um défice de cobertura neste aspeto, assim como referido no estado da arte).

Sugere-se, ainda, a utilização deste padrão em situações onde se verifiquem fluxos altamente complexos, com a necessidade de um controlo elevado das várias alterações de dados e ações realizadas. Este tipo de utilização poderá contribuir para um aumento da observabilidade e controlabilidade do software.

5.2.1.10 Comentar corretamente e oportunamente

A carência de comentários foi apontada por membros da equipa de desenvolvimento como um fator que dificultava a legibilidade e compreensão do código, o que acabou por ser corroborado ao estudar estes conceitos (Boswell et al., 2012). Portanto, na solução estabeleceu-se um conjunto de regras a serem seguidas para que o código tenha comentários adequados onde necessário. Para uma melhor compreensão das regras expostas, nalgumas situações colocaram-

se exemplos do que deve ser feito e do que não deve ser feito. Sumariamente, apresentar-se-ão de seguida as regras colocadas no documento (Boswell et al., 2012):

- Os comentários devem ser curtos e elucidativos;
- Comentários que expliquem o obvio são desnecessários para a compreensão do software – ler comentários toma tempo, por isso não se deve desperdiçar tempo a ler algo que não traga valor;
- Deve-se ter rigor no que se comenta – comentários ambíguos ou incorretos podem induzir em erro;
- Tomadas de decisão e pensamento tomado devem ser colocadas nos comentários;
- Ao comentar, deve-se ter em conta que no futuro pode ser outra pessoa a ler o código em questão – desenvolvedor deve tentar pôr-se no papel de um outro programador que possa vir a ser confrontado com aquele software;
- Devem-se comentar melhorias futuras e eventuais erros detetados no código, recorrendo para tal a palavras-chave como "TODO" e "FIXME", que possam de facto contribuir para que essa informação seja lida;
- Devem-se comentar todos os métodos, explicando o que estes tratam, quais os seus parâmetros de entrada e parâmetros de saída. Os comentários explicativos de cada método devem seguir uma das estruturas definida pela biblioteca de *JSDoc*, que passará a ser de utilização obrigatória, como será explicado mais à frente (*Use JSDoc*, n.d.).

De seguida, na Figura 25, é apresentado um excerto da secção de comentários, com algumas das regras previamente referidas.

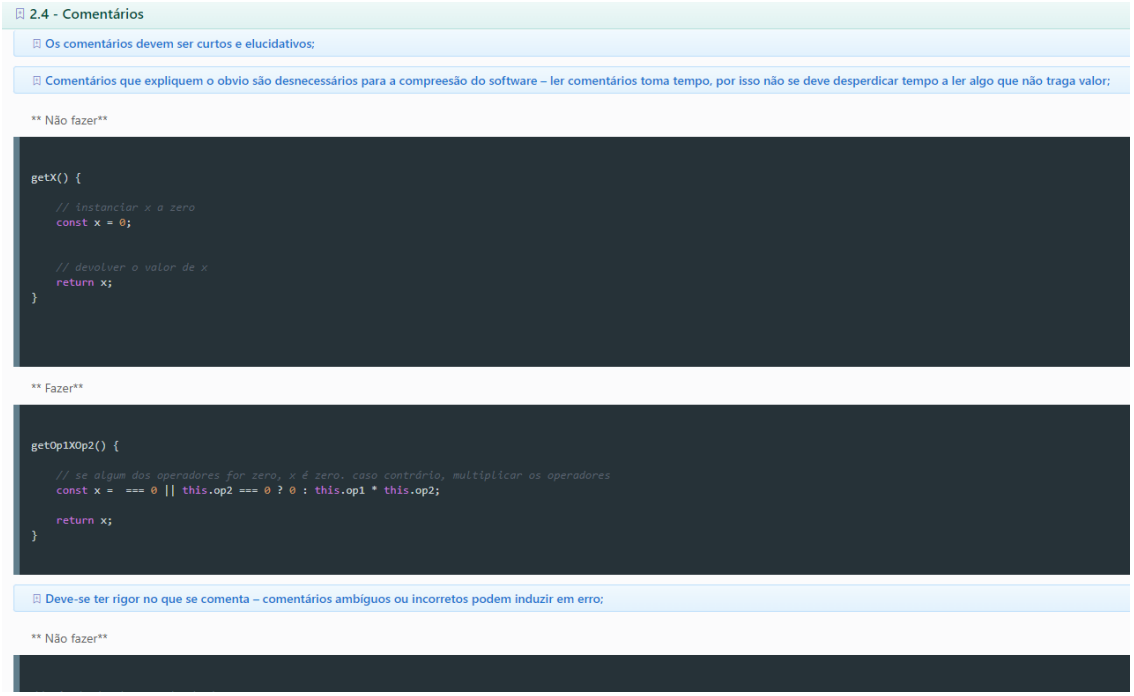

Figura 25 – Excerto de guia onde são apresentadas algumas regras sobre a escrita de comentários no código.

5.2.1.11 Simplificar, uniformizar e reutilizar código

A dificuldade em entender código complexo escrito por terceiros foi também apontada pelos desenvolvedores como um fator que lhes dificulta a testagem do software. Relacionado com este fator foi também apontado o facto de diferentes programadores optarem por diferentes abordagens enquanto constroem o seu software.

Desta forma, para além de definir regras relativamente aos comentários a colocar, colocou-se algum apoio que visa ajudar os programadores a escreverem código mais uniforme, simples, legível e compreensível para terceiros. O apoio colocado para a simplificação do código assenta em fatores como, por exemplo, a comparação de argumentos, simplificação de ciclos, ações condicionais, nomenclaturas, retornos antecipados em métodos, entre outros (Boswell et al., 2012).

De seguida, na Figura 26, pode-se ver um excerto das sugestões colocadas com vista a simplificar e uniformizar o código.

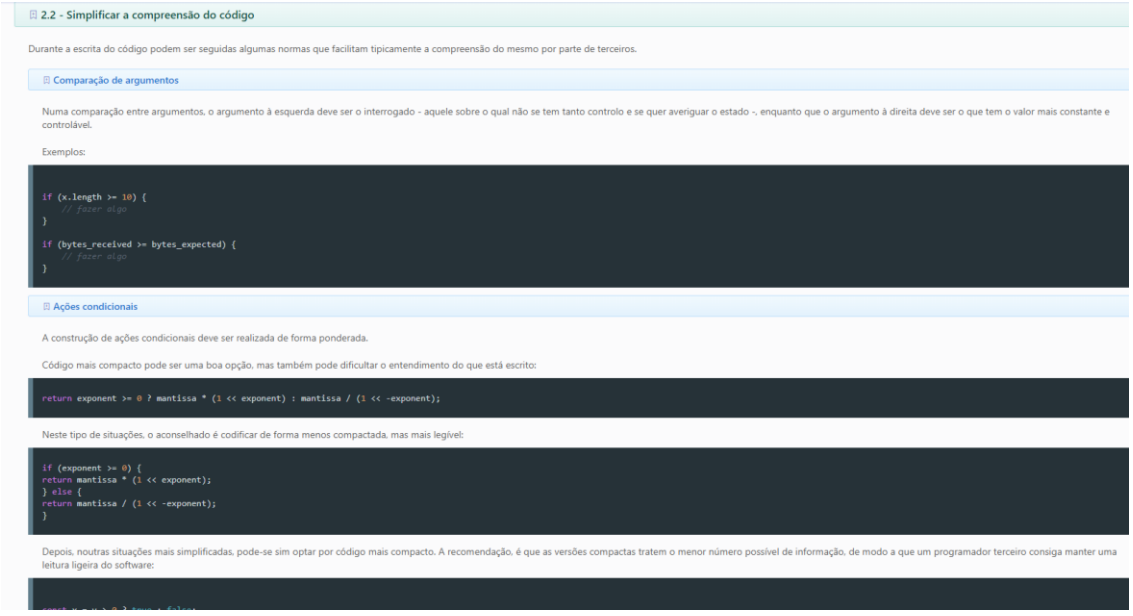

Figura 26 – Excerto do guia relativo à apresentação de sugestões para simplificação e uniformização do código.

Concomitantemente, desenvolveram-se, no contexto desta dissertação, algumas funcionalidades – nas bibliotecas internas da organização-, de modo a uniformizar a codificação de aspetos que foram detetados como de interesse e utilização comum nos vários projetos da empresa, potenciando, assim, a reutilização de código e combatendo, simultaneamente, a existência de código redundante. Este tipo de abordagem - de codificação de software, de modo que este possa ser reutilizável e da própria reutilização de software já desenvolvido- foi também reforçado e promovido no guia.

5.2.1.12 Nomenclaturas e estrutura dos testes

Foram definidos alguns aspetos no que diz respeito à construção de testes automáticos, devido ao facto de os testes automáticos inicialmente existentes não seguirem todos a mesma lógica.

Em primeiro lugar, definiu-se que cada teste deve explicar, seguindo nomenclatura de negócio, o que está a ser testado e qual o resultado esperado para esse mesmo teste, de modo que seja possível entender mais facilmente, em caso de erro, que conceito de negócio está abrangido. Este aspeto visa resolver um dos pontos referidos pelos desenvolvedores no questionário inicialmente aplicado.

Em segundo lugar, foi definido como regra que os testes automáticos devem ser estruturados segundo o padrão *Arrange Act Assert* (AAA), começando pela preparação das variáveis necessárias para a execução do teste, passando pela execução do código a ser testado e terminando com a comparação entre o resultado obtido e o esperado (Vance, 2014). Desta forma, todos os testes desenvolvidos na empresa terão uma lógica coerente e uniforme.

5.2.1.13 Especificação de casos de teste

Denotou-se, aquando da recolha de opiniões por parte da equipa de desenvolvimento, alguma dificuldade no que diz respeito à definição do que deve e não deve ser testado, para uma determinada funcionalidade, ou seja, verifica-se uma incerteza aquando da formação de casos de teste. Esta perceção vai ainda de encontro aos objetivos da dissertação. Isto acontece porque a especificação de casos de uso é um passo fundamental no que concerne à testagem do software (Heumann, 2001).

Um dos aspetos que se deve ter em consideração na especificação de casos de teste é o fluxo normal do caso de uso – o que tipicamente se verifica na maior parte das vezes (Heumann, 2001). Adicionalmente, devem-se ainda considerar os vários fluxos alternativos de cada caso de uso – os que representam comportamentos de carater opcional, alternativo ou excecional (Heumann, 2001). Esta variância de fluxos encontra-se exemplificada na Figura 27.

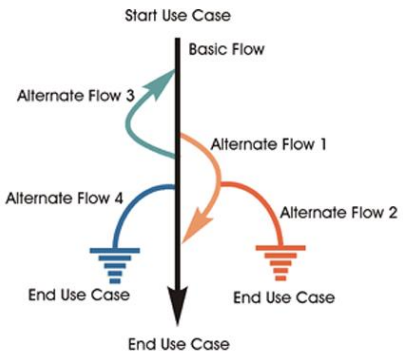

Figura 27 – Fluxo básico de eventos e fluxos alternativos (Heumann, 2001).

Tendo em conta o especificado, definiu-se como regra que deve haver pelo menos um caso de teste para cada cenário (fluxo), sendo que nalguns casos pode-se justificar a existência de mais do que um teste, para validar, por exemplo, casos limite (Heumann, 2001).

Para além do especificado, poder-se-ia definir a adoção de estratégias de documentação detalhada dos diferentes casos de teste a aplicar, recorrendo por exemplo a matrizes (Heumann, 2001). No entanto, depois de um debate com o chefe da equipa de desenvolvimento da

Sysnovare, entendeu-se que a própria definição de cada caso de uso deve ser o suficiente para a construção dos testes, sendo uma especificação adicional de casos de teste não justificativa – e difícil de enquadrar nas práticas atuais da equipa de consultoria - para aquilo que são tipicamente os problemas solucionados na empresa e a forma como o software é estruturado e desenvolvido. Para além disso, foi referido que o desenvolvedor tem acesso aos níveis de cobertura do software, havendo ainda a definição de nível mínimo de cobertura nos 95%, o que deverá contribuir para o desenvolvedor cobrir a maior parte dos fluxos presentes num determinado software.

5.2.1.14 Test Driven Development (TDD)

Como explicado no estado da arte, na empresa, os testes são realizados depois da codificação do respetivo software, seguindo uma abordagem de Test Last Development. No entanto, como explicado anteriormente, existem outros processos, como o TDD, que potenciam frequentemente a cobertura do software (ver 3.1.2).

Tendo em conta esta informação, e depois de um debate com o chefe da equipa de desenvolvimento, definiu-se, no guia, que esta estratégia deve ser apenas seguida aquando da deteção de erros no software – ou seja, quando um erro é detetado, deve-se escrever um conjunto de testes que permitam validar a situação em questão, devendo ser possível executar esses testes (com sucesso) depois da correção do dito erro. Nos restantes casos, não se deve seguir o TDD por algumas razões, como:

- Existe uma grande parte de software já desenvolvido que não se encontra testado, sendo que utilizar-se-á a técnica de TLD para o fazer;
- Grande parte dos desenvolvedores está pouco familiarizado com o processo, pelo que não se pretende que exista um grande esforço na compreensão/aplicação do mesmo, não sendo este considerado um aspeto fundamental para cumprimento da solução. Prefere-se, ao invés disso, que os desenvolvedores dediquem o seu tempo a perceber/seguir a restante informação colocada no guia.

5.2.1.15 Depuração do software

Foi verificado no estado da arte que a observabilidade e a controlabilidade são dois fatores que impactam fortemente na testabilidade do software. Da mesma forma, percebeu-se que não existia, no início da dissertação, nenhuma ferramenta para realizar uma depuração mais detalhada do software, em momento de execução, havendo apenas a escrita de informações para um ficheiro de registo.

Assim, definiu-se como regra que os desenvolvedores devem privilegiar, em momento de desenvolvimento e aquando da realização de testes manuais no software, a utilização de uma ferramenta de depuração, especialmente em situações onde estão a tentar detetar a origem de um determinado erro. De modo a promover e facilitar a utilização deste tipo de ferramenta, foi realizado um estudo sobre estas ferramentas no contexto da tecnologia *NodeJs*, e tendo em conta o IDE privilegiado pelos desenvolvedores, *Visual Studio Code*. Assim, colocou-se no guia uma explicação de como os desenvolvedores podem incorporar o *debugger* deste mesmo IDE

no software desenvolvido pela empresa (*Debugging in Visual Studio Code*, n.d.). Na Figura 28, está apresentada a configuração necessária para contemplar o *debugger* no IDE previamente referido, sendo ainda colocada alguma informação adicional, que pode redirecionar os desenvolvedores para informações extra que possam querer consultar.

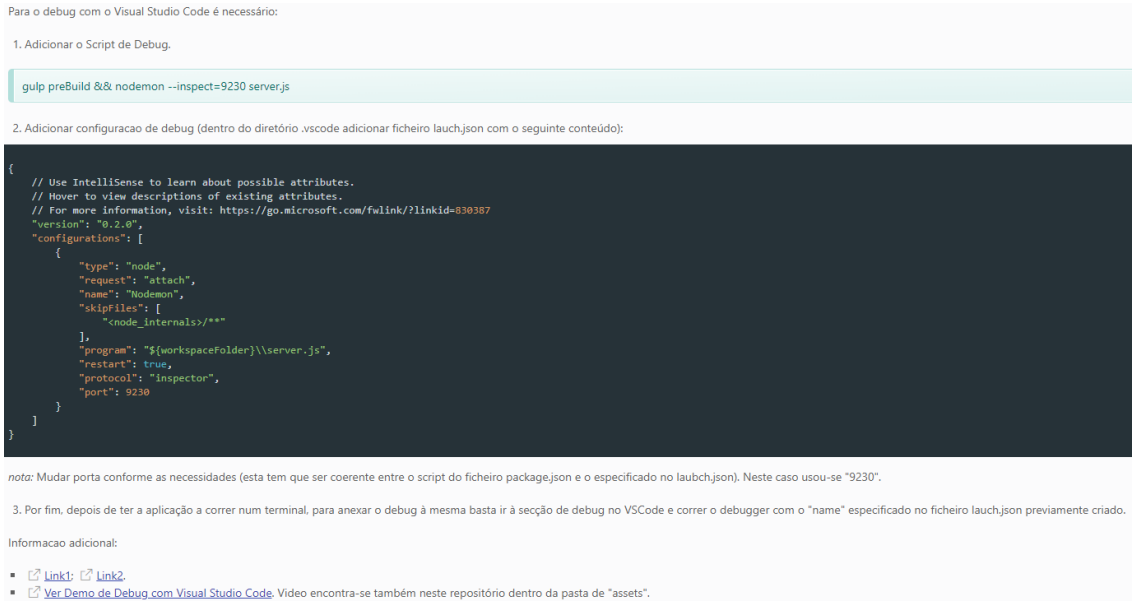

Figura 28 – Explicação de configuração do *debugger* no *Visual Studio Code.*

Não obstante a especificação desta prática de depuração, foi assente no guia que a escrita de informação para ficheiros de registo previamente realizada deve continuar, de modo que essa informação possa ser consultada ao longo do tempo, permitindo também perceber certos comportamentos que possam surgir apenas em ambientes de produção, ou seja, quando o software esteja a ser utilizado pelos clientes.

5.2.1.16 Redução de dependências na testagem

A redução das dependências aquando da testagem do software foi apresentada como um dos objetivos para esta dissertação, tendo-se verificado ainda, posteriormente, que nenhum software que comunicava com entidades externas se encontrava coberto por testes automáticos, o que representava um problema. Assim, este aspeto revelou-se muito importante no contexto desta dissertação.

Para reduzir as dependências, começou-se por tentar perceber de que tipo de dependências se verificavam. Assim, verificou-se que as dependências dizem respeito, em grande parte, a comunicações das aplicações com sistemas externos – através, por exemplo, de pedidos *SOAP*, *REST*, *FTP* e *SMTP* -, comunicações com bases de dados, preparação de ficheiros de configuração e leitura/escrita de ficheiros no sistema operativo. Notaram-se ainda alguns casos onde não se testavam variâncias do software devido ao ambiente no qual os testes automáticos eram executados, não havendo controlo de variáveis dinâmicas, como por exemplo a hora atual.

De modo a reduzir as dependências referidas, fez-se uma análise de como se poderia, de alguma forma, controlar o ambiente no qual os testes automáticos são executados, removendo as dependências indesejadas. Depois de um estudo, percebeu-se que isto poderia ser feito recorrendo à aplicação de ferramentas que permitam a aplicação de *Stubs*, *Mocks*, *Spies* e *Dummies* (Doglio, 2018).

De seguida, investigou-se que bibliotecas(s) se poderiam utilizar, em *NodeJs*, para responder a esta necessidade. Encontraram-se várias bibliotecas que possibilitavam a aplicação destes conceitos, no entanto a maior parte delas eram relativamente limitadas, tendo em conta que apenas cobriam parte das necessidades, como por exemplo a biblioteca *nock*, que apenas se encontrava preparada para simular pedidos HTTP (*Nock - Npm*, n.d.). Depois de algum estudo, encontrou-se uma biblioteca, *SinonJs*, que permite aplicar todos os conceitos previamente referidos, tendo esta biblioteca uma forte utilização por parte da comunidade desenvolvedora, contando com perto de cinco milhões de transferências semanais no repositório *npm* e tendo uma manutenção recorrente por parte da sua equipa de desenvolvimento/manutenção (*API Documentation - Sinon.JS*, n.d.; *Sinon - Npm*, n.d.).

A seleção de uma biblioteca única para suprimir as dependências externas foi tida em consideração, de modo a simplificar o processo de adaptação da equipa de desenvolvimento e de modo a facilitar a aplicação destes conceitos.

Apesar de útil, a utilização destes conceitos, no contexto da empresa, deve seguir algumas regras, tendo em conta que o ambicionado é apenas suprimir dependências externas e não deixar de testar corretamente as diferentes componentes do software (substituindo funções fulcrais por comportamento controlado).

Concomitantemente, definiu-se um conjunto de regras de quando aplicar e não aplicar estes conceitos. Adicionalmente, colocaram-se ainda algumas dicas/sugestões na componente de apoio à tomada de decisão, para suprimir eventuais aspetos que possam ser variáveis e dependentes do contexto onde se encontra inseridos.

Definiu-se como regra que todas as comunicações com sistemas externos, recorrendo, por exemplo, a protocolos como REST, SOAP, FTP e SMTP devem ver simuladas, como se pode ver na Figura 29.

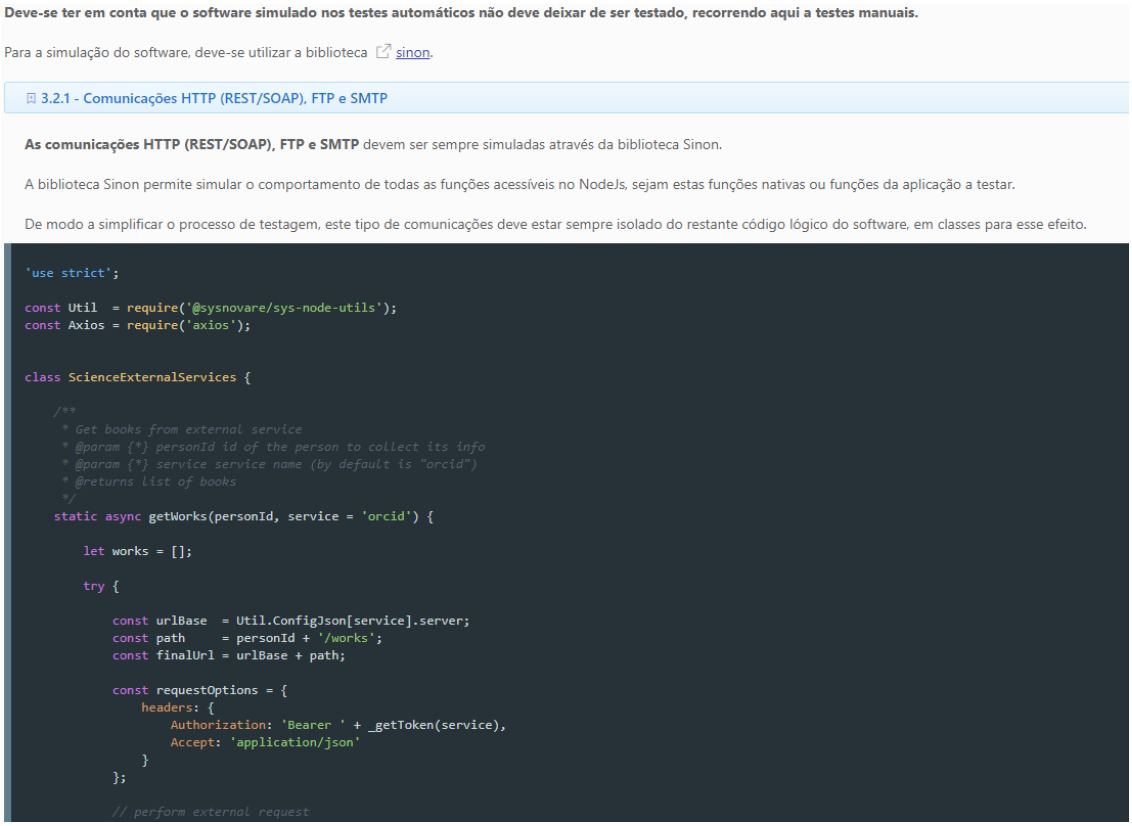

Figura 29 – Excerto do guia onde é explicado que certas comunicações devem ser sempre simuladas nos testes automáticos.

Note-se que é ainda explicado que as comunicações com sistemas externos devem estar separadas logicamente no código, de modo que a simulação do seu comportamento afete o mínimo possível a cobertura do software, respeitando ainda o *Single Responsibility Principle* (Ingeno, 2018; Robert C. Martin, 2009).

Relativamente à comunicação com as bases de dados internas da empresa, decidiu-se, como regra, não simular o seu comportamento, por um conjunto de razões que será de seguida apresentado:

- Os testes automáticos já realizados integram as várias bases de dados, havendo já muito trabalho realizado para preparar o acesso e comunicação com as mesmas, não sendo proveitoso, no momento do projeto, modificar todas as estruturas já prontas;
- Existe ainda muito código desenvolvido na linguagem de base de dados, sendo esse código, como por exemplo procedimentos e funções, reutilizado nalgumas situações pelas aplicações NodeJs. Assim, é proveitoso manter esta integração nos testes automáticos, permitindo, de certa forma, testar indiretamente o software anteriormente desenvolvido na tecnologia que se encontra a ser migrada;
- Nas comunicações com a base de dados, são aplicadas *queries* através do código em *NodeJs*, sendo importante testar que as *queries* estão bem elaboradas para comunicar com a base de dados em questão.

No entanto, ficou em nota quem no futuro a simulação da comunicação com as bases de dados internas pode vir a ser uma solução, tendo que ser sempre avaliado com muito cuidado a situação onde se poderia aplicar.

Relativamente à configuração de ficheiros para a execução dos testes automáticos, definiu-se como regra que estas configurações devem ser realizadas no arranque dos testes automáticos, através da execução de um script que carrega em memória as configurações pretendidas, tendo um comportamento homogéneo ao longo do tempo.

Outros dados que sejam necessários para a execução dos testes automáticos, mas não sejam importantes para a validação dos mesmos devem ser, em regra, injetados. Este tipo de situações acontece, por exemplo, no que diz respeito às credenciais de um utilizador. Toda a componente de autenticação e validação é testada na biblioteca respetiva, não sendo responsabilidade da aplicação que a incorpora testar esses aspetos. Assim, esse tipo de dados deve ser injetado no próprio do teste, conforme apresentado no guia, como se pode verificar na Figura 30.

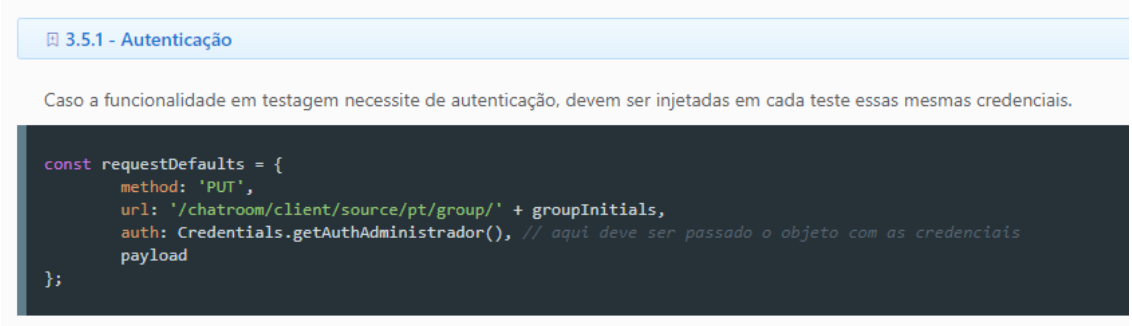

Figura 30 – Excerto do guia onde é explicado que as credenciais de utilizador devem ser injetadas, aquando da execução de testes automáticos.

Existem ainda situações onde a utilização, ou não, da biblioteca para simulação de comportamentos está dependente de uma determinada situação ou contexto específico, como por exemplo:

- Simular o comportamento de código lógico desenvolvido, por exemplo para centralizar o foco de testagem num determinado componente específico – simular, por exemplo, a leitura de um ficheiro do sistema se constituir um ponto menor de uma determinada funcionalidade;
- Controlar variáveis dinâmicas como por exemplo a data e hora do sistema;

No guia, são explicadas algumas situações onde isto pode acontecer, dando, em paralelo, potenciais soluções para colmatar essas mesmas situações, como se pode verificar na Figura 31.

Figura 31 – Exemplo de simulação de fatores dinâmicos, com enfâse na simulação de uma determinada data.

5.2.1.17 Regras definidas pelo chefe da equipa de desenvolvimento

Para além dos aspetos resultantes do estudo efetuado nesta dissertação, existe um conjunto de normas/regras que foram colocadas no guia, a pedido do chefe da equipa de desenvolvimento da empresa, de modo a garantir o seu cumprimento. Nesse conjunto de informação estão incluídas normas como:

- Utilizar como ferramenta de testagem automática a ferramenta já utilizada no início da dissertação, *HapiLab* (Nguyen et al., 2015). Esta decisão deriva do facto de ter sido conduzido previamente um estudo no que diz respeito à escolha desta mesma tecnologia;
- A cobertura do software desenvolvido não deve ser menor que 95%. Este valor foi sendo apresentado ao longo da dissertação e é de facto uma referência. Pretende-se que com o estabelecimento deste valor o software tenha sempre uma cobertura elevada, abrindo espaço para eventuais questões que possam carecer de um maior custo de testagem, dando por isso 5% de folga. Adicionalmente, e como referido anteriormente, havendo um nível de cobertura entendido como elevado, existe uma maior potencialidade de grande parte de casos de teste de cada caso de uso serem cobertos;
- Devem ser executados os testes automáticos já existentes num projeto antes de iniciar desenvolvimentos no mesmo, de modo a despistar eventuais erros que possam, ou não, ser causados por esses novos desenvolvimentos;
- Não se deve dar como finalizado um determinado desenvolvimento de um caso de uso, sem antes testar o mesmo recorrendo a testes automáticos.
- Todos os casos de uso incluídos numa versão do software devem estar testados através de testes automáticos.

As regras aqui definidas, nesta última secção, são entendidas como essenciais pelo autor e pelo chefe da equipa de desenvolvimento para sensibilizar e responsabilizar os desenvolvedores

naquilo que é o processo de desenvolvimento e testagem do software, evitando se possível alguns comportamentos que caiam fora do esperado.

5.2.2 Ferramentas de apoio à codificação

Inicialmente, idealizava-se a inclusão no Guia de apenas ferramentas de validação/verificação do software desenvolvido, de modo que este fosse monitorizado em momento de desenvolvimento. No entanto, no decorrer da dissertação, percebeu-se que outras ferramentas poderiam contribuir para uma maior testabilidade do software, tendo sido por isso algumas delas também incluídas na solução.

Da mesma forma, não se pretendia que as ferramentas apresentadas no guia fossem de utilização obrigatória – pensava-se que seriam apenas um complemento, de utilização arbitrária por parte de cada desenvolvedor. No entanto, depois de um debate com o chefe da equipa de desenvolvimento, percebeu-se que algumas das ferramentas recolhidas poderiam ter também elas um caráter de utilização obrigatória, de modo a "controlar" o software de cada pessoa, obrigando, de certa forma, a seguir um conjunto de regras "impostas".

Todas as ferramentas presentes no guia foram explicadas e demonstradas aos constituintes da equipa de desenvolvimento.

De seguida, serão apresentadas as ferramentas de apoio à codificação incorporadas no guia, sendo explicada a sua relevância e clarificado o seu caráter (facultativo/obrigatório). No guia, estas ferramentas encontram-se divididas em duas secções: uma com as ferramentas de utilização obrigatória e outra com as ferramentas de utilização opcional.

5.2.2.1 Debugger

Já foi referenciado anteriormente o caráter obrigatório da utilização de uma ferramenta de depuração, entendendo-se que esta potencia a observabilidade e controlabilidade do software, tendo sido não só sugerida uma ferramenta para este efeito, bem como explicada a sua configuração (ver 5.2.1.15).

5.2.2.2 JSDoc

A falta de documentação do software foi apontada pelos desenvolvedores como um aspeto que poderá afetar negativamente a testabilidade do software. Para além disso, tendo em ainda em consideração a análise realizada acerca dos fatores que influenciam a testabilidade, percebeu-se que a existência de documentação adequada pode potenciar o aumento da testabilidade.

Assim, fez-se uma pesquisa acerca de que ferramentas poderiam ser incorporadas na empresa, de modo a colmatar esta necessidade, não obrigando a um esforço acrescido.

Uma das ferramentas indicadas no guia é o *JSDoc*. Esta ferramenta permite a geração de documentação do software, tipicamente em formato HTML, recorrendo para tal aos comentários presentes no código fonte, à semelhança do que acontece por exemplo com *Javadoc* ou *phpDocumentor* (*Use JSDoc*, n.d.). A utilização desta ferramenta é de caráter

obrigatório, devendo os desenvolvedores ter o cuidado de escrever comentários que possibilitem uma geração documental coerente – o facto de ter um caráter obrigatório, poderá potenciar mais a escrita de comentários, por parte dos desenvolvedores, sendo que são estes comentários que alimentam a informação apresentada na documentação gerada. Na Figura 32 pode-se ver um excerto do resultado da geração da documentação.

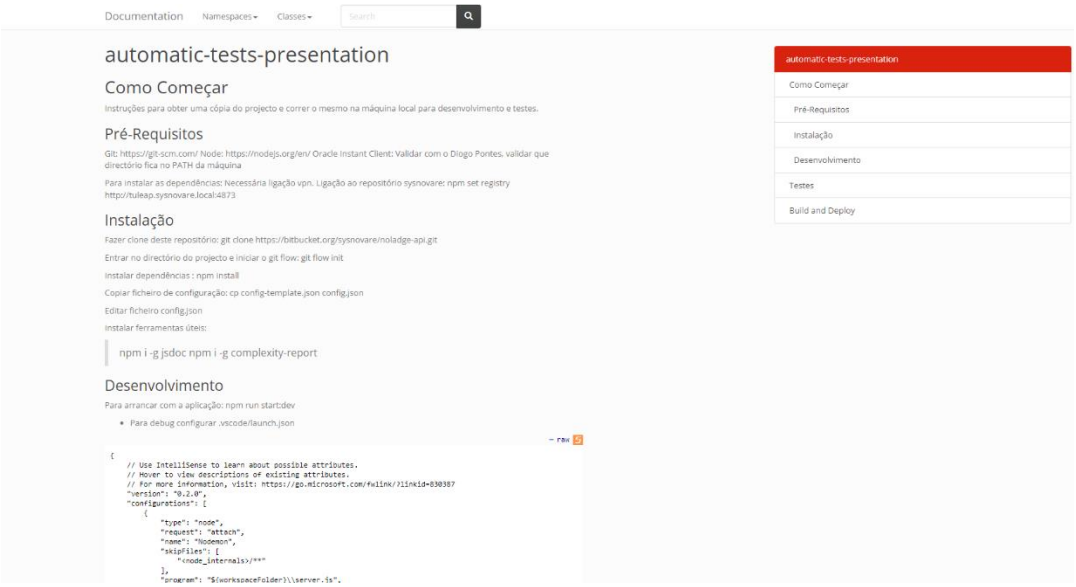

Figura 32 – Excerto do relatório *JSDoc* gerado para um determinado software.

Com a utilização do *JSDoc* é possível ter, de forma automática, documentação inerente a um software, com um nível detalhado de granularidade.

5.2.2.3 Swagger

Incorporou-se ainda no guia o *Swagger*, que é uma *framework* que permite especificar as diferentes interfaces de cada aplicação, permitindo um maior acompanhamento da evolução e estruturação da mesma (*API Documentation*, n.d.). Esta documentação pode ser disponibilizada em vários formatos e pode facilitar processos de integração com entidades externas ou mesmo processos internos.

A utilização do *Swagger* é também ela obrigatória, tendo-se definido a utilização da biblioteca *hapi-swagger* (*Hapi-Swagger - Npm*, n.d.). Tendo a utilização desta biblioteca devidamente configurada, a documentação gerada pelo *Swagger* pode ser consultada através de um endereço da própria aplicação, como se pode verificar na Figura 33.

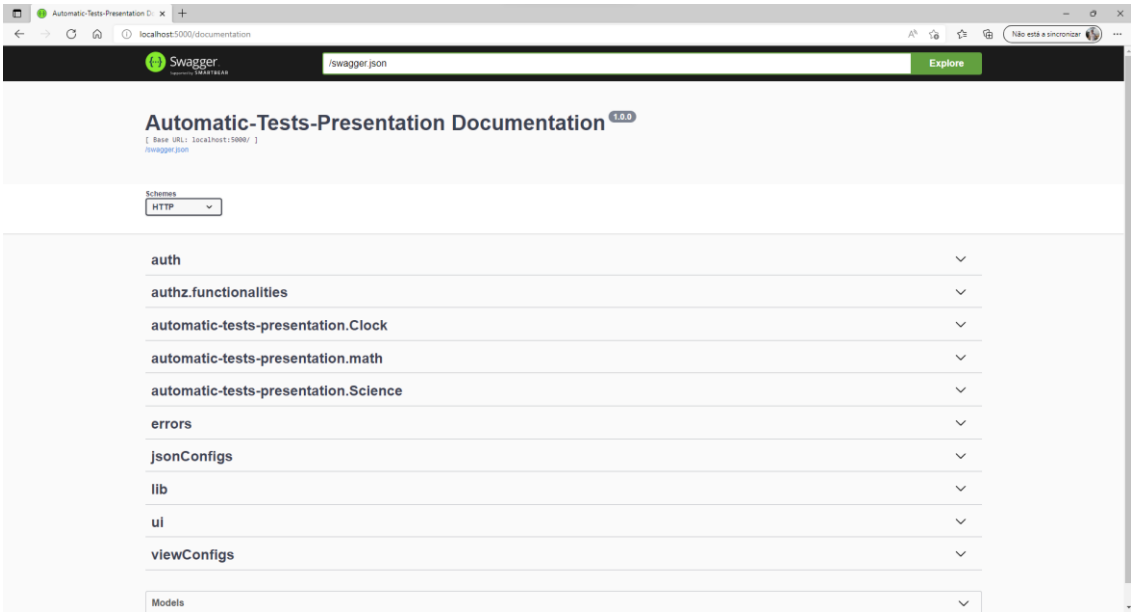

Figura 33 – Exemplo da informação apresentada pelo Swagger.

É de salientar que apenas as rotas pretendidas serão expostas através desta ferramenta, sendo possível ocultar informação mais sensível (*Hapi-Swagger - Npm*, n.d.).

5.2.2.4 Arkit

Esta ferramenta permite gerar diagramas, em vários formatos, como *SVG*, *PNG* ou *PlantUML*, que representam os vários componentes de uma aplicação, apresentando as dependências existentes entre classes e a forma como estas se encontram organizadas em cada diretório (*Arkit - Npm*, n.d.). Um exemplo de um diagrama gerado por esta ferramenta pode ser visto na Figura 34.

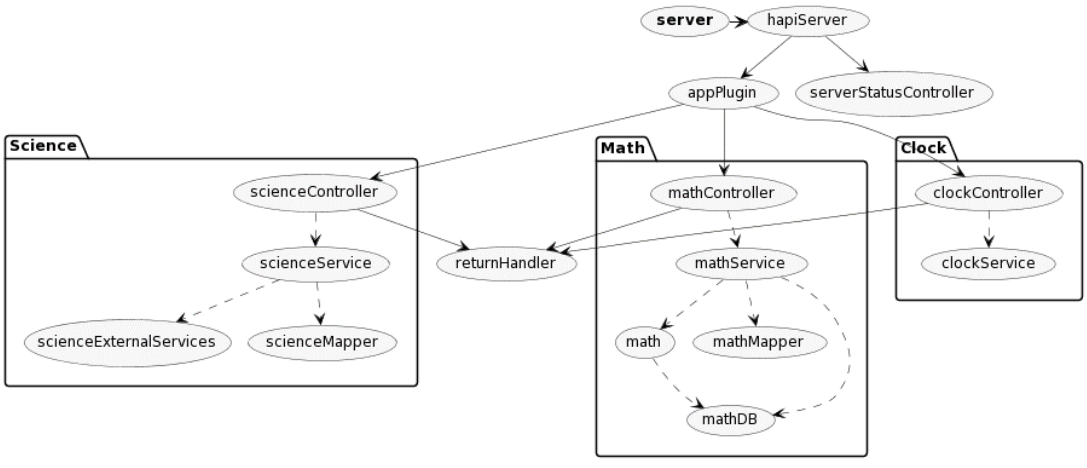

Figura 34 – Diagrama gerado pelo *Arkit* para projeto exemplificativo.

Os diagramas são gerados tendo como pilar uma configuração especificada, podendo esta configuração contemplar, inclusivamente, as bibliotecas internas utilizadas.

Havendo pouca documentação do software na empresa, esta é uma ferramenta útil para fazer uma geração relativamente automática e rápida de documentação, podendo ainda haver reutilização das configurações utilizadas para a geração destes mesmos diagramas. Entende-se que esta documentação, apesar de não seguir, segundo se apurou, nenhuma notação declarada, como acontece, por exemplo, no UML, permite um entendimento suficientemente rico e esclarecedor para ser considerada uma mais-valia.

A utilização desta ferramenta é de caráter opcional, tendo em conta que a sua utilização não afeta diretamente a qualidade do código, ao contrário do que acontece, por exemplo, com o JSDoc e com o Swagger, que obrigam os desenvolvedores a terem um cuidado acrescido na sua codificação, nomeadamente através da escrita de comentários e preparação de interfaces das APIs, para que estes dois tipos de documentação possam ser efetivamente gerados com informação relevante.

De modo a facilitar a utilização desta ferramenta, foi também colocado num guia um exemplo de uma configuração para a mesma.

5.2.2.5 ESLint

Como já explicado no estado da arte, verificou-se que o ESLint é uma ferramenta versátil e que responde às necessidades da empresa, no que diz respeito à possibilidade de análise da qualidade de código, nomeadamente em momento de codificação.

O facto de ser configurável por projeto, através de um ficheiro de configuração, é muito importante para a empresa, porque permite que diferentes aplicações possam ter diferentes regras, permitindo ainda que estas regras sejam modificáveis em paralelo com o código desenvolvido.

Adicionalmente, e ainda recorrendo ao seu aspeto de configurabilidade, no ESLint é possível especificar as regras desejadas, que podem, por exemplo, controlar o cumprimento de aspetos definidos no guia, como acontece na Figura 35, onde é despoletado um aviso de incumprimento da regra de comentação obrigatória de cada método – necessária para gerar a documentação com o JSDoc.

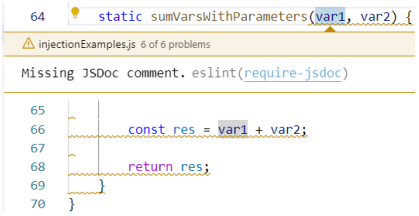

Figura 35 – Aviso apresentado no IDE pelo incumprimento de regra de comentário para JSDoc.

Tendo em conta o descrito, o ESLint foi definido no guia como de utilização obrigatória, de modo a ser possível controlar, em momento de desenvolvimento, o cumprimento de um conjunto de regras. De modo a contribuir para o trabalho dos desenvolvedores, colocou-se no guia uma breve explicação de como configurar esta ferramenta, como se pode verificar na Figura 36.

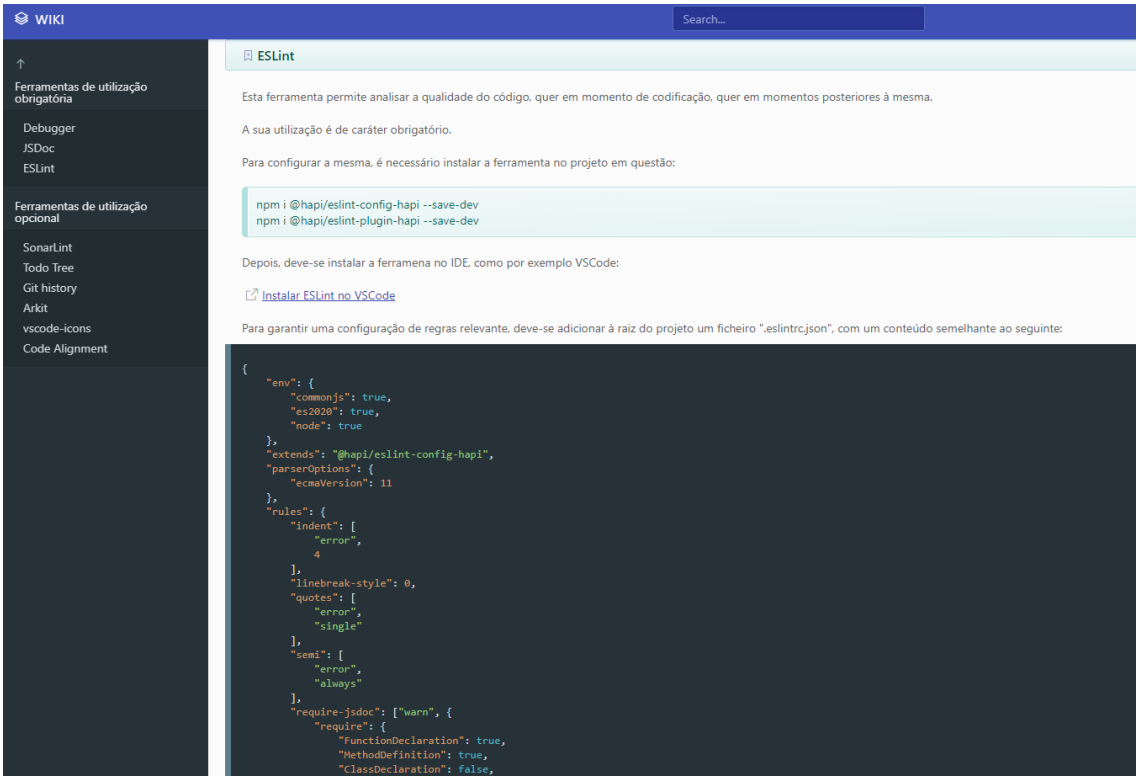

Figura 36 – Excerto do guia com parta da explicação da configuração do ESLint.

5.2.2.6 Todo Tree

Esta extensão para *Visual Studio Code* permite realçar no código um conjunto de palavras-chave, como TODO ou FIXME, fornecendo ainda uma vista de árvore da sua utilização ao longo de um determinado projeto (*Todo Tree - Visual Studio Marketplace*, n.d.). A sua utilização é de caráter opcional, não se considerando que a esta ferramenta seja fulcral para um aumento da testabilidade, podendo, alternativamente, os desenvolvedores realizar uma pesquisa pelas palavras desejadas, de forma manual.

5.2.2.7 Git History

Esta extensão para *Visual Studio Code* permite ter algum controlo sobre a evolução do software ao longo do tempo, sendo possível acompanhar, no próprio IDE, em momento de codificação, quando é que um ficheiro foi alterado, quem realizou essas alterações e porquê, por exemplo (*Git History - Visual Studio Marketplace*, n.d.).

Esta ferramenta é de utilização opcional. No entanto, deve-se salientar que poderá auxiliar o desenvolvedor em processos de codificação, permitindo que este possa entender mais facilmente o enquadramento de cada alteração num determinado ficheiro, podendo facilitar a compreensão do mesmo, podendo, em último caso, o desenvolvedor contactar com o outro desenvolvedor que codificou o código respetivo, de modo a suprimir alguma dúvida/necessidade, sendo, por estes motivos, aqui enquadrada.

5.2.2.8 Vscode-icons

O fator visual impacta na legibilidade do software, o que, por sua vez, impacta na testabilidade (Boswell et al., 2012). Assim, aspetos como uniformização de formas de programação, organização do software e alinhamento de código podem facilitar a leitura e compreensão de um determinado software, tendo em conta que este tipo de aspetos compõem um conjunto de "normas" que as tornam naturais.

Sabendo isto, recomenda-se (carácter opcional) aos desenvolvedores a utilização, nos seus IDE, de extensões que distingam os tipos de ficheiros e diretórios, tendo em conta um conjunto de aspetos – como o seu tipo e nome -, ambicionando, assim, que seja possível realizar uma leitura e avaliação global do projeto de forma mais célere, compacta e adequada. A ferramenta sugerida para o efeito é a *vscode-icons* (para *Visual Studio Code*), sendo uma extensão muito utilizada, contando já com mais de dez milhões de transferências (*Vscode-Icons - Visual Studio Marketplace*, n.d.).

5.2.2.9 Code Alignment

Como já foi referido anteriormente, o alinhamento lógico do código pode contribuir para uma maior legibilidade do software, potenciando assim a testabilidade (Boswell et al., 2012).

Assim, sugere-se (carácter opcional) a integração do *Code Alignment* no IDE em utilização, de modo que seja possível, alinhar por caracteres específicos e de uma só vez determinados excertos de código (*Code Alignment - Visual Studio Marketplace*, n.d.).

5.3 Conclusões

Desenvolveu-se uma solução (guia) que visa aumentar a testabilidade do software, através da uniformização de decisões realizadas pelos desenvolvedores, aquando da codificação, e através do estabelecimento de regras que devem ser seguidas pelos mesmos. A solução construída tem em consideração o conhecimento adquirido no estado da arte, contemplando conceitos que visam solucionar aspetos referidos pelos desenvolvedores e potenciar particularidades que aumentem a testabilidade do software. Tratando-se de uma solução que visa resolver vários aspetos afetos à testabilidade, pode não apresentar a melhor resposta para cada aspeto, mas esta decisão partiu da empresa, que apenas ambicionava resolver um grande conjunto de particularidades de forma satisfatória. A solução desenvolvida encontra-se disponível da Wiki da Sysnovare, tendo havido sessões de contextualização da mesma, com a equipa de desenvolvimento.

6 Avaliação

Neste capítulo é feita a avaliação da solução desenvolvida, aferindo-se se a mesma cumpre com os objetivos definidos inicialmente.

6.1 Metodologia de Avaliação

A avaliação da solução realizou.se, em grande parte, pelo autor e pelo chefe da equipa de desenvolvimento, tendo ainda existido um contributo da equipa de desenvolvimento da Sysnovare.

A avaliação possui três dimensões distintas: o guia construído, o grau de testabilidade e a opinião da equipa de desenvolvimento. Os aspetos avaliados nas três dimensões estão diretamente relacionados com os objetivos do projeto.

O guia e o grau de testabilidade foram avaliados pelo autor e pelo chefe da equipa de desenvolvimento da Sysnovare.

A avaliação do grau de testabilidade decorreu após um período de cerca de mês e meio onde o guia foi seguido por uma equipa de desenvolvimento num determinado projeto piloto. Assim, poder-se-á entender a adequação do guia ao problema apresentado.

Na avaliação, existiu um contributo da equipa de desenvolvimento, que foi responsável por responder a um questionário, de modo a entender as melhorias sentidas, relativamente à testabilidade, tendo sido através deste questionário averiguada a opinião da equipa de desenvolvimento. Este questionário (ver Anexo I) foi disponibilizado e analisado depois do período de aplicação do guia, como se tinha já verificado para o grau de testabilidade. Decidiu-se contemplar este aspeto como uma dimensão separada porque se percebeu, daquilo que foi o estudo realizado e até do questionário inicialmente colocado, que o programador/desenvolvedor é uma parte fulcral para a construção de um software com alta testabilidade, sabendo ainda que esta dissertação, apesar de ser potencialmente aplicável em contextos externos, não deixa de ter origem num problema específico de uma empresa e do seu grupo de desenvolvimento.

Com a finalização da avaliação realizada, foi possível verificar se existiram melhorias na testabilidade do software, tendo sido para isso definido um “valor esperado” de qualidade resultante apresentado pela ferramenta de avaliação a ser utilizada. Desta forma, foi possível verificar se a solução tratou os aspetos que de facto impactavam a testabilidade no software da Sysnovare.

6.2 QEF - Quantitative Evaluation Framework

Para avaliar a qualidade de uma solução, é necessário recorrer a meios que permitam obter uma medida que quantifique o grau de sucesso na persecução de uma determinada particularidade (Escudeiro & Bidarra, 2008). Uma solução pode ser avaliada tendo em consideração um conjunto de fatores. Cada fator pode ser subdividido num grupo de características. Existem características que são diretamente mensuráveis – apresentam métricas associadas - e características indiretamente mensuráveis – descritas e calculadas, eventualmente, recorrendo a fatores diretamente mensuráveis.

O *"Quantitative Evaluation Framework"* (QEF) é uma ferramenta de avaliação que permite avaliar a qualidade tendo em consideração uma série de dimensões, os seus fatores e os requisitos que constituem esses mesmos fatores (Escudeiro & Bidarra, 2008). Assim, para avaliar uma solução recorrendo ao QEF, é necessário perceber as dimensões associadas, os fatores que influenciam estas dimensões e que requisitos estão associados a cada fator. Adicionalmente, dentro de cada dimensão, é necessário pesar cada fator e cada requisito associado, de modo a evidenciar o impacto dos mesmos no resultado final.

Para a avaliação da solução foram levantados uma série de aspetos que permitem aferir a sua adequação ao problema, sendo grau de concretização da solução calculado através da aplicação do QEF, tendo em conta os objetivos propostos.

Caso a qualidade obtida seja maior ou igual a um valor arbitrário (definido pela empresa), pode-se concluir que a solução elaborada vai de encontro às necessidades do problema, tendo sido cobertos aspetos fundamentais para o aumento da testabilidade no contexto da empresa.

Tendo em conta que se visa avaliar o aumento testabilidade do software e recorrendo à metodologia de avaliação delineada, preparou-se o **QEF** para contemplar **três dimensões**: **testabilidade**, **guia** e **opinião da equipa de desenvolvimento**.

Na **primeira dimensão** são analisados **três fatores**:

- Dependências – onde o se analisam as dependências existentes no projeto piloto para processos de testagem (ver Anexo D);
- Cobertura – onde é averiguada a existência de testes adequados e o nível de cobertura (ver Anexo C);
- Métricas – onde se acompanha a evolução dos valores de um conjunto selecionado de métricas de testabilidade (ver Anexo E).

A Cobertura e as Dependências são avaliadas tendo em conta o delineado nos objetivos do projeto. O fator das métricas não era apresentado inicialmente nos objetivos do projeto, mas pelo estudo que se realizou percebeu-se que as métricas podem ser um forte indicador empírico da testabilidade do software, decidindo-se por isso, incluir um conjunto de métricas – SLOC, TLOC e WMC -, selecionado no estado da arte, ao qual se adicionou uma métrica afeta à previsão de erros (detalhada mais à frente em 6.3.4).

Já a **segunda dimensão**, ou seja, o guia, é constituída por **dois fatores**: Conteúdos do Guia, onde se averigua se o guia cobre todos os pontos pretendidos no que concerne à temática de testabilidade (ver Anexo G); Disponibilização do Guia: onde se analisa a estratégia de disponibilização do guia para a equipa de desenvolvimento, tendo em conta o explanado nos objetivos iniciais (ver Anexo H). Esta dimensão visa avaliar o cumprimento dos requisitos inicialmente colocados dobre o Guia (ver Tabela 1).

A primeira e segunda dimensão, de acordo com o já explicado, são avaliadas apenas pelo autor e pelo chefe da equipa de desenvolvimento da Sysnovare.

Na **terceira dimensão** – Opinião da Equipa de Desenvolvimento - averigua-se o entendimento dos desenvolvedores sobre o Guia realizado e a forma como este se adequa às suas necessidades, no que diz respeito ao aumento da testabilidade do software. Esta dimensão tem apenas um fator: Opinião (ver Anexo F). Para se obter a informação pretendida, disponibilizou-se um questionário (ver Anexo I).

Como já foi dito anteriormente, a avaliação da solução é realizada após um período de cerca de mês e meio, durante o qual a equipa de desenvolvimento aplicou o guia num projeto piloto, de modo que os dados afetos a duas das dimensões - testabilidade e opinião da equipa de desenvolvimento - tenham como pilar um período de experimentação que os sustente.

Note-se que os valores apresentados no QEF foram definidos pelo chefe da equipa de desenvolvimento da Sysnovare. A estruturação das várias dimensões, fatores e pesos relativos de requisitos pode ser consultada em anexo (ver Anexo J).

Os dados resultantes do QEF transmitem uma ideia global da adequação da solução para o problema colocado.

6.3 Projeto Piloto

Como já foi explicado, para a análise do impacto da utilização do guia na empresa, escolheu-se um projeto piloto para servir de referência para a avaliação da solução. A escolha do projeto piloto foi efetuada pelo chefe da equipa de desenvolvimento em conjunto com o autor.

Analisando todos os projetos possíveis, tendo em consideração o planeamento dos mesmos, envergadura e equipa de desenvolvimento, optou-se por definir o software de recursos humanos como projeto piloto. Este software encontra-se dividido em vários módulos.

A equipa de desenvolvimento deste projeto é constituída por quatro membros, não sendo nenhum deles o autor que desenvolveu o guia, o que pode fornecer uma ideia mais precisa do efeito do guia em desenvolvedores alheios à sua construção.

Sendo este o projeto que permitiu a averiguação de melhoria da testabilidade após a aplicação do guia, tornou-se relevante comparar não só a cobertura dos seus componentes, como

também outro tipo de métricas – previamente selecionadas no estado da arte antes e depois da aplicação do guia.

Os dados apurados que dizem respeito às métricas escolhidas, bem como aos dados de cobertura, serão utilizados na avaliação final.

De modo a facilitar a comparação entre os vários valores, colocar-se-á nas seguintes secções esta mesma informação.

6.3.1 Cobertura

Na Tabela 8 estão apresentados os níveis de cobertura da aplicação piloto, em momentos antecedentes e posteriores à aplicação do guia.

Tabela 8 – Cobertura, do projeto piloto, antes e depois da aplicação do guia.

Nome da Componente	Cobertura antes da aplicação do guia	Cobertura depois da aplicação do guia
rhsuite_api	71.62%	84.00%
rhsuite-external-services	74.71%	97.08%
rhsuite_xlsx_export	97.13%	97.13%
rhsuite-chatbot	71.19%	71.19%
Média Total	~ 78.66%	~ 87.35%

De todos componentes da aplicação piloto, dois deles tiveram alterações em termos de cobertura. Em ambos os casos, estas alterações foram positivas, tendo a percentagem de cobertura aumentado – havendo até uma das componentes a passar o valor de 95% de cobertura. Num contexto global, no que diz respeito à aplicação piloto, a cobertura aumentou de aproximadamente 79% para cerca de 87%.

6.3.2 Complexidade do software – WMC

Para a averiguação da complexidade do software, aferiu-se a média da complexidade de cada classe integrante das diversas componentes do projeto piloto.

Os dados recolhidos poderão ser consultados na Tabela 9.

Tabela 9 – Média da complexidade ciclomática média por classe, no projeto piloto, antes e depois da aplicação do guia.

Nome da Componente	Complexidade ciclomática média por método antes da aplicação do guia	Complexidade ciclomática média por método depois da aplicação do guia
rhsuite_api	~ 8.48	~ 7.31
rhsuite-external-services	~ 2.78	~ 1.83
rhsuite_xlsx_export	~ 1.67	~ 1.67
rhsuite-chatbot	~ 8.6	~ 8.6
Média Total	~ 5.38	~ 4.85

Como se pode ver, antes da aplicação do guia, a média de complexidade ciclomática nos diversos componentes era de aproximadamente 5.38, sendo o componente de *rhsuite_api* o que contava com uma maior complexidade. Depois da aplicação do guia, a média de complexidade ciclomática nos componentes diminuiu para aproximadamente 4.85, devido às diminuições verificadas nas componentes de *rhsuite_api* e *rhsuite-external-services*. É ainda de salientar que o *rhsuite-api* continua a ser a componente com maior complexidade, apesar da diminuição já observada.

6.3.3 Esforço de desenvolvimento vs Esforço de Testagem – SLOC vs TLOC

Para comparar o esforço ao processo de testagem em detrimento do processo de desenvolvimento, recolheu-se informação acerca do número de linhas de código afeto a cada um destes aspetos, de modo que fosse possível realizar algumas críticas. Para tal, recolheu-se o número de linhas médio de código fonte (SLOC) e de código de testagem (TLOC), no projeto piloto.

Tabela 10 – Comparação do esforço de desenvolvimento e testagem, através do número de linhas afetas a cada uma destas componentes.

Componente	Média de SLOC antes da aplicação do guia	Média de TLOC antes da aplicação do guia	Média de rácios entre médias antes da aplicação do guia	Média de SLOC depois da aplicação do guia	Média de TLOC depois da aplicação do guia	Média de rácios entre médias depois da aplicação do guia
rhsuite_api	203	196	~ 0.97	221	193	~ 0.87
rhsuite-external-services	128	~ 41	~ 0.32	137	104	~ 0.76
rhsuite_xlsx_export	42	59	~ 1.40	42	59	~ 1.40
rhsuite-chatbot	137	135	~ 0.99	137	135	~ 0.99
Média Total	~ 128	~ 108	~0.92	~ 134	~ 123	~ 1.01

Como já se verificou anteriormente, das quatro componentes que integram o projeto piloto, apenas duas sofreram alterações. Destas duas componentes, uma diminuiu e outra aumentou o rácio entre o número de linhas de testagem e o número de linhas de código fonte. Efetivamente, uma das componentes modificadas, *rhsuite_api*, apresentou uma melhoria neste aspeto. No entanto, a outra componente modificada, *rhsuite-external-services*, piorou. A razão da última componente referida ter piorado no rácio entre linhas de testagem e linhas de código fonte deve-se ao facto de terem sido adicionados muitos testes automáticos a esta componente (comparando com o inicialmente existente). Esta justificação pode ser comprovada com o aumento significativo da cobertura desta componente – de cerca de 75% para aproximadamente 97% (ver Tabela 8).

6.3.4 Previsão de erros

Sabendo que a origem deste problema se relaciona com a necessidade de detetar atempadamente erros, evitando que estes se manifestem aquando da utilização por parte do cliente final, decidiu-se incluir, no conjunto de métricas a ser avaliado, os dados afetos a uma estimativa de erros possíveis, sendo este valor suportado pela "estimativa de erros entregues de Halstead" (Govil, 2020).

De modo a comparar o número de erros previstos inicialmente existente com o conjeturado no final da aplicação do guia, recolheu-se a média (por classe) de erros previstos, para cada projeto antes e depois da aplicação do guia. Estes valores podem ser vistos na Tabela 11.

Tabela 11 – Erros previstos no projeto piloto, antes e depois da aplicação do guia.

Componente	Média de Erros previstos antes da aplicação do guia	Média de Erros previstos depois da aplicação do guia
rhsuite_api	~1.71	~1.64
rhsuite-external-services	~1.33	~0.81
rhsuite_xlsx_export	~0.25	~0.25
rhsuite-chatbot	~1.05	~1.05
Média Total	~1.09	~0.94

Como se pode verificar, a quantidade média de erros previstos diminuiu para as duas componentes alteradas, *rhsuite_api* e *rhsuite-external-services*, fazendo com que a média de erros previstos na aplicação piloto diminuísse. Esta diminuição é positiva e vai de encontro aos objetivos da empresa.

6.4 Testabilidade

A testabilidade (referida anteriormente como "grau de testabilidade") é uma das dimensões avaliadas através do QEF.

Tal com explicado anteriormente, a testabilidade foi validada através de três fatores: "Dependências", "Cobertura" e "Métricas". Para cada um dos requisitos integrantes de cada fator atribuíram-se os valores previamente escalonados para cada grau de cumprimento destes mesmos requisitos – definidos pelo chefe da equipa de desenvolvimento. Nesta análise utilizar-se-á sempre como referência a aplicação piloto.

No que concerne ao primeiro fator – dependências –, de um conjunto de seis requisitos, apenas um não foi cumprido e outro foi parcialmente atingido, tendo os restantes sido alcançados com sucesso, como se pode ver na Figura 37.

Dimension	Testabilidade			
Factor	Dependências			
		Wfk - Fullfilment (%)		
Requirement	Metric Evaluation	0	50	100
FTD001 - Sistemas Externos	Para a Testagem, o software está dependente de sistemas externos?	Sim.		Não.
FTC002 - Configuração	Para a Testagem, o software requer configuração particular (diferente da existente para a sua execução normal) de ficheiros?	Sim.	Sim, mas o existe um template do ficheiro de configuração no repositório para esse efeito.	Não.
FTC003 - Simulação software	Existe a possibilidade de simular o comportamento do software para suprimir a carência de sistemas externos aquando da testagem?	Não.		Sim.
FTC004 - Isolabilidade	Existem casos na escrita de Testes Automáticos onde é necessário tratar/preparar entidades terceiras para testar uma determinada entidade?	Sim.		Não.
FTC005 - Dependencia de Sistemas Operativos	A execução dos Testes está dependente da máquina/sistema operativo nos quais estes são executados?	Sim.		Não.
FTC006 - Dependencia entre Testes	Existe dependência entre testes, nomeadamente na ordem pela qual estes são executados?	Sim.	Sim, mas não se verifica para todo o software.	Não.

Figura 37 - Requisitos afetos ao fator de "dependências".

Deve-se explicar o não atingimento completo de todos os requisitos do fator de "dependências". O primeiro requisito em falta é o FTC004, tendo em conta que existe, no projeto piloto, um conjunto de testes que necessita de uma preparação de entidades antes da testagem de outras entidades, devido a relações de dependência de dados e à insuficiente preparação do *schema* de base de dados. Este requisito encontra-se diretamente relacionado com o FTC006, tendo em conta que existem testes que devem ser realizados antes de outros, precisamente devido à necessidade de sediar a base de dados com todos os dados necessários.

No que diz respeito ao segundo fator enunciado – cobertura -, de um conjunto de seis requisitos, um foi cumprido com sucesso, quatro foram parcialmente cumpridos e um não foi atingido, como se pode ver na Figura 38.

Dimension	Testabilidade			
Factor	Cobertura			
		Wfk - Fullfilment (%)		
Requirement	Metric Evaluation	0	50	100
FTC001 - Testes Automáticos	Existem testes automáticos para cada componente da aplicação piloto?	Não.		Sim.
FTC002 - Casos de Uso	Existem testes automáticos que cubram todos casos de uso da aplicação piloto?	Não.		Sim.
FTC003 - Percentagem de Cobertura	Nivel médio de cobertura da aplicação piloto - média de cobertura de todos os componentes da aplicação piloto.	Menos de 80%.	Entre 80% a 95%.	Mais de 95%.
FTC004 - Diferentes Fluxos	São testados diferentes fluxos para os casos de uso da aplicação piloto?	Não.	Sim, mas não para todos os casos de uso.	Sim.
FTC005 - Casos de erro	São forçados casos de erro ao software desenvolvido?	Não.	Sim, mas não para todos os casos de uso.	Sim.
FTC006 - Casos limite	São forçados casos limite para funcionalidades específicas do software desenvolvido?	Não.	Sim, mas não para todos os casos de uso.	Sim.

Figura 38 – Requisitos afetos ao fator de "cobertura".

Torna-se relevante clarificar o não cumprimento de parte dos requisitos do fator de “cobertura”. O requisito FTC002 não foi cumprido porque não existem nas componentes *rhsuite_api* e *rhsuite-chatbot* testes automáticos para todos os casos de uso. O requisito FTC003 não foi atingido completamente porque a cobertura média da aplicação piloto, assim como aferido anteriormente (ver 6.3.1), apresenta um valor de aproximadamente 87% - o que já foi uma melhoria significativa de quase 10%, num período de tempo curto, mas continua a ser inferior ao valor de 95% pretendido pela empresa. Os requisitos FTC004, FTC005 e FTC006 não foram cumpridos na íntegra porque não existem, nas componentes de *rhsuite_api* e *rhsuite-chatbot*, testes para diferentes fluxos, casos de erro e casos limite para todos os casos de uso existentes. No fator de cobertura, apesar de apenas um requisito ter sido cumprido na íntegra, é de realçar que se verificaram melhorias em todos os aspetos afetos aos requisitos colocados, para as duas componentes que sofreram alterações durante o período no qual esta avaliação se baseia.

Relativamente ao terceiro fator referido – métricas -, de um grupo de três requisitos, dois foram cumpridos com sucesso e um foi parcialmente atingido, como se pode ver na Figura 39.

Dimension	Testabilidade			
Factor	Métricas			
		Wfk - Fullfilment (%)		
Requirement	Metric Evaluation	0	50	100
FTM001 - SLOC vs TLOC	O rácio entre o número de linhas de codigo de testagem e o número de linhas do código fonte diminuiu, no projeto piloto?	Não.	Sim, nalguns dos componentes com alterações.	Sim, em todos os componentes com alterações.
FTM002 -WMC	Existiu diminuição da complexidade ciclomática média das classes que constituem os componentes do projeto piloto?	Não.	Sim, nalguns dos componentes com alterações.	Sim, em todos os componentes com alterações.
FTM003 - Previsão de erros (Halstead)	Houve uma diminuição dos erros previstos, no projeto piloto?	Não.	Sim, nalguns dos componentes com alterações.	Sim, em todos os componentes com alterações.

Figura 39 – Requisitos afetos ao fator de “métricas”.

O requisito em falta para o fator de “métricas” é o FTM001. Este requisito encontra-se diretamente relacionado com o rácio entre o número de linhas de código de testagem e o número de linhas de código fonte. Os valores obtidos neste requisito estão relacionados com o facto de uma das componentes ter aumentado significativamente a sua cobertura, como já explicado anteriormente (ver 6.3.3).

6.5 Guia

O guia é uma das dimensões avaliadas através do QEF.

Como já foi explicado anteriormente, sobre a dimensão do guia, foram analisados dois fatores fundamentais: “conteúdos do guia” e “disponibilização do guia”.

Para cada um dos requisitos integrantes de cada fator atribuíram-se os valores previamente escalonados para cada grau de cumprimento destes mesmos requisitos – definidos pelo chefe da equipa de desenvolvimento. Na Figura 40 e Figura 41 estão apresentados os requisitos de cada fator, com o valor atribuído selecionado com um círculo.

		Wfk - Fullfilment (%)		
Requirement	Metric Evaluation	0	50	100
FPCP001 - Regras	O Guia apresenta regras que devem ser (sempre que possível) seguidas para aumentar a Testabilidade.	Não.		Sim.
FPCP002 - Apoio à decisão	O Guia apoia a decisão do desenvolvedor nalgumas questões ambíguas.	Não.		Sim.
FPCP003 - Autoexplicativo	O Guia é autoexplicativo, não havendo necessidade de consultar informações externas para entender o nele perpetuado.	Não.	Sim, mas apresenta sugestão de informação adicional a ser consultada, de modo culmatar.	Sim.
FPCP004 - Sugere Ferramentas	O Guia sugere a utilização de ferramentas que potenciem a melhoria da qualidade de código, nomeadamente da Testabilidade.	Não.		Sim.
FPCP005 - Para o desenvolvedor	O Guia tem em conta as dificuldades reportadas pela equipa de desenvolvimento, apresentando soluções para as mesmas.	Não.	Sim, para algumas das dificuldades apresentadas.	Sim, para todas as dificuldades apresentadas.

Figura 40 – Requisitos afetos ao fator de "conteúdos do guia".

		Wfk - Fullfilment (%)	
Requirement	Metric Evaluation	0	100
FPDP001 - Disponibilidade	O Guia está acessível a qualquer membro da equipa de desenvolvimento.	Não.	Sim.
FPDP002 - Modificabilidade	O Guia é editável, de modo a potenciar a sua evolução ao longo do tempo.	Não.	Sim.
FPDP003 - Adequado ao Contexto da Empresa	O Guia está integrado nas práticas atuais da empresa, de modo a potenciar a sua utilização.	Não.	Sim.

Figura 41 – Requisitos afetos ao fator de "disponibilização do guia".

Como se pode verificar na Figura 40, dois dos requisitos colocados não foram cumpridos na íntegra para o fator de "conteúdos do guia". O FPCP003 não foi cumprido na sua integridade porque ao longo do guia existem algumas referências a documentação externa, que podem ajudar os membros da equipa de desenvolvimento a entender mais profundamente alguns dos temas e ferramentas apresentados no guia. O FPCP005 não se encontra também ele no valor máximo porque alguns dos aspetos apresentados pelos membros da equipa de desenvolvimento no primeiro questionário de resposta aberta não foram resolvidos.

Para o fator de "disponibilização do guia" cumpriram-se com todos os requisitos propostos.

6.6 Opinião da Equipa de Desenvolvimento

A Opinião da Equipa de Desenvolvimento é uma das dimensões avaliadas no QEF.

Relativamente à Opinião da Equipa de Desenvolvimento, tentou-se perceber se os membros da equipa sentiram melhorias no que diz respeito à testabilidade do software, tendo em especial consideração os aspetos que colocaram no questionário realizado no início do projeto. Desta forma, aplicou-se um segundo questionário (ver Anexo I) aos desenvolvedores que inicialmente responderam ao primeiro inquérito.

As perguntas colocadas no questionário final tratam variáveis qualitativas (categóricas) nominais – que não apresentam ordem natural entre as suas categorias (Meireles, n.d.). Na Tabela 12 são apresentados os valores obtidos para cada variável nas perguntas efetuadas.

Tabela 12 – Frequências relativas para respostas a perguntas aplicadas no questionário final.

Pergunta	Frequência relativa por resposta (%)		
	Sim	Não	Não aplicável
1	44.4%	0%	55.6%
2	100%	0%	----
3	77.8%	22.2%	----
4	100%	0%	----
5	100%	0%	----
6	55.6%	0%	44.4%
7	88.9%	11.1%	----
8	100%	0%	----

Cada uma das perguntas colocadas no questionário está diretamente relacionada com um requisito da Opinião da Equipa de Desenvolvimento. Na Figura 42, pode-se verificar cada um dos requisitos e o respetivo valor atribuído.

Dimension	Equipa Desenvolvimento			
Factor	Opinião			
		Wfk - Fullfilment (%)		
Requirement	Metric Evaluation	0	50	100
FTOED001 - Sugestões	Os desenvolvedores sentem que as suas sugestões iniciais foram tidas em consideração.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.
FTOED002 - Legibilidade/Compreensibilidade	Os desenvolvedores consideram o guia benéfico para um melhor entendimento do software desenvolvido.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.
FTED003 - Esforço	Os desenvolvedores sentem menos esforço para testar o software.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.
FTOED004 - Frequência de Testagem	Os desenvolvedores realizam testes ao software - nomeadamente testes automaticos.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.
FTOED005 - Dificuldades	Os desenvolvedores sentem que as suas maiores dificuldades associadas à testagem de software foram suprimidas.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.
FTOED006 - Testes de software Desenvolvido por Outrem	Os desenvolvedores entem que o esforço para testar software desenvolvido por terceiros diminuiu.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.
FTOED007 - Testar nova funcionalidade	Os desenvolvedores sentem um menor custo associado à adição e testagem de uma nova funcionalidade.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.
FTOED008 - Testar funcionalidade alterada	Os desenvolvedores sentem um menor custo associado à alteração e testagem de uma funcionalidade já existente	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.

Figura 42 – Requisitos afetos ao fator de "Opinião da Equipa de Desenvolvimento".

O cumprimento parcial do requisito FTOED001 pode dever-se ao facto de nem todas as sugestões terem sido aplicadas no guia, podendo ainda dever-se ao facto de os desenvolvedores não se recordarem ao certo do que responderam no primeiro questionário. O atingimento parcial do requisito FTOED006 pode estar relacionado com o curto período de aplicação e habituação ao guia, não tendo ainda existido, eventualmente, contacto de alguns desenvolvedores com software desenvolvido por terceiros, onde já se verificassem os conceitos padronizados no guia.

6.7 Limitações

São algumas, as limitações que podem ser apontadas à avaliação efetuada. Estas limitações serão seguidamente apresentadas.

A primeira limitação relaciona-se com o período de tempo do estudo. Considera-se este período demasiado reduzido. Desta forma, entende-se que um período de tempo superior poderia possibilitar o atingimento resultados que representassem mais corretamente a qualidade da solução desenvolvida.

A segunda limitação a enunciar prende-se com os aspetos considerados na avaliação. Depois de um estudo sobre o tema, entendeu-se que o QEF cumpria com o pretendido, permitindo uma avaliação balanceada, tendo em consideração um conjunto de aspetos. A escolha destes aspetos esteve altamente relacionada com o considerado relevante para a organização que promoveu o projeto. No entanto, percebe-se que as dimensões e fatores utilizados nesta mesma avaliação condicionam o nível de qualidade apresentado. Assim, existe a consciência de que o nível de qualidade da solução poderia diferir, caso tivessem sido escolhidos diferentes dimensões/fatores/requisitos.

A terceira limitação a apresentar consiste na validade de algumas das respostas apresentadas pelos membros da equipa de desenvolvimento, no questionário aplicado aquando da avaliação da solução. Estando algumas das perguntas relacionadas com respostas manifestadas num questionário anterior, e tendo em consideração o fosso temporal existente entre a disponibilização dos dois questionários, algumas das respostas podem não representar a realidade.

6.8 Conclusões

Foram tidas em consideração três dimensões para avaliar a solução apresentada: testabilidade, guia e opinião da equipa de desenvolvimento. Cada uma destas dimensões é constituída por um conjunto de fatores e respetivos requisitos. Para se avaliar a solução, utilizou-se como referência um projeto piloto. Neste projeto piloto, todas as dimensões apresentaram resultados considerados satisfatórios, tendo sido na dimensão da testabilidade onde se verificou uma menor concretização, o que pode ser explicado pelo curto período de aplicação do guia, face à dimensão já existente do projeto piloto. Utilizou-se o QEF na avaliação efetuada, tendo-se obtido um valor de qualidade final de 86%, para a solução. Este valor é satisfatório e passa o mínimo estabelecido pela empresa. No entanto, considera-se que poderia ainda ser melhor, se o período de estudo fosse maior.

7 Conclusão

Nesta secção analisa-se, de forma global e sintetizada, todo o trabalho efetuado. Para tal, apresentam-se os objetivos alcançados, as limitações verificadas e o trabalho futuro, realizando-se, ainda, uma apreciação final do planeamento, organização, solução e aprendizagens pessoais.

7.1 Objetivos Alcançados

Com esta dissertação pretendeu-se desenvolver uma solução para a problemática de uma baixa testabilidade do software desenvolvido pela Sysnovare.

O problema colocado foi devidamente contextualizado e explorado. Para tal, começou-se por introduzir o mesmo, passando-se, posteriormente, para um estudo mais detalhado, no estado da arte, sobre o tema e sobre a situação interna inicialmente verificada na organização. No estudo referido, analisaram-se vários aspetos sobre a testabilidade, como os fatores que a influenciam, as métricas relacionadas, tendo-se ainda aferido algumas ferramentas que permitem controlar aspetos relacionados com a mesma. Relativamente à situação interna da organização, conseguiu-se perceber quais os processos seguidos e quais as práticas padronizadas, tendo-se ainda recolhido uma opinião da equipa de desenvolvimento acerca da temática em estudo, de modo que a solução desenvolvida pudesse ter em conta esta informação.

Realizou-se uma análise de valor, na qual se entendeu a potencialidade deste projeto, especialmente no que diz respeito à melhoria da qualidade do software desenvolvido pela Sysnovare. Esta melhoria do software pode potenciar a melhoria da sua reputação junto dos seus clientes atuais e potenciais, podendo aumentar a competitividade da empresa no mercado onde atua.

Desenhou-se a solução a desenvolver, tendo em consideração os requisitos e objetivos colocados pela organização. Construiu-se uma solução que respeitou esse mesmo desenho efetuado e que teve em atenção a informação adquirida no estado da arte. Desta forma, existe coerência entre o desenho e a solução.

Avaliou-se a solução desenvolvida, tendo em consideração um conjunto de dimensões e fatores relacionados com os objetivos/requisitos do projeto.

No final, obteve-se uma solução que cumpre com a maior parte dos requisitos e objetivos impostos, tendo um valor de qualidade de 86% (valor calculado pela ferramenta utilizada na avaliação da solução, ver Anexo J). O valor de qualidade obtido supera o mínimo estabelecido pela empresa.

7.2 Limitações

Foram detetadas algumas limitações ao longo do desenvolvimento do projeto.

A primeira limitação está relacionada com a solução apresentada e consiste no tempo de estudo/pesquisa. Este projeto está associado a uma empresa em particular, sendo o autor um ativo dessa mesma empresa. Existe a consciência que as soluções apresentadas podem nem sempre ser as ideais em contextos teóricos. No entanto, foi explicitamente explicado ao autor, pela empresa, que no contexto onde o estudo foi realizado, não se pretendia necessariamente uma solução ótima, mas sim uma solução que cumprisse com o conjunto de objetivos e requisitos impostos. Portanto, cada um dos conteúdos analisados poderia ter sido estudado com uma maior profundidade. No entanto, esta abordagem limitaria a diversidade de informação apresentada. Assim, optou-se por um estudo menos aprofundado de cada tema, mas com uma maior diversidade de conceitos.

A segunda limitação relaciona-se com a avaliação da solução. Realizou-se uma reunião com os diferentes gestores de projeto, de modo que fosse possível adaptar o planeamento de cada desenvolvedor, para que estes pudessem estudar e aplicar o guia devidamente. No entanto, os resultados finais sobre a testabilidade podem não representar o verdadeiro impacto do guia, tendo em conta que o período do estudo é demasiado reduzido para contemplar todas as possíveis retificações ao software já desenvolvido. Assim, pensa-se que os resultados da dimensão de testabilidade (ver 6.4), seriam melhores com um período de análise substancialmente alargado. Relativamente à avaliação da solução, deve-se salientar que a escolha de diferentes dimensões e fatores poderia conduzir a diferentes resultados.

A terceira limitação está relacionada com os desenvolvedores. Não havendo um controlo rigoroso do cumprimento/incumprimento do especificado no guia, é difícil no momento atual verificar rapidamente que membros da equipa de desenvolvimento seguem o guia.

A quarta limitação a assinalar reside no facto de que poder-se-ia ter estudado formas de melhorar aspetos como a especificação e desenho de funcionalidades. No entanto, por opção

da empresa, foi decidido não alterar, pelo menos de momento, estes processos. Com a melhoria destes processos, poder-se-ia suprimir algumas das dificuldades inicialmente relatadas pelos desenvolvedores.

7.3 Trabalho Futuro

O trabalho realizado nesta dissertação enquadra-se numa fase inicial de potenciação do aumento da qualidade do software desenvolvido na Sysnovare.

Primeiramente, deverá ser feita uma monitorização rigorosa ao software desenvolvido e ao cumprimento de aspetos colocados no guia. Não foi objetivo deste projeto a preparação de ferramentas que controlassem de forma rigorosa a qualidade do software, analisando por exemplo cada alteração submetida pelos desenvolvedores no repositório de versionamento do software. No entanto, no futuro, utilizar-se-á o Jenkins (já mencionado neste documento), para realizar este tipo de tarefas, prevendo-se ainda a sua integração com outras plataformas que forneçam informações detalhadas acerca da qualidade do software.

Em segundo lugar, o guia elaborado deve ser atualizado sempre que necessário e até, eventualmente, alargado a outros atributos de qualidade.

7.4 Apreciação Final

Nesta secção realiza-se uma apreciação final, tendo em consideração o planeamento realizado, a organização que promoveu o projeto, a solução apresentada e o trabalho e aprendizagens do autor.

7.4.1 Planeamento do Projeto

O planeamento realizado revelou-se adequado e realista, tendo-se conseguido cumprir com o delineado.

Este aspeto foi sempre realçado tanto pelo supervisor como pelo orientador do projeto, tendo-se revelado fundamental para a realização de um trabalho bem estruturado, cumprindo os prazos delineados.

7.4.2 Organização

A Sysnovare desempenhou um papel fundamental neste projeto. Permitiu uma gestão saudável e balanceada entre este projeto e as restantes funções que o autor desempenha na empresa, tendo-lhe sido dada uma liberdade adequada no que diz respeito à gestão de tempo, que se revelou fundamental para o atingimento da solução e finalização do projeto. Dentro da

Sysnovare, é ainda de salientar a disponibilidade de todos os membros da equipa de desenvolvimento, que, desde início, contribuíram para que a solução aqui apresentada permitisse de facto solucionar o problema pretendido. Dentro da equipa de desenvolvimento, é de acentuar o papel ativo, assertivo e oportuno do chefe da equipa de desenvolvimento, que foi o supervisor deste projeto.

7.4.3 Solução

A solução alcançada, como já referido anteriormente, cumpre com os objetivos e requisitos propostos pela empresa. Espera-se que a solução no futuro possa continuar a contribuir para a melhoria da testabilidade do software, podendo sofrer melhorias ao longo do tempo, tendo em conta que não é uma solução perfeita nem intemporal.

7.4.4 Trabalho e aprendizagens pessoais

Existe uma consciencialização não só das limitações existentes para este projeto, como também das próprias limitações do autor. Sendo o autor, entre outros, trabalhador-estudante e atleta de alta competição, de ginástica acrobática, a concretização deste projeto obrigou a uma rigorosa gestão pessoal, para a qual contribuiu positivamente a Sysnovare, pelo que se reitera aqui, novamente, um agradecimento.

Deve ser salientada a aprendizagem alcançada com a realização deste projeto, que permitiu ao autor aprofundar a temática da testabilidade e os conceitos com ela relacionados.

Está presente no autor um sentimento de felicidade pela potencialidade deste projeto contribuir para uma melhoria no software desenvolvido pela Sysnovare, podendo, assim, melhorar a sua reputação e a satisfação por parte dos seus clientes.

Tendo em conta a solução atingida e sabendo todo o trabalho efetuado, prevalece, ainda, um sentimento de orgulho e dever cumprido.

8 Referências

Abdullah, A., Khan, M. H., & Srivastava, R. (2015). Flexibility: A Key Factor to Testability. *International Journal of Software Engineering & Applications*, *6*(1), 89–99. https://doi.org/10.5121/ijsea.2015.6108

Allman, E. (2012). Managing technical debt. *Communications of the ACM*, *55*(5), 50–55. https://doi.org/10.1145/2160718.2160733

Aniche, M., & Gerosa, M. A. (2015). Does test-driven development improve class design? A qualitative study on developers' perceptions. *Journal of the Brazilian Computer Society*, *21*(1). https://doi.org/10.1186/s13173-015-0034-z

Ann Kitchenham, B., Budgen, D., & Brereton, P. (2015). *Evidence-Based Software Engineering and Systematic Reviews*.

API Documentation. (n.d.). Retrieved April 21, 2022, from https://swagger.io/solutions/api-documentation/

API documentation - Sinon.JS. (n.d.). Retrieved April 14, 2022, from https://sinonjs.org/releases/v13/

arkit - npm. (n.d.). Retrieved April 22, 2022, from https://www.npmjs.com/package/arkit

Badri, L., Badri, M., & Toure, F. (2011). An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes. In *International Journal of Software Engineering and Its Applications* (Vol. 5, Issue 2).

Badri, M., Badri, L., & Flageol, W. (2016). Source and Test Code Size Prediction A Comparison between Use Case Metrics and Objective Class Points. *11th International Conference on Evaluation of Novel Software Approaches to Software Engineering* . https://commons.apache.org/proper/commons-email/

Badri, M., & Toure, F. (2012). Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes. *Journal of Software Engineering and Applications*, *05*(07), 513–526. https://doi.org/10.4236/jsea.2012.57060

Bafandeh Mayvan, B., Rasoolzadegan, A., & Ghavidel Yazdi, Z. (2017). The state of the art on design patterns: A systematic mapping of the literature. *Journal of Systems and Software*, *125*, 1339–1351. https://doi.org/10.1016/j.jss.2016.11.030

Balogun, A., Basri, S., Akintola, A., Bajeh, A. O., Bajeh, A. O., Oluwatosin, O.-J., Akintola, A. G., & Balogun, A. O. (2020). Object-Oriented Measures as Testability Indicators: An Empirical Study. *Journal of Engineering Science and Technology*, *15*(2), 1092–1108. https://www.researchgate.net/publication/340580611

Baudry, B., le Traon, Y., Sunyé, G., & Jézéquel, J.-M. (2003). *Measuring and improving design patterns testability*. https://hal.inria.fr/hal-00794846

Boswell, D., Foucher, T., Cambridge, B., Farnham, Köln, Sebastopol, & Tokyo. (2012). *The Art of Readable Code*.

Bruntink, M., & van Deursen, A. (2004). Predicting class testability using object-oriented metrics. *Proceedings - Fourth IEEE International Workshop on Source Code Analysis and Manipulation*, 136–145. https://doi.org/10.1109/SCAM.2004.16

Bruntink, M., & van Deursen, A. (2006a). An empirical study into class testability. *Journal of Systems and Software*, *79*(9), 1219–1232. https://doi.org/10.1016/j.jss.2006.02.036

Buchanan, J., & Gardiner, L. (2003). A comparison of two reference point methods in multiple objective mathematical programming. *European Journal of Operational Research*, *149*(1), 17–34. https://doi.org/10.1016/S0377-2217(02)00487-3

Buchmann, F., Henney, K., & Schmidt, D. (2007). *Pattern-Oriented Software Architecture*.

Canal, C., Pimentel, E., & Troya, J. M. (2001). Compatibility and inheritance in software architectures. In *Science of Computer Programming* (Vol. 41). www.elsevier.com/locate/scico

Chowdhary, V. (2009). Practicing testability in the real world. *Proceedings - 2nd International Conference on Software Testing, Verification, and Validation, ICST 2009*, 260–268. https://doi.org/10.1109/ICST.2009.53

Choy, S. O., & Ng, K. C. (2007). Implementing wiki software for supplementing online learning. In *Australasian Journal of Educational Technology* (Vol. 23, Issue 2).

Codacy. (2022). *Codacy quickstart (5 min) - Codacy docs*. https://docs.codacy.com/getting-started/codacy-quickstart/

Code alignment - Visual Studio Marketplace. (n.d.). Retrieved April 26, 2022, from https://marketplace.visualstudio.com/items?itemName=cpmcgrath.Codealignment

Crivello, A. (n.d.). *Best Static Code Analysis Tools in 2022 | Compare Reviews on 90+ | G2*. Retrieved February 9, 2022, from https://www.g2.com/categories/static-code-analysis

Debugging in Visual Studio Code. (n.d.). Retrieved April 21, 2022, from https://code.visualstudio.com/docs/editor/debugging

Dobles, I., Martínez, A., & Quesada-López, C. (2019). Comparing the effort and effectiveness of automated and manual tests. *14th Iberian Conference on Information Systems and Technologies*.

Doglio, F. (2018). Scaling Your Node.js Apps. In *Scaling Your Node.js Apps*. Apress. https://doi.org/10.1007/978-1-4842-3991-9

Dresch, A., Daniel, ·, Lacerda, P., Antônio, J., & Antunes, V. (2015). *Design Science Research A Method for Science and Technology Advancement*.

Ebert, C., & Cain, J. (2016). Cyclomatic Complexity. *IEEE Software*, *33*(6), 27–29. https://doi.org/10.1109/MS.2016.147

Engineering Standards Committee of the IEEE Computer Society, S. (2010). *Systems and software engineering-Vocabulary Ingénierie des systèmes et du logiciel-Vocabulaire*. www.iso.org

Escudeiro, P. M., & Bidarra, J. (2008). Quantitative Evaluation Framework (QEF). *Revista Ibérica de Sistemas e Tecnologias de Informação*. https://www.researchgate.net/publication/257579051

ESLint. (2022a). *ESLint - Pluggable JavaScript linter*. https://eslint.org/

ESLint. (2022b). *Getting Started with ESLint - ESLint - Pluggable JavaScript linter*. https://eslint.org/docs/user-guide/getting-started

ESLint. (2022c). *Node.js API - ESLint - Pluggable JavaScript linter*. https://eslint.org/docs/developer-guide/nodejs-api

Ethan Trostler, M. (n.d.). *Testable JavaScript*. www.it-ebooks.info

Evans, E., & Fowler, M. (2014). *Domain-driven design - Tackling Complexity in the Heart of Software*.

Fenton, S. (2018). Pro TypeScript. In *Pro TypeScript*. Apress. https://doi.org/10.1007/978-1-4842-3249-1

Filho, F. G. S., Lelli, V., Santos, I. D. S., & Andrade, R. M. C. (2020, December 1). Correlations among Software Testability Metrics. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3439961.3439972

Fowler, M. (2011). *Patterns of Enterprise Application Architecture*.

Gao, J., & Shih, M. C. (2005). A component testability model for verification and measurement. *Proceedings - International Computer Software and Applications Conference*, *2*, 211–218. https://doi.org/10.1109/COMPSAC.2005.17

Garousi, V., Felderer, M., & Kılıçaslan, F. N. (2019). A survey on software testability. In *Information and Software Technology* (Vol. 108, pp. 35–64). Elsevier B.V. https://doi.org/10.1016/j.infsof.2018.12.003

Gencel, C., & Demirors, O. (2007). Conceptual Differences Among Functional Size Measurement Methods. *Proceedings - 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007*, 305–313. https://doi.org/10.1109/ESEM.2007.35

Ghafari, M., Eggiman, M., & Nierstrasz, O. (2019). Testability First! *International Symposium on Empirical Software Engineering and Measurement*, *2019-Septemer*. https://doi.org/10.1109/ESEM.2019.8870170

Gill, N. (2021, October 15). *Software Testability Metrics and its Various Types*. https://www.xenonstack.com/insights/software-testability-metrics?fbclid=IwAR1H2cwopMosJFKoPdxq0a2mb_-wWo9m33H5jyMilzPuYx5KRP51gg0LUQI

Git History - Visual Studio Marketplace. (n.d.). Retrieved April 26, 2022, from https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory

Goel, N., & Gupta, M. (2012). Testability Estimation of Framework Based Applications. *Journal of Software Engineering and Applications*, *05*(11), 841–849. https://doi.org/10.4236/jsea.2012.511097

González, A., Piel, É., & Gross, H. G. (2009). A model for the measurement of the runtime testability of component-based systems. *IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2009*, 19–28. https://doi.org/10.1109/ICSTW.2009.9

Govil, N. (2020). *Applying Halstead Software Science on Different Programming Languages for Analyzing Software Complexity*.

Graf, A., & Maas, P. (2014). Customer value from a customer perspective – a comprehensive review. In *Service Value als Werttreiber* (pp. 59–87). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-02140-5_3

hapi-swagger - npm. (n.d.). Retrieved April 21, 2022, from https://www.npmjs.com/package/hapi-swagger

Harmes, Ross., & Diaz, Dustin. (2008). *Pro JavaScript design patterns*. Apress.

Heumann, J. (2001). *Generating Test Cases From Use Cases*. http://www.therationaledge.com/content/jun_01/m_cases_jh.html

Hue, C., Hanh, D., & Binh, N. (2018). *A Transformation-Based Method for Test Case Automatic Generation from Use Cases*.

Hunter, T. (2020). *Distributed Systems with Node.js*. O'Reilly Media, Inc.

IBM. (n.d.). *What is Software Testing and How Does it Work?* Retrieved June 3, 2022, from https://www.ibm.com/topics/software-testing

Ingeno, Joseph. (2018). *Software Architect's Handbook : Become a Successful Software Architect by Implementing Effective Architecture Concepts.* Packt Publishing Ltd.

Janzen, D. S., & Saiedian, H. (2006). On the influence of test-driven development on software design. *Software Engineering Education Conference, Proceedings*, *2006*, 141–148. https://doi.org/10.1109/CSEET.2006.25

JetBrains. (2022a). *Code Analysis with Error Checking & Code Annotations - Features | ReSharper*. https://www.jetbrains.com/resharper/features/code_analysis.html

JetBrains. (2022b). *ReSharper: The Visual Studio Extension for .NET Developers by JetBrains*. https://www.jetbrains.com/resharper/promo/

Kedemo, M. (2015, December). Moving Testability Into New Dimensions. *Testing Trapaze* .

Khan, R. A., & Mustafa, K. (2009). Metric based testability model for object oriented design (MTMOOD). *ACM SIGSOFT Software Engineering Notes*, *34*(2), 1–6. https://doi.org/10.1145/1507195.1507204

Kim, T., Kim, H., Khan, M., Kiumi, A., Fang, W., & Sle¸zak, D. (2010). *Advances in Software Engineering*.

Koen, P. A., Bertels, H. M. J., & Kleinschmidt, E. J. (2014). Managing the front end of innovation-part II: Results from a three-year study. *Research Technology Management*, *57*(3), 25–35. https://doi.org/10.5437/08956308X5703199

Koen, P., Ajamian, G., Burkart, R., Clamen, A., Davidson, J., D'Amore, R., Elkins, C., Herald, K., Incorvia, M., Johnson, A., Karol, R., Seibert, R., Slavejkov, A., & Wagner, K. (2001). Providing clarity and a common language to the "fuzzy front end." *Research Technology Management*, *44*(2), 46–55. https://doi.org/10.1080/08956308.2001.11671418

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, *29*(6), 18–21. https://doi.org/10.1109/MS.2012.167

Lewis, W. (2009). *Software Testing and Continuous Quality Improvement*.

Li, Z., Seco, D., & Rodríguez, A. E. S. (2019). Microservice-oriented platform for internet of big data analytics: A proof of concept. *Sensors (Switzerland)*, *19*(5). https://doi.org/10.3390/S19051134

Mardan, A. (2018). Practical Node.js. In *Practical Node.js*. Apress. https://doi.org/10.1007/978-1-4842-3039-8

Maurizio Gabbrielli, & Simone Martini. (2010). *Programming Languages: Principles and Paradigms*. http://www.springer.com/series/7592

Meireles, M. (n.d.). *O Processo Estatístico: tipos de variáveis - ProQuest*. Retrieved June 3, 2022, from https://www.proquest.com/docview/2598730577?pq-origsite=gscholar&fromopenview=true

Mouchawrab, S., Briand, L. C., & Labiche, Y. (2005). A measurement framework for object-oriented software testability. *Information and Software Technology*, *47*(15), 979–997. https://doi.org/10.1016/j.infsof.2005.09.003

Nguyen, V., Bretoi, D., Preul, W., & Benson, L. (2015). *Developing a hapi Edge A Rich Node.js Framework for Apps and Services*.

Nicola, S., Ferreira, E. P., & Ferreira, J. J. P. (2012). A novel framework for modeling value for the customer, an essay on negotiation. *International Journal of Information Technology and Decision Making*, *11*(3), 661–703. https://doi.org/10.1142/S0219622012500162

nock - npm. (n.d.). Retrieved April 20, 2022, from https://www.npmjs.com/package/nock

O'Keeffe, M., & Ócinnéide, M. (2006). Search-based software maintenance. *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, 249–258. https://doi.org/10.1109/CSMR.2006.49

Osterwalder, A., & Pigneur, Y. (2003). Modeling Value Propositions in E-Business. In *University of Lausanne*.

Osterwalder, A., Pigneur, Y., Bernarda, G., & Smith, A. (2014). *Value Proposition Design*.

Pacheco Lacerda, D., Dresch, A., Proença, A., Valle, J. A., & Júnior, A. (2013). *Design Science Research: método de pesquisa para a engenharia de produção Design Science Research: a research method to production engineering*.

Perforce. (2022). *Klocwork for C, C++, C#, Java, JavaScript, and Python | Perforce*. https://www.perforce.com/products/klocwork

Pradeesh, J. (2021). *5 JavaScript Static Analysis Tools - DZone Web Dev*. https://dzone.com/articles/javascript-static-analysis-tools

Rabee Shaheen, M., du Bousquet, L., & du Bousquet Survey, L. (2010). *Survey of source code metrics for evaluating testability of object oriented systems*. https://hal.inria.fr/hal-00953403

Rains, J. A. (2018). *What are the Functions of Function Analysis*.

Richards, M. (W. M. (2015). *Software architecture patterns: understanding common architecture patterns and when to use them*.

Robert C. Martin. (2009). *Clean Code*.

Roy S. Freedman. (1991). Testability of Software Components. *IEEE TRansactions on Software Engineering*.

Saaty, T. L., & Katz, J. M. (1990). How to make a decision: The Analytic Hierarchy Process. *European Journal of Operational Research*, *48*, 9–26.

Sampaio, A. (2015). Improving Systematic Mapping Reviews. *ACM SIGSOFT Software Engineering Notes*, *40*(6), 1–8. https://doi.org/10.1145/2830719.2830732

Schrammel, P. (2020). *How Testable is Business Software?* https://github.com/apache/tika

Sharma R. M. (2014). Quantitative Analysis of Automation and Manual Testing. *International Journal of Engineering and Innovative Technology*.

Sharma, R., & Saha, A. (2018). A Systematic Review of Software Testability Measurement Techniques. *2018 International Conference on Computing*.

Silva, J. (2011). A survey on algorithmic debugging strategies. *Advances in Engineering Software*, *42*(11), 976–991. https://doi.org/10.1016/j.advengsoft.2011.05.024

Singhani, H., & Suri, P. (2015). *Object Oriented Software Testability (OOST) Metrics Analysis*. https://doi.org/10.7753/IJCATR0405.1006

sinon - npm. (n.d.). Retrieved April 14, 2022, from https://www.npmjs.com/package/sinon

SonarQube. (2022). *SonarQube Documentation | SonarQube Docs*. https://docs.sonarqube.org/latest/

Spirlandeli, C., De, C. E., & Roland, F. (2019). A utilização de testes automatizados no desenvolvimento de software. *Revista EduFatec: Educação, Tecnologia e Gestão*.

Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM Model for Agile Methodology. *International Conference on Computing, Communication and Automation*.

Sysnovare. (2015a). *Sysnovare | innovative solutions - sobre nós*. https://www.sysnovare.pt/portal/#sobre-nos

Sysnovare. (2015b). *Sysnovare | innovative solutions - soluções*. https://www.sysnovare.pt/portal/#solucoes

Tarlinder, A. (2017). *Developer Testing*.

Terragni, V., Salza, P., & Pezzè, M. (2020). Measuring software testability modulo test quality. *IEEE International Conference on Program Comprehension*, 241–251. https://doi.org/10.1145/3387904.3389273

Todo Tree - Visual Studio Marketplace. (n.d.). Retrieved April 26, 2022, from https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree

Ulkhaq, M. M., Wijayanti, W. R., Zain, M. S., Baskara, E., & Leonita, W. (2018). Combining the AHP and TOPSIS to evaluate car selection. *ACM International Conference Proceeding Series*, 112–117. https://doi.org/10.1145/3195612.3195628

Use JSDoc. (n.d.). Retrieved April 15, 2022, from https://jsdoc.app/about-getting-started.html

Vacek, J. (2006). Structuring The New Product Development Processes. In *AEDS*.

Vance, S. (2014). *Quality Code*.

Vázquez-Ingelmo, A., García-Holgado, A., & García-Peñalvo, F. (2020). C4 model in a Software Engineering subject to ease the comprehension of UML and the software development process. *2020 IEEE Global Engineering Education Conference*.

vscode-icons - Visual Studio Marketplace. (n.d.). Retrieved April 26, 2022, from https://marketplace.visualstudio.com/items?itemName=vscode-icons-team.vscode-icons

Anexo A – Lista de Objetivos

Tópico	Objetivos
Cobertura	Existência de testes automáticos para todas as componentes das aplicações.
	Existência de testes automáticos para os diferentes casos de uso.
	Existência de testes automáticos para diferentes fluxos em cada caso de uso.
	Existência de testes de erro.
	Existência de testes para situações limite.
	Nível de cobertura das aplicações superior a 95%.
Equipa de Desenvolvimento	Deve sentir que o guia é uma mais-valia para o seu trabalho.
	Esforço sentido para a testagem deve ser reduzido.
	Deve existir favorecimento de testes automáticos, em detrimento dos manuais.
	Esforço para testar software desenvolvido por terceiros deve encontrar-se na mesma medida que o sentido para testar o próprio software (face ao inicialmente sentido).
	Esforço sentido para adicionar e testar uma nova funcionalidade deve ser reduzido (face ao inicialmente sentido).
	Esforço sentido para alterar e testar uma funcionalidade já existente deve ser reduzido.
	O software não deve depender de softwares ou integrações externas para ser testado.

Dependências	O software não deve requerer configurações específicas para a testagem.
	As execuções de testes não devem depender de sistemas externos.
	Não deve ser necessário construir objetos de entidades terceiras para testar o comportamento de uma determinada entidade.
	A execução dos testes não deve estar dependente dá máquina do desenvolvedor, nomeadamente do Sistema Operativo da mesma.
	Não devem existir dependências entre testes, sendo que todos os testes deveriam poder ser executados de forma individual.

Anexo B – Questionário com Parecer da Equipa de Desenvolvimento (Início da Dissertação)

Parecer da equipa de desenvolvimento da Sysnovare quanto à Testabilidade do software

No âmbito da Dissertação intitulada "Relação entre Práticas de Codificação e Testabilidade de Software", promovida pela Sysnovare, e integrada no Mestrado Em Engenharia Informática, no Instituto Superior de Engenharia do Porto, torna-se essencial levantar um conjunto de informações.
As questões deste questionário visam recolher informações relacionadas com a Testabilidade sentida, por parte da equipa de desenvolvimento de software da Sysnovare, no software construído em NodeJs.

O email do inquirido (**null**) foi gravado ao enviar este formulário.
*Obrigatório

1. Email *

2. Realiza/Realizou testes ao software da Sysnovare? *

   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

3. Que dificuldades sente/sentiu ao testar o código por si desenvolvido? *

4. Que dificuldades sente/sentiu ao testar o código desenvolvido por outrem? *

5. Em termos gerais, que dificuldades sente/sentiu ao testar software, nomeadamente através de Testes Automáticos? *

6. Considere a Testabilidade como o grau de facilidade com que testa o software, sendo que uma maior Testabilidade significa que o software é mais facilmente testável. Na sua opinião, o que poderá potenciar uma maior Testabilidade no software desenvolvido na Sysnovare? *

Anexo C – Avaliação Testabilidade (Cobertura)

Dimension	Testabilidade				
Factor	Cobertura				
			Wfk - Fullfilment (%)		
Requirement	Metric Evaluation	0	50		100
FTC001 - Testes Automáticos	Existem testes automáticos para cada componente da aplicação piloto?	Não.			Sim.
FTC002 - Casos de Uso	Existem testes automáticos que cubram todos casos de uso da aplicação piloto?	Não.			Sim.
FTC003 - Percentagem de Cobertura	Nivel médio de cobertura da aplicação piloto - média de cobertura de todos os componentes da aplicação piloto.	Menos de 80%.	Entre 80% a 95%.		Mais de 95%.
FTC004 - Diferentes Fluxos	São testados diferentes fluxos para os casos de uso da aplicação piloto?	Não.	Sim, mas não para todos os casos de uso.		Sim.
FTC005 - Casos de erro	São forçados casos de erro ao software desenvolvido?	Não.	Sim, mas não para todos os casos de uso.		Sim.
FTC006 - Casos limite	São forçados casos limite para funcionalidades específicas do software desenvolvido?	Não.	Sim, mas não para todos os casos de uso.		Sim.

Anexo D – Avaliação Testabilidade (Dependências)

Dimension	Testabilidade			
Factor	Dependências			
			Wfk - Fullfilment (%)	
Requirement	Metric Evaluation	0	50	100
FTD001 - Sistemas Externos	Para a Testagem, o software está dependente de sistemas externos?	Sim.		Não.
FTC002 - Configuração	Para a Testagem, o software requer configuração particular (diferente da existente para a sua execução normal) de ficheiros?	Sim.	Sim, mas o existe um template do ficheiro de configuração no repositório para esse efeito.	Não.
FTC003 - Simulação software	Existe a possibilidade de simular o comportamento do software para suprimir a carência de sistemas externos aquando da testagem?	Não.		Sim.
FTC004 - Isolabilidade	Existem casos na escrita de Testes Automáticos onde é necessário tratar/preparar entidades terceiras para testar uma determinada entidade?	Sim.		Não.
FTC005 - Dependencia de Sistemas Operativos	A execução dos Testes está dependente da máquina/sistema operativo nos quais estes são executados?	Sim.		Não.
FTC006 - Dependencia entre Testes	Existe dependência entre testes, nomeadamente na ordem pela qual estes são executados?	Sim.	Sim, mas não se verifica para todo o software.	Não.

Anexo E – Avaliação Testabilidade (Métricas)

Dimension	Testabilidade					
Factor	Métricas					
Requirement	Metric Evaluation	Wfk - Fullfilment (%)				
		0	50	100		
FTM001 - SLOC vs TLOC	O rácio entre o número de linhas de codigo de testagem e o número de linhas do código fonte diminuiu, no projeto piloto?	Não.	Sim, nalguns dos componentes com alterações.	Sim, em todos os componentes com alterações.		
FTM002 -WMC	Existiu diminuição da complexidade ciclomática média das classes que constituem os componentes do projeto piloto?	Não.	Sim, nalguns dos componentes com alterações.	Sim, em todos os componentes com alterações.		
FTM003 - Previsão de erros (Halstead)	Houve uma diminuição dos erros previstos, no projeto piloto?	Não.	Sim, nalguns dos componentes com alterações.	Sim, em todos os componentes com alterações.		

Anexo F – Avaliação equipa de desenvolvimento (Opinião)

Dimension	Equipa Desenvolvimento				
Factor	Opinião				
Requirement	Metric Evaluation	Wfk - Fullfilment (%)			
		0	50	100	
FTOED001 - Sugestões	Os desenvolvedores sentem que as suas sugestões iniciais foram tidas em consideração.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	
FTOED002 - Legibilidade/Compreensibilidade	Os desenvolvedores consideram o guia benéfico para um melhor entendimento do software desenvolvido.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	
FTED003 - Esforço	Os desenvolvedores sentem menos esforço para testar o software.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	
FTOED004 - Frequência de Testagem	Os desenvolvedores realizam testes ao software - nomeadamente testes automaticos.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	
FTOED005 - Dificuldades	Os desenvolvedores sentem que as suas maiores dificuldades associadas à testagem de software foram suprimidas.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	
FTOED006 - Testes de software Desenvolvido por Outrem	Os desenvolvedores entem que o esforço para testar software desenvolvido por terceiros diminuiu.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	
FTOED007 - Testar nova funcionalidade	Os desenvolvedores sentem um menor custo associado à adição e testagem de uma nova funcionalidade.	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	
FTOED008 - Testar funcionalidade alterada	Os desenvolvedores sentem um menor custo associado à alteração e testagem de uma funcionalidade já existente	Menos de 30 % dos interrogados respondeu que sim.	Entre 30 a 70% dos interrogados respondeu que sim.	Mais de 70% dos interrogados respondeu que sim.	

Anexo G – Avaliação guia (Conteúdos do Guia)

Dimension	Guia			
Factor	Conteúdos do guia			
			Wfk - Fullfilment (%)	
Requirement	Metric Evaluation	0	50	100
FPCP001 - Regras	O Guia apresenta regras que devem ser (sempre que possível) seguidas para aumentar a Testabilidade.	Não.		Sim.
FPCP002 - Apoio à decisão	O Guia apoia a decisão do desenvolvedor nalgumas questões ambíguas.	Não.		Sim.
FPCP003 - Autoexplicativo	O Guia é autoexplicativo, não havendo necessidade de consultar informações externas para entender o nele perpetuado.	Não.	Sim, mas apresenta sugestão de informação adicional a ser consultada, de modo culmatar	Sim.
FPCP004 - Sugere Ferramentas	O Guia sugere a utilização de ferramentas que potenciem a melhoria da qualidade de código, nomeadamente da Testabilidade.	Não.		Sim.
FPCP005 - Para o desenvolvedor	O Guia tem em conta as dificuldades reportadas pela equipa de desenvolvimento, apresentando soluções para as mesmas.	Não.	Sim, para algumas das dificuldades apresentadas.	Sim, para todas as dificuldades apresentadas.

Anexo H – Avaliação guia (Disponibilização do Guia)

Dimension	Guia			
Factor	Disponibilização do Guia			
			Wfk - Fullfilment (%)	
Requirement	Metric Evaluation		0	100
FPDP001 - Disponibilidade	O Guia está acessível a qualquer membro da equipa de desenvolvimento.		Não.	Sim.
FPDP002 - Modificabilidade	O Guia é editável, de modo a potenciar a sua evolução ao longo do tempo.		Não.	Sim.
FPDP003 - Adequado ao Contexto da Empresa	O Guia está integrado nas práticas atuais da empresa, de modo a potenciar a sua utilização.		Não.	Sim.

Anexo I - Questionário com opinião da equipa de desenvolvimento (final da dissertação)

Melhorias sentidas pela Equipa de Desenvolvimento - Depois da Aplicação de guia

No âmbito da Dissertação intitulada "Guia para aumentar a testabilidade do software", promovida pela Sysnovare, e integrada no Mestrado em Engenharia Informática, no Instituto Superior de Engenharia do Porto, torna-se essencial levantar um conjunto de informações.
As questões deste questionário visam recolher informações relacionadas com a adequação da solução construída no âmbito da resolução do problema da baixa testabilidade do software desenvolvido na Sysnovare.

O email do inquirido (**null**) foi gravado ao enviar este formulário.
*Obrigatório

1. Email *

2. 1 - Sente que as sugestões por si apresentadas anteriormente, caso as tenha realizado, no primeiro questionário de resposta aberta, foram tidas em consideração no guia? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Não aplicável

3. 2 - No seu entendimento, com a aplicação do guia, existe um maior entendimento do software, quer em cada funcionalidade, quer no software como um todo? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

4. 3 - O esforço sentido para a testagem do Software é reduzido, face ao sentido antes de aplicar o guia? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

5. 4 - Realiza, presentemente, testes - nomeadamente testes automáticos - ao código que desenvolve? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

6. 5 - Viu suprimidas as suas maiores dificuldades associadas à realização de testes? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

7. 6 - Sente que o esforço para testar software desenvolvido por terceiros reduziu, com a aplicação do guia? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Não aplicável

8. 7 - Sente que o custo inerente à adição de uma funcionalidade e sua respetiva testagem foi reduzido com a aplicação do guia? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

9. 8 - Sente que o custo inerente à alteração de uma funcionalidade e sua respetiva testagem foi reduzido com a aplicação do guia? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

Anexo J - QEF

q	D	Qi	Dimension	Q_j	W_{ij} (Factor Weight j in Dim i) [0,1]	Factor	rw_{jk} (requirement weight k in Factor j) {2, 4, 6, 8, 10}	Requirement	wf_k % requirement fulfillment k) [0,100]
86%	0,36	66,67	Testabilidade	75	0,40	Dependências	10	FTD001 - Sistemas Externos	100
							2	FTC002 - Configuração	100
							4	FTC003 - Mocks	100
							6	FTC004 - Isolabilidade	0
							10	FTC005 - Dependencia de Sistemas Operativos	100
							8	FTC006 - Dependencia entre Testes	50
				50	0,40	Cobertura	10	FTC001 - Testes Automáticos	100
							10	FTC002 - Casos de Uso	0
							10	FTC003 - Percentagem de Cobertura	50
							6	FTC004 - Diferentes Fluxos	50
							6	FTC005 - Casos de erro	50
							4	FTC006 - Casos limite	50
				83,333	0,20	Métricas	2	FTM001 - SLOC vs TLOC	50
							2	FTM002 -WMC	100
							2	FTM003 - Previsão de erros	100
		90	Equip. Desenvolvimento	90	1,00	Opinião Equipa Desenvolvimento	6	FTOED001 - Sugestões	50
							10	FTOED002 - Legibilidade/Compreensibilidade	100
							10	FTED003 - Esforço	100
							10	FTOED004 - Frequência de Testagem	100
							6	FTOED005 - Dificuldades	100
							6	FTOED006 - Testes de software Desenvolvido por Outrem	50
							6	FTOED007 - Testar nova funcionalidade	100
							6	FTOED008 - Testar funcionalidade alterada	100
		89,06	Guia	100	0,38	Disponibilização do Guia	10	FPDP001 - Disponibilidade	100
							8	FPDP002 - Modificabilidade	100
							8	FPDP003 - Adequado ao Contexto da Empresa	100
				82,5	0,63	Conteúdos do Guia	10	FPCP001 - Regras	100
							8	FPCP002 - Apoio à decisão	100
							8	FPCP003 - Autoexplicativo	50
							8	FPCP004 - Sugere Ferramentas	100
							6	FPCP005 - Para o desenvolvedor	50