INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Avaliação de Qualidade de Desenho de Software por Cobertura e Sincronismo com o Código

JOÃO FERNANDO CORREIA QUERIDO
Outubro de 2023

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Avaliação de Qualidade de Desenho de Software por Cobertura e Sincronismo com o Código

João Fernando Correia Querido

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

Orientador: Alexandre Bragança

Co-orientador: Emanuel Silva

Júri:

Presidente:

Dr. Carlos Ferreira, Professor, DEI/ISEP

Vogais:

Dr. Nuno Bettencourt, Professor, DEI/ISEP

Porto, outubro 2023

Dedicatória

A realização deste trabalho é dedicada especialmente aos meus pais, que tornaram possível a realização de mais uma etapa na minha vida e que sempre acreditaram no meu esforço. Quero também deixar um agradecimento particular aos meus amigos que me apoiaram e ajudaram em todo o percurso académico.

João Querido

Resumo

Esta dissertação foi desenvolvida num ambiente académico, debruçando-se sobre a temática da avaliação da qualidade do desenho de software por cobertura e sincronismo com o código, ou seja, pretender perceber se o código desenvolvido está totalmente apoiado no desenho elaborado previamente.

Tem como objetivo encontrar a ferramenta mais adequada para a geração de um relatório de semelhança entre um diagrama de sequência elaborado manualmente e um excerto de código também elaborado manualmente.

Durante o desenvolvimento da dissertação foram investigadas diversas alternativas para que fossem abordados todos os possíveis meios que permitissem avaliar a qualidade do desenho do software por cobertura e sincronismo com o código.

Em suma, durante todo o processo de investigação da ferramenta mais adequada, foram utilizados processos de engenharia de software, nos quais se destacam a análise de requisitos, investigação e a testabilidade.

Palavras-chave: Avaliador de controlo e qualidade de diagramas de software sincronizados com o código, Qualidade de Desenho de Software, UML, Código, Diagrama de sequência

Abstract

This dissertation was developed in an academic environment, focusing on the theme of assessing the quality of software design by coverage and synchronisation with the code, i.e. trying to understand whether the code developed is fully supported by the design drawn up previously.

Its aim is to find the most suitable tool for generating a similarity report between a manually drawn sequence diagram and a manually drawn code extract.

During the development of the dissertation, various alternatives were investigated so that all possible means of assessing the quality of the software design by coverage and synchronisation with the code could be addressed.

In short, throughout the process of investigating the most suitable tool, software engineering processes were used, including requirements analysis, research and testability.

Keywords: Evaluator of control and quality of software diagrams synchronized with the code, Quality of Software Design, UML, Code, System Sequence Diagram

Agradecimentos

Em primeira instância o agradecimento é dirigido aos meus pais, que me apoiaram em todos os momentos, estando sempre disponíveis para me ajudar, ouvir e aconselhar, nunca deixando de acreditar nas minhas capacidades.

De seguida agradeço aos meus colegas de mestrado e a todos os meus amigos que me acompanharam desde o primeiro ano da licenciatura até este momento, sendo os mesmos responsáveis por me ampararem e auxiliarem a ultrapassar todas adversidades do meu percurso académico.

Deixo também um agradecimento à minha entidade patronal Critical Manufacturing e aos demais colegas e amigos de profissão, pela trivialidade com que me permitiram conciliar o mestrado com a minha vida profissional.

Agradeço também à instituição ISEP, em especial ao departamento de Engenharia Informática (DEI) pela passagem de conhecimento durante todo o percurso na Licenciatura, aliado ao conhecimento também adquirido no Mestrado em Engenharia de Software.

Por fim, agradeço ao meu orientador, Alexandre Bragança, e ao coorientador, Emanuel Silva, por todo o suporte prestado. A disponibilidade de ambos para a revisão do trabalho, todas as críticas construtivas e sugestões, fornecidas pelos mesmos, foram cruciais para o resultado e qualidade da dissertação final.

Declaração de Integridade

Declaro ter realizado com integridade a presente dissertação e corroboro não ter recorrido à prática de plágio, bem como qualquer outra forma de utilização inadequada ou adulteração de informações ou dos resultados nas diversas etapas da sua elaboração.

Mais declaro ter-me comprometido a realizar a análise, especificação, conceção, desenvolvimento, teste e manutenção do software de acordo com os oito princípios do IEEE-CS/ACM sobre Ética em Engenharia de Software e Práticas Profissionais. (Institute for Electrical and Electronics Engineers and Association for Computing Machinery, 2019)

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

AD	*Activity Diagram*
AHP	*Analytic Hierarchy Process*
BPMN	*Business Process Model and Notation*
CD	*Continuous Delivery*
CI	*Continuous Integration*
EQSOFT	Engenharia e Qualidade de Software
FAST	*Function Analysis System Technique*
GPL	*General Public License*
GUI	*Graphical User Interface*
IDE	*Integrated Development Environment*
ISEP	Instituto Superior de Engenharia do Porto
MDA	*Model Driven Architecture*
MDT	*Model Driven Testing*
NCD	*New Concept Development*
OMG	*Object Management Group*
PSM	*Platform specific models*
QMOOD	*Quality Model for Object-Oriented Design*
SAST	*Static Application Security Testing*
SCM	*Supply chain management software*
SDLC	*Software Development Life Cycle*
SSD	*System sequence diagram*
TMDEI	Tese / Dissertação / Estágio
UC	*Use Case*
UI	*User Interface*

UML *Unified Modeling Language*

UX *User Experience*

XMI X*ML Metadata Interchange*

1 Introdução

Neste capítulo, é descrita a visão geral da dissertação "Avaliação de Qualidade de Desenho de Software por Cobertura e Sincronismo com o Código". Em primeira instância, é esclarecido o contexto em que a dissertação está inserida. De seguida, é descrito o problema e indicados os respetivos objetivos pretendidos. É também, neste capítulo, detalhada a abordagem utilizada para a resolução do problema e os resultados esperados com a utilização da ferramenta escolhida. Por fim, é apresentada a estrutura deste documento.

1.1 Enquadramento/Contexto

A vigente dissertação foi elaborada em ambiente académico, no Lab EQSOFT - Laboratório de Engenharia e Qualidade de Software, DEI, ISEP, como dissertação para a unidade curricular de TMDEI, presente no mestrado de Engenharia Informática – Engenharia de Software do Instituto Superior de Engenharia do Porto. Possibilitou ao autor da presente dissertação adquirir novos conhecimentos ao nível da investigação, bem como diferentes metodologias de trabalho.

No ambiento académico, bem como no mercado de trabalho, existe a necessidade de avaliar a qualidade de código de software em conjunto com o desenho que lhe deu origem. Já existem inúmeras ferramentas *open-source* direcionadas a avaliar a qualidade do código-fonte, bem como a cobertura do mesmo através de testes e possíveis *bugs* ou *code-smells*, mas com lacunas na análise da componente de desenho. Surgiu então a necessidade de avaliar de forma equivalente o desenho de software para ser possível compreender se o mesmo deu origem a todo o código produzido, aliado ainda à possível ausência de representação de alguns excertos no desenho de software.

Posto isto, esta dissertação surgiu dessa mesma necessidade e expõe a abordagem adotada na tentativa de encontrar uma ferramenta adequada (ou combinação de ferramentas), direcionada a ser utilizada em ambiento académico e atender às necessidades do Lab EQSOFT.

1.2 Problema

O problema que a vigente dissertação pretende explorar e quiçá resolver prende-se com o facto de que os modelos ou diagramas *Platform specific models* (PSM) desenvolvidos num projeto muitas vezes não são totalmente refletidos aquando da geração código. Logo, é necessário perceber se o código desenvolvido está de acordo com o modelo e qual a cobertura do modelo face a esse mesmo código.

Atualmente é possível encontrar diversas ferramentas para controlo e melhoria do código desenvolvido. Porém, essas ferramentas têm como principal funcionalidade a avaliação da qualidade do software ao nível das linguagens de programação (implementação) e raramente se debruçam ao nível do desenho da solução. Ou seja, existem diversas soluções com suporte para análise da qualidade do código, mas o mesmo não acontece para análise da qualidade de modelos ou diagramas PSM por cobertura e sincronismo com o código.

Por fim, é de realçar que será tido em foco diagramas ou modelos em UML, devido a sua vasta utilização e, em particular, em diagramas de sequência (SSD).

1.3 Objetivos

O principal objetivo desta dissertação visa propor uma solução que permita avaliar a qualidade do desenho do software por cobertura e sincronismo com o código, focando-se em *standards* com forte utilização como por exemplo UML ou XMI e/ou ferramentas de uso comum, como por exemplo, o PlantUML.

Contudo, existem diversas ferramentas qualificadas para atingir o objetivo principal desta dissertação podendo ser distinguidas pela sua extensibilidade, integração e custo.

Logo, o código, quanto à linguagem utilizada e dimensão, bem como a linguagem do diagrama previamente elaborado, são fatores que pressupõe diferentes hipóteses(Prasad, Rao and Rehani, 2001) de escolha de ferramenta, sendo assim necessário explorar as hipóteses mais relevantes e que permitam chegar à solução aplicável em qualquer situação (Myers, 2018; The Software Engineering Authority, 2020).

A solução proposta deve, preferencialmente, ser direcionada para tecnologias *open-source*, pois tem como finalidade ser utilizada em ambientes académicos de ensino, de conceção e implementação de software.

1.4 Abordagem

A vigente dissertação enquadra-se no âmbito de um processo científico, mais concretamente um *design science research*. É possível corroborar tal afirmação, pois esta dissertação pressupõe algumas das características que definem um *design science research* como a já referida

utilização em meio académico, a existência de múltiplos casos de estudo, como o código a analisar ser de diferentes linguagens, dimensões e complexidade e também a possibilidade da dissertação permitir que seja publicado de uma forma resumida em poucas páginas (Wohlin and Runeson, 2021).

Inicialmente, é aconselhado a realização de um estudo sobre a área de avaliação e controlo de desenho de software, bem como, sobre a área de avaliação e controlo do software, descrito no subcapítulo "Enquadramente/Contexto".

De seguida, o problema da dissertação deverá ser caracterizado de um modo mais pormenorizado a fim de averiguar o método mais eficaz e que melhor se adequada para avaliar e controlar o desenho de software, caracterizado no Subcapítulo "Problema".

Após este estudo deve ser realizada uma pesquisa sobre quais são as ferramentas de avaliação e controlo de desenho de software existentes, sendo que este estudo está exposto no Capítulo "Estado da Arte".

Posteriormente, também no Capítulo "Estado da Arte", será elaborada uma análise sobre a extensão e compatibilidade das ferramentas encontradas para, deste modo, ser possível detetar as limitações das mesmas.

Por fim, deve ser efetuado um caso de estudo no Capítulo "Solução" , que não inclui o desenvolvimento de um protótipo, mas sim a escolha de um já existente. O caso de estudo está subdividido em duas partes de complexidade crescente, sendo a primeira mais elementar e a segunda com maior complexidade, para que seja assegurado o valor e utilidade da ferramenta escolhida aquando da sua utilização com um repositório exemplo.

Em suma, no Capítulo "Conclusão", os resultados provenientes do estudo serão apreciados para que possam ser recolhidas as conclusões relacionadas com a possibilidade de utilização da ferramenta, principalmente, em ambiente académico.

Denote se que todo o processo descrito anteriormente é iterativo e incremental, logo é expectável que, eventualmente, possa existir uma diferença nos objetivos apresentados na primeira iteração.

1.5 Contribuições

A vigente dissertação terá como finalidade contribuir para auxiliar o meio académico, fomentando a avaliação de software produzido pelos estudantes, nomeadamente nos diagramas e modelos de software em conjunto com o código.

Para que seja possível apoiar os já existentes meios de avaliação será incorporada uma ferramenta de avaliação e controlo qualidade de diagramas de software de sincronismo com código, dentro das diversas alternativas apresentadas ao longo desta dissertação.

1.6 Estrutura da Dissertação

Após serem concluídos os subcapítulos precedentes que têm como fim contextualizar e oferecer ao leitor a correta perceção de qual é o objetivo do trabalho realizado, é igualmente fulcral perceber a estrutura da dissertação.

O Capítulo 2 descreve o estado de arte. São apresentadas as aplicações existentes e mais utilizadas para a avaliação e controlo de qualidade de desenho e código de *software,* bem como, os *standards* mais utilizados. Além disso, são também apresentados alguns trabalhos associados com a área da dissertação desenvolvida que a organização em questão pretende.

O Capítulo 3 expõe a análise de valor que tem como objetivo apresentar o valor do problema apresentado, bem como a sua solução para que deste modo seja possível identificar a oportunidade a perseguir. É também apresentada a análise *SWOT* com o objetivo de ver esclarecido o propósito de perseguir da oportunidade já identificada. De seguida, é também especificada a proposta de valor, através modelo da *value propostion* de Osterwalder. Por fim, é exposto o diagrama *FAST* e o método AHP para comparar as diversas ferramentas possíveis para a implementação da solução, no sentido de auxiliar a tomada de decisão final.

No Capítulo 4 é apresentada uma descrição aprofundada da análise da solução escolhida no âmbito do problema, assim como a análise dos requisitos e fundamentas as decisões optadas.

O Capítulo 5 destina-se a descrever a implementação da solução escolhida para colmatar o problema.

No Capítulo 6 é apresentada a avaliação do sistema implementado, através de um estudo das métricas e técnicas utilizadas, onde vão ser analisados os resultados obtidos. Por fim, é também realizada uma apreciação dos mesmos resultados pelo autor.

Por último, é no Capítulo 7 que são apresentadas as mais relevantes conclusões sobre a dissertação elaborada. Também são descritas algumas limitações encontradas na solução desenvolvida, assim como possíveis pontos de melhoria e as próximas etapas a realizar para que o mesmo possa ser integrado, num futuro próximo, em ambiente académico.

1.7 Sumário

Em suma, neste capítulo é possível identificar o problema, modelos ou diagramas desenvolvidos num projeto muitas vezes não são totalmente refletidos aquando da geração código, os objetivos da dissertação.

Também é deliniado o principal objetivo desta dissertação visa propor uma solução que permita avaliar a qualidade do desenho do software por cobertura e sincronismo com o código. Bem como, a abordagem nos moldes de um processo científico, sendo a sua contribuição como de auxílio ao meio académico e não só.

Por fim é também apresentada a estrutura da dissertação composta por mais 6 capítulos sendo eles (2) Estado de Arte, (3) Análise de Valor, (4) Análise E Desenho Da Solução, (5) Experimentação E Avaliação, (6) Avaliação da Solução e (7) Conclusão.

2 Estado da Arte

Neste capítulo é apresentado o resultado do estudo realizado sobre conceitos e tecnologias relevantes para a compreensão do problema e desenho da solução ideal. São apresentados conceitos teóricos indispensáveis à compreensão do âmbito em que a dissertação está inserida. Também são descritas algumas das abordagens existentes para avaliar e controlar o desenho de software.

O capítulo inicia com a descrição do que é a revisão de código bem como os diferentes tipos e algumas das principais ferramentas capazes de fazer esta revisão. De seguida será apresentado a revisão de desenho de software, os diferentes tipos da mesma. Após esta descrição são enumeradas e caracterizadas ferramentas direcionadas a avaliar *User Interface* (UI) e ferramentas com foco na análise dos diagramas. Por fim, serão especificados alguns trabalhos idênticos à vigente dissertação.

2.1 Revisão de Código

A revisão de código, célebre por *code review* é a metodologia onde uma ou várias pessoas efetuam, como o nome indica, uma revisão ao código desenvolvido, possibilitando assim a identificação de *bugs* e excertos de código que não estão de acordo com os princípios adotados pelo projeto ou em relação à *framework* que está a ser utilizada. Um exemplo é a utilização de variáveis denominadas de “a” ou “x” cuja nomenclatura não reflete corretamente o significado sobre a entidade que representam.

Este processo existe, pois, o software é escrito por seres humanos e, como sabemos, ninguém é perfeito, nós somos passíveis de erros. Sabendo disso, é possível relacionar que um software costuma ter vários erros que precisam ser corrigidos antes da implantação, por isso, a sua revisão é indispensável (nata.house, 2022).

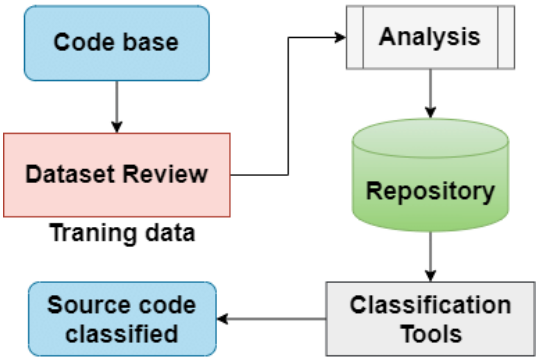


Figura 1 - Arquitetura da análise de revisão de código (Adam Khleel and Károly, 2020)

Existem três tipos distintos de revisão de código: (1) Automatizada, (2) Manual e (3) Mista.

2.1.1 Revisão de Código Automatizada

A revisão é considerada do tipo "Automatizada" se for efetuada exclusivamente com o auxílio de uma ferramenta.

Este processo de revisão permite paralelamente aos princípios já enumerados acima:(nata.house, 2022)(Daityari, 2023)

- Detetar se existem inconsistências no código;
- Localizar operações não permitidas ou pouco seguras, como *loops* infinitos ou apontadores nulos;
- Organizar e exibir os ficheiros atualizados durante uma etapa específica do projeto;
- Avaliar a eficácia do processo de revisão do código com métricas específicas.

A principal vantagem deste tipo de revisão está relacionada com a agilidade de todo o processo, pois é poupado bastante tempo de dedicação à tarefa.

Todavia acarreta uma grande desvantagem relacionada com a sua capacidade ser bastante limitada quando comparada com a de um individuo, sendo assim possível que alguns excertos de código sejam postos de parte ou ignorados pela ferramenta.

2.1.2 Revisão de Código Manual

Este tipo de revisão é elaborado por uma pessoa, normalmente com o papel do programador de software.

O processo no fundo resume-se a analisar linha a linha de código procurando possíveis vulnerabilidades. Por isso, todo o processo exige que a pessoa designada a esta tarefa tenha experiência e competências na área (codegrip, 2020).

As principais vantagens desta abordagem são as seguintes:(Malik, 2022)

- Encontrar problemas que nenhuma outra técnica pode identificar;
- Possibilidade dos seus utilizadores despenderem tempo a analisar o código em proporção com o tamanho do mesmo.

Por outro lado, as principais desvantagens centram-se no tempo consumido pela tarefa, que pode ser extenso e o nível elevado de dificuldade em encontrar todas as anomalias no código.

2.1.3 Revisão de Código Mista

Por último, existe a revisão mista que, como o seu nome sugere, é uma mistura entre a revisão automatizada e a revisão manual.

Por norma, é realizada primeiro a revisão automatizada e depois a revisão manual para que seja possível obter feedback mais assertivo pelos revisores em conjunto com o feedback transmitido pela ferramenta (nata.house, 2022), possibilitando assim complementar as desvantagens de uma abordagem com a utilização da outra.

Contudo, a desvantagem do tempo que todo o processo irá necessitar é agravada pela utilização das duas abordagens.

2.2 Avaliador de Controlo E Qualidade de Código de Software

Na presente secção são apresentadas as ferramentas orientadas para avaliar e controlar a qualidade do código para que deste modo seja possível avaliar com maior exatidão e precisão a qualidade do código de software.

A denominada ferramenta de revisão de código tem como objetivo automatizar o processo de revisão de código para que um revisor se concentre apenas no código (Daityari, 2023).

Estas ferramentas são concebidas para fornecer feedback imediato ao programador sobre os problemas que podem estar presentes no código. Desta forma é possível localizar as vulnerabilidades do código numa fase precoce o que é bastante útil, pois habitualmente a localização de vulnerabilidades tende a ser realizada numa fase mais tardia do ciclo *Software Development Life Cycle* (SDLC) (Lima, 2022).

2.2.1 Review board

Review board é uma ferramenta *open-source* e autónoma de revisão de código, que pode ser instalada em máquinas locais ou na nuvem, apenas com custo para o serviço de hospedagem (Beanbag, 2001).

O *workflow* da ferramenta é considerado padrão permitindo também a integração com ferramentas de análise estática, como por exemplo verificadores de estilo e plataformas CI/CD, pois o Review board é uma ferramenta de revisão híbrida necessitando da revisão manual combinada com as ferramentas de software. O Review board ainda possibilita racionalizar o fluxo de trabalho de revisão com outras ferramentas como Slack, Asana, Jenkin, Trello, entre outras (Greiler, 2023).

No Quadro 1 são apresentados os principais benefícios e as limitações da ferramenta Review board.

Quadro 1 - Benefícios e limitações da ferramenta Review board

Benefícios	**Limitações**
▪ *Open-source*; ▪ Suporta integrações consideradas *3rd party*; ▪ Sustenta a revisão automática de código; ▪ Integração CI.	▪ UI antiquada; ▪ Impossibilidade de integração com IDE's.

2.2.2 Codestriker

A ferramenta Codestriker é uma aplicação web gratuita e *open-source* utilizada para a revisão de código de forma colaborativa (SoftwareTestingHelp, 2023). Esta ferramenta suporta também revisões a código gerado por um sistema SCM (SourceForge, 2001).

No Quadro 2 são apresentados os principais benefícios e as limitações da ferramenta Codestriker.

Quadro 2 - Benefícios e limitações da ferramenta Codestriker

Benefícios	**Limitações**
▪ Possibilidade de ser executada nas principais plataformas e navegadores; ▪ Licenciado pela GPL; ▪ Minimiza o trabalho em papel, os problemas, comentários e decisões ficam registados numa base de dados.	▪ *Workflow* complexo; ▪ Impossibilidade de integração com IDE's.

Contudo, tem como principal desvantagem o seu *workflow* pois é considerado complexo e com a agravante de não possuir integração com qualquer IDE.

2.2.3 Crucible

Crucible é uma ferramenta de revisão de código desenvolvida pela Atlassian. É considerada uma das melhores pela diversidade de sistemas de controlo de versão que é capaz de integrar para além do Git, como BitBucket Server, Mercurial, CVS, Subversion e Perforce.

O *workflow* de comentários desta ferramenta permite aos utilizadores assinalar qualquer parte do código com apenas um comentário informativo ou assinalar que são necessárias alterações. Por outro lado, os comentários podem ser acrescentados como esboços, possibilitando ao revisor refletir sobre o feedback dado anteriormente e, se necessário, fazer alterações aos comentários à medida que obtém mais informação do código e contexto em que foi desenvolvido (Greiler, 2023).

No Quadro 3 são apresentados os principais benefícios e as limitações da ferramenta Crucible.

Quadro 3 - Benefícios e limitações da ferramenta Crucible

Benefícios	**Limitações**
▪ Ferramenta flexível, suportando diversas abordagens de trabalho e dinâmicas de equipa; ▪ Permite a sua utilização no *pre-commit*; ▪ Permite a sua utilização no pos-*commit*.	▪ Não é *open-source*; ▪ Inexistência de mecanismos que possibilitem que a ferramenta seja utilizada para avaliar código através de critério de aprovação mais rigorosos.

2.2.4 Collaborator

A Collaborator é uma ferramenta reconhecida por ser a mais abrangente quando se trata de obter uma qualidade de código crítica.

As suas principais funcionalidades permitem a visualização das alterações ao código, a adição de comentários, a definição de regras para a revisão e notificações automáticas para assegurar que o término da revisão está dentro do prazo definido (SoftwareTestingHelp, 2023).

No Quadro 4 são apresentados os principais benefícios e as limitações da ferramenta Collaborator (smartbear, 2023) (Greiler, 2023).

Quadro 4 - Benefícios e limitações da ferramenta Collaborator

Benefícios	**Limitações**
▪ Integração ágil e flexível; ▪ Possibilidade de ser integrada com cerca de onze diferentes SCM's; ▪ Possibilidade de ser integrada com IDE's, nomeadamente, Eclipse e Visual Studio.	▪ Elevado custo da licença base; ▪ Collaborator depende de outras ferramentas, como Git, para poder funcionar totalmente; ▪ A personalização da interface é limitada.

2.2.5 NDepend

NDepend é uma ferramenta .NET que possibilita uma análise profunda na base do código desenvolvido. O website oficial da ferramenta NDepend[1] declara que a ferramenta permite aos programadores, arquitetos e executivos tomar decisões inteligentes sobre os projetos (ZEN PROGRAM HLD, 2023).

Esta ferramenta permite calcular as métricas padrão para compreender melhor o projeto em questão, verificar o código do projeto e respeita as regras de qualidade padrão. Também permite criar as suas próprias métricas e aplicar regras personalizadas (Tomassetti, 2017).

Contudo, não é recomendada a sua utilização em código com uma longa lista de construtores, principalmente se os mesmos forem constituídos por um grande número de parâmetros (Tomassetti, 2017).

No Quadro 5 são apresentados os principais benefícios e as limitações da ferramenta NDepend.

[1] https://www.ndepend.com/

Quadro 5 - Benefícios e limitações da ferramenta NDepend

Benefícios	**Limitações**
▪ Elabora uma análise detalhada do código, incluindo métricas de código, regras personalizadas e visualizações avançadas; ▪ Possibilidade de ser integrada com IDE's, nomeadamente, Jenkins e Visual Studio; ▪ Suporte a diversas linguagens de programação como Java e C#.	▪ Elevado custo da licença base; ▪ Configuração complexa; ▪ Dependência de outras ferramentas como .NET Framework e o Visual Studio.

2.2.6 SonarQube

O SonarQube[2] é uma ferramenta de análise de código estático que ajuda a garantir a qualidade do código. Esta ferramenta foi concebida com o propósito de ser utilizada ao longo do ciclo de vida do software, desde a fase de desenvolvimento até o lançamento do produto. Além disso a ferramenta é capaz de analisar código de diversas linguagens de programação, como C#, Java, JavaScript, Ruby, Python, entre outras.

No Quadro 6 são apresentados os principais benefícios e as limitações da ferramenta SonarQube (Alves, 2020).

Quadro 6 - Benefícios e limitações da ferramenta SonarQube

Benefícios	**Limitações**
▪ Suporte de diversas linguagens de programação; ▪ Rápida identificação de problemas de qualidade de código; ▪ *Work flow* fácil e intuitivo; ▪ Possibilidade de ser integrada com IDE's, nomeadamente, Eclipse, Visual Studio e Maven.	▪ Apenas deteta os problemas de código e não procede à correção dos mesmos; ▪ Apesar de ter um *work flow* fácil a sua configuração é complexa; ▪ Elevado consumo de recursos do sistema em que está instalada.

[2] https://www.sonarsource.com/products/sonarqube/

2.2.7 Comparação Entre as Ferramentas de Controlo E Qualidade de Código de Software

A Tabela 1 apresenta uma síntese comparativa de algumas características, escolhidas através da combinação de diferentes fontes, entre as várias ferramentas estudadas (Adam Khleel and Károly, 2020)(GitHub, 2019; GitLab, 2021).

Tabela 1 – Comparação das diferentes ferramentas de revisão de código

	Review Board	Codestriker	Crucible	Collaborator	NDepend	Sonarqube
Integração com ferramentas de controlo de versões	**Sim**	**Sim, através de um *plug-in***	**Sim**	**Sim**	**Sim**	**Sim**
Integração com IDE's	**Não**	**Não**	**Não**	**Sim**	**Sim**	**Sim**
Revisões no *pre commit*	**Sim**	**Não**	**Sim**	**Sim**	**Sim**	**Sim**
Revisões no *post commit*	**Sim**	**Não**	**Sim, Patchreview**	**Sim**	**Sim**	**Sim**
Work flow	**Fácil**	**Complexo**	**Fácil**	**Fácil**	**Fácil**	**Fácil**
Revisão automática de código	**Não**	**Não**	**Não**	**Não**	**Não**	**Sim**
UI	**Antiquada**	**Antiquada**	**Moderna**	**Moderna**	**Moderna**	**Moderna**
Análise ao desenho	**Não**	**Não**	**Não**	**Não**	**Sim, apenas para aplicações em .NET**	**Sim, mas limitado**

2.3 Revisão E Avaliação Do Desenho de Software

A revisão e a análise de desenho de software são sistemáticas, abrangentes e bem documentadas, visando verificar se os requisitos de desenho especificados na fase inicial do projeto são os adequados e se o desenho vai de encontro a todos os requisitos especificados. Além disso, também ajudam a identificar os problemas, se os mesmos existirem, no processo de elaboração do desenho (Siddiqui Ammar, 2017).

A revisão do desenho de software é constituída por três principais tipos de revisão:

- Revisão preliminar do desenho;
- Revisão crítica do desenho;
- Revisão do desenho programada.

Figura 2 - Principais tipos de revisão do desenho de software

Existem ainda mais quatro revisões que não foram consideradas, em virtude da sua relevância ser considerada inferior às representadas na Figura 2, são elas: (Campbell, 2021)

- Revisão de requisitos;
- Revisão do sistema/conceção do desenho;
- Revisão Final do desenho;
- Revisão da prontidão para produção.

Contudo devido a não serem tão relevantes, quando comparadas com as três referencias das na Figura 2, serão apenas detalhadas com maior pormenor a revisão preliminar, crítica e programada do desenho.

2.3.1 Revisão Preliminar Do Desenho

Por norma, em primeira instância, é elaborada a revisão preliminar. Esta revisão é feita com a presença dos clientes e utilizadores, permitindo que o desenho de software seja revisto pelo cliente antes de avançar para a próxima fase.

Desta forma, é possível transpor para o cliente como irá funcionar o sistema para satisfazer as necessidades do mesmo. Esta revisão é considerada do tipo formal e tem como principais objetivos: assegurar que os requisitos do software são refletidos na arquitetura do software, assim como assegurar que a estrutura de dados será coerente com o domínio e por fim, mas não menos importante, avaliar os fatores de qualidade (Campbell, 2021).

Por outro lado, também é possível detetar anomalias no desenho nesta revisão e, caso essas anomalias sejam graves, a equipa de desenvolvimento irá rever o mesmo. Se essa revisão

determinar uma nova conceção do mesmo, a revisão preliminar do desenho será novamente agendada com o cliente (Thakur, 2023).

2.3.2 Revisão Crítica Do Desenho

Em segundo lugar, é elaborado uma revisão crítica em que integram apenas elementos com conhecimento técnico como analistas de desenho, programadores de desenho e ao auditor do sistema.

Esta revisão é conduzida para servir os seguintes propósitos:(Thakur, 2023)

- Assegurar que não existem lacunas nos desenhos técnicos e conceptuais;
- Verificar se o projeto em revisão satisfaz os requisitos de projeto estabelecidos nas especificações;
- Avaliar criticamente a funcionalidade e o estado de maturidade do desenho;
- Clarificar o desenho a pessoas fora do desenvolvimento da mesa. Deste modo o desenho técnico fica mais claro e fácil de compreender.

À semelhança da revisão preliminar do desenho esta é também uma revisão de cariz formal e, se forem detetadas discrepâncias no processo de revisão crítica do desenho, as falhas são avaliadas com base na sua gravidade. Uma falha menor é resolvida pela equipa de revisão. Por outro lado, já uma falha maior pode levar a equipa de revisão a rever o desenho técnico proposto (Haughtondesign, no date).

2.3.3 Revisão Do Desenho Programada

De seguida, é efetuada a revisão do desenho do programa com os programadores a fim de obter feedback antes do desenho ser implementado.

A reunião é conduzida com o objetivo de assegurar:(Thakur, 2023)

- A viabilidade da conceção detalhada;
- A consistência da interface de acordo com o desenho arquitetónico;
- A compatibilidade da especificação do desenho com a linguagem de implementação;
- A compreensão da proposta de desenho por parte da equipa de implementação.

Esta é também uma revisão considerada formal que, em caso de sucesso, resultará sempre em considerações relacionadas com os planos de codificação antes do início da codificação.

2.4 Avaliador de Controlo E Qualidade de Desenho de Interface de Utilizador

Como já referenciado, existem inúmeras razões para que a revisão do desenho de interface de utilizador seja feita. No entanto, por norma, estas revisões acontecem quando há necessidade de reunir todas as pessoas envolvidas no desenvolvimento do projeto para partilhar informação e envolver todos no processo de *brainstorming* para decidir sobre as próximas ações (Campbell, 2021).

De modo a auxiliar este processo de avaliação e controlo do desenho de interface de utilizador existem dois tipos de ferramentas que são utilizadas: (1) as ferramentas de verificação do desenho de interface de utilizador e (2) as ferramentas visuais de testes de regressão.

2.4.1 Ferramentas de Verificação Do Desenho De Interface de Utilizador

Estas ferramentas comparam o desenho de interface de utilizador e o código, assegurando que o código corresponde à implementação correta do desenho de interface de utilizador.

As principais funcionalidades da sua utilização são:(AMIQ EDA, 2023)

- Aumentar a velocidade e a qualidade do desenvolvimento do código adicionado;
- Compreender facilmente o código fonte complexo ou que se encontre mal documentado;
- Simplificar a manutenção do código antigo e das bibliotecas reutilizáveis;
- Acelerar a aprendizagem de linguagens e metodologias;
- Gerar automaticamente documentação de código fonte.

Segue abaixo uma breve análise de cada uma das ferramentas escolhidas.

2.4.1.1 Parasoft

A ferramenta Parasoft fornece uma variedade de recursos de análise de código através da utilização de técnicas de análise estática para identificar possíveis problemas de código e fornecer feedback aos programadores em tempo real (Parasoft, 2023).

A Parasoft também pode ser usada para automação de testes e integração contínua, permitindo que as equipas de desenvolvimento elaborem testes de software com um maior nível de eficiência.

No Quadro 7 são apresentados os principais benefícios e as limitações da ferramenta Parasoft (Parasoft SOAtest reviews, rating and features 2023 | PeerSpot, 2021) (Trofimov, 2021) .

Quadro 7 - Benefícios e limitações da ferramenta Parasoft

Benefícios	Limitações
▪ Variedade de recursos de análise de código, incluindo análise estática e dinâmica, e automação de testes; ▪ Suporte a várias linguagens de programação, incluindo C/C++, Java, .NET e Python; ▪ Possibilidade ser integrado em vários ambientes de desenvolvimento.	▪ Elevado custo; ▪ Complexidade.

2.4.1.2 LDRA

O LDRA é uma ferramenta de análise de código que permite garantir a conformidade com os padrões de segurança de software, como MISRA, CERT e DO-178B. Esta ferramenta assegura uma análise detalhada do código e, por conseguinte, permite identificar possíveis problemas de segurança e erros.

O LDRA também pode ser usado para criar testes automatizados e medir a cobertura de código, garantindo que todo o código está coberto por testes (LDRA, 2023).

No Quadro 8 são apresentados os principais benefícios e as limitações da ferramenta LDRA (Mark, 2018).

Quadro 8 - Benefícios e limitações da ferramenta LDRA

Benefícios	Limitações
▪ Conformidade com padrões de segurança de software, como MISRA, CERT e DO-178B; ▪ Análise detalha de código.	▪ Elevado custo; ▪ Dificuldade de integração com outras ferramentas de desenvolvimento de software.

2.4.2 Ferramentas de Teste de Regressão Visual

Estas ferramentas comparam o desenho de interface de utilizador esperado com a representação real no código, assegurando que o desenho de interface de utilizador é refletido com precisão no código e verificando se a disposição e os elementos visuais se alinham com as expectativas.

Existem os seguintes tipos de testes de regressão visual: (Lambdatest, no date)

- Visual Manual – são testes de regressão funcionais elaborados manualmente pelos programadores sem utilizar ferramentas de teste de automação;
- Comparação de *Layout* – comparam o tamanho e a posição dos elementos da interface do utilizador em vez de pixels. Logo, se o tamanho da posição de um elemento mudar, o teste falhará;
- Comparação *Pixel* por *Pixel* – este método compara duas capturas de ecrã e analisa-as a um nível de pixel;
- Comparação Estrutural – este teste compara a estrutura do DOM para determinar as alterações de marcação HTML;
- Comparação visual da IA - este tipo de teste visual utiliza duas imagens usando ML e IA;
- Comparação com base no DOM – estes testes são baseados na comparação baseada no DOM. Analisa o mesmo DOM antes e depois de uma mudança de estado e assinala as diferenças.

O objetivo dos testes de regressão visual é assegurar que a experiência do utilizador é visualmente perfeita, focando-se em verificar a aparência e a usabilidade da UI após uma mudança de código (Shain, 2022).

Em suma, a principal funcionalidade destas ferramentas é impedir que *bugs* visuais, com um possível custo elevado, sejam incluídos aquando da instalação do software em produção. Este cenário é comum pois os testes funcionais tradicionais funcionam através da simples validação da entrada e saída de dados (Shain, 2022).

2.4.2.1 Testlio

Testlio é uma plataforma de testes de software que usa uma combinação de testes manuais e automatizados para identificar bugs, falhas e outros problemas que podem afetar a funcionalidade e a experiência do utilizador de um software (Testlio Inc, no date).

Além disso, é possível integrar o Testlio com outras ferramentas como Jira, Slack e Trello, para ajudar as equipas a gerir o seu fluxo de trabalho.

No Quadro 9 são apresentados os principais benefícios e as limitações da ferramenta Testlio (Ryan, 2022).

Quadro 9 - Benefícios e limitações da ferramenta Testlio

Benefícios	**Limitações**
▪ Escalabilidade, pode ser usada em diferentes plataformas e dispositivos; ▪ Flexibilidade, pode ser usado em diferentes fases do ciclo de vida do desenvolvimento do software, desde a conceção até o lançamento final.	▪ Elevado custo; ▪ Necessário tempo de familiarização com a interface e com os processos envolvidos.

2.4.2.2 Wraith

Wraith é uma ferramenta de teste de regressão visual *open-source* que ajuda a detetar problemas visuais em websites (Blooman *et al.*, 2019).

O funcionamento desta ferramenta resume-se a capturar *screenshots* de uma página da web e comparar essas capturas com versões anteriores da página. Dessa forma, é possível detetar alterações visuais na página, como mudanças de *layout*, cor ou fonte (Sharma, 2017).

Além de que, o Wraith é uma ferramenta altamente configurável, podendo ser integrado com sistemas como o Jenkins, o TravisCI e o TeamCity e, também pode ser utilizado em diversos *browsers*, incluindo Google Chrome, Firefox e PhantomJS.

No Quadro 10 são apresentados os principais benefícios e as limitações da ferramenta Wraith.

Quadro 10 - Benefícios e limitações da ferramenta Wraith

Benefícios	**Limitações**
▪ Facilidade de integração; ▪ Rápido e escalável.	▪ Apenas especializada em testes de regressão visual; ▪ identifica falhas em imagens que não são necessariamente problemas reais da aplicação.

2.4.2.3 Percy

Percy é uma ferramenta de teste visual automatizada que permite garantir que a interface do software está a funcionar corretamente. A ferramenta é usada para capturar e comparar visualmente as alterações num qualquer website e detetar problemas de layout, estilo ou outros problemas visuais (BrowserStack, 2023).

A ferramenta Percy tem um funcionamento idêntico à Wraith, apesar desta ter a particularidade de ser integrada nos seus testes automatizados já existentes. Dessa forma, quando os testes são executados, a Percy captura uma série de imagens de cada página e compara com versões antecedentes, assinalando as diferenças entre as versões (Kumar, 2022).

No Quadro 11 são apresentados os principais benefícios e as limitações da ferramenta Percy.

Quadro 11 - Benefícios e limitações da ferramenta Percy

Benefícios	**Limitações**
▪ Facilmente integrada com várias ferramentas de teste e controlo de versões, como o GitHub, o Bitbucket e o GitLab; ▪ Suporta vários *browsers*, como o Google Chrome, o Firefox e o Safari.	▪ Custo elevado; ▪ Obrigatoriedade de estar conectado à internet; ▪ Integração de Percy em um pipeline de CI/CD é complexa.

2.5 Avaliador de Controlo E Qualidade de Diagramas de Software Sincronizados Com O Código

Da mesma forma que a qualidade do código e do desenho de interface de utilizador são controlados e avaliados, os diagramas, normalmente sobre a forma de diagramas de sequência ou SSD que especificam e representam as diversas funcionalidades de um determinado projeto, também requerem uma avaliação e controlo sobre os mesmos.

Deste modo, é necessário verificar se os diagramas concebidos previamente são refletidos no código desenvolvido. Por conseguinte, na presente secção, são apresentados os dois métodos que verificam esta condição.

O método inicial requer a combinação de uma ferramenta como o Visual Studio, Tracealyzer ou PlantUML com uma comparação manual por parte do utilizador de diferenças entre diagramas. O papel da ferramenta neste método está relacionado com a geração de um novo diagrama através do código desenvolvido e utilizando uma funcionalidade destas ferramentas denominada de *built-in sequence diagram generation*, denote-se que nem todas a ferramentas investigadas podem suportar esta funcionalidade.

Após a geração do diagrama o utilizador compara-os para identificar as semelhanças e diferenças. De seguida, irá determinar a percentagem do diagrama que é representada no seu código, enumerando os eventos e trocas de mensagens presentes em ambos os diagramas e de seguida dividindo pelo número total eventos e trocas de mensagens presentes no diagrama original.

Em segunda instância, apenas é utilizada uma ferramenta como o Enterprise Architect, UModel, ou Visual Paradigm que, para além de agregarem a mesma funcionalidade de geração de diagramas que o Visual Studio, Tracealyzer ou PlantUML, também tem incorporada uma funcionalidade que permite comparar o diagrama gerado a partir do código com o diagrama existente.

Esta funcionalidade irá salientar as diferenças entre os dois diagramas providenciando uma percentagem de similaridade.

2.5.1 Visual Studio

O Visual Studio[3] é um IDE (ambiente de desenvolvimento integrado), que oferece uma ampla gama de recursos para programadores de software, capaz de suportar várias linguagens de programação, incluindo C#, C++, Python, entre outras.

Esta ferramenta disponibiliza um *debugger*, um editor de código, ferramentas de criação de GUI, integração com controle de versão, bem como outras ferramentas que ajudam no

[3] https://visualstudio.microsoft.com/#vs-section

desenvolvimento e controlo de software e, ainda, de geração de diagramas através de código (Pedamkar, 2019).

No Quadro 12 são apresentados os principais benefícios e limitações da linguagem de programação Visual Studio (Pedamkar, 2019) (Microsoft Visual Studio Reviews 2023 - Capterra, 2023).

Quadro 12 - Benefícios e limitações da ferramenta Visual Studio

Benefícios	**Limitações**
▪ Suporte diversas linguagens de programação; ▪ Interface intuitiva; ▪ Integração com tecnologias de controle de versão.	▪ Elevado custo para obter a licença para versão comercial; ▪ Elevada a complexidade criada pela agregação de diversas funcionalidades.

2.5.2 Tracealyzer

Tracealyzer[4] é uma ferramenta de software para visualização e análise do comportamento de sistemas embebidos em tempo real. Esta mesma ferramenta recolhe e analisa padrões do sistema, considerados arquivos de registro que guardam a execução do sistema ao longo do tempo. Estes padrões são gerados por um sistema embutido interligado a um computador que por sua vez está a proceder à execução do Tracealyzer.

No Quadro 13 são apresentados os principais benefícios e limitações da ferramenta Tracealyzer (Carnica Technology, no date) (Percepio Tracealyzer Reviews & Ratings 2023, 2023).

Quadro 13 – Benefícios e limitações da ferramenta Tracealyzer

Benefícios	**Limitações**
▪ Análise em tempo real do desempenho do sistema; ▪ Integração com IDE's como Visual Studio e Eclipse; ▪ Permite gerar diagramas através de *system traces*; ▪ Consultar o agendamento de tarefas e de que forma a interrupção de uma tarefa afeta o desempenho do sistema.	▪ Elevado custo para obter a licença para versão comercial; ▪ Complexidade de utilização que dificultará o processo de aprendizagem em programadores juniores.

[4] https://percepio.com/tracealyzer/

2.5.3 PlantUML

PlantUML[5] é uma ferramenta de modelação de software *open-source* que permite desenhar diagramas UML de modo rápido através de uma sintaxe simples e intuitiva. Suporta ainda diferentes tipos de diagramas, incluindo diagramas de classe, de sequência, de atividade, entre outros. Além disso, permite que seja integrada em diversos *code editors* que são capazes de gerar diagramas com base no código(Cherniavska, 2019). Em suma, o PlantUML é uma ferramenta para programadores de software que necessitem de elaborar diagramas de forma rápida e eficiente;

No Quadro 14 são apresentados os principais benefícios e limitações da ferramenta PlantUML (Terlecki, 2019).

Quadro 14 - Benefícios e limitações da ferramenta PlantUML

Benefícios	**Limitações**
▪ Suporta diversos tipos de diagramas; ▪ A qualidade dos diagramas gerado é considerada de alto nível e são visualmente atraentes para os utilizadores.	▪ Limitada ao nível de recursos e funcionalidades; ▪ Inexistência de uma interface gráfica.

2.5.4 Enterprise Architect

Enterprise Architect[6] é uma ferramenta de desenho e modelação de software da empresa Sparx Systems e suporta vários padrões de modelação de diagramas, como o UML e BPMN.

A ferramenta possui ainda uma funcionalidade de geração de diagramas baseado no código fornecido pelo programador, que também é capaz de comparar o diagrama gerado com o diagrama previamente elaborado (SparxSystems, 2000a).

No Quadro 15 são apresentados os principais benefícios e limitações da linguagem de programação Enterprise Architect (SparxSystems, 2000b) (Software Adivce, 2006).

Quadro 15 - Benefícios e limitações da ferramenta Enterprise Architect

Benefícios	**Limitações**
▪ Suporta diversos padrões de modelação e geração de código a partir de diagramas e o seu inverso; ▪ Integração com ferramentas de desenvolvimento de software.	▪ Custo elevado para obtenção de uma licença de utilização base; ▪ Elevada complexidade de utilização pois a vasta quantidade de recursos pode criar dificuldades de interpretação ao utilizador.

[5] https://plantuml.com/
[6] https://sparxsystems.com/products/ea/

2.5.5 UModel

UModel[7] é uma ferramenta de modelação de diagramas em UML desenvolvida pela Altova. A ferramenta permite que sejam utilizados na modelação em UML diagramas de sequência, classe, entre outros. UModel também inclui funcionalidades de geração automática de código tendo por base diagramas, bem como o inverso. Além disso, possibilita a integração com diversas ferramentas de desenvolvimento de software, como Visual Studio e Eclipse.

No Quadro 16 são apresentados os principais benefícios e limitações da ferramenta UModel.

Quadro 16 - Benefícios e limitações da ferramenta UModel

Benefícios	**Limitações**
▪ Funcionalidade de geração de automática de código e diagramas; ▪ Interface gráfica intuitiva e personalizável; ▪ Integração com ferramentas de desenvolvimento de software.	▪ Custo elevado para obtenção de uma licença de utilização base; ▪ Limitação de recursos e funcionalidades quando comparada com ferramentas Enterprise Architect; ▪ Necessidade de ser hospedado, num computador com bastantes recursos.

2.5.6 Visual Paradigm

O Visual Paradigm[8] é uma ferramenta de modelação e desenho de software que aceita a utilização de distintos padrões de modelação, incluindo UML, BPMN, ERD, entre outros.

Esta ferramenta possibilita a criação de diagramas que representam diferentes vistas de um sistema, como requisitos de negócios, arquitetura, desenho de software, modelação de dados, processos de negócios e fluxos de trabalho.

Também suporta uma funcionalidade de geração de diagramas com base em excertos de código, realçando as diferenças entre o diagrama gerado e o diagrama previamente elaborado manualmente (Visual Paradigm, 2002).

No Quadro 17 são apresentados os principais benefícios e limitações da ferramenta Visual Paradigm (Visual Paradigm, 2002) (PeerSpot, 2023).

[7] https://www.altova.com/umodel
[8] https://www.visual-paradigm.com/

Quadro 17 - Benefícios e limitações da ferramenta Visual Paradigm

Benefícios	Limitações
▪ Suporte a diferentes padrões de modelação; ▪ Permite que diversas pessoas trabalhem em simultâneo no mesmo projeto; ▪ Funcionalidade de geração de automática de código e diagramas.	▪ Licença paga para ser utilizada; ▪ Interface pouco personalizável; ▪ Tempo de espera ao carregar grandes projetos, ou seja, contém nele um elevado número de diagramas e modelos.

2.5.7 Comparação Entre as Ferramentas de Controlo E Qualidade de Diagramas de Software Sincronizados Com O Código

Nas Tabela 2 e Tabela 3 é apresenta uma síntese comparativa entre as diferentes ferramentas previamente enunciadas. As características escolhidas para comparar as mesmas têm como objetivo sobressair as principais diferenças.

Tabela 2 – Comparação entre ferramentas de controlo e qualidade de diagramas de software sincronizados com o código - parte 1

	Visual Studio	**Tracealyzer**	**PlantUML**
Geração de código a partir de diagramas	**Através de plugins ou extensões**	**-**	**-**
Geração de diagramas a partir de código	**Apenas em alguns tipos de projeto e linguagem de programação**	**-**	**Apenas através de *snipets* básicos de texto**
Comparação de similaridade de diagramas	**Manuel e através de *plugins***	**Manual**	**Manual**
Tipos de diagramas	**Classe Sequência Atividade Componente Implantação**	**Estado Sequência Atividade Comunicação Chamada**	**Classe Sequência Atividade Componentes Implantação Cronograma Rede Sintaxe**
	Visual Studio	**Tracealyzer**	**PlantUML**

Padrões de modelação de diagramas	**UML BPMN ERD DSL Visual Studio**	**Visualizações e gráficos personalizados *system traces* Tracealyzer**	**UML BPMN Wireframe Gantt PlantUML**
Versão demo	**Sim**	**Sim**	**Sim**
Versão comercial	**Sim**	**Sim**	**Não**
Open-source	**Não**	**Não**	**Sim**

Tabela 3 – Comparação entre ferramentas de controlo e qualidade de diagramas de software sincronizados com o código - parte 2

	Enterprise Architect	**UModel**	**Visual Paradigm**
Geração de código a partir de diagramas	**Sim**	**Sim, para diagramas UML**	**Sim**
Comparação de similaridade de diagramas	**Manual**	**Manual**	Manual
Tipos de diagramas	**Classe Sequencia Atividade Componentes Implantação Casos de Uso Estado Comunicação Perfil Package Rede**	**Classe Sequência Atividade Componente Implantação Objetos Comunicaçã o Perfil Casos de Uso**	**Classe Sequência Atividade Componente Implantação Casos de Uso Comunicação Estado Perfil Interação Temporização Estrutura composta**
Padrões de modelação de diagramas	**UML BPMN TOGAF ArchiMate SysML**	**UML**	**UML BPMN ERD DFD SysML**
Versão demo	**Não**	**Sim, com tempo limitado**	**Sim**
Versão comercial	**Sim**	**Sim**	**Sim**
Open-source	**Não**	**Não**	**Não**

2.6 Tecnologias

Na seguinte secção são apresentadas algumas tecnologias utilizadas para desenvolver código e diagramas de software. Esta escolha foi realizada tendo por base combinação entre as principais

tecnologias usadas no desenvolvimento de software, compatibilidade destas com as ferramentas de avaliação e controlo dos diagramas de software sincronizados com o código e a sua utilização em ambiente académico.

Em primeira instância, são apresentadas as tecnologias para desenvolvimento de código de software e, de seguida, são apresentadas as tecnologias para desenvolvimento de diagramas e modelos de software.

2.6.1 Java

A linguagem de programação Java[9] é uma linguagem de alto nível, orientada a objetos e com uma sintaxe simples, desenvolvida pela Sun Microsystems em 1995 e é considerada como uma das linguagens mais populares utilizada em todo o mundo(Cass, 2023).

Quando comparada com outras linguagens de programação a linguagem Java é utilizada em uma grande diversidade de cenários como aplicações GUI, sistemas de *back-end* avançados e no desenvolvimento de jogos (Strato Flow, 2023).

No Quadro 18 são apresentados os principais benefícios e limitações da linguagem de programação Java (Data Flair, 2018; IBM, 2023).

Quadro 18 - Benefícios e limitações da linguagem Java

Benefícios	**Limitações**
▪ Fácil de aprender; ▪ Independente; ▪ Orientado a objetos; ▪ *Multithreading*.	▪ Custo elevado para a licença comercial; ▪ Lenta; ▪ Problemas de performance limitada.

2.6.2 C#

C#[10] é uma linguagem de programação relativamente recente, de alto nível, orientada a objetos, desenvolvida pela Microsoft e foi elaborado para ser uma linguagem segura, robusta e fácil de aprender, auxiliando o desenvolvimento de aplicações para a plataforma .NET.

No Quadro 19 são apresentados os principais benefícios e limitações da linguagem de programação C# (Deborah, 2019)(Payne, 2023a)(Payne, 2023b).

[9] https://www.java.com/
[10] https://learn.microsoft.com/en-us/dotnet/csharp/

Quadro 19 - Benefícios e limitações da linguagem C#

Benefícios	Limitações
▪ Excelente performance; ▪ Compatibilidade multiplataforma como Windows, Linux e MacOS; ▪ Suporta programação orientada a objetos e programação funcional.	▪ Extremamente dependente de ferramentas e plataformas da Microsoft, limitando a sua interoperabilidade com outras plataformas: ▪ Problemas de performance em aplicações como jogos.

2.6.3 Python

A linguagem Python[11] é uma linguagem de programação de alto nível, interpretada e de uso geral. É da autoria de Guido van Rossum em 1991 e é uma das linguagens de programação mais populares do mundo(Stackscale, 2023) .

No Quadro 20 são apresentados os principais benefícios e limitações da linguagem de programação Python (Novotny, 2023)(Vartanian, 2022).

Quadro 20 - Benefícios e limitações da linguagem Python

Benefícios	Limitações
▪ Grande variedade de bibliotecas e frameworks disponíveis; ▪ Compatibilidade multiplataforma como Windows, Linux e MacOS; ▪ Utlizada para desenvolver diversos tipos de aplicações como aplicações desktop a aplicações web e científicas.	▪ Performance limitada; ▪ Baixa segurança; ▪ Problemas de compatibilidade entre versões, por exemplo entre versão 2 e 4 do Python.

2.6.4 UML

O UML [12] conhecida por linguagem de modelação unificada. É uma linguagem gráfica padronizada para a modelação através de diagramas e modelos de sistemas de software orientados a objetos. Esta linguagem foi desenvolvida pela OMG e é vivamente utilizada em todo o mundo pela comunidade de programadores de software (Visual Paradigm, 2017).

No Quadro 21 são apresentados os principais benefícios e limitações da linguagem UML (Poest, 2020)(Indeed Editorial Team, 2022).

[11] https://www.python.org/
[12] https://www.omg.org/spec/UML/

Quadro 21 - Benefícios e limitações da linguagem UML

Benefícios	Limitações
▪ Utlizada para modelar sistemas complexos; ▪ Abstrai os detalhes técnicos e tem em foco os conceitos de alto nível; ▪ Padronizada e globalmente usada.	▪ Dificulta a atualização da modelação do sistema o mesmo evolui; ▪ Devido ao seu nível de abstração pode levar a abstrações excessivas e dessa forma criar dificuldades aos programadores de interpretação do sistema.

2.6.5 XMI

XMI[13] é uma linguagem de modelação, que permite a troca de informações entre ferramentas de modelação baseadas em XML. Foi desenvolvida pelo grupo de trabalho da OMG e é considera uma extensão do formato XML.

No Quadro 22 são apresentados os principais benefícios e limitações da linguagem XMI (OMG, 2015).

Quadro 22 - Benefícios e limitações da linguagem XMI

Benefícios	Limitações
▪ Formato padronizado e suportado por várias ferramentas de desenvolvimento de software; ▪ Integração com IDE's; ▪ Confere a troca de informações de metadados entre várias ferramentas de desenvolvimento de software.	▪ O ficheiro gerado tem um tamanho considerável implicando algum tempo para serem carregados; ▪ Como foi desenvolvido para trabalhar com UML, pode não ser tão fácil de trabalhar com outras linguagens de modelação.

2.6.6 BPMN

A linguagem gráfica BPMN [14] é uma linguagem modelação de processos de negócios, desenvolvida pela OMG e utilizada para descrever e visualizar processos de negócios em uma ampla variedade de indústrias globalmente.

No Quadro 23 são apresentados os principais benefícios e limitações da linguagem BPMN (Semenchuk, 2017)(kissflow, 2023).

[13] https://www.omg.org/spec/XMI/
[14] https://www.omg.org/spec/BPMN/

Quadro 23 - Benefícios e limitações da linguagem BPMN

Benefícios	Limitações
▪ Notação simples e padronizada para modelar processos de negócios; ▪ Interface gráfica intuitiva; ▪ Adaptável tanto processos simples como complexos.	▪ Difícil de entender em processos muito complexos e com muitos detalhes; ▪ Limitada utilidade em outras áreas que não sejam processos de negócios e fluxos de trabalho.

2.6.7 Compatibilidade Das Linguagens De Programação E Modelação

Nesta secção é apresentada a Tabela 4 que indica se as diferentes linguagens, quer de programação quer de modelação, são compatíveis com todas ferramentas apresentadas previamente, sejam estas focadas na revisão e análise de código, bem como, do desenho da Interface gráfica ou do desenho do *software*.

Tabela 4 – Compatibilidade das linguagens com as diferentes ferramentas

	Java	C#	Python	UML	BPMN	XMI
Review Board	**Sim**	**Sim**	**Sim**	**Não**	**Não**	**Não**
Codestriker	**Sim**	**Sim**	**Sim**	**Não**	**Não**	**Não**
Crucible	**Sim**	**Sim**	**Sim**	**Não**	**Não**	**Não**
Collaborator	**Sim**	**Sim**	**Sim**	**Não**	**Não**	**Não**
NDepend	**Não**	**Sim**	**Não**	**Não**	**Não**	**Não**
SonarQube	**Sim**	**Sim**	**Sim**	**Não**	**Não**	**Não**
Parasoft	**Sim**	**Sim**	**Não**	**Não**	**Não**	**Não**
LDRA	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**
Testlio	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**
Wraith	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**
Percy	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**
Visual Studio	**Sim**	**Sim**	**Sim**	**Sim**	**Não**	**Não**
Tracealyzer	**Não**	**Não**	**Não**	**Não**	**Não**	**Não**
PlantUML	**Apenas diagramas que representam código desta linguagem**	**Apenas diagramas que representam código desta linguagem**	**Apenas diagramas que representam código desta linguagem**	**Sim**	**Não**	**Não**
Enterprise Architect	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**
UModel	**Sim**	**Sim**	**Sim**	**Sim**	**Não**	**Sim**
Visual Paradigm	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**

2.7 Trabalhos Relacionados

Na presente secção é feita uma revisão de artigos, projetos em curso e soluções atualmente utilizadas relativas ao tema da vigente dissertação, avaliação e controlo de qualidade de desenho de software.

Em virtude de ser um tema pouco explorado e face ao desconhecimento por parte do meio envolvente foram encontrados um número reduzido de trabalhos relevantes e com uma data de publicação recente.

2.7.1 Model-Based Testing Using UML Activity Diagrams: A Systematic Mapping Study

No artigo de Ahmad (Ahmad *et al.*, 2019) é apresentada uma visão abrangente das abordagens existentes sobre testes baseados em modelos, utilizando padrão UML ADs.

Este artigo relaciona-se com o tema da corrente dissertação pois, prevê que sejam utilizados diagramas, mais concretamente UML AD, que sejam totalmente refletidos em código, especificamente em testes. Paralelamente, também é referido no artigo que o tema foi pouco explorado até ao momento, o que coincide com atual dissertação.

O artigo expõe uma visão abrangente relativamente ao estado de arte sobre testes baseados em modelos UML AD, sendo que estes modelos são normalmente utilizados como base para o desenvolvimento de testes.

Os UML ADs são diagramas semiformais utilizados para representar atividades sequenciais e concorrentes, os objetos que são produzidos e consumidos e qual a ordem de execução das diferentes ações.

Na Figura 3 está representado um exemplo de UML AD com um parâmetro de input.

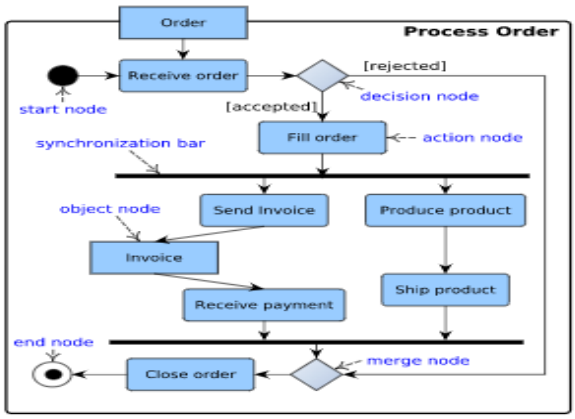


Figura 3 - UML AD com parâmetro de input (OMG, 2005)

Em suma, foi possível concluir através deste artigo que (1) os UML AD não estão a ser utilizados para testes não funcionais, (2) apenas algumas abordagens foram validadas contraestudos de casos industriais realistas, (3) a maioria das abordagens visa domínios de aplicação muito restritos, e (4) existe atualmente uma clara falta de abordagens holísticas para testes baseados em modelos utilizando modelo UML AD (Ahmad *et al.*, 2019).

2.7.2 Comparing Model Coverage and Code Coverage in Model Driven Testing: An Exploratory Study

Neste segundo artigo, de. Amalfitano (Amalfitano *et al.*, 2016) são exploradas duas metodologias, MDT e o MDA, num caso de estudo com especial foco no MDT. Este artigo coincide com a vigente tese apenas no apuramento de quais as principais razões para que um modelo não reflita totalmente o código gerado;

Na Figura 4 é demonstrada a comunicação entre o MDA, representado do lado esquerdo da figura, e o MDT, representado do lado direito da figura (Amalfitano *et al.*, 2016).

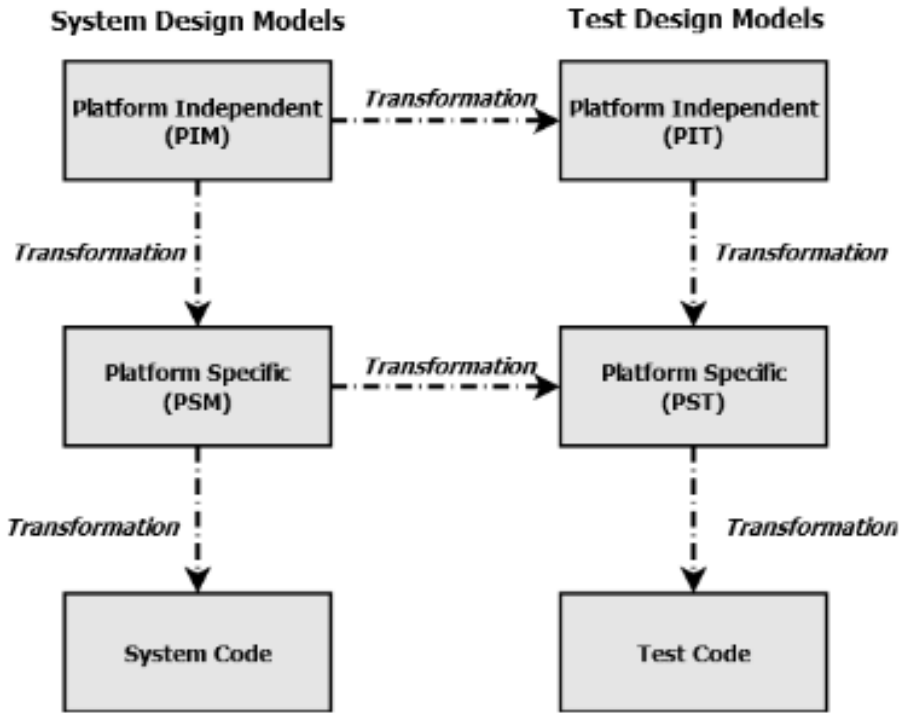


Figura 4 - MDA e MDT

Ao longo do artigo é descrito um estudo exploratório a fim de avaliar as diferenças que existem entre a cobertura do modelo pelos casos de teste e a cobertura de código alcançada aquando da execução da geração automática de código. Além disso, são identificados os principais fatores que permitem a existência de diferenças em relação à cobertura de código.

Os principais fatores para existirem tais diferenças são (1) a ferramenta que implementa as regras de transformação do PSM em código, (2) adequação de critérios para os testes ao nível do modelo, sendo transpostos na geração do código para testes e (3) o estilo adotado pelo programador para representar o comportamento do sistema a nível do PSM (Amalfitano *et al.*, 2016).

2.7.3 Avaliação Da Qualidade de Software Com Base Em Modelos UML

Na dissertação de Bertrán Macía (Bertrán Macía, 2009) é proposto um conjunto de estratégias para identificar, em modelos e diagramas UML, problemas específicos e recorrentes ao nível do desenho como: *Long Parameter List, God Class, Data Class, Shotgun Surgery, Misplaced Class e God Package*. De seguida com a utilização do modelo da qualidade QMOOD, foram avaliados diagramas de classes ao nível do desenho de software e ainda foi utilizada a ferramenta QCDTool para automatizar estes mecanismos de avaliação.

Deste modo, é possível considerar que a dissertação de Bertrán Macía tem como ponto de ligação com a dissertação redigida no presente documento a deteção de variações entre o código desenvolvido e o que foi representado em diagramas.

Os mecanismos desenvolvidos por Bertrán Macía foram avaliados no contexto de dois estudos experimentais distintos, sendo que o primeiro estudo avaliou a exatidão, precisão e *recall* das estratégias de deteção propostas, apresentando como resultado os benefícios e desvantagens da aplicação.

Por outro lado, o segundo estudo avaliou a utilidade da aplicação do modelo da qualidade QMOOD em diagramas UML. Desta forma, este segundo estudo permitiu identificar nos diagramas de classes as variações das propriedades de desenho e, consequentemente, dos atributos da qualidade nos sistemas analisados (Bertrán Macía, 2009).

Na Figura 5 estão representados os diferentes níveis do QMOOD e as suas relações(Bansiya and Davis, 2002).

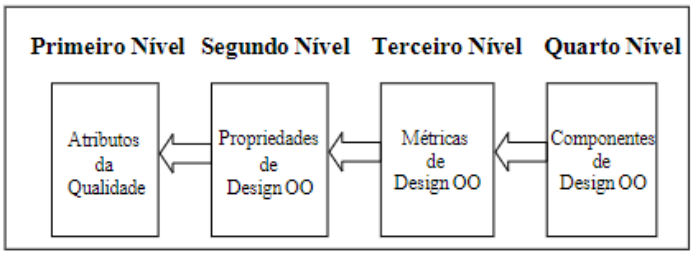


Figura 5 - Níveis e relações no QMOOD

Resumidamente, com esta dissertação de Bertrán Macía é possível concluir que, a análise da qualidade dos sistemas, por norma, baseia-se exclusivamente no código e devido a essa razão a deteção de problemas na análise da qualidade deve ser efetuada durante todas as etapas do projeto, não se focando apenas na implementação.

2.8 Sumário

Neste capítulo, inicialmente, foram clarificados 3 tipos de Revisão de Código: (1) Automatizada, (2) Manual e (3) Mista, sendo também exploradas algumas das ferramentas mais utilizadas como SonaQube e Review board.

De seguida, foram também expostos os 3 tipos de Revisão E Avaliação Do Desenho de Software: (1) Revisão Preliminar Do Desenho, (2) Revisão Crítica Do Desenho e (3) Revisão Do Desenho Programada, acompanhados de uma breve explicação do que são Ferramentas de Verificação Do Desenho e de Teste de Regressão Visual.

Posteriormente, foram investigadas e comparadas ferramentas de Avaliação de Controlo E Qualidade de Diagramas de Software Sincronizados Com O Código como Visual Paradigm e PlantUML, bem como algumas das principais linguagens de programação e modelação como Java, C#, UML e XMI e a sua compatibilidade com as anteriores ferramentas.

Por fim, foram apresentados alguns trabalhos relacionados ao da vigente dissertação realçando alguns aspetos em comum e outros mais divergentes.

3 Análise de Valor

Neste capítulo é exposta a análise de valor da dissertação, desde a identificação e análise da oportunidade, passando pela proposta de valor e terminando análise e abordagem do problema com recurso a um diagrama FAST e ao método AHP com base em critérios que permitam decifrar qual a melhor opção de resolução.

Deste modo, foi aplicado o modelo NCD para auxiliar no processo de identificação e análise da oportunidade. O modelo NCD é composto por cinco elementos do modelo: (1) identificação da oportunidade, (2) a análise da oportunidade, (3) a geração e aperfeiçoamento de ideias, (4) a seleção de ideias e (5) o desenvolvimento do conceito e da tecnologia (Teza *et al.*, 2013).

Adicionalmente aos cinco elementos do modelo, para identificar a proposta a oportunidade também é necessário ter em conta os fatores de influência, como a capacidade da organização, estratégia de negócio, o ambiente externo da organização e ainda pelo fator motor que agrega a liderança e cultura organizacional.

Na Figura 12 (consultar Anexo A) do encontra-se representado o modelo NCD traduzido (Teza *et al.*, 2013).

Na presente secção é apresentada a oportunidade resultante do problema introduzido na secção 1.2 e proposto pelo Lab EQSOFT. A principal oportunidade originada pelo problema visa encontrar a melhor ferramenta ou combinação de ferramentas que permitam saber qual a percentagem de um diagrama que foi refletida no código.

Como referido na secção 0, existem algumas soluções para avaliar a qualidade do código tendo por base a qualidade dos modelos e diagramas elaborados, seja o código direcionado para testes ou para desenvolvimentos. No entanto, nesta avaliação apenas são apontados os possíveis defeitos quer ao nível de desenho quer ao nível de código que refletem as diferenças entre ambos.

Deste modo, nenhuma das soluções previamente exploradas em outros artigos ou dissertações apresentam de modo claro e específico a percentagem de código que está coberto pelo seu respetivo desenho, bem como, a ausência de excertos do código no seu respetivo desenho.

3.1 Análise Da Oportunidade

Nesta secção é explicitada como foi elaborada a análise da oportunidade apresentada na secção **Erro! A origem da referência não foi encontrada.**, para que deste modo seja possível c ompreender se faz sentido dar seguimento ou não à mesma.

Por conseguinte, foi adotada a estratégia denominada de *Strategic framing,* que tem como objetivo determinar quais os pontos fortes e fracos da oportunidade. Neste sentido, foi elaborada uma análise SWOT, representada na Figura 6, que permite identificar quais são as forças e fraquezas, fatores internos, e as oportunidades e ameaças e fatores externos (Fesenfeld *et al.*, 2021)(Hallahan, 2008).

Forças

- Facilidade de pesquisa de ferramentas capazes de gerar código a partir de diagramas;
- Facilidade de pesquisa de ferramentas capazes de gerar diagramas a partir de código;
- Reutilização de ferramentas do Lab ESOFT para avaliar o desenho com sincrónico com o código.

Fraquezas

- Complexidade para perceber qual a funcionalidade ou combinação de funcionalidades em ferramentas de análise de software que permite comparar desenho e o código;
- Inexistência de uma funcionalidade que compare diretamente código com o desenho do mesmo.

Oportunidades

- A procura por uma solução capaz de indicar qual a percentagem de código coberto pelo desenho é complexa;
- Inexistência de conhecimento de algum artigo ou dissertação que aborde o tema desta dissertação em concreto;
- Possibilidade de auxiliar o meio de avaliação da qualidade do software ao nível do desenho no meio académico.

Ameaças

- Inexistência de conhecimento de algum artigo ou dissertação que aborde o tema desta dissertação em concreto e por sua vez torna mais difícil a pesquisa de informação sobre o tema;
- Existência de diversas ferramentas focadas para a análise de código, interface e desenho, mas que não tem como principal foco o tema desta dissertação.

Figura 6 - Análise SWOT

Após a realização desta análise é possível comprovar que as oportunidades identificadas devem ser seguidas, pois irão acrescentar um valor enorme ao meio académico por via de um processo científico.

3.2 Proposta de Valor

Na seguinte secção é apresentado um modelo CANVAS, também denominado de modelo de Osterwalder, com o objetivo de demonstrar de que forma esta dissertação poderá trazer valor a cada uma das partes interessadas, Figura 7 [15].

A proposta de valor pode ser definida como a correlação entre as necessidades dos consumidores, *value for the customer,* e os benefícios que o produto origina com a sua utilização, *benefits*. Desta forma, também é possível analisar qual *perceived value* e os *scraficies* necessários.

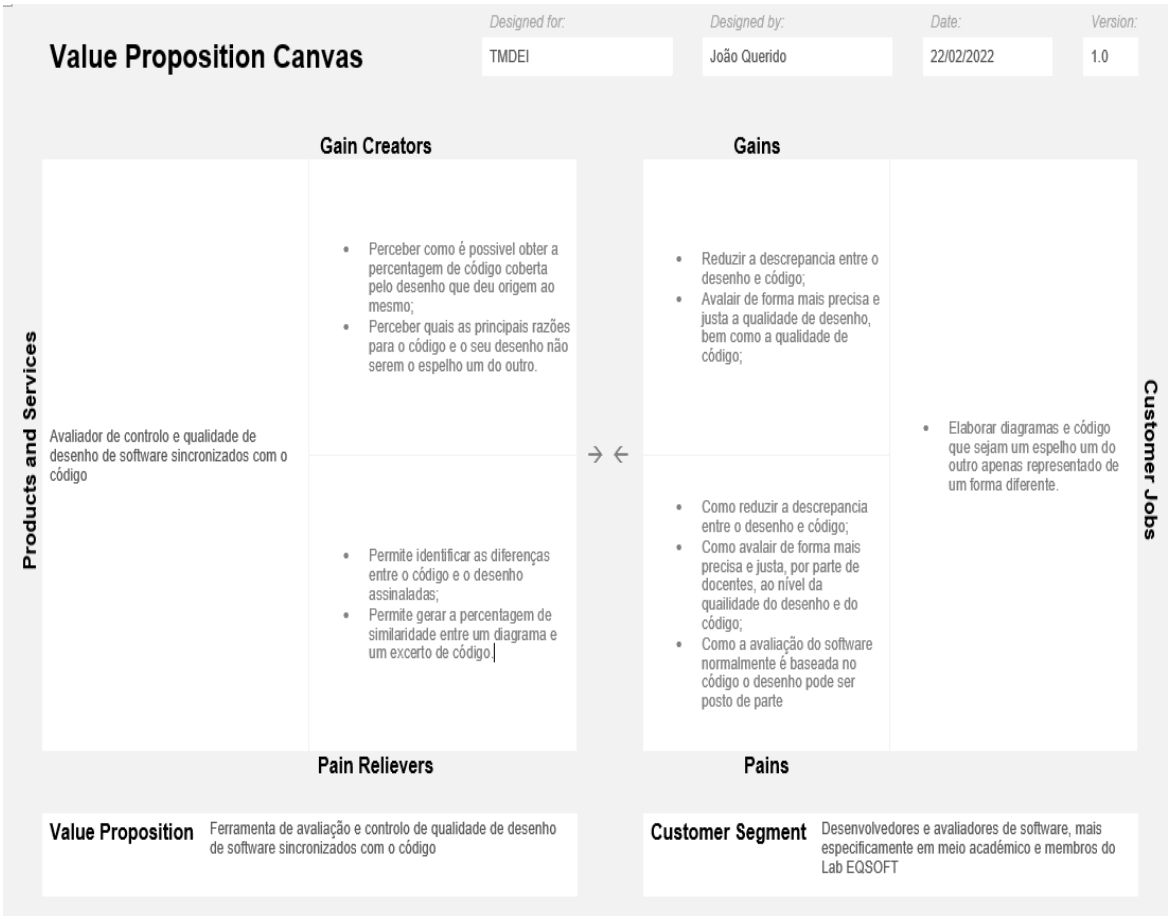


Figura 7 - Modelo de Ostwewalder

[15] https://www.strategyzer.com/

3.3 Métodos E Técnicas

3.3.1 Técnica FAST

Nesta secção é exposta a técnica FAST que se baseia no desenvolvimento de um diagrama para identificar qual o âmbito da dissertação, através da análise e compreensão das funcionalidades, da visualização das relações entre as mesmas e quais as que necessitam de um maior cuidado e enfoque(Sundar, 2020).

Desta forma, o desenvolvimento com recurso ao diagrama FAST torna possível auxiliar a equipa de desenvolvimento relativamente aos seguintes pontos (Nicola, 2017):

- Melhorar a compressão sobre o projeto;
- Identificar quais as funcionalidades em falta;
- Definir, simplificar e clarificar o problema;
- Organizar e compreender as relações entre as funcionalidades;
- Identificar a funcionalidade base do projeto, processo ou produto;
- Melhorar a comunicação e o consenso;
- Estimular a criatividade.

Na Figura 13 [16] (consultar Anexo B) está representado o diagrama de FAST desta dissertação(Borza, 2011)(Porhassann and Tony, 2009).

Em suma, recorrendo ao desenvolvimento deste diagrama foi exequível delinear e definir os passos necessários para alcançar o objetivo definido.

3.3.2 Método AHP

Nesta secção, é apresentado o método AHP, criado por Thomas Saaty no ano de 1980(Saaty and Windt, 1980), que se resume a um método matemático aplicado na tomada de decisão, através de critérios quantitativos e qualificativos (Saaty, 1987).

Este método é divido pelas seguintes sete fases (Nicola, 2018):

1. Construção da árvore hierárquica de decisão;
2. Comparação das alternativas e critérios;
3. Prioridade relativa de cada critério;
4. Avaliar a consistência das prioridades relativas;
5. Construção da matriz de comparação paritária para cada critério, considerando cada uma das alternativas selecionadas;
6. Obter a prioridade composta para as alternativas;
7. Escolha da alternativa;

[16] https://lucid.co

Na primeira fase é necessário perceber a decisão a ser tomada de modo que seja possível iniciar a elaboração da árvore hierárquica. Tendo por base a finalidade desta dissertação, consiste em encontrar a ferramenta ideal para avaliação de diagramas a partir de sincronismo com o código. Também é necessário definir quais são os critérios com maior relevância para este estudo, bem como as alternativas a considerar(Saaty and Windt, 1980).

Tendo por base a investigação previamente elaborada, os critérios a ter em conta são: (1) a possibilidade de gerar de código a partir de diagramas, (2) a possibilidade de gerar de diagramas a partir de código, (3) a possibilidade comparar a similaridade de diagramas, (4) as tecnologias quer ao nível de modelação de diagramas, quer ao nível do código suportados. A árvore hierárquica está representada na Figura 8.

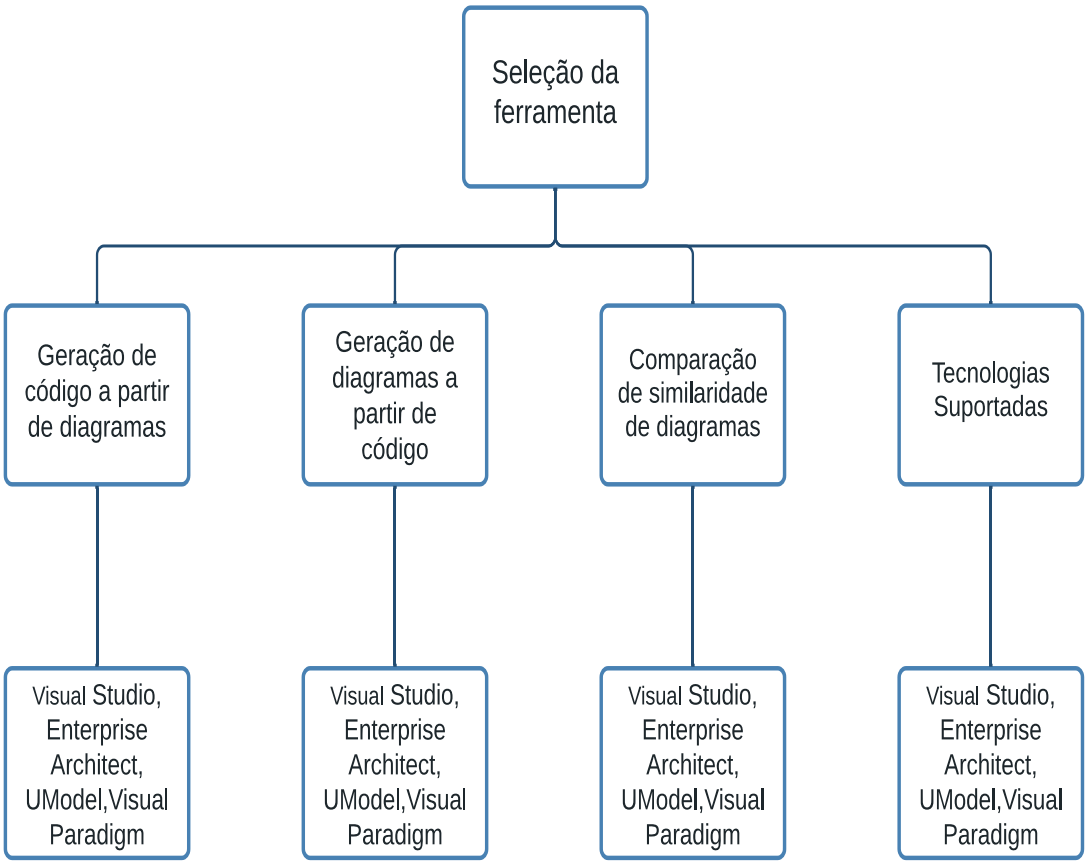


Figura 8 – Arvóre Hierarquica

De seguida, no segundo passo, são estabelecidas as prioridades entre os diversos elementos de cada nível hierárquico, através da elaboração de uma matriz de comparação. Deste modo, será necessário recorrer à Escala Fundamental criado por Thomas Saaty(consultar Anexo C), Tabela 5, para comparar os diversos critérios.

Tabela 5 – Matriz de comparação para os critérios

Critérios	Geração de código a partir de diagramas	Geração de diagramas a partir de código	Comparação de similaridade de diagramas	Tecnologias suportadas
Geração de código a partir de diagramas	1	1/3	1/5	1/3
Geração de diagramas a partir de código	3	1	1	5
Comparação de similaridade de diagramas	5	1	1	5
Padrões de modelação de diagramas	3	1/5	1/5	1
Soma total	**12**	**2 1/2**	**2 2/5**	**11 1/3**

Com base na matriz de comparação é elaborada a seguinte matriz de comparação:

$$A = \begin{bmatrix} 1 & 1/3 & 1/5 & 1/3 \\ 3 & 1 & 1 & 5 \\ 5 & 1 & 1 & 5 \\ 3 & 1/5 & 1/5 & 1 \end{bmatrix}$$

Na terceira etapa do método AHP tem como foco priorizar as demais alternativas, normalizar os valores da matriz de comparação previamente concebida, para igualar todos os critérios a uma mesma unidade, e por fim obter o vetor de prioridades, para identificar a ordem de importância de cada critério(Nicola, 2018).

Na Tabela 6 está representada a matriz de comparação normalizada com a prioridade relativa por cada um dos critérios.

Tabela 6 – Matriz de comparação normalizada com a prioridade relativa para cada um dos critérios

Critérios	Geração de código a partir de diagramas	Geração de diagramas a partir de código	Comparação de similaridade de diagramas	Tecnologias suportadas	**Prioridade Relativa**
Geração de código a partir de diagramas	0.0833	0.1316	0.0833	0.0294	**0.0819**
Geração de diagramas a partir de código	0.2500	0.3947	0.4167	0.4412	**0.3756**
Comparação de similaridade de diagramas	0.4167	0.3947	0.4167	0.4412	**0.4173**
Tecnologias suportadas	0.2500	0.0789	0.0833	0.0882	**0.1251**

Tendo por base os valores obtidos na matriz normalizada foi obtido o seguinte vetor próprio:

$$A = \begin{bmatrix} 1 & 1/3 & 1/5 & 1/3 \\ 3 & 1 & 1 & 3 \\ 5 & 1 & 1 & 5 \\ 3 & 1/3 & 5 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0{,}08 & 0{,}13 & 0{,}08 & 0{,}03 \\ 0{,}25 & 0{,}39 & 0{,}42 & 0{,}44 \\ 0{,}42 & 0{,}39 & 0{,}42 & 0{,}44 \\ 0{,}25 & 0{,}08 & 0{,}08 & 0{,}07 \end{bmatrix} \rightarrow X \cong \begin{bmatrix} 0{,}08 \\ 0{,}38 \\ 0{,}42 \\ 0{,}13 \end{bmatrix}$$

Logo, foi possível auferir que o critério com mais peso é possibilidade comparar a similaridade de diagramas, aproximadamente 0,42, e por outro lado o critério com menos peso, aproximadamente 0,08, a geração de código a partir de diagramas.

Na quarta etapa é necessário avaliar a consistências das prioridades relativas, posto isto é calculada a Razão de Consistência (RC), que permite medir a consistência das avaliações em relação a grandes amostras de juízo aleatórios(Nicola, 2018)(Saaty and Windt, 1980).

A razão de consistência é calculada pela divisão do índice de consistência (IC) por um índice aleatório (IR) como demonstrado na seguinte formula:

$$RC = \frac{IC}{IR} \quad (1)$$

O índice aleatório foi obtido através da utilização do índice calculado para matrizes quadradas de ordem n, sendo que n representa o número de critérios utilizados, Figura 9(Saaty and Windt, 1980)(Nicola, 2018).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Figura 9 - Valores de IR para matrizes quadradas

Dado que o número de critérios é quatro, o valor índice de aleatório a considerar é de 0,9.

De seguida é necessário calcular o índice calculado através da equação (2), em que n representa o número de critérios utilizados e o λ_{max} o maior valor próprio da matriz de comparação par a par.

$$IC = \frac{\lambda_{max} - n}{n - 1} \tag{2}$$

Por conseguinte, para calcular o valor de λ_{max} é determinada média da divisão de cada linha dos vetores resultantes, através da seguinte equação:

$$A * X = \lambda_{max} * X \tag{3}$$

Após a substituição dos valores já conhecidos de A, matriz de comparação, e X, vetor de próprio, a equação traduz em:

$$\begin{bmatrix} 0{,}08 & 0{,}13 & 0{,}08 & 0{,}03 \\ 0{,}25 & 0{,}39 & 0{,}42 & 0{,}44 \\ 0{,}42 & 0{,}39 & 0{,}42 & 0{,}44 \\ 0{,}25 & 0{,}08 & 0{,}08 & 0{,}07 \end{bmatrix} * \begin{bmatrix} 0{,}08 \\ 0{,}38 \\ 0{,}42 \\ 0{,}13 \end{bmatrix} = \lambda_{max} \begin{bmatrix} 0{,}08 \\ 0{,}38 \\ 0{,}42 \\ 0{,}13 \end{bmatrix} \tag{4}$$

Ao simplificar a multiplicação obtemos o seguinte resultado:

$$\begin{bmatrix} 0{,}33 \\ 1{,}41 \\ 1{,}83 \\ 0{,}58 \end{bmatrix} = \lambda_{max} \begin{bmatrix} 0{,}08 \\ 0{,}38 \\ 0{,}42 \\ 0{,}13 \end{bmatrix} \tag{5}$$

Para calcular o valor de λ_{max} é necessário colocar o mesmo em evidencia e efetuar o cálculo da média da divisão de cada linha dos vetores resultantes:

$$\lambda_{max} = \frac{(0{,}33/0{,}08) + (1{,}41/0{,}38) + (1{,}83/0{,}42) + (0{,}58/0{,}13)}{4} \approx 4{,}21 \tag{6}$$

Visto que já foi calculado o λ_{max} , é utilizado o seu valor para prosseguir com calculo do índice de consistência:

$$IC = \frac{4{,}17 - 4}{4 - 1} \approx 0{,}07 \quad (7)$$

Finalmente, com a obtenção do valor do índice de consistência aliado ao já obtido valor de índice aleatório obtemos a seguinte razão de consistência:

$$RC = \frac{0{,}07}{0{,}9} \approx 0{,}08 \quad (8)$$

Em suma, como o valor da razão de consistência é menor que 0,1 comprava que os valores das prioridades relativas utilizados são consistentes(Nicola, 2018).

No quinto passo é necessário repetir o processo anterior de: (1) calcular as matrizes de comparação, (2) proceder à normalização das mesmas e (3) obter o vetor próprio de cada critério, mas desta vez em relação a cada uma das quatro alternativas.

Para cada critério foram elaboradas as Tabelas de 7 a 14, respetivamente para a matriz de comparação e para a matriz de comparação normalizada associado ao respetivo vetor próprio.

Tabela 7 – Matriz de comparação para o critério gerar de código a partir de diagramas

Geração de código a partir de diagramas	Visual Studio	Enterprise Architect	UModel	Visual Paradigm
Visual Studio	1	3	5	3
Enterprise Architect	1/3	1	5	3
UModel	1/5	1/5	1	1/5
Visual Paradigm	1/3	1/3	5	1
Soma total	**1 6/7**	**4 1/2**	**16**	**7 1/5**

Tabela 8 – Matriz de comparação normalizada com a prioridade relativa para o critério gerar de código a partir de diagramas

Geração de código a partir de diagramas	Visual Studio	Enterprise Architect	UModel	Visual Paradigm	**Prioridade Relativa**
Visual Studio	0.53571	0.66176	0.31250	0.41667	**0.4817**
Enterprise Architect	0.17857	0.22059	0.31250	0.41667	**0.2821**
UModel	0.10714	0.04412	0.06250	0.02778	**0.0604**
Visual Paradigm	0.17857	0.07353	0.31250	0.13889	**0.1759**

Tabela 9 – Matriz de comparação para o critério gerar de diagramas a partir de código

Geração de diagramas a partir de código	Visual Studio	Enterprise Architect	UModel	Visual Paradigm
Visual Studio	1	1/5	1/3	1/5
Enterprise Architect	5	1	5	1/3
UModel	3	1/5	1	1/5
Visual Paradigm	5	3	5	1
Soma total	**14**	**4 2/5**	**11 1/3**	**1 3/4**

Tabela 10 – Matriz de comparação normalizada com a prioridade relativa para o critério de gerar de diagramas a partir de código

Geração de diagramas a partir de código	Visual Studio	Enterprise Architect	UModel	Visual Paradigm	**Prioridade Relativa**
Visual Studio	0.0714	0.0455	0.0294	0.1154	**0.0654**
Enterprise Architect	0.3571	0.2273	0.4412	0.1923	**0.3045**
UModel	0.2143	0.0455	0.0882	0.1154	**0.1158**
Visual Paradigm	0.3571	0.6818	0.4412	0.5769	**0.5143**

Tabela 11 – Matriz de comparação para o critério comparação de similaridade de diagramas

Comparação de similaridade de diagramas	Visual Studio	Enterprise Architect	UModel	Visual Paradigm
Visual Studio	1	1/5	1/3	1/5
Enterprise Architect	5	1	5	3
UModel	3	1/5	1	1/5
Visual Paradigm	5	1/3	5	1
Soma total	**14**	**1 3/4**	**11 1/3**	**4 2/5**

Tabela 12 – Matriz de comparação normalizada com a prioridade relativa para o critério de comparação de similaridade de diagramas

Comparação de similaridade de diagramas	Visual Studio	Enterprise Architect	UModel	Visual Paradigm	**Prioridade Relativa**
Visual Studio	0.0714	0.1154	0.0294	0.0455	**0.0654**
Enterprise Architect	0.3571	0.5769	0.4412	0.6818	**0.5143**
UModel	0.2143	0.1154	0.0882	0.0455	**0.1158**
Visual Paradigm	0.3571	0.1923	0.4412	0.2273	**0.3045**

Tabela 13 – Matriz de comparação para o critério das tecnologias suportadas

Tecnologias suportadas	Visual Studio	Enterprise Architect	UModel	Visual Paradigm
Visual Studio	1	3	5	5
Enterprise Architect	1/3	1	5	3
UModel	1/5	1/5	1	1/3
Visual Paradigm	1/5	1/3	3	1
Soma total	**1 3/4**	**4 1/2**	**14**	**9 1/3**

Tabela 14 – Matriz de comparação normalizada com a prioridade relativa para o critério das tecnologias suportadas

Tecnologias suportadas	Visual Studio	Enterprise Architect	UModel	Visual Paradigm	**Prioridade Relativa**
Visual Studio	0.5769	0.6618	0.3571	0.5357	**0.5329**
Enterprise Architect	0.1923	0.2206	0.3571	0.3214	**0.2729**
UModel	0.1154	0.0441	0.0714	0.0357	**0.0667**
Visual Paradigm	0.1154	0.0735	0.2143	0.1071	**0.1276**

Após este estudo procedeu se à alteração da árvore hierárquica com a adição dos pesos tanto para os critérios como para as alternativas, Figura 10:

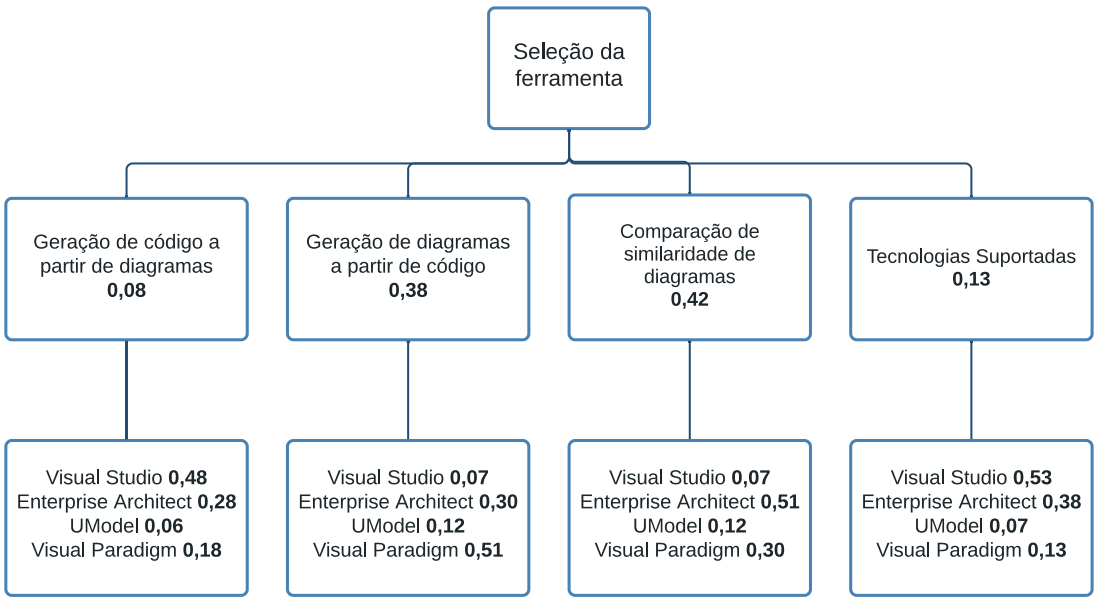


Figura 10 - Árvore hierárquica com pesos

De seguida, para obter as propriedades compostas por alternativa, através da multiplicação do vetor de prioridade de cada critério, pelo respetivo vetor de prioridade de cada alternativa pelo mesmo critério:

$$\begin{bmatrix} 0{,}48 & 0{,}07 & 0{,}07 & 0{,}53 \\ 0{,}28 & 0{,}30 & 0{,}51 & 0{,}27 \\ 0{,}06 & 0{,}12 & 0{,}12 & 0{,}07 \\ 0{,}18 & 0{,}51 & 0{,}30 & 0{,}13 \end{bmatrix} * \begin{bmatrix} 0{,}08 \\ 0{,}38 \\ 0{,}42 \\ 0{,}13 \end{bmatrix} = \begin{bmatrix} 0{,}16 \\ 0{,}39 \\ 0{,}11 \\ 0{,}35 \end{bmatrix} \quad (9)$$

Finalmente, após este cálculo foi possível perceber que as ferramentas mais aptas em todos os aspetos analisados são a Enterprise Architect com 0,39 e o Visual Paradigm com 0,35 apenas separadas por 0,04.

3.4 Sumário

No capítulo da Análise Valor é analisada a oportunidade apresentada no capítulo 1, com a utilização da análise SWOT, a proposta de valor com o auxílio do modelo de Ostwewalder e ainda é apresentada a técnica FAST e o método AHP.

Por conseguinte foram assinaladas como principais ferramentas a ser avaliadas a Enterprise Architect e o Visual Paradigm.

4 Análise E Desenho Da Solução

Neste capítulo é exposta a análise do problema apresentado na secção 1.2, através de uma apreciação sobre o domínio do mesmo. Numa fase seguinte serão especificados os requisitos funcionais e não funcionais. Por fim, será definido o desenho da solução, através das diversas vistas do sistema, permitindo compreender como o mesmo irá funcionar.

4.1 Domínio Do Problema

De modo a que seja possível representar e compreender as relações entre as entidades do mundo real recorre-se à construção de um modelo de domínio. A construção do modelo de domínio só é possível através da compreensão dos requisitos ao nível do sistema, da identificação e representação das relações entre as entidades (Koopman, 2020).

Na Figura 11 está representado o modelo domínio do sistema, que visa encontrar a melhor ferramenta para o problema em questão.

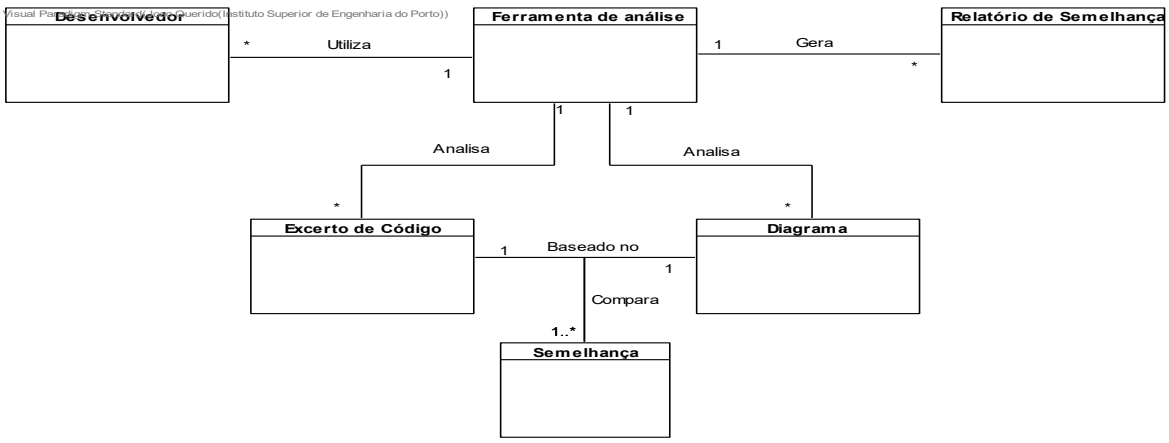


Figura 11 - Modelo de domínio

Através do modelo de domínio elaborado é possível concluir que o principal objetivo da ferramenta é a geração do relatório de semelhança. Em síntese, para a geração do relatório a ferramenta irá comparar o grau de semelhança entre o excerto de código e o diagrama providenciados pelo desenvolvedor, bem como os critérios de semelhança a considerar.

4.2 Requisitos Funcionais e Não Funcionais

Ainda que o problema da presente dissertação não resulte no desenvolvimento de uma nova ferramenta, é necessário identificar o que as ferramentas escolhidas para análise devem permitir fazer, bem como as suas limitações.

Na subseções 4.2.1 e 0encontram-se todos os requisitos que as ferramentas devem incluir sendo estes funcionais e não funcionais.

4.2.1 Requisitos Funcionais

Em resumo, os requisitos funcionais, assim como o nome pressupõe, refletem as funcionalidades que as ferramentas estão qualificadas a realizar. Deste modo, é devido as estas funcionalidades que o sistema das ferramentas é definido de um modo geral.

Os requisitos funcionais encontram-se descritos no presente documento sobre a forma de UC's, possibilitando constatar de que forma o utilizador interage com a ferramenta.

No caso em concreto da vigente dissertação, o único requisito funcional tem como finalidade automatizar ao máximo o cenário típico em que a comparação entre o excerto de código escrito manualmente e o diagrama previamente elaborado, também manualmente, é concebida através de uma comparação visual entre os dois.

Na Figura 12 encontra-se representado o diagrama de casos correspondente às funcionalidades assinaladas como base principal da ferramenta a utilizar.

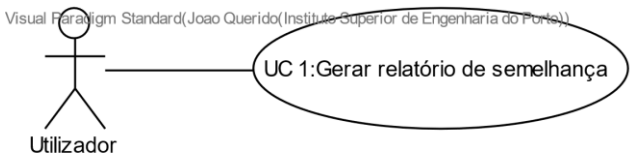


Figura 12 – Diagrama de casos de uso

UC1: Gerar relatório de semelhança
Em última instância, do mesmo modo que os dois casos de uso descritos este caso de uso pode ser desempenhado por qualquer utilizador que tenha acesso à ferramenta e aos diagramas UML. Neste caso de uso o objetivo é que o utilizador seja capaz de gerar um relatório em que sejam identificadas as diferenças entre os dois diagramas e que, deste modo, seja gerado uma (%) de similaridade entre os dois.

Na Tabela 15 encontra-se uma especificação mais aprofundada e específica do caso de uso através da apresentação do formato completo

Tabela 15 - UC1: Gerar relatório de semelhança

ARTEFACTO	**DESCRIÇÃO**
ATOR	Utilizador
OBJETIVO	Gerar um relatório de semelhança entre um diagrama elaborado previamente e o excerto de código correspondente.
PRÉ-CONDIÇÕES	O diagrama e o excerto de código devem existir e ter sido elaborados manualmente.
PÓS-CONDIÇÕES	O relatório será gerado num ficheiro pdf.
FLUXO BÁSICO	1. O utilizador inicia a ferramenta; 2. A ferramenta solicita o diagrama e o excerto de código; 3. O utilizador fornece o diagrama e o excerto de código; 4. A ferramenta mostra o relatório de semelhança.
FLUXO ALTERNATIVO	3a. O utilizador pede à ferramenta a tradução do excerto de código para o formato de um diagrama; 4a. A ferramenta exporta os dois diagramas em formato XMI e, auxiliada por um script, gera o relatório de semelhança.

4.2.2 Requisitos Não Funcionais

Os requisitos não funcionais serão baseados no padrão FURPS (Medin *et al.*, 2023) que é constituído por cinco categorias:

- (F)*untionality*;
- (U)*sability*;
- (R)*eability;*
- (P)*erformance;*
- (S)u*pportability*;

Cada uma destas categorias será apresentada seguidamente em maior detalhe, bem como, o seu enquadramento na ferramenta a utilizar.

1.Funcionalidade

Tem como objetivo identificar funcionalidades que não se enquadram nos casos de uso descritos nos requisitos funcionais. Na presente dissertação foram identificados os subsequentes requisitos:

- O relatório de semelhança deve ser gerado no formato PDF;
- A geração de diagramas e código deverá ser possível em mais do que uma linguagem.

2.Usabilidade
É também relevante avaliar a usabilidade do software para com o seu utilizador final, ou seja, a interação com a interface. Foi apenas identificado nesta categoria como requisito que a ferramenta a utilizar incorpore uma interface acessível, intuitiva e de utilização trivial.

3.Confiabilidade
É necessário aferir a confiabilidade que abrange a integridade e compatibilidade de todo o sistema. Alguns dos requisitos nesta categoria são a gravidade das falhas, recuperação e previsibilidade. Neste contexto, não foram identificados nenhuns requisitos pois a ferramenta a utilizar não os prevê.

4.Desempenho
Ao nível do desempenho, requisitos como a duração do tempo de resposta e recursos consumidos são os principais constituintes desta categoria. Na vigente dissertação foram considerados os seguintes requisitos:

- Tempo de reposta inferior a cinco segundos para gerar um diagrama com base no excerto de código;
- Tempo de reposta necessário inferior a cinco segundos para gerar um código com base no diagrama.

Considerou-se que 5 segundos seria o tempo de resposta máximo para manter uma experiência de utilização razoável.

5.Suportabilidade
A suportabilidade avalia diversos parâmetros como a testabilidade, adaptabilidade, manutenibilidade, compatibilidade e escalabilidade. Contudo apenas foram considerados os seguintes requisitos no âmbito da presente dissertação:

- A ferramenta deve permitir a integração de *plugins*, extensões e outras ferramentas;
- A ferramenta deve adaptar as suas funcionalidades às mais diversas linguagens de programação.

4.3 Desenho

O vigente subcapítulo está dividido em duas partes, sendo estas, arquitetura da solução e comparação da arquitetura e características das duas ferramentas. O objetivo é compreender a arquitetura da solução a implementar e a sua estrutura, apresentando assim, para cada uma das ferramentas, as vistas de implementação e uma comparação das suas características.

4.3.1 Arquitetura Da Solução

De modo a apresentar a arquitetura da solução escolhida para o problema da presente dissertação, procedeu-se a elaboração de um diagrama de implementação de alto nível (Figura 13).

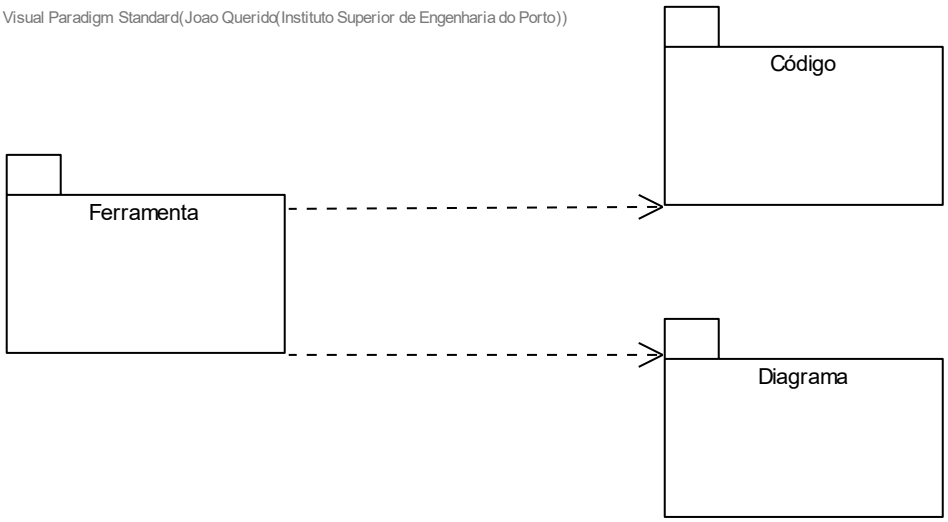


Figura 13 - Vista de implementação da solução

Nesta vista de implementação, tal como no Modelo Domínio, é possível denotar que a implementação da solução apenas depende do excerto de código e do diagrama, ou seja, ferramenta irá gerar o relatório de semelhança após ser providenciado o excerto de código e o diagrama elaborados previamente e manualmente.

4.3.2 Comparação Da Arquitetura e Características Das Diferentes Ferramentas

Para que seja possível avaliar e comparar a arquitetura das ferramentas Visual Paradigm e Enterprise Architect também foram desenhadas a vistas de implementação de ambas (Figura 14 e 15)

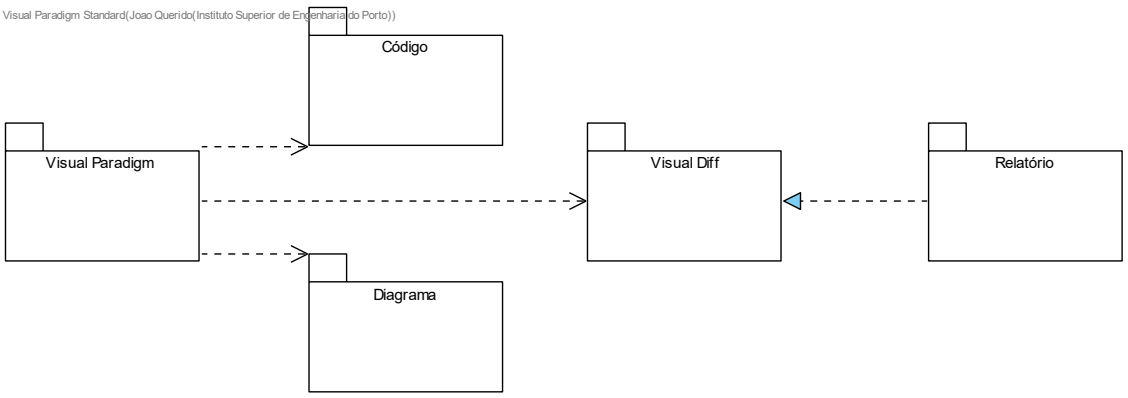


Figura 14 - Vista de implementação da ferramenta Visual Paradigm

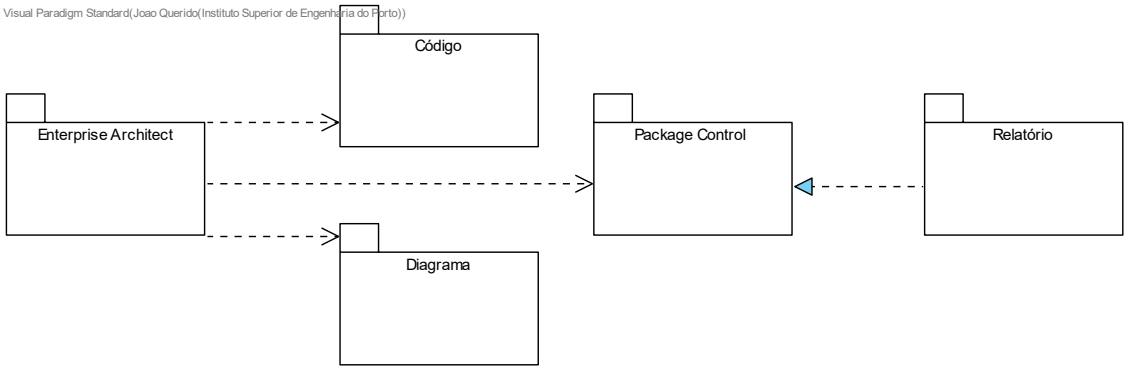


Figura 15 - Vista de implementação da ferramenta Enterprise Architect

Apesar de terem funcionalidades destintas para a comparação e posterior geração de um relatório, Visual DIff no caso do Visual Paradigm e Package Control na ferramenta Entrepise Architect, o desfecho final idêntico, com a geração de um relatório em que é dado enfase ás diferenças entre o diagrama e o excerto de código transformado em diagrama.

Ao nível das características entre as duas ferramentas não foram denotas diferenças arquiteturais significativas, posto isto ao invés de selecionar uma das mesmas para ser avaliada ao nível da implementação no Capítulo 5 “Implementação Da Solução” , serão ambas avaliadas e posteriormente comparadas.

Deste modo, será possível identificar a ferramenta mais adequada para o problema da vigente solução, tendo como alternativa a ferramenta não selecionada.

4.4 Sumário

No capítulo da Análise E Desenho Da Solução foi apresentado o modelo domínio, de modo a identificar qual o principal requisito do sistema da ferramenta a utilizar para colmatar o problema da vigente dissertação e de que modo interagem entre si as diferentes entidades.

Após a elaboração do modelo domínio foram identificados os requisitos funcionais e não funcionais. Foi definido o caso de uso e foram utilizadas as diferentes categorias do padrão FURPS para assinalar os requisitos não funcionais da solução.

Em última instância, é apresentada a arquitetura das duas soluções e realizada uma comparação entre as características das duas, indicando por sua vez que em ao invés de selecionar uma delas serão ambas avaliadas ao nível da implementação.

5 Implementação Da Solução

Neste capítulo será descrita a solução adotada para a resolução do problema e, por fim, será apresentada uma apreciação ao trabalho realizado.

5.1 Descrição da Implementação

No vigente subcapítulo situa-se a descrição detalhada da solução implementada. Para tornar a leitura mais acessível, o capítulo foi dividido em quatro secções. Deste modo, será possível expor com maior clareza como a solução será aplicada nas diferentes ferramentas escolhidas previamente, bem como as conclusões a retirar.

Denote-se que a solução escolhida incide na utilização da ferramenta Visual Paradigm, contudo ambas as ferramentas foram testadas e avaliadas. Na secção específica de cada ferramenta, é apresentado todo o procedimento necessário para a geração de um relatório de similaridade. Na secção de conclusões será apresentada informação adicional, incluindo as vantagens, desvantagens e a justificação para a escolha da ferramenta Visual Paradigm.

Para a geração do relatório de semelhança é de realçar que foram considerados quer a geração do mesmo a partir da comparação de dois diagramas de sequência, bem como, a partir da comparação de dois excertos de código. Ainda foram consideradas como relevantes na escolha da ferramenta as funcionalidades de *reverse engineering* e de geração de código.

Por conseguinte, o fluxo de teste e avaliação em ambas a ferramenta avançou de forma similar por meio dos seguintes passos:

1. Verificação da comparação entre diagramas na ferramenta é apenas manual ou automática? É possível gerar um relatório após essa comparação?

2. A ferramenta providencia funcionalidades de *reverse engineering*, ou seja, geração de diagramas a partir de código? Se sim de que forma, quais os tipos de diagramas capazes de gerar e qual a sua compatibilidade com as demais linguagens de programação?
3. A ferramenta possui mecanismo de geração de código? Se sim a partir de que tipo de diagramas é capaz de gerar código e em que linguagens de programação é possível ser gerado?

5.1.1 Visual Paradigm

A ferramenta Visual Paradigm já descrita e analisada no Capítulo 2, Estado da Arte, foi uma das duas ferramentas escolhida para teste e avaliação devido a dois principais fatores:

1. Ferramenta com segunda maior pontuação aquando da utilização do método AHP no Capítulo 3, Análise de Valor, com uma pontuação de 0.35;
2. Ferramenta com que os docentes e alunos da instituição ISEP estão familiarizados e está disponibilizada uma licença, ainda que *standard*, acessível a todos.

De modo a sintetizar e auxiliar a compressão da implementação da solução nesta ferramenta, encontram-se abaixo as cinco diferentes etapas elaboradas, cuja descrição é complementada com imagens autoexplicativas. Por fim, serão apresentadas a principais vantagens e desvantagens da utilização desta ferramenta para a implementação pretendida.

1.Elaboração do diagrama base

Em primeira instância foi elaborado manualmente um diagrama de sequência (Figura 16) que, em teoria, será o diagrama em que o desenvolver se irá basear aquando do desenvolvimento da funcionalidade.

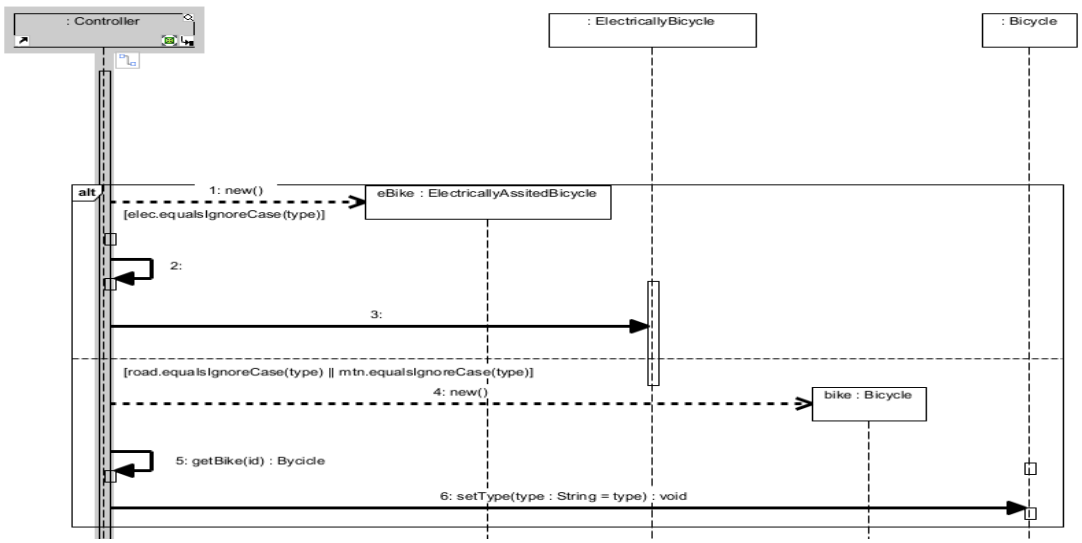


Figura 16 - Diagrama elaborado manualmente

2.Elaboração do diagrama a partir de código

De seguida será necessário gerar um diagrama automaticamente que representa o código elaborado. Isto apenas é necessário pois a ferramenta Visual Paradigm não possui nenhuma funcionalidade que compare, diretamente, excerto de código com um diagrama de sequência.

Esta funcionalidade apenas está acessível em duas versões da ferramenta: *Enterprise* e *Professional*, tendo ainda a limitação de apenas disponibilizar que sejam gerados diagramas de sequência se o código for elaborado na linguagem de programação Java.

Para aceder a esta funcionalidade é necessário navegar para o separador "*Tools*", selecionar a opção "*Code*" e dentro irá aparecer a opção "*Instant Reverse Java to Sequence Diagram…*", tal como ilustra a Figura 17.

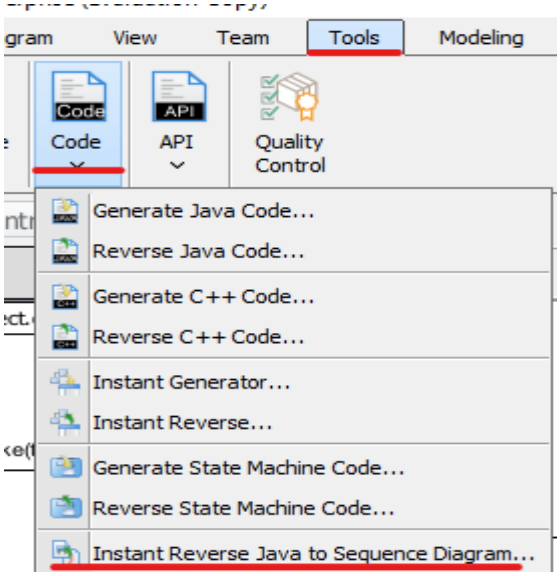


Figura 17 - Funcionalidade de geração automática de diagramas

Após selecionar a opção será aberta uma janela onde é requisitado o caminho para o projeto que contém o excerto de código que será utilizado para gerar o diagrama. Também será necessário selecionar qual o primeiro método da classe Java que será traduzido, como está exposto nas Figuras 18 e 19.

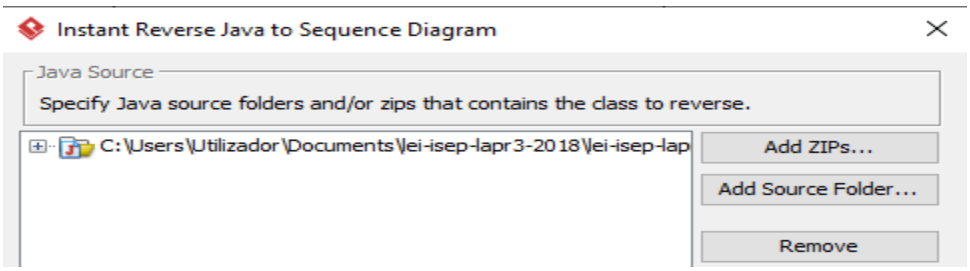


Figura 18 - Janela do caminho para o projeto do código base

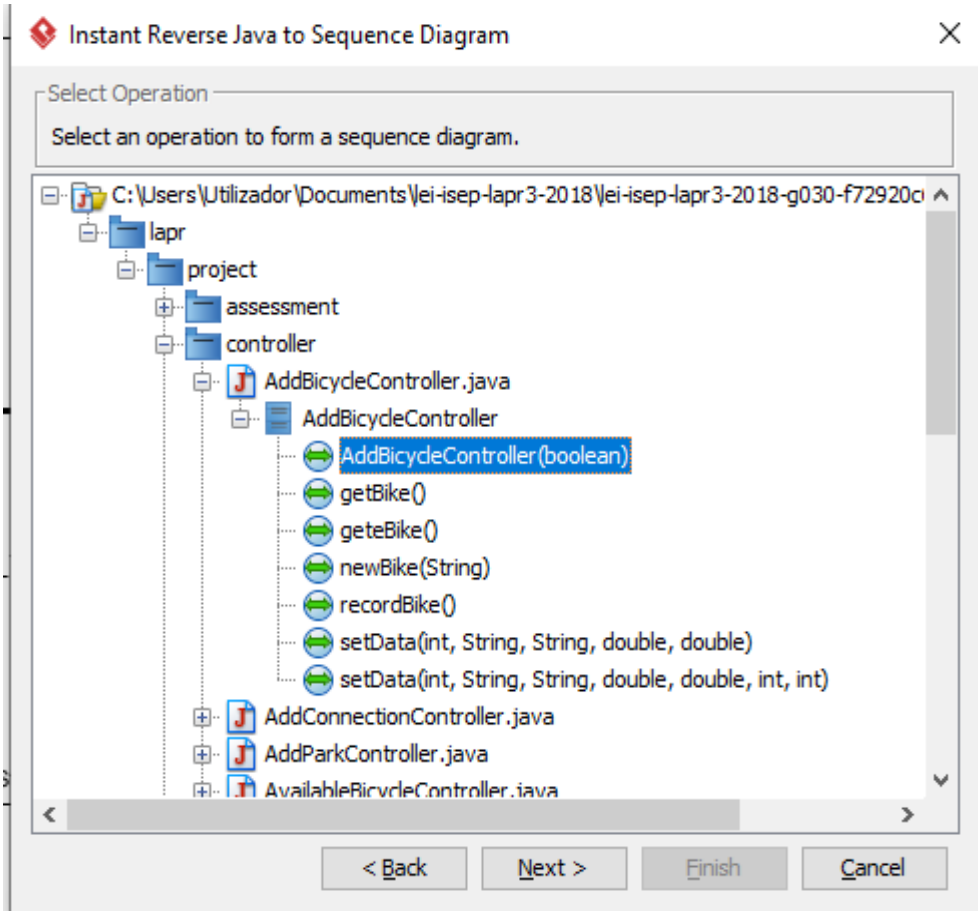


Figura 19 - Janela do caminho para o primeiro método da classe Java

Por fim, o Visual Paradigm disponibiliza também a opção de gerar o restante diagrama selecionando cada uma das já existentes "*Messages*", como é demonstrado na Figura 20.

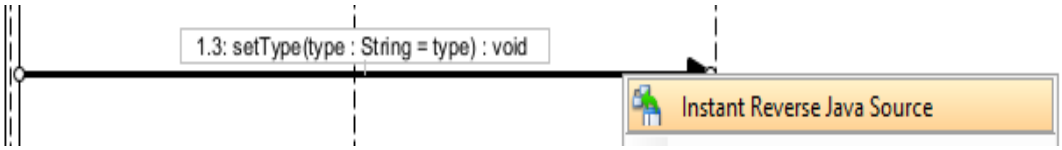


Figura 20 - Funcionalidade "Instant *Reverse Java Source*"

3.Comparação dos diagramas

Após a elaboração do diagrama a partir de código é necessário comparar os mesmos e para isso foram analisadas e testadas duas alternativas:

1. Utilização da funcionalidade Visual Diff do Visual Paradigm;
2. Exportação dos diagramas no formato XMI e, através de um *powershell script,* os dois são comparados.

Para aceder à funcionalidade Visual Diff é necessário navegar para o separador "Modeling", selecionar a opção "Visual Diff" e dentro irá ter uma janela que permite selecionar os dois diagramas, como é demonstrado nas Figuras 21 e 22.

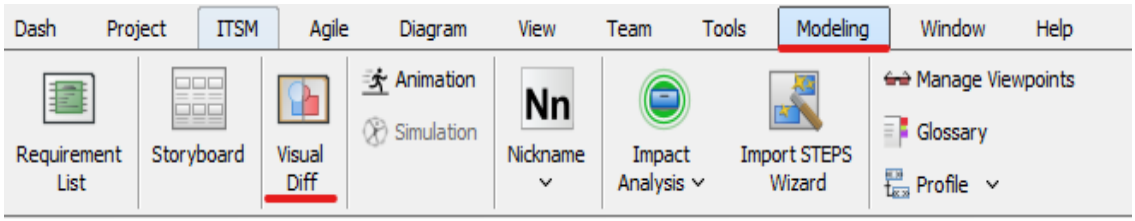


Figura 21 - Funcionalidade Visual Diff

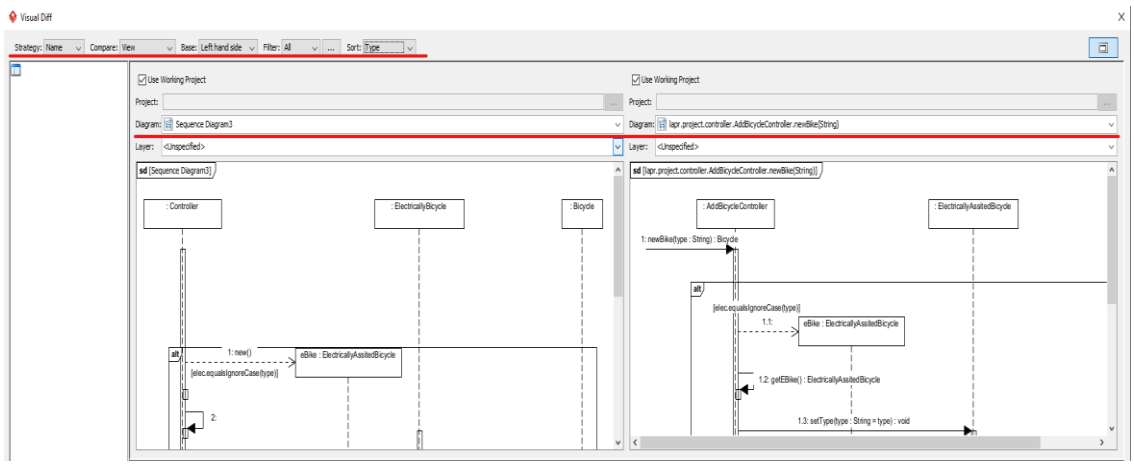


Figura 22 – Janela de comparação do Visual Diff

O Visual Diff disponibiliza filtros pelo tipo de modificações por novas, eliminadas ou alteradas, qual a estratégia a adotar como ID, nome ou *transitor* e ainda ordenar as diferenças por nome ou tipo.

Por outro lado, com a exportação dos diagramas para no formato XMI, aliado ao script elaborado com base em algoritmos encontrados na *Web* como o *LevenshteinDistance*(Nam, 2019) e (Øyvind Kallstad, 2016)serão analisadas as diferenças entre os dois ficheiros.

Na Figura 23 abaixo é possível visualizar os passos necessários para exportar o diagrama no formato XMI.

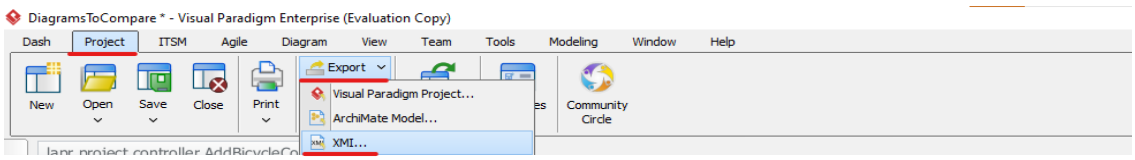


Figura 23 – Exportar diagramas no formato XMI

Na Figura 24 é possível visualizar o script elaborado inicialmente para gerar uma (%) de similaridade entre os dois ficheiros.

```powershell
# Define the paths to the two XMI files you want to compare
$file1Path = "C:\Users\Utilizador\Desktop\GenerateCode\file1.xmi"
$file2Path = "C:\Users\Utilizador\Desktop\GenerateCode\file2.xmi"

# Read the contents of the XMI files as text
$file1Content = Get-Content $file1Path
$file2Content = Get-Content $file2Path

# Convert the contents to lowercase for case-insensitive comparison
$file1Content = $file1Content.ToLower()
$file2Content = $file2Content.ToLower()

# Function to calculate percentage similarity
function Get-Percentage-Similarity ($s1, $s2) {
    $commonCharacters = 0
    $totalCharacters = [math]::max($file1Content.Length, $file2Content.Length)

    for ($i = 0; $i -lt $file1Content.Length; $i++) {
        if ($file2Content.Contains($file1Content[$i])) {
            $commonCharacters++
        }
    }

    return ($commonCharacters / $totalCharacters) * 100
}

# Calculate the percentage similarity
$similarityPercentage = Get-Percentage-Similarity $file1Content $file2Content

# Output the result
Write-Host "Percentage Similarity: $similarityPercentage%"
```

Figura 24 - *Script*

4.Geração do relatório de similaridade

Desta forma, para que seja gerado o relatório de similaridade, é necessário ter por base as duas diferentes abordagens para a comparação dos relatórios, pois se no caso da funcionalidade Visual Diff é através da opção "*Export PDF…*" (Figura 25) que será gerado o relatório no formato PDF que contém os dois diagramas e a lista das diferenças entre ambos.

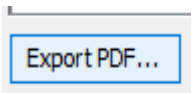


Figura 25 – Funcionalidade de exportação

Em contrapartida, na utilização do script procedeu-se apenas a uma alteração (Figura 26) para ser possível exportar o resultado também para um ficheiro PDF, através da criação de um report em HTML e com inspiração em funções como o *ConvertToPDF*(Silva, 2016).

```
# Create an HTML report with the result
$htmlContent = @"
<!DOCTYPE html>
<html>
<head>
    <title>Similarity % Comparison Report</title>
</head>
<body>
    <h1>Similarity % Comparison Report</h1>
    <p>Percentage Similarity: $similarityPercentage%</p>
</body>
</html>
"@

# Specify the path where you want to save the HTML report
$htmlReportPath = "C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.html"

# Save the HTML content to the specified file
$htmlContent | Set-Content -Path $htmlReportPath -Force

# Specify the path where you want to save the PDF report
$pdfReportPath = "C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.pdf"

# Convert the HTML report to a PDF using wkhtmltopdf
Start-Process -FilePath "C:\Program Files\wkhtmltopdf\bin\wkhtmltopdf.exe" -ArgumentList "$htmlReportPath $pdfReportPath" -Wait

# Display a message indicating that the report has been generated
Write-Host "HTML report has been generated and saved to $htmlReportPath"
Write-Host "PDF report has been generated and saved to $pdfReportPath"
```

Figura 26 - Alteração ao script

5.Avaliação

Em última instância, foi elaborado o Quadro 24 onde se encontram enumeradas quais as vantagens e desvantagens da utilização da ferramenta Visual Paradigm.

Quadro 24 - Vantagens e desvantagens da ferramenta Visual Paradigm

Vantagens	**Desvantagens**
▪ Comparação dos diagramas lado a lado; ▪ Alargada escolha de filtros e estratégias na funcionalidade Visual Diff; ▪ Permite a geração de diagramas de sequência a partir de código, funcionalidade *Instant Reverse Java to Sequence Diagram*; ▪ Permite a geração de diagramas de classes a partir de diferentes linguagens de programação como Java, C, C#, XML e Hibernate; ▪ Permite a geração de código em Java, C++, C#, PHP, entre outros, tendo por base um diagrama de classes.	▪ Visual Diff tem algumas falhas caso a estratégia de comparação não seja a mais adequada pois o tamanho das ligações pode ser considerado uma diferença; ▪ O relatório gerado pelo Visual Diff não calcula a (%) de similaridade apenas anota as diferenças; ▪ A funcionalidade *Instant Reverse Java to Sequence Diagram* apenas está acessível nas versões *Enterprise* e *Professional*;

5.1.2 Enterprise Architect

A ferramenta Enterprise Architect também já descrita e analisada no Capítulo 2, Estado da Arte, foi uma das duas ferramentas escolhida para teste e avaliação devido a ter obtido a melhor pontuação aquando da utilização do método AHP no Capítulo 3, Análise de Valor, com uma pontuação de 0,39.

De forma similar ao que foi elaborado para a ferramenta Visual Paradigm encontram-se abaixo as cinco diferentes etapas elaboradas, complementadas com imagens autoexplicativas. De seguida serão também apresentadas as principais vantagens e desvantagens da utilização desta ferramenta para a implementação pretendida.

1.Elaboração do diagrama base

Inicialmente foi elaborado manualmente um diagrama de sequência, que em teoria será o diagrama em que o desenvolver se irá basear aquando do desenvolvimento da funcionalidade, idêntico ao que se encontra representado na Figura 16.

2.Elaboração do diagrama a partir de código

O Entrerprise Architect não inclui por base uma funcionalidade que permita gerar diagramas de sequência a partir de código, contudo após uma pesquisa mais aprofundada através da integração de *plug-ins* já é permitido.

Neste caso em concreto o *plug-in* é o "*HSDc Seq* "que, como já referido anteriormente, permite que sejam gerados diagramas de sequência em UML através de código, dando ainda possibilidade de escolher qual o nível de profundidade desejado. Um exemplo seria *Controller*, *Service*, *Model*, *Repository* que equivale ao nível 4.

Para iniciar o processo de geração do diagrama é necessário criar um projeto base de diagramas de sequência e apagar todos os diagramas gerados automaticamente pela ferramenta, pois apenas é obrigatório ter um projeto base para agregar o código e os diagramas que serão gerados posteriormente.Nas Figuras 27 e 28 está representado o processo acima assinalado:

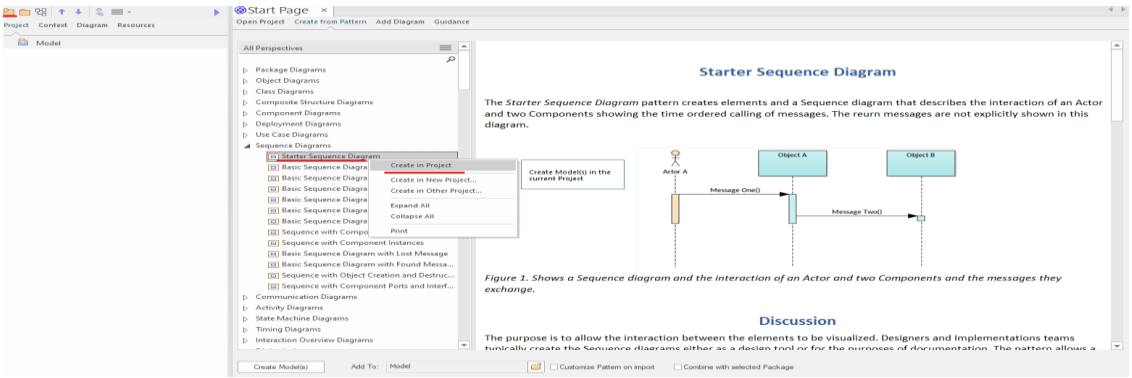


Figura 27 - Criação projeto base

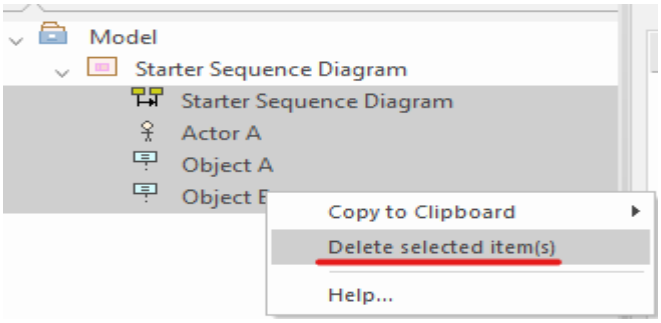


Figura 28 - Eliminação dos diagramas gerados automaticamente

Após a geração do projeto base estar terminada é também necessário que seja importado o código que servirá como base para a geração do diagrama de sequência final. Desta forma é necessário indicar onde se encontra o mesmo bem como a sua linguagem de programação.

As Figuras 29 e 30 ilustram os passos necessários a efetuar:

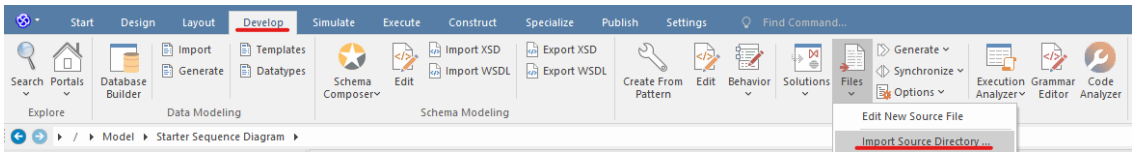


Figura 29 - Como importar o código

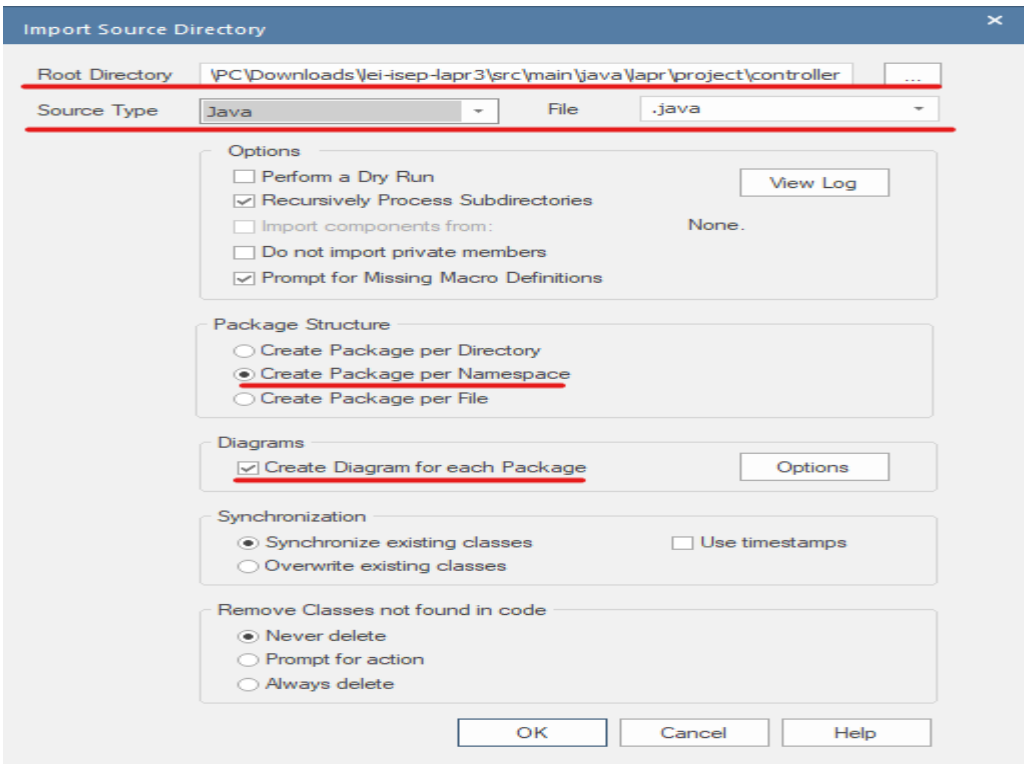


Figura 30 - Menu para importar código

Por fim, após todo o código ser importado, é necessário escolher um método inicial para gerar o diagrama através do *plugin* "*HSDc Seq*" acima mencionado. Esta funcionalidade, se instalada corretamente, estará disponível no menu "*Specialize*" aquando da seleção do método inicial, como está demonstrado na Figura 31.

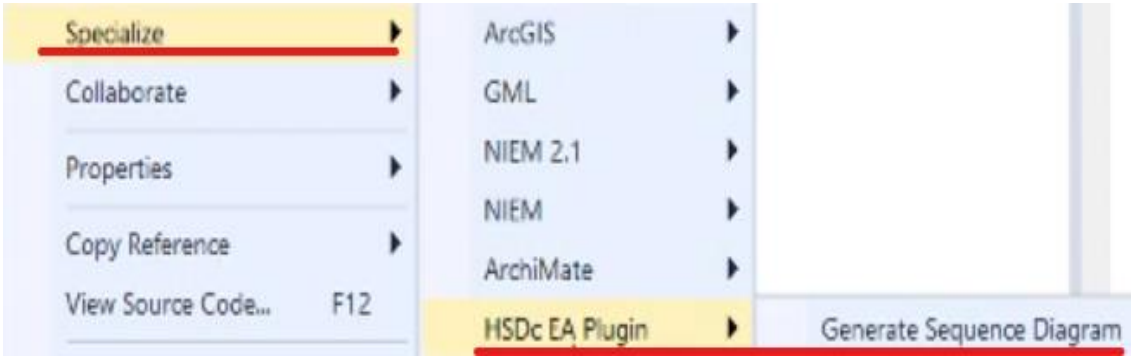


Figura 31 - *HSDc Seq plug-in*

3.Comparação dos diagramas

De forma análoga ao Visual Paradigm, o Enterprise Architect permite que seja realizada uma comparação visual entre os dois diagramas para denotar as principais diferenças. Permite também que seja feita uma comparação automática através da funcionalidade "*Compare Package to XMI*" que irá enaltecer as principais diferenças entre os diagramas de sequência.

Para aceder e utilizar corretamente esta funcionalidade foram efetuados os seguintes passos ilustrados nas Figuras 32 e 33.

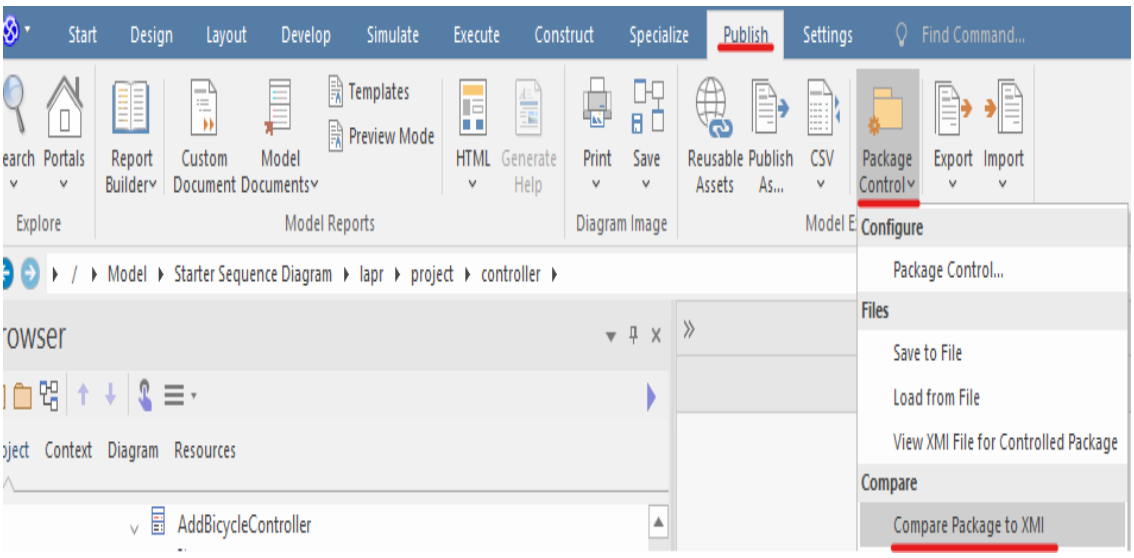


Figura 32 - Como aceder à funcionalidade *Compare Package to XMI*

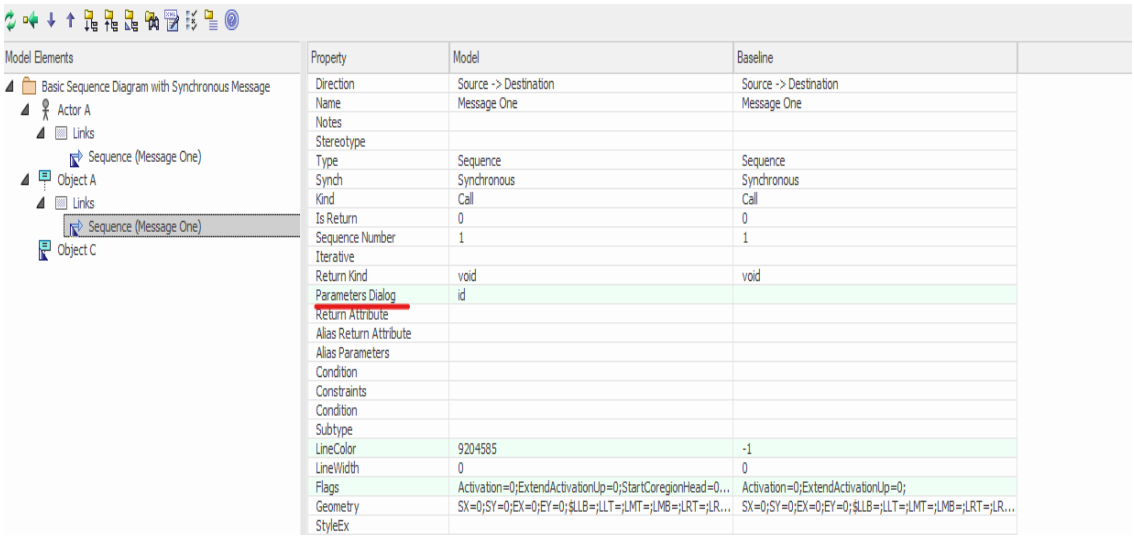


Figura 33 - Menu de comparação

4.Geração do relatório de similaridade

Ao contrário da ferramenta Visual Paradigm, a ferramenta Enterprise Architect apenas permite que a geração de um relatório seja feita para um ficheiro no formato XML contendo as diferenças entre ambos os diagramas, ainda que também possibilite que sejam escolhidas quais as opções desta comparação, como é exposto na Figura 34.

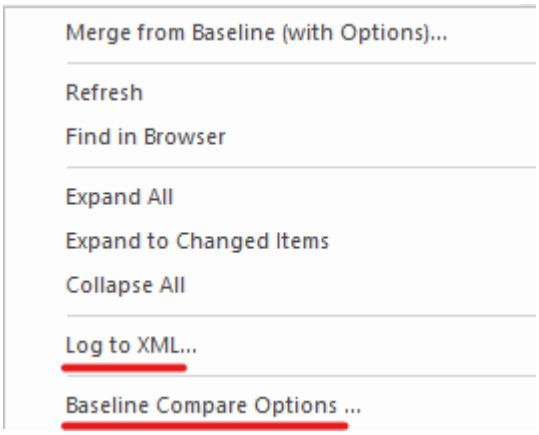


Figura 34 – Geração do relatório

Contudo, e para que o resultado desta comparação seja mais fidedigno, foi adotada a utilização do script anteriormente elaborado, presente nas Figuras 24 e 26. Deste modo, apenas será necessário exportar os dois diagramas XMI, assim como na ferramenta Visual Paradigm.

5.Avaliação

Em última instância, foi elaborado o Quadro 25 onde se encontram enumeradas as vantagens e desvantagens, mais relevantes, da utilização da ferramenta Enterprise Architect.

Quadro 25 - Vantagens e desvantagens da ferramenta Enterprise Architect

Vantagens	**Desvantagens**
▪ Comparação dos diagramas lado a lado; ▪ Alargada escolha de filtros e estratégias na funcionalidade *Compare Package to XMI:* ▪ Permite a geração de diagramas de sequência a partir de código em linguagens de programação como diferentes Java, C, C#, através do *plug-in HSDc Seq*;	▪ A funcionalidade *HSDc Seq* apenas está acessível através da instalação do *plug-in* não nativo à ferramenta; ▪ A instalação de *plug-ins* e a sua integração não são triviais; ▪ A funcionalidade *Compare Package to XMI* apenas disponibiliza a geração de um relatório no formato XML e o relatório gerado pelo não calcula a (%) de similaridade apenas anota as diferenças.

5.1.3 Conclusões

Após avaliar os prós e contras das duas ferramentas escolhidas para implementar a solução pretendida, a ferramenta escolhida foi o Visual Paradigm.

A escolha desta ferramenta deve-se a três principais fatores:

1. Familiarização: Sendo que o principal objetivo é que a ferramenta seja usada em meio académico, profissional também, mas principalmente académico, todos os intervenientes no processo estão mais acostumados a esta e por isso será economizado imenso tempo ao utilizar esta ferramenta ao invés do Enterprise Architect;
2. Económico: A licença necessária para aceder a funcionalidade *Instant Reverse Java to Sequence Diagram* no Visual Paradigm tem atualmente um custo mensal de 39 dólares, já o Enterprise Architect na sua licença mais barata tem um custo mensal de 76 dólares;
3. Funcional: Ao nível funcional as duas ferramentas apesar de serem bastante equivalentes, apenas é oferecido no Visual Paradigm a funcionalidade nativa de gerar diagramas de sequência através de código e de geração de um relatório de comparação no formato PDF, ainda que sem a (%) de similaridade.

5.2 Sumário

Neste capítulo encontra-se descrita a implementação da solução de modo a resolver o problema da vigente dissertação. Inicialmente é definido qual o fluxo de teste e avaliação enumerando as principais premissas do mesmo e enumerando a ferramenta Visual Paradigm como a escolhida.

De seguida é descrito passo a passo como foi implementada a solução em ambas ferramentas com recurso a figuras, sendo que no fim é feito um levantamento das principais vantagens e desvantagens de cada ferramenta.

Finalmente, são apresentados os principais motivos para a escolha da ferramenta Visual Paradigm em detrimento da ferramenta Enterprise Architect.

6 Experimentação E Avaliação

Neste capítulo será apresentada a hipótese de investigação, seguida dos indicadores e fontes de informação e ainda da metodologia a ser adotadas experiências a realizar. De seguida serão apresentadas as experiências realizadas e a avaliação dos resultados obtidos com as mesmas.

6.1 Especificação Da Hipótese de Investigação

São vistas como hipóteses de investigação os pressupostos definidos com base nos objetivos e no problema, estes serão analisados e examinados durante todo o processo de avaliação.

Estas mesmas hipóteses funcionam como trampolim para a concretização de investigações sobre os pressupostos definidos inicialmente. Desta forma será possível avaliar se os objetivos e perspetivas definidas previamente foram concluídas com ou sem sucesso.

Na corrente dissertação o principal objetivo é encontrar uma solução que permita avaliar a qualidade do desenho do software por cobertura e sincronismo com o código, logo a hipótese a pôr à prova é:

"Existe forma de detetar se um diagrama de software é totalmente refletido no código desenvolvido posteriormente".

6.2 Identificação Dos Indicadores E Fontes de Informação

Para que a solução final seja sólida e robusta serão aplicados alguns métodos de avaliação para identificar quais os indicadores que permitem aceitar ou rejeitar a hipótese com um elevado grau de confiança.

Será utilizado um indicador, como principal fonte para corroborar a hipótese acima apresentada. Este indicador está relacionado com a utilização de ferramentas de modelação de software, sendo que estas possuem funcionalidades integradas para verificar a conformidade entre o diagrama e o código. Alguns exemplos são o UModel, Enterprise Architect, e Visual Paradigm.

Desta forma será possível obter o grau de cobertura do modelo face ao código, apenas nos casos em que o código não é gerado automaticamente, mas sim por um programador. Caso contrário se os modelos e diagramas estiverem sincronizados com o código a cobertura seria de 100%.

Um exemplo em que este indicador será útil enquadra-se numa situação em que num diagrama de sequência estão representadas duas classes (A e B) em que A comunica com B. Já no código estão representadas também duas classes (C e B), sendo que a classe B é comum nos dois. Apenas será verificado qual o código coberto pelo diagrama, mas na classe C terá de ser verificado o nome da mesma que não está conforme o diagrama, bem como a sua implementação.

6.3 Descrição Da Metodologia de Avaliação

Tendo por base o indicador apresentado anteriormente, a metodologia de avaliação resultará em dois distintos aspetos:

- A escolha de análise diagramas de sequência, é dado um maior enfase aos diagramas de sequência, pois estes descrevem todo o processo de implementação de um requisito. Deste modo, é também enfatizada a importância dos modelos e diagramas na engenharia de software, promovendo o uso dos mesmos;
- A análise do código, comparando o código com o diagrama de modo a verificar se o primeiro está totalmente refletido no segundo. Deste forma, se a afirmação anterior se confirmar, então o código deve ser organizado de forma semelhante ao diagrama, caso contrário será possível encontrar evidências claras das diferenças entre os dois. Este indicador será analisado e descrito em mais detalhe no Capítulo 5, Implementação da Solução.

6.4 Experiências

Nesta secção serão expostas as experiências realizadas na ferramenta Visual Paradigm tendo como base a metodologia de avaliação indicada (secção 6.3) e por conseguinte os seus indicadores (secção 6.2).

Esta secção será dividida em subsecções denominadas de “Experiências” em que o nível de complexidade dos diagramas elaborados vai aumentando em cada uma, ou seja, o número de ligações e classes vai ser cada vez mais elevado. Estas experiências têm como sustento a

utilização da ferramenta apenas para transformar código em diagramas e proceder à sua exportação no formato XMI, aliada à comparação dos dois ficheiros através do *script,* já apresentado no Capítulo 5, Implementação da Solução.

Por fim, será apresentada uma alternativa também avaliada, que tem por base a funcionalidade base do Visual Paradigm, Visual Diff, capaz de exportar as diferenças entre os dois diagramas para um ficheiro no formato PDF.

6.4.1 Experiência Base

A primeira experiência tem por base a comparação entre dois diagramas de sequência exatamente iguais, procedendo a sua exportação no formato XMI na ferramenta Visual Paradigm.

Após a exportação de ambos será executado o *script* que tem como objetivo comparar os caracteres comuns entre ambos os diagramas. Neste caso em específico irá gerar um relatório com a indicação de 100% de semelhança (consultar Anexo D). Também é possível denotar esta percentagem no *output* do *script* aquando da sua execução, Figura 35.

```
PS C:\Users\Utilizador\Desktop\GenerateCode> .\Similarity%.ps1
Percentage Similarity: 100%
HTML report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.html
PDF report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.pdf
```

Figura 35 - Consola do *output* do *script*

6.4.2 Experiência Complexidade Nível 1

De seguida, foi incrementada a complexidade da experiência de comparação entre os diagramas, ou seja, foi alterada uma ligação no diagrama quer ao nível da sua escrita, bem como a inclusão de parâmetros, Figura 36 e 37.

Figura 36 - Ligação inicial

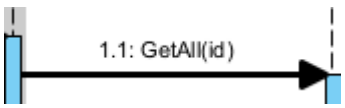


Figura 37 - Ligação modificada

Após esta alteração foi novamente executado o *script* denotando-se, ainda que pouco significativa, um decréscimo na (%) de similaridade entre os dois diagramas de sequência, Figura 38.

```
PS C:\Users\Utilizador\Desktop\GenerateCode> .\Similarity%.ps1
Percentage Similarity: 99.978282115322%
HTML report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.html
PDF report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.pdf
```

Figura 38 - Consola do *output* do *script* dos diagramas com diferenças pouco significativas

6.4.3 Experiência Complexidade Nível 2

De modo a incrementar a complexidade, foram adicionadas duas novas ligações ao diagrama para proceder à criação de uma nova entidade "Material" e do seu "DTO". Esta alteração encontra-se exposta na Figura 39.

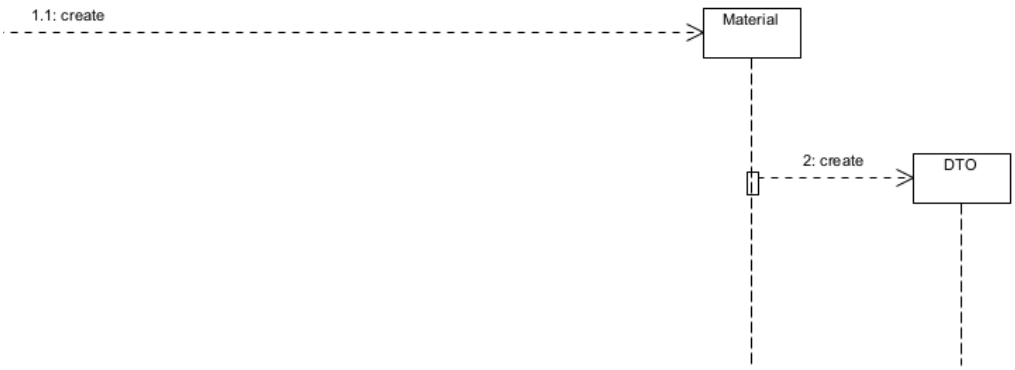


Figura 39 - Ligações adicionais

Com esta alteração foi novamente executado o *script* denotando-se um decréscimo mais significativo na (%) de similaridade entre os dois diagramas de sequência, em mais de um ponto percentual, Figura 40.

```
PS C:\Users\Utilizador\Desktop\GenerateCode> .\Similarity%.ps1
Percentage Similarity: 98.4808047628824%
HTML report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.html
PDF report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.pdf
```

Figura 40 - Consola do *output* do *script* dos diagramas com ligações adicionais

6.4.4 Experiência Complexidade Nível 3

Por fim, a última experiência contempla, além das ligações já adicionadas na experiência anterior, uma alteração nas ligações a uma classe já existente, bem como o nome da mesma tal como apresentado na Figura 41.

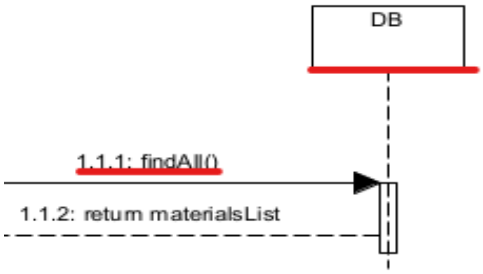


Figura 41 - Ligações atualizadas

Todavia, com esta última experiência voltou a ser executado o *script* e denotou-se, novamente pouco significativo, um decréscimo na (%) de similaridade entre os dois diagramas de sequência, Figura 42.

```
PS C:\Users\Utilizador\Desktop\GenerateCode> .\Similarity%.ps1
Percentage Similarity: 98.3473619380004%
HTML report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.html
PDF report has been generated and saved to C:\Users\Utilizador\Desktop\GenerateCode\comparison_report.pdf
```

Figura 42 - Consola do output do script dos diagramas com ligações adicionais e ligações atualizadas

6.4.5 Alternativa

A principal alternativa às experiências acima expostas é a utilização da funcionalidade Visual Diff já apresentada no Capítulo 5, Implementação da Solução, da ferramenta Visual Paradigm.

Esta funcionalidade permite que seja gerado um relatório em formato PDF destacando as diferenças entre os dois diagramas de sequência. O Visual Diff como já referenciado, disponibiliza filtros pelo tipo de modificações por novas, eliminadas ou alteradas, qual a estratégia a adotar como ID, nome ou *transitor* e ainda ordenar as diferenças por nome ou tipo (consultar Anexo E).

6.5 Análise Aos Resultados Das Experiências

Tendo por base as experiências e a alternativa acima apresentadas foi possível determinar que nenhuma das abordagens é considerada completa por si só. Ou seja, a combinação entre a geração de um relatório com as principais diferenças entre os dois diagramas através da funcionalidade Visual Diff e a execução do *script* para ter uma ideia mais precisa se essas diferenças são ou não impactantes no desenvolvimento final dos diagramas será a abordagem ideal.

No entanto, é de salientar que o *script* elaborado deverá ser aprimorado para não contemplar na comparação entre diagramas a estrutura base de um ficheiro XMI, pois esta estrutura é

comum para todos os ficheiros válidos mesmo que não sejam ficheiros relacionados com os que foram utilizados nas experiências acima assinaladas.

6.6 Sumário

Neste capítulo, é descrita a hipótese de investigação *"Existe forma de detetar se um diagrama de software é totalmente refletido no código desenvolvido posteriormente"*. São também descritos os indicadores e fontes de informação.

De seguida, é apresentada a metodologia de avaliação, focada em diagramas de sequência e na análise do código, comparando o código com o diagrama de modo a verificar se o primeiro está totalmente refletido no segundo.

Em última instância, foram realizadas algumas experiências com a ferramenta Visual Paradigm aliada a execução de um *script* de modo a auxiliar a geração de um relatório de semelhança, sendo ainda elaborada uma análise aos resultados destas experiências.

Em suma, o caso de estudo e as experiências realizadas permitiram obter resultados prometedores, com evidências de que é possível obter um relatório de semelhança através da aplicação da solução proposta, a ferramenta Visual Paradigm em conjunto com o *script*.

7 Conclusão

Neste capítulo final será elaborado um levantamento sobre o projeto da vigente dissertação, onde serão exibidos quais os objetivos alcançados, o trabalho que ficou por executar, as limitações encontradas e, finalmente, uma apreciação final de todo o trabalho realizado.

7.1 Objetivos Atingidos

Primeiramente, elaborou-se uma pesquisa tecnológica sobre as existentes revisões de código e quais as principais ferramentas capazes de efetuar esta revisão. De seguida, elaborou-se uma pesquisa tecnológica sobre as existentes revisões de desenho de software, bem como as principais ferramentas aconselhadas para efetuar este tipo de revisão. Por fim, foi efetuado um estudo sobre a compatibilidade das mesmas ferramentas com as principais linguagens de programação e de desenho de software.

Por consequência desta pesquisa, foi possível atingir o principal objetivo da vigente dissertação que seria identificar a ferramenta mais adequada para avaliar a qualidade do desenho do software por cobertura e sincronismo com o código.

Contudo, como foram encontradas duas ferramentas capazes de cumprir este objetivo - Visual Paradigm e Enterprise Architect - foram efetuados alguns testes de modo a descobrir qual seria a mais adequada para o âmbito da presente dissertação. Deste modo, foi possível verificar que a ferramenta Visual Paradigm, apesar de incompleta, é a mais adequada.

7.2 Limitações E Proposta Para Trabalho Futuro

Neste subcapítulo serão expostas as principais limitações ao trabalho desenvolvido, assim como as possíveis futuras melhorias quer ao trabalho de pesquisa quer à ferramenta escolhida.

Aquando da geração do relatório de semelhança entre o excerto de código e o digrama, ambos elaborados previamente e manualmente, um dos elementos que este relatório deveria conter era a (%) de semelhança entre os dois. Contudo, o relatório que é gerado nativamente pela ferramenta apenas contém as diferenças sendo esta considerada a sua principal limitação.

Por outro lado, para a escolha da ferramenta, visto que nativamente não existia nenhuma ferramenta capaz de comparar código com um diagrama, foi necessário transformar o excerto de código num diagrama de sequência. Para proceder à comparação na ferramenta escolhida, Visual Paradigm, é preciso que a licença da mesma seja a versão *Professional* ou *Enterprise* para a funcionalidade estar disponível, embora apenas seja possível importar código na linguagem de programação Java. A versão trial destas licenças tem apenas a duração de trinta dias limitando o tempo e o número de testes possíveis de realizar.

A principal melhoria ao nível de investigação, seria explorar uma ferramenta que não foi investigada, a IBM Rational Software Architect Designer. Certamente, seria uma boa alternativa quer ao Visual Paradigm quer ao Enterprise Architect.

Por fim, para tornar a ferramenta escolhida a ideal, seria interessante o desenvolvimento de um *plug-in* ou extensão capaz de gerar o relatório de semelhança com a (%) de similaridade, bem como a compatibilidade com todas as mais diversas linguagens de programação, não se cingindo apenas ao Java.

7.3 Apreciação Final

Devido a esta dissertação se enquadrar num ambiente académico e ter como principal enfoque a vertente de pesquisa, o autor teve a oportunidade de realizar um projeto diferente do que geralmente é realizado durante o curso. Isto possibilitou que o autor adquirisse novas competências que não tinha até então esmiuçado, sendo que desta forma fortaleceu o seu desenvolvimento ao nível tecnológico. Por outro lado, o projeto da vigente dissertação encorajou o autor também a um nível pessoal, pondo à prova as suas capacidades de iniciativa própria, autocrítica e organização entre a vida profissional, pessoal e académica.

O constante feedback, principalmente na fase inicial de pesquisa, com os orientadores da dissertação e o seu acompanhamento, foram fatores fulcrais para o crescimento pessoal do autor, alargando e aprofundando os seus conhecimentos.

Referências

Adam Khleel, N.A. and Károly, N. (2020) *Tools, processes and factors influencing of code review megtekintése*. Available at: https://ojs.uni-miskolc.hu/index.php/multi/article/view/415/315.

Ahmad, T. *et al.* (2019) 'Model-based testing using UML activity diagrams: A systematic mapping study', *Computer Science Review*. Elsevier Ireland Ltd, pp. 98–112. Available at: https://doi.org/10.1016/j.cosrev.2019.07.001.

Alves, M. (2020) *Qualidade de código com SonarQube | by Marina Alves | Tdx Oficial | Medium*. Available at: https://medium.com/tdx-oficial/qualidade-de-c%C3%B3digo-com-sonarqube-9ed754d88bda.

Amalfitano, D. *et al.* (2016) 'Comparing model coverage and code coverage in model driven testing: An exploratory study', in *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshops, ASEW 2015*. Institute of Electrical and Electronics Engineers Inc., pp. 70–73. Available at: https://doi.org/10.1109/ASEW.2015.18.

AMIQ EDA (2023) *Design and Verification Tools (DVT) IDE for e, SystemVerilog, VHDL, and PSS | Eclipse Plugins, Bundles and Products - Eclipse Marketplace*. Available at: https://marketplace.eclipse.org/content/design-and-verification-tools-dvt-ide-e-systemverilog-vhdl-and-pss.

Bansiya, J. and Davis, C.G. (2002) 'A hierarchical model for object-oriented design quality assessment', *IEEE Transactions on Software Engineering*, 28(1), pp. 4–17. Available at: https://doi.org/10.1109/32.979986.

Beanbag, I. (2001) *Review Board: Time for a code review upgrade*. Available at: https://www.reviewboard.org/.

Bertrán Macía, I. (2009) 'Isela Macía Bertrán Avaliação da Qualidade de Software com Base em Modelos UML'.

Blooman, D. *et al.* (2019) *GitHub - bbc/wraith: Wraith — A responsive screenshot comparison tool*. Available at: https://github.com/bbc/wraith.

Borza, J. (2011) *FAST Diagrams: The Foundation for Creating Effective Function Models General Dynamics Land Systems*.

BrowserStack (2023) *Percy | Visual testing as a service*. Available at: https://percy.io/?r=qal-vrtt.

Campbell, M. (2021) *Collaborative Engineering 101: Types of Design Reviews*. Available at: https://www.colabsoftware.com/post/collaborative-engineering-101-design-review-types.

Carnica Technology (2018) *TRACEALYZER | Benefits | Insights to Runtime World | PERCEPIO | Carnica Technology*. Available at: https://www.carnica-technology.com/visual-analysis/tracealyzer-benefits/.

Cass, S. (2023) *The Top Programming Languages 2023 - IEEE Spectrum*. Available at: https://spectrum.ieee.org/the-top-programming-languages-2023 (Accessed: 28 September 2023).

Cherniavska, M. (2019) *PlantUML: Diagrams as Code - Java Code Geeks - 2023*. Available at: https://www.javacodegeeks.com/2019/02/plantuml-diagrams-code.html (Accessed: 19 February 2023).

codegrip (2020) *Manual vs Automated Code Review - Codegrip*. Available at: https://www.codegrip.tech/productivity/manual-vs-automated-code-review/.

Daityari, S. (2023) *13 Melhores Ferramentas de Revisão de Código para Desenvolvedores (Edição 2023)*. Available at: https://kinsta.com/pt/blog/ferramentas-de-revisao-de-codigo/.

Data Flair (2018) *Pros and Cons of Java | Advantages and Disadvantages of Java - DataFlair*. Available at: https://data-flair.training/blogs/pros-and-cons-of-java/ (Accessed: 20 February 2023).

Deborah (2019) *C Sharp - Features, Advantages and Disadvantages*. Available at: https://urbannaturale.com/c-sharp-features-advantages-and-disadvantages/ (Accessed: 20 February 2023).

Fesenfeld, L.P. *et al.* (2021) 'The role and limits of strategic framing for promoting sustainable consumption and policy', *Global Environmental Change*, 68. Available at: https://doi.org/10.1016/J.GLOENVCHA.2021.102266.

GitHub (2019) *What to look for in a code review | eng-practices*. Available at: https://google.github.io/eng-practices/review/reviewer/looking-for.html.

GitLab (2021) *What are the most important features for code review tools? | GitLab*. Available at: https://about.gitlab.com/topics/version-control/what-are-best-code-review-tools-features/.

Greiler, M. (2023) *10 Best Code Review Tools In 2023 | Awesome Code Reviews*. Available at: https://www.awesomecodereviews.com/tools/best-code-review-tools/#review-board.

Hallahan, K. (2008) 'Strategic Framing'.

Haughtondesign (2020) *How do you carry out an effective Critical Design Review (CDR)? | Haughton Design*. Available at: https://haughtondesign.co.uk/how-do-you-carry-out-an-effective-critical-design-review-cdr/.

IBM (2023) *Advantages of Java - IBM Documentation*. Available at: https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java (Accessed: 20 February 2023).

Indeed Editorial Team (2022) *What Is a UML and What Are Its Benefits? | Indeed.com Canada*. Available at: https://ca.indeed.com/career-advice/career-development/what-is-a-uml (Accessed: 21 February 2023).

Institute for Electrical and Electronics Engineers, Inc. and Association for Computing Machinery (2019) *Code of Ethics | IEEE Computer Society*. Available at: https://www.computer.org/education/code-of-ethics.

kissflow (2023) *A Beginner's Guide to Business Process Model and Notion (BPMN)*. Available at: https://kissflow.com/workflow/bpm/what-is-bpmn/ (Accessed: 21 February 2023).

Koopman, P. (2020) 'Software Architecture & High Level Design'. Available at: https://goo.gl/WnciF3 (Accessed: 26 February 2023).

Kumar, P. (2022) *Visual Regression Testing using Percy | by Pavan Kumar S | TestVagrant | Medium*. Available at: https://medium.com/testvagrant/visual-regression-testing-using-percy-94f3d03b26cc.

Lambdatest (2023) *Visual Regression Testing Tutorial: Comprehensive Guide With Best Practices*. Available at: https://www.lambdatest.com/learning-hub/visual-regression-testing#different-methods.

LDRA (2023) *LDRA are market leaders in verification and software quality tools*. Available at: https://ldra.com/.

Lima, A. (2022) *As 5 principais ferramentas de código aberto e gratuito de análise de código estático em 2020 – Acervo Lima*. Available at: https://acervolima.com/as-5-principais-ferramentas-de-codigo-aberto-e-gratuito-de-analise-de-codigo-estatico-em-2020/.

Malik, K. (2022) *Code Review: Manual VS Automated*. Available at: https://www.codiga.io/blog/code-review-manual-vs-automated/.

Mark, J. (2018) *Functionally Safe Positioning with the LDRA Tool Suite | Business Wire*. Available at: https://www.businesswire.com/news/home/20181206005119/en/Functionally-Safe-Positioning-with-the-LDRA-Tool-Suite.

Medin, H.F. *et al.* (2023) 'HOW TO IMPROVE WEB APPLICATION DEVELOPMENT USING A LAYERED REFERENCE MODEL AND QUALITY INDICATORS', *Journal of Engineering Research*, 3(10), pp. 2–16. Available at: https://doi.org/10.22533/at.ed.3173102330032.

Myers, B. (2018) *Let's do program design reviews, not just code reviews | by Bob Myers | Medium*. Available at: https://torazaburo.medium.com/lets-do-program-design-reviews-not-just-code-reviews-dcb5788925ef.

Nam, E. (2019) *Understanding the Levenshtein Distance Equation for Beginners | by Ethan Nam | Medium*. Available at: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0 (Accessed: 29 September 2023).

nata.house (2022) *Revisão de código: descubra o que você precisa saber sobre o assunto*. Available at: https://natahouse.com/pt/revisao-de-codigo-descubra-o-que-voce-precisa-saber-sobre-o-assunto.

Nicola, S. (2017) 'Análise de Valor (Value Analysis) FAST and QFD Techniques'.

Nicola, S. (2018) 'ANÁLISE DE VALOR INESC-TEC'.

Novotny, J. (2023) *A Programmers' Guide to Python: Advantages & Disadvantages | Linode*. Available at: https://www.linode.com/docs/guides/pros-and-cons-of-python/ (Accessed: 20 February 2023).

OMG (2005) 'Unified Modeling Language: Superstructure version 2.0 formal/05-07-04'.

OMG (2015) 'XML Metadata Interchange (XMI) Specification'. Available at: http://www.omg.org/spec/XMI/20131001/XMI.xsd (Accessed: 21 February 2023).

Øyvind Kallstad (2016) *LevenshteinDistance*. Available at: https://www.powershellgallery.com/packages/Communary.PASM/1.0.43 (Accessed: 6 October 2023).

Parasoft (2023) *Automated Testing to Deliver Superior Quality Software | Parasoft*. Available at: https://www.parasoft.com/.

Payne, R. (2023a) *Advantages of C# | CodeGuru.com*. Available at: https://www.codeguru.com/csharp/c-sharp-advantages/ (Accessed: 20 February 2023).

Payne, R. (2023b) *Disadvantages of C# | CodeGuru.com*. Available at: https://www.codeguru.com/csharp/c-sharp-disadvantages/ (Accessed: 20 February 2023).

Pedamkar, P. (2019) *What is Visual Studio Code? | Features And Advantages | Scope & Career*. Available at: https://www.educba.com/what-is-visual-studio-code/.

PeerSpot (2022) *Parasoft SOAtest reviews, rating and features 2023 | PeerSpot*. Available at: https://www.peerspot.com/products/parasoft-soatest-reviews#products-show.

Poest, A. (2020) *4 Advantages and Disadvantages of UML Diagrams for Companies - PC ZONE*. Available at: https://www.pczone.co.uk/4-advantages-and-disadvantages-of-uml-diagrams-for-companies/ (Accessed: 21 February 2023).

Porhassann, K. and Tony, M. (2009) 'Implementation of Value Methodology concept within the IranianConstruction Industry Paper Title Implementation of Value MethodologyConcept within the Iranian Construction Industry'.

Prasad, S., Rao, A. and Rehani, E. (2001) *DEVELOPING HYPOTHESES & RESEARCH QUESTIONS DEVELOPING HYPOTHESIS AND RESEARCH QUESTIONS DEVELOPING HYPOTHESES & RESEARCH QUESTIONS*.

Ryan, T. (2022) *Testlio Reviews 2023: Details, Pricing, & Features | G2*. Available at: https://www.g2.com/products/testlio/reviews.

Saaty, R.W. (1987) 'THE ANALYTIC HIERARCHY PROCESS-WHAT IT IS AND HOW IT IS USED', 9(5), pp. 161–176.

Saaty, T. and Windt, Y. (1980) 'Marketing_Applications_of_the_Analytic'.

Semenchuk, M. (2017) *Using BPMN diagrams: pros and cons | by Max Semenchuk | 4IRE | Medium*. Available at: https://medium.com/4ire/benefits-of-bpmn-b154904e7468 (Accessed: 21 February 2023).

Shain, D. (2022) *What is Visual Regression Testing? Tools & Examples - Applitools*. Available at: https://applitools.com/blog/visual-regression-testing/.

Sharma, S. (2017) *Visual Regression Testing Using Wraith*. Available at: https://www.axelerant.com/blog/visual-regression-testing-using-wraith.

Siddiqui Ammar, H. (2017) *6-Software Design Reviews (Object Oriented Software Engineering - BNU…*. Available at: https://pt.slideshare.net/HafizAmmarSiddiqui/6software-design-reviews-object-oriented-software-engineering-bnu-spring-2017.

smartbear (2023) *Collaborator - Code Review Tool for Teams | SmartBear*. Available at: https://smartbear.com/product/collaborator/.

SoftwareTestingHelp (2023) *13 BEST Code Review Tools For Developers in 2023 [SELECTIVE]*. Available at: https://www.softwaretestinghelp.com/code-review-tools/#5_Codestriker.

SourceForge (2001) *Codestriker: Homepage*. Available at: https://codestriker.sourceforge.net/.

SparxSystems (2000a) *Full Lifecycle Modeling for Business, Software and Systems | Sparx Systems*. Available at: https://sparxsystems.com/products/ea/ (Accessed: 19 February 2023).

SparxSystems (2000b) *Key Benefits | Enterprise Architect User Guide*. Available at: https://sparxsystems.com/enterprise_architect_user_guide/14.0/benefits_and_features/key_benefits.html (Accessed: 19 February 2023).

Stackscale (2023) *Most popular programming languages in 2023 [Ranking]*. Available at: https://www.stackscale.com/blog/most-popular-programming-languages/ (Accessed: 28 September 2023).

Strato Flow (2023) *How is Java Used in Software Development in 2023: The Ultimate Guide - Stratoflow*. Available at: https://stratoflow.com/use-of-java-in-software-development/#use (Accessed: 20 February 2023).

Sundar (2020) *Function Analysis and System Technique - FAST diagram - ExtruDesign*. Available at: https://extrudesign.com/function-analysis-and-system-technique-fast-diagram/?utm_content=cmp-true (Accessed: 23 February 2023).

Terlecki, D. (2019) *PlantUML as go-to UML CASE tool*. Available at: https://blog.termian.dev/posts/plantuml/ (Accessed: 19 February 2023).

Testlio Inc (2012) *Testlio | Functional QA, user experience testing for web and mobile apps*. Available at: https://testlio.com/.

Teza, P. *et al.* (2013) '28_2013_Direcionadoresdoprocessodeinovaao'.

Thakur, D. (2023) *Software Design Reviews in Software Engineering - Computer Notes*. Available at: https://ecomputernotes.com/software-engineering/softwaredesignreviews#Types_of_Software_Design_Reviews.

The Software Engineering Authority (2020) *Design & Code Reviews*. Available at: https://www.softwareengineeringauthority.com/index.php/tools/13-software-engineering-disciplines/4-design-code-reviews.

Tomassetti, G. (2017) *NDepend: Static Analysis For Software Architects - Review*. Available at: https://tomassetti.me/ndepend/.

Vartanian, E. (2022) *Why learn Python? 5 advantages and disadvantages*. Available at: https://www.educative.io/blog/why-learn-python (Accessed: 20 February 2023).

Visual Paradigm (2002) *Benefits of Designing Database with Visual Paradigm*. Available at: https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3584/85402_benefitsofde.html (Accessed: 19 February 2023).

Visual Paradigm (2017) *What is Unified Modeling Language (UML)?* Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (Accessed: 20 February 2023).

Wohlin, C. and Runeson, P. (2021) 'Guiding the selection of research methodology in industry–academia collaboration in software engineering', *Information and Software Technology*, 140. Available at: https://doi.org/10.1016/j.infsof.2021.106678.

ZEN PROGRAM HLD (2023) *Improve your .NET code quality with NDepend*. Available at: https://www.ndepend.com/.

Anexo A – Modelo NCD

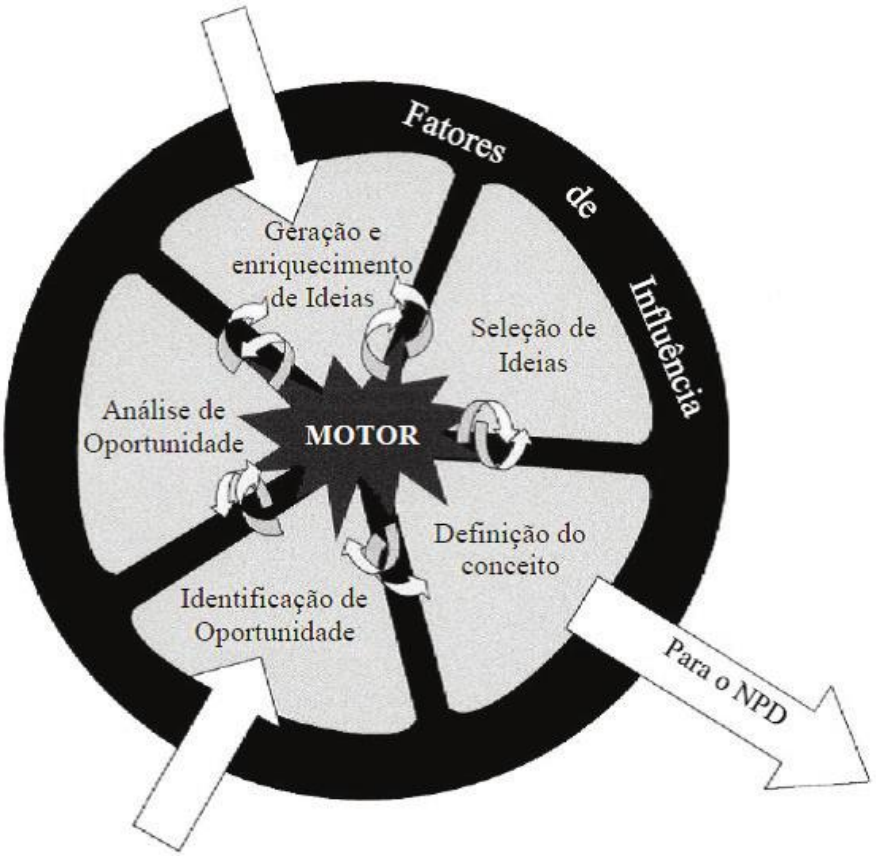


Figura 43 - Modelo NCD

Anexo B – Diagrama de FAST

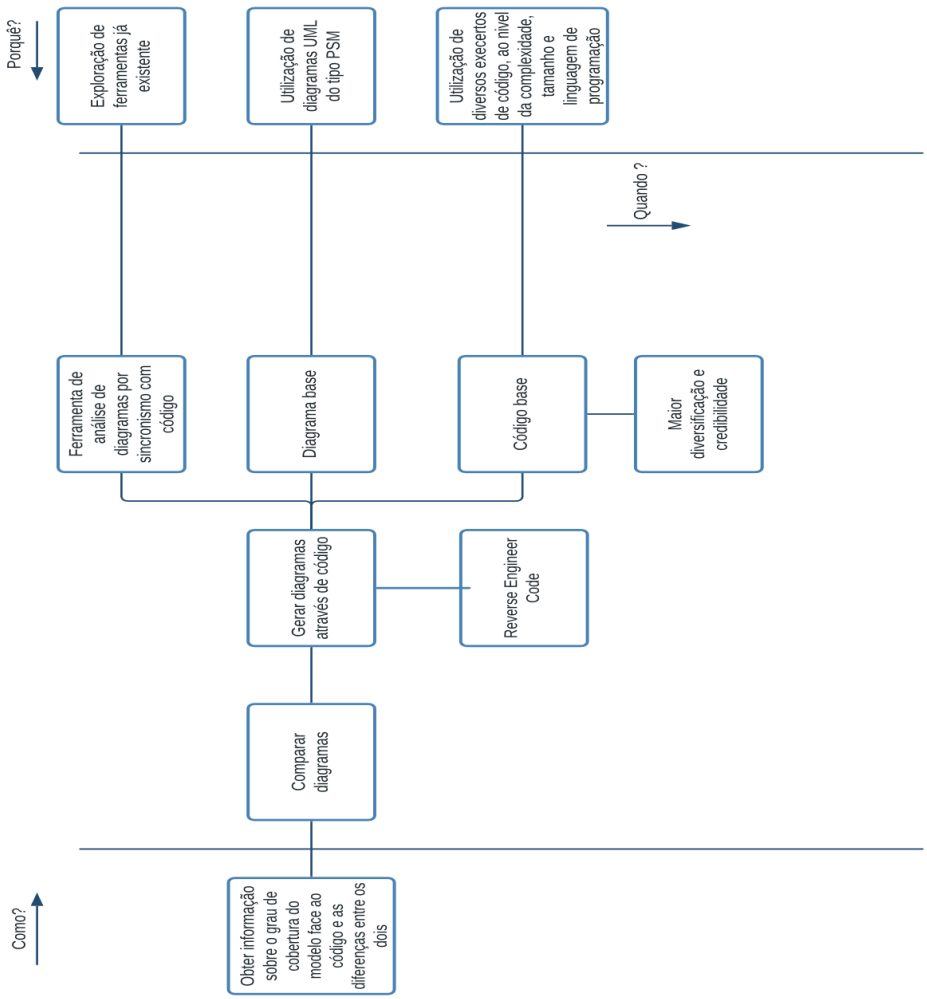


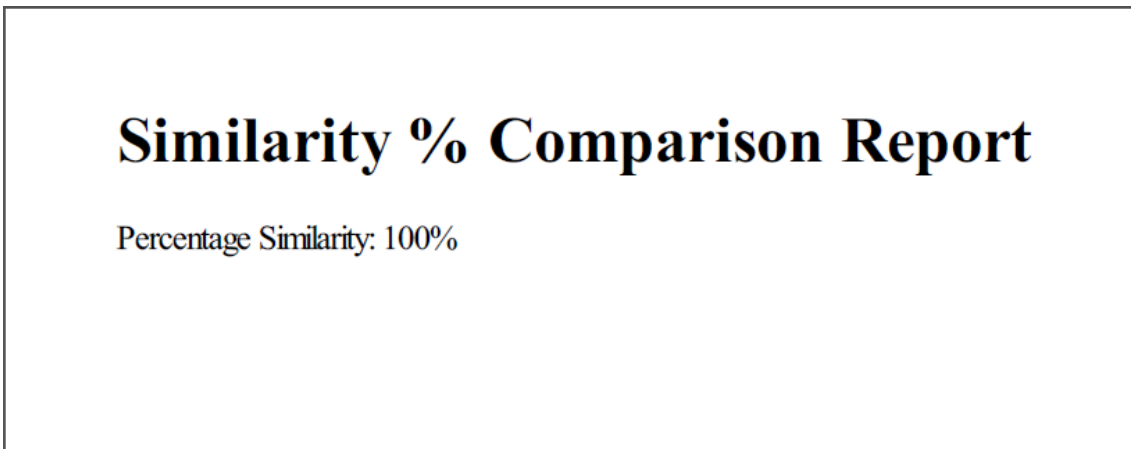
Figura 44 - Diagrama de FAST

Anexo C – Valores Para O Nível de Importância

Nivel de importância	Definição	Explicação
1	Igual importânica	As duas atividades contribuem igualmente para o objetivo
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Figura 45 - Tabela de valores para o nível de importancia

Anexo D – Exemplo relatório de semelhança gerado pelo *script*



Similarity % Comparison Report

Percentage Similarity: 100%

Figura 46 - Relatório de Semelhança

Anexo E – Exemplo relatório gerado pelo Visual Diff

Summary

Strategy:	Name
Compare:	View
Base:	Left hand side
Filter:	All
Sort:	Type

Ignored Model Types

Actor
Activation

Figura 47 - Elementos ignorados pelo Visual Diff

→1.1: GetAllMaterials() (New)
→1.1: getAllMaterials() (Deleted)
Controller (Deleted)
MaterialRepository (New)
MaterialsController (New)
MaterialsService (New)
Repository (Deleted)
Service (Deleted)

Figura 48 - Diferenças entre diagramas detetadas pelo Visual Diff