Universidade do Minho
Escola de Engenharia

Departamento de Sistemas de Informação

Luís Miguel Carvalho Correia

Desenvolvimento de uma *framework* de métricas para projetos ágeis

Tese de mestrado em Engenharia e Gestão de Sistemas de Informação

Desenvolvido sobre a orientação do

Professor Doutor Pedro Abreu Ribeiro

Outubro de 2018

“Ever tried. Ever failed. No matter. Try again. Fail again. Fail better.” (Samuel Beckett)

Agradecimentos

Nesta fase de desenvolvimento de dissertação, pude contar com o apoio de várias pessoas, sem as quais não teria conseguido com sucesso esta etapa.

Gostava de começar por agradecer em especial aos meus pais que não deixaram de acreditar em mim e que suportaram todo o meu percurso educacional e pessoal, nunca limitando as escolhas por mim feitas.

À minha irmã que esteve sempre predisposta a dar a sua opinião ou auxílio, independentemente do problema, e que sofreu com muita da minha angústia ou frustração ao longo do percurso pelo ensino superior. Obrigado pelo teu voto de confiança e por agires em minha defesa.

Para a minha restante família que sempre me apoiou durante todo o percurso e que sempre tiveram elevadas expetativas daquilo que poderia ser.

Aos meus colegas que durante todo o percurso académico compreenderam a frustração dos obstáculos que tínhamos de superar e só com entreajuda conseguíamos ultrapassar essas barreiras.

Em especial ao Nuno Santos, investigador do Centro de Computação Gráfica, que disponibilizou muito do seu tempo para ajudar com o caso de estudo exposto na dissertação, bem como acompanhou a documentação feita acerca do mesmo. O meu sincero muito obrigado por todo o apoio prestado, por vezes fora do local de trabalho.

Ao professor Pedro Ribeiro que tornou tudo possível com este tema, e que foi mais além ao conseguir um caso de estudo onde foi possível obter dados reais da *framework* em uso.

Resumo

Este trabalho tem por base o desenvolvimento de uma *framework* de métricas no ramo da gestão de projetos, com foco nos projetos que são desenvolvidos através de metodologias ágeis. Para isso, será analisado o caso de um projeto real *ongoing* e a *framework* será desenvolvida e experimentada com vista ao aperfeiçoamento do mesmo.

Com este projeto de dissertação pretendemos fazer um levantamento de toda a informação relacionada com a engenharia de software, metodologias ágeis e métricas que são utilizadas para fazer controlo de projetos de software. Isto vai também envolver um estudo de metodologias tradicionais para compreender limitações e benefícios em contraste com as metodologias ágeis e o impacto que estas têm nas organizações.

Uma abordagem ágil tem como pressuposto um agilizar dos processos da equipa de desenvolvimento, com rápidas entregas de software produzido ao cliente para que haja constante feedback entre o cliente e a organização. Assim, requisitos que não estejam a ir ao acordo do planeamento podem mais facilmente ser alterados.

A *framework* criada foi testada num caso real de desenvolvimento de software, apesar de a ideia inicial ser o aperfeiçoamento de uma *framework* depois de testar em projetos reais, mas já terminados e só de seguida passar ao projeto real de caso de estudo.

Palavras-chave: Metodologias ágeis, métricas, engenharia de software, gestão de projetos.

Abstract

The assumption behind this paper is the development of a metric framework in the branch of project management, with focus on project being developed through agile methodologies. In order to do so, a real and ongoing project was analyzed, and the framework developed and tested to improve it.

With this dissertation project, it's intended to absorb all the knowledge related with software engineering, agile methodology and metrics used to control software projects. This will involve a study of traditional methodology to understand the limitations and benefits in contrast with the agile methodology and the impact this has on organizations.

An agile approach has as assumption, expedite the team development processes with quick software delivery produced to the client, having constant feedback between client and organization. This away, requirements that are not going according to plan can easily be altered.

The created framework was tested in a real case of software development, even though the initial idea was to perfect the framework after testing in already finished projects and only then pass onto the real case study project.

Keywords: Agile methodologies, metrics, software engineering, project management

Conteúdo

Índice de ilustrações

Índice de tabelas

Lista de siglas e acrónimos

AC – *Actual Cost*

APC – *Actual Percentage Complete*

ASD – *Agile Software Development*

BAC – *Budget At Completion*

BVD – *Business Value Delivered*

CCG – Centro de Computação Gráfica

cVIG – *Computer Vision Interaction and Graphics*

CPI – *Cost Performance Index*

CV – *Cost Variance*

DSRM – *Design Science Research Methodology*

EPMQ – *Engineering Process, Maturity & Quality*

EV – *Earned Value*

IS – *Information Systems*

ISO – *International Organization for Standardization*

IPMA – *International Project Management Association*

JIT – *Just In Time*

KPI – *Key Performance Indicator*

LeanSD – *Lean Software Development*

OO – *Object Oriented*

OKR – *Objetives and Key Results*

PPC – *Planned Percentage Complete*

PV – *Planned Value*

PM – *Project Management*

PMBOK – *Project Management Body of Knowledge*

PMI – *Project Management Institute*

PPC – *Planned Percentage Complete*

PRINCE2 – *PRojects IN Controlled Environment*

RUP – *Rational Unified Process*

SDLC – *Software Development Life Cycle*

SDM – *Software Development Methodology*

SI – Sistemas de Informação

SPI – *Schedule Performance Index*

SV – *Schedule Variance*

SWEBOK – *Software* Engineering Body of Knowledge

UH4SP – *Unified Hub for Smart Plants*

UML - *Unified Modeling Language*

WIP – *Work In Progress*

XP – *eXtreme Programming*

1. Introdução

1.1 Contextualização

Esta dissertação surge no ramo da engenharia de software, mais concretamente na área de conhecimento da gestão dos projetos, que tem grande responsabilidade no desenvolvimento do Sistema de Informação (SI).

Um Sistema de Informação para apoio à gestão de projetos, segundo o SEVOCAB (*Software and Systems Engineering Vocabulary*), consiste em ferramentas e técnicas utilizadas para juntar, integrar e disseminar o output do processo da gestão de projeto. É utilizado para suportar todos os aspetos do projeto, desde o início até ao fecho, e pode incluir sistemas manuais e automáticos (1/SC7, n.d.).

Para a gestão de projetos de SI, há duas abordagens utilizadas na engenharia de software, abordagem tradicional e abordagem ágil.

A abordagem tradicional serviu durante anos como base para a gestão de projetos, pois preenchia todos os requisitos a que um projeto tinha de responder, tendo como caraterísticas o elevado detalhe no levantamento de requisitos no início do projeto, extensa documentação acerca do projeto e o cliente não acompanhava todo o processo de desenvolvimento, participando apenas no início quando era apresentado o problema.

Nas últimas décadas surgiu uma abordagem ágil que veio colmatar algumas lacunas deixadas pela anterior. Assim, esta abordagem preza mais a comunicação com o cliente como algo de grande relevo para o sucesso do projeto e assim encontra-se preparada para mudanças que possam surgir a meio do ciclo de vida do projeto por escolha do cliente. Para manter o cliente sempre atualizado o desenvolvimento é feito através de *sprints* sequenciais e gerando valor de negócio para o cliente no fim de cada *sprint*, em vez do desenvolvimento acontecer todo de uma vez só por um longo período de tempo. Assim cada vez mais existem organizações a adotar este tipo de abordagem, havendo algumas metodologias que adotam certos aspetos que outras não acham tão relevantes.

Nesta dissertação é feito um estudo sobre as mais relevantes, mas a mais focada é o *Scrum*, pois é a utilizada no caso real de estudo.

Como referido anteriormente, a abordagem ágil surgiu para ultrapassar algumas falhas da abordagem tradicional, contudo a abordagem ágil tem também alguns aspetos de possível melhoria, e para colmatar isso surgem cada vez mais abordagens híbridas que possuem os pontos fortes das diferentes abordagens ou combinam aspetos relevantes de diferentes metodologias, para atingir o sucesso do projeto em desenvolvimento. É possível afirmar que cada vez mais a gestão de um projeto difere de equipa para a equipa, tendo o gestor do projeto um papel fulcral em saber o que a sua equipa precisa, para conseguir ter sucesso na entrega do projeto e assim na entrega de valor ao cliente.

Para um gestor conseguir perceber o rumo do projeto, este recorre a métricas que lhe permitem monitorizar o projeto com o passar do tempo e assim fazer ajustes necessários caso haja algum problema.

No que toca à escolha de métricas para um projeto, existem várias que podem ser utilizadas para a medição do estado do projeto. Uma das mais relevantes englobam-se nas técnicas de *Earned Value Management* (EVM) que permitem observar os custos que uma equipa está a ter no projeto e assim prevenir possíveis perdas. Outra área com bastantes dados passíveis de análise é dentro do trabalho da equipa, no desenvolvimento de software, pois são eles que estão responsáveis por definir os requisitos e assim entregar valor ao cliente o mais depressa possível. Tudo isto deve ser feito tendo sempre em conta o cliente e a comunicação com ele, para que todo o trabalho elaborado seja o que o cliente tem em mente.

Um dos pilares a ter em consideração numa abordagem ágil é que esta não costuma ter grande foco nas áreas circundantes ao desenvolvimento. Como tal, o número de métricas escolhidas para um projeto ágil deve ser tido em consideração para não tornar essa área da gestão algo burocrática na equipa. Para facilitar isso, nesta dissertação as métricas vão estar divididas em diferentes áreas do desenvolvimento do projeto, facilitando assim a medição, visto que apenas os elementos que contribuem com os dados têm de ser consultados para fornecerem informação.

1.2 Finalidade e principais objetivos

Pretende-se com esta dissertação, um estudo inicial sobre as duas escolas que existem para a gestão de projetos, tradicional ou ágil. Numa segunda fase serão analisadas as abordagens híbridas, que pretendem encapsular os pontos fortes de cada uma das referidas acima.

Depois de realizado o estudo, pretende-se fazer um levantamento das métricas mais utilizadas em projetos de software e uma seleção das que se encaixam no universo ágil. Algumas das métricas encontradas focam-se em áreas como o *Earned Value Management* (EVM), a relação entre os *stakeholders* num projeto, o controlo do desenvolvimento do projeto, entre outras. Esta *framework* vai estar dividida por áreas em que um projeto desenvolvimento pode passar, para que assim seja mais fácil a sua utilização.

Seguidamente será aplicada a *framework* construída com as métricas selecionadas, num caso de estudo em contexto real de desenvolvimento de software. Serão retiradas todas as conclusões inerentes ao caso de estudo e assim será proposta a versão final da *framework*.

1.3 Estrutura do documento

Nesta secção será feita uma descrição mais detalhada da estrutura do documento. No capítulo um, é feita uma introdução à temática em estudo, bem como, é apresentada a finalidade e principais objetivos do estudo.

No segundo capítulo é explicada a abordagem metodológica usada no trabalho, bem como, a estratégia de pesquisa e um plano de atividades para as tarefas associadas à dissertação. No capítulo seguinte está a revisão da literatura onde se faz a exposição de todo o estudo feito até à data a nível das metodologias ágeis e tradicionais, da gestão de projetos e possíveis métricas a utilizar para o estudo.

No capítulo quatro apresenta-se a *framework* construída para a dissertação e as métricas selecionadas em cada área. É também neste capítulo que vemos a aplicação da *framework* num projeto de software real.

Por fim, no último capítulo serão apresentadas todas as conclusões do trabalho efetuado nesta dissertação.

2. Caraterização do estudo

2.1 Abordagem metodológica

A abordagem a ser utilizada para o desenvolvimento e validação da dissertação vai ser a *Design Science Research Methodology* (DSRM). A *Design Science* cumpre três objetivos que, são eles a consistência associada com a literatura anteriormente analisada, fornecer um modelo de processo nominal para fazer DSR e fornecer um modelo mental para apresentar e avaliar investigações feitas na área dos Sistema de Informação (SI) (Peffers et al., 2006).

O processo inclui seis passos, figura 1, para o desenvolvimento de um trabalho de investigação, no ramo dos SI. De seguida os passos são enumerados e explicados.

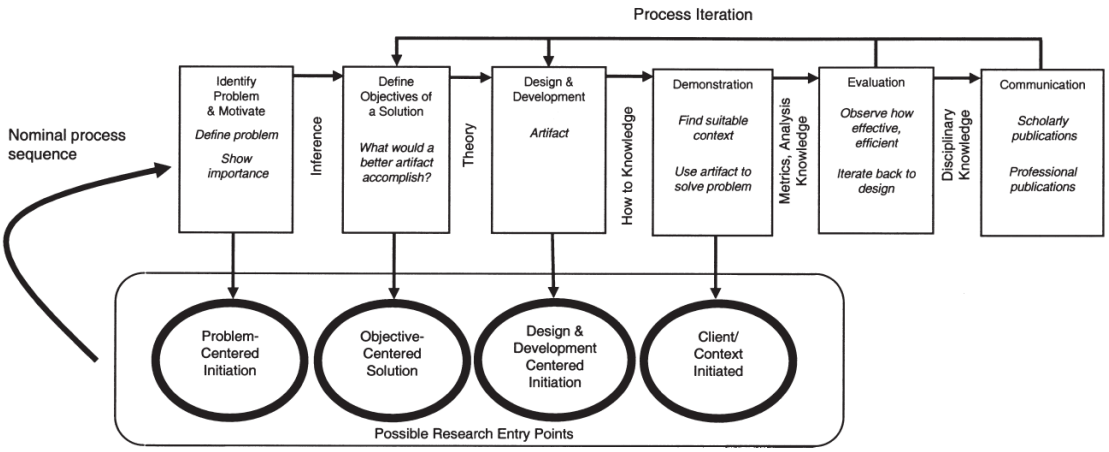

Figura 1. Modelo de Design Science Research retirado de (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007)

1. Identificação do problema e motivação – define o problema específico de investigação e justifica o valor da solução. A definição do problema é usada para a criação de um artefacto que fornece a solução. É aconselhável para perceber a complexidade da solução, fragmentar conceptualmente o problema.
2. Definir os objetivos para a solução – infere os objetivos da solução a partir da definição do problema e informa o que é possível e fazível. Os objetivos podem classificar-se como quantitativos ou qualitativos.

3. Design e desenvolvimento – criação do artefacto. Um artefacto com base na DSRM, pode ser qualquer objeto concebido no qual uma contribuição de investigação está embutida no design. Esta atividade inclui a determinação das características desejadas no artefacto e a sua arquitetura. É necessário um conhecimento teórico que pode ser criado para suportar a solução.
4. Demonstração – demonstra o uso do artefacto para a resolução de uma ou mais instâncias do problema. Isto pode envolver o seu uso em experiências, simulações, casos de estudo ou outras atividades.
5. Avaliação – observa e mede a maneira como o artefacto suporta a solução do problema. Esta atividade envolve a comparação dos objetivos da solução com os resultados observados do artefacto. É necessário ter conhecimento das métricas relevantes e técnicas de análise.
6. Comunicação – comunica o problema e a sua importância, a sua utilidade, a sua eficácia, entre outros aspetos, a pessoas reconhecidas da área em investigação. Se for uma investigação escolar, os investigadores utilizam, a estrutura deste processo para estruturar artigo (Peffers et al., 2007).

2.2 Estratégia pesquisa elaborada

Neste capítulo será feito uma explicação da abordagem utilizada para a pesquisa de informação em relação ao tema abordado e como foi feita a seleção dos artigos a utilizar entre os vários resultados existentes.

Inicialmente para ter o conhecimento genérico do tema, utilizou-se alguns referenciais conhecidos na área como o PMBOK e o SWEBOK para perceber as áreas da engenharia de software e da gestão de projetos. Seguidamente a pesquisa foi detalhada ao nível da gestão feita nos projetos, tradicional ou ágil, e das métricas utilizadas pelas abordagens.

Para a realização da pesquisa dos artigos selecionados, foram utilizados os motores de busca mais referenciados na temática do estudo. Sendo assim, os mais utilizados foram o *ScienceDirect*, o SCOPUS, que possuem vários artigos de várias áreas de investigação, a *Springer*, o *Institute of Electrical and Electronics Engineers* (IEEE), o RepositoriUM, e os Repositórios Científicos de

Acesso Aberto de Portugal (RCAA). Inicialmente, a pesquisa foi feita com recurso ao *Google Scholar* que permitia vários resultados e que, por sua vez, redirecionava muitas das vezes aos motores de busca supracitados. Com o passar do tempo e o avançar da dissertação, a pesquisa começou a ser feita nos próprios motores, utilizando o *Google Scholar* quando os resultados eram escassos ou desatualizados.

No que toca à seleção de artigos, a pesquisa muitas vezes utilizada incidia sobre a temática como "*agile methodology*", "*project management*", "software *metrics*", "*engineering*", entre outras, quando em certos casos a pesquisa tinha de ser mais refinada. Para filtrar os resultados existentes, a maioria dos artigos provém de um intervalo temporal entre 2015 e a data atual, exceto alguns casos pontuais.

Na tabela 1, estão listados os artigos utilizados para esta dissertação e o tema a que eles estão associados.

1. *Project Management*
2. *Agile Methodology*
3. *Engineering*
4. *Scrum*
5. *Software Metrics*

Tabela 1. Lista dos artigos utilizados e temas associados

Artigos	**1**	**2**	**3**	**4**	**5**
(PMI, 2013)	X		X		
(IEEE, 2014)	X	X	X		
(Sulaiman, Barton, & Blackburn, 2006)	X	X			X
(Kupiainen, Eetu; Mäntyla, Mika V.; Itkonen, 2015)		X			X
(Turk, France, & Rumpe, 2002)		X			
(Ahmad, Dennehy, Conboy, & Oivo, 2017)		X	X		
(Bassil, 2012)	X	X			
(Jeff Schwaber, Ken; Sutherland, 2016)	X	X		X	
(Špundak, 2014)	X	X			
(Ahmad et al., 2017)		X	X		

Artigos	1	2	3	4	5
(Fenton, Norman; Bieman, 2014)			X		X
(Fernandez, 2016)6	X	X			
(Jeff; Schwaber, Ken; Sutherland, 2017)		X	X	X	
(Wells, 1999)	X	X			
(Hartmann & Dymond, 2006)		X			X
(Mujtaba, Feldt, & Petersen, 2010)		X	X		X
(Peffers et al., 2007)			X		
(Peffers et al., 2006)					
(Varajão & Cruz-Cunha, 2013)	X		X		
(Ferreira, L.; Lopes, N.; Ávila, P. S.; Castro, H.; Varela, M. L. R.; Putnik, G. D.; Martinho, R.; Rijo, R.; Miranda, I. M.; Cruz-Cunha, 2017)	X		X		
(Radujković, Mladen; Sjekavica, 2017)	X				
(Atkinson, 1999)	X				X
(Wysocki, 2006)	X				
(Henderson-Sellers, Collins, Due, & Graham, 2001)			X		
(IBM, 2001)	X		X		
(Aked, 2003)	X		X		X
(Cockburn & Alistair, 2002)	X	X			
(Cockburn, 2004)	X	X			
(Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002)	X	X	X		
(Palmer & Felsing, 2002)	X	X	X		
(Poppendieck, Mary; Cusumano, 2003)	X	X			
(Poppendieck, 2007)	X	X			
(Poppendieck & Cusumano, 2012)	X	X			
(Beck, 2000)	X	X			
(Pulford, Kuntzmann-Combelles, & Shirlaw, 1996)	X		X		
(Hayes, Garcia-Miller, Lapham, Wrubel, & Chick, 2014)		X	X		X
(Ghosh, 2015)	X	X			X
(Nkiwane1, Meyer, & Steyn, 2016)	X				X
(Anderson, 2005)			X		

Artigos	1	2	3	4	5
(Hodgetts, 2004)		X	X		
(Haugen, 2006)		X			X
(Middleton, Taylor, Flaxel, & Cookson, 2007)		X	X		X
(Abbas, Gravell, & Wills, 2010), 2010	X				
(P. S. M. dos Santos, Varella, Dantas, & Borges, 2013)	X	X			
(Janus, André; Schmietendorf, Andreas; Dumke, Reiner; Jäger, 2012)		X			X
(Trimble & Webster, 2013)	X	X	X		
(Polk, 2011)		X		X	
(Seikola, Loisa, & Jagos, 2011)		X	X		
(Petersen & Wohlin, 2011)		X			X
(Kohlbacher, Stelzmann, & Maierhofer, 2011)	X	X		X	X
(Staron, Meding, & Söderqvist, 2010)	X				X
(Green, 2011)		X		X	X
(Kniberg, 2009)		X		X	

2.3 Plano de atividades

De seguida, tabela 2, vai ser exposto o cronograma das atividades realizadas, para desenvolver o trabalho, desde o início do trabalho em setembro 2017, até à sua conclusão.

Atividade 1: Escolha do tema de dissertação e plano de trabalhos.

Atividade 2: Revisão de conceitos inerentes ao problema.

Atividade 3: Redação do projeto de dissertação.

Atividade 4: Construção da *framework*.

Atividade 5: Aplicação da nova *framework* em projetos reais.

Atividade 6: Redação do relatório de dissertação.

Tabela 2. Plano de atividades da dissertação

Atividade / Mês	A1	A2	A3	A4	A5	A6
Setembro 2017	X					
Outubro 2017	X	X				X
Novembro 2017		X				X
Dezembro 2017		X	X			X
Janeiro 2018			X			X
Fevereiro 2018				X		X
Março 2018				X		X
Abril 2018				X	X	X
Maio 2018					X	X
Junho 2018					X	X
Julho 2018					X	X
Agosto 2018					X	X
Setembro 2018						X
Outubro 2018						X

3. Revisão de literatura

3.1 Engenharia de software

Este projeto de dissertação centra-se na área da engenharia de software. O IEEE (IEEE, 1997) define engenharia de software como "*(A) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (B) The study of approaches as in definition (A)*".

A engenharia de software consegue influenciar várias disciplinas do mundo académico, não estando afeta só às de engenharia, como também, à matemática, à gestão de projetos, gestão de qualidade, entre outras. Com um leque tão vasto de áreas de uso para a engenharia de software, a IEEE *Computer Society* desenvolveu um livro de conhecimento para a engenharia de software denominado *Software Engineering Body of Knowledge* (SWEBOK), onde consta o conhecimento necessário para o sucesso na profissão de engenheiro de software.

A construção do livro começou em 1990, tendo a publicação da primeira edição acontecido em 1995. O SWEBOK tem estabelecido 5 objetivos:

- Promover uma perspetiva consistente da engenharia de software globalmente;
- Especificar o âmbito e clarificar o lugar da engenharia de software no que toca a outras disciplinas como a ciência de computação, gestão de projetos, engenharia de computação e matemáticas;
- Caraterizar os conteúdos da disciplina de engenharia de software;
- Fornecer um acesso por tópicos para o corpo de conhecimento da engenharia de software;
- Fornecer uma fundação para o desenvolvimento curricular e certificação individual (IEEE, 2014).

Como anteriormente referido, uma das áreas onde gestão de projetos tem impacto é sobre a engenharia de software. Cada vez mais há uma maior preocupação pelo controlo de um projeto e por saber o rumo que ele está a ter

no seu ciclo de vida. Assim, a engenharia de software é uma área bastante relevante para o sucesso de um projeto de SI. Na versão de 2014 do SWEBOK estão listadas 15 áreas de conhecimento (*Knowledge Areas*) nas quais um projeto de engenharia de software é abrangido:

- Requisitos de software;
- Design de software;
- Construção de software;
- Teste de software;
- Manutenção de software;
- Gestão de configuração de software
- Gestão de engenharia de software
- Processo de engenharia de software
- Modelos e métodos de engenharia de software;
- Qualidade de software;
- Prática profissional de engenharia de software;
- Economia de engenharia de software;
- Fundações de computação;
- Fundações matemáticas;
- Fundações de engenharia;

3.2 Gestão de projetos

Varajão & Cruz-Cunha explicam-nos que a gestão de projetos compreende o planeamento, delegação, monitorização e controlo de todos os aspetos de um projeto, e mantendo a motivação dos vários *stakeholders* envolvidos, de modo a alcançar os objetivos traçados para o projeto, cumprindo o desempenho em termos de tempo, custo, qualidade, âmbito, benefícios e risco (Varajão & Cruz-Cunha, 2013).

Atualmente a gestão de projetos é suportada por vários standards como o *Individual Competence Baseline* (ICB), desenvolvido pelo *International Project Management Association* (IPMA), o *Project Management Body Of Knowledge* (PMBOK), proveniente do PMI (*Project Management Institute*) ou o *PRojects IN*

Controlled Environment (PRINCE2) da OGC (*Office of Government Commerce*) (Ferreira, L.; Lopes, N.; Ávila, P. S.; Castro, H.; Varela, M. L. R.; Putnik, G. D.; Martinho, R.; Rijo, R.; Miranda, I. M.; Cruz-Cunha, 2017).

O PMI define a gestão de projetos como a aplicação de conhecimentos, competências, ferramentas e técnicas para projetar atividades que vão ao encontro aos requisitos do projeto. A gestão do projeto é obtida através da aplicação e integração de grupos de processos de gestão. Estes grupos consistem em:

- Iniciação;
- Planeamento;
- Execução;
- Monitorização e controlo;
- Encerramento.

A gestão de um projeto está normalmente associada a tarefas como o levantamento de requisitos, adaptação das especificações, planos e abordagens para as diferentes preocupações e expetativas dos vários *stakeholders* e balancear as exigências competitivas para âmbito, qualidade, tempo e custo (PMI, 2013).

O sucesso associado à gestão do projeto é uma parte que envolve o sucesso global do projeto. Existem modelos capazes de medir o sucesso que um projeto atingiu. Na figura 2, é possível observar o primeiro modelo utilizado para medir a gestão de um projeto, olhando para critérios como o tempo, custo e a qualidade, intitulado "*The Iron Triangle*". Nos dias de hoje sabemos que um gestor é responsável, não só por estes fatores, como também por outras áreas, integração, âmbito, recursos humanos, comunicação, risco, aquisições e gestão de *stakeholders*. Outros modelos capazes de medir o sucesso são o PMPA – *Project Management Success Assessment* ou modelos de maturidade de gestão dentro da organização como *Project Excellence Model* (Atkinson, 1999; Radujković, Mladen; Sjekavica, 2017).

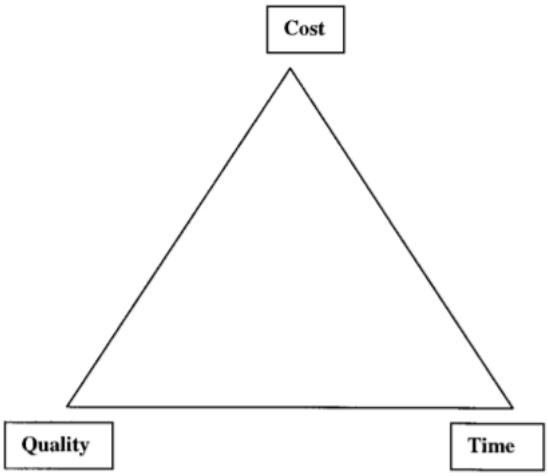

Figura 2. The Iron Triangle retirado de (Atkinson, 1999)

3.3 Abordagem tradicional versus abordagem ágil

A abordagem a ter na gestão de projetos pode variar de acordo com o projeto em desenvolvimento ou com a filosofia adotada pela equipa de trabalho. Existem assim duas abordagens, uma tradicional e uma abordagem ágil, criada nos anos 90, e começa a existir algum foco sobre aproveitamento dos pontos fortes de cada abordagem, formando assim uma abordagem híbrida das duas anteriores.

A ideia sobre a qual assenta a abordagem tradicional é a de que os projetos são relativamente simples, previsíveis e lineares com fronteiras claramente definidas, o que facilita no planeamento e acompanhamento sem grandes alterações. O objetivo da gestão tradicional é a otimização e eficiência ao cumprir com um acompanhamento detalhado do projeto. A desvantagem com uma abordagem tradicional é a de que, como o desenvolvimento de software está em constante alteração, as mesmas técnicas e métodos utilizados num projeto por vezes não se aplicam noutro. Outra desvantagem a considerar é a de que um projeto se encontra isolado do seu ambiente e as mudanças nos planos iniciais são inevitáveis, devido a ajustes imprevisíveis e mudanças dinâmicas no ambiente do projeto (Špundak, 2014).

Assim, para ultrapassar as desvantagens da abordagem tradicional, surgiram novas abordagens que enfatizam a diferença para com a tradicional (tabela 3). O nome mais comum é abordagem ágil, apesar de poder existir outras

designações com praticamente a mesma ideia, como abordagem *lean*, *extreme* e *adaptative*. Esta abordagem é caraterizada pela adaptabilidade durante o ciclo de vida do projeto, por ser mais centrada na comunicação e colaboração entre os membros da equipa, o que faz com que os elementos estejam mais envolvidos na tomada de decisão e a comunicação pode ser formal como informal (Fernandez, 2016; Špundak, 2014).

Tabela 3. Diferença entre abordagem ágil e tradicional adaptado de (Špundak, 2014)

Caraterística	Abordagem tradicional	Abordagem ágil
Requisitos	Requisitos iniciais claros; baixa taxa de mudança	Criativa, inovadora; requisitos não especificados
Utilizadores	Não se envolvem	Colaboração próxima e frequente
Documentação	Documentação formal necessária	Conhecimento implícito
Tamanho do projeto	Projetos grandes	Projetos pequenos
Suporte organizacional	Usar processos existentes; organizações maiores	Preparada para aceitar abordagem ágil
Membros de equipa	Variação esperada; equipas distribuídas	Equipas no mesmo local; equipa mais reduzida
Criticidade do sistema	Falha do sistema, consequências sérias	Sistemas menos críticos
Planeamento do projeto	Linear	Adaptativo; Extremo

Na figura 3 é feita uma comparação de cinco abordagens (as 3 primeiras tradicionais e as restantes ágeis) usando as cinco fases genéricas (âmbito, design, construção, teste e implementação) do ciclo de vida de desenvolvimento de software (Fernandez, 2016).

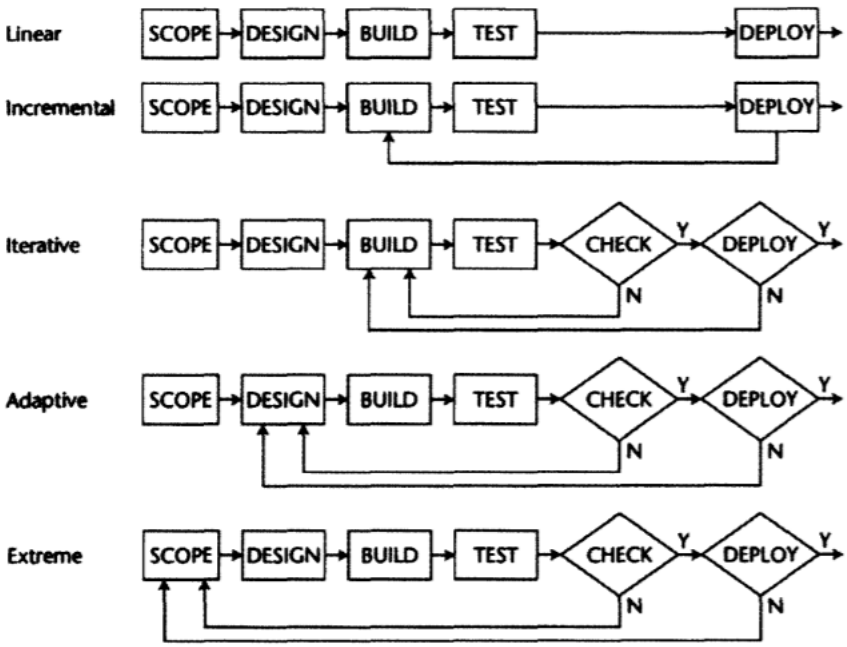

Figura 3. Estratégias de gestão de projetos com base na complexidade e incerteza retirado de (Wysocki, 2006)

Sutherland, Schwaber e Cockburn criaram em 2001 o manifesto para *Agile Software Development* (ASD). Este manifesto rege-se pelos seguintes valores (Sutherland, Jeff; Schwaber, Ken; Cockburn, 2001):

- Indivíduos e interações em vez de processos e ferramentas;
- Software a trabalhar em vez de documentação compreensiva;
- Colaboração com o cliente em vez da negociação de contrato;
- Responder à mudança em vez de seguir um plano.

Para um melhor desenvolvimento de software, o Manifesto Ágil sugere doze princípios:

1. A prioridade é satisfazer o cliente através de entregas rápidas e contínuas de software com valor;
2. Aceitar mudança de requisitos, mesmo que sejam tardias no desenvolvimento. Processos ágeis aceitam mudança para uma vantagem competitiva do cliente;
3. Entregar software funcional frequentemente, desde um par de semanas até um par de meses, com a preferência pela escala menor;

4. Cliente e programadores têm de trabalhar junto, diariamente ao longo do projeto;
5. Construir projetos à volta de indivíduos motivados. Oferecer o ambiente e suporte necessário, e acreditar que eles cumprem o trabalho;
6. O método mais eficiente e eficaz de transmitir informação para e com a equipa de desenvolvimento é conversar cara-a-cara;
7. Software a funcionar é a primeira medida de progresso;
8. Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, programadores, e utilizadores devem ser capazes de manter um fluxo constante indefinidamente;
9. Atenção contínua para excelência técnica e bom design reforça agilidade;
10. Simplicidade – a arte de maximizar a quantidade de trabalho não desenvolvido é essencial;
11. A melhor arquitetura, requisitos e designs emergem de equipas auto-organizadas;
12. Em intervalos regulares, a equipa reflete em como ser mais efetiva, depois sintoniza e ajusta o seu comportamento de acordo.

Como abordado anteriormente, há cada vez mais tentativas de fazer uma abordagem com o melhor da abordagem tradicional e da ágil porque ambas têm limitações no que toca à gestão de projetos. Focando no estudo da abordagem ágil, algumas limitações que esta possui passam por suporte limitado, para ambientes de desenvolvimento distribuídos, para subcontratação e, para construir artefactos reutilizáveis. Além do referido, nota-se também, limitação para suporte ao desenvolvimento envolvendo grandes equipas, limitação em suportar o desenvolvimento de software crítico e suporte limitado para desenvolver software complexo e grande (Turk et al., 2002).

De seguida, são expostas algumas metodologias, tradicionais e ágeis, para desenvolvimento de software, com maior foco sobre o desenvolvimento software ágil. Definimos um ciclo de vida de desenvolvimento de software (SDLC – *Software Development Life Cycle*) como uma abordagem capaz de construir e manter sistemas de informação, design e sistemas industriais.

3.4 Metodologias tradicionais

3.4.1 *Waterfall*

Uma das mais antigas metodologias para desenvolvimento de software é designada por *Waterfall*. Este modelo, proposto por Winston W. Royce, compreende cinco fases, figura 4, que devem ser completadas uma a seguir à outra, de uma forma sequencial (Bassil, 2012).

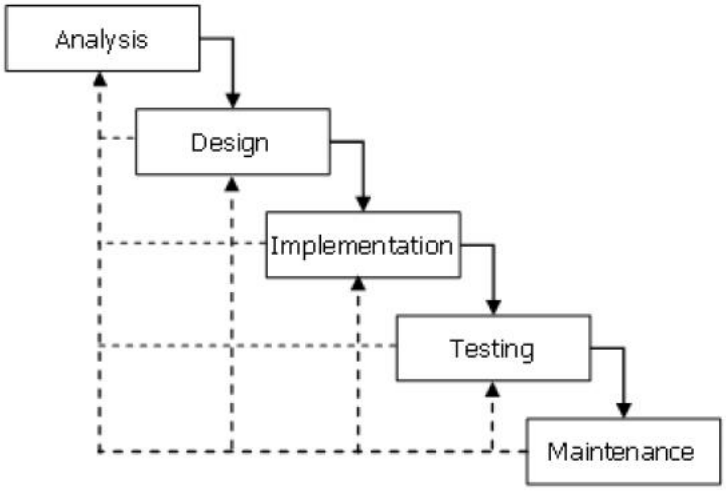

Figura 4. Ciclo de vida do modelo Waterfall retirado de (Bassil, 2012)

Fase de análise: fase na qual são definidos os requisitos funcionais e não funcionais. Para os requisitos funcionais são frequentemente definidos casos de uso, utilizando a linguagem de modelação UML (*Unified Modeling Language*). Para os requisitos não funcionais temos limitações, constrangimentos ou propriedades como escalabilidade, disponibilidade, desempenho e padrões de qualidade.

Fase de design: é feito o planeamento da solução para o software. São discutidos aspetos como design dos algoritmos, design da arquitetura de software, esquema conceptual e diagrama lógico da base de dados, interface gráfico do utilizador, entre outros.

Fase de implementação: quando os requisitos de negócio e especificações de design são realizados e colocados na componente de utilização final, seja ele um website, programa ou uma base de dados.

Fase de teste: também conhecida como verificação e validação, nesta fase testa-se o produto com intuito de perceber se este preenche os requisitos acordados na primeira fase e corrigir bugs ou falhas que possam ter surgido.

Fase de manutenção: encontrado algum bug na fase anterior, na fase de manutenção esses erros são corrigidos e aperfeiçoa-se o desempenho e qualidade do software. Adicionalmente podem ser acrescentados novos requisitos ou melhorada a fiabilidade do programa.

3.4.2 Rational Unified Process (RUP)

Uma abordagem muito utilizada na engenharia de software para o desenvolvimento de produtos orientado a objetos é a *framework* criada pela *International Business Machines* (IBM) denominada de *Rational Unified Process* (Henderson-Sellers et al., 2001).

O RUP é uma *framework* abrangente e, flexível para desenvolvimento de projetos de software que incorpora uma abordagem iterativa. A *framework* (ver figura 5) encontra-se dividida em quatro fases: *Inception*, *Elaboration*, *Construction* e *Transition*.

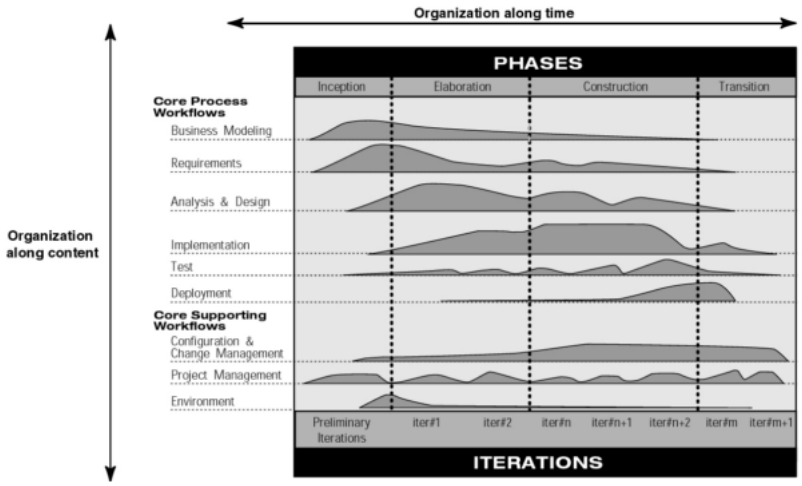

Figura 5. Fluxo de trabalho na framework RUP retirado de (IBM, 2001)

Para facilitar a divisão de trabalho pelos diferentes membros numa equipa que esteja a utilizar o RUP, esta possui seis disciplinas de engenharia de software que são a modelação de negócio, os requisitos, a análise e design, a implementação, o teste, e o *deployment*. Existem, no entanto, alguns autores que estendem estas disciplinas com mais três que servem de apoio ou suporte. São elas a gestão de configurações e mudança, a gestão do projeto e o ambiente (Aked, 2003).

Todas estas disciplinas têm um esforço associado ao longo do projeto, não tendo uma alocação fixa apenas a uma dada fase. Assim, a disciplina de análise e design, por exemplo, encontra-se bastante presente em todo o ciclo de vida do projeto. A finalização de uma fase tem associada uma data que é marcada como *milestone* e que produz um ou vários artefactos. Uma desvantagem a realçar na utilização desta *framework* é a documentação excessiva que propõe, algo que não é praticável na metodologia ágil. O RUP é uma *framework* que utiliza a linguagem de modelação UML, para descrever as ações/comportamentos que o projeto irá ter perante os atores à sua volta (IBM, 2001).

3.5 Metodologias ágeis

3.5.1 *Crystal*

A "*Crystal family"* é um conjunto de metodologias que enfatizam entregas frequentes, comunicação próxima e melhoria reflexiva. Assim, a escolha de qual metodologia *Crystal* usar depende de duas dimensões, do tamanho e da criticidade do sistema. "*The name "Crystal" comes from my characterization of project along two dimensions, size and criticality, matching that of minerals, color and hardness*" (Cockburn, 2004).

Existem três metodologias centrais denominadas de *Crystal Clear* para projetos pequenos com equipas até 6 elementos, podendo ir até 10 em alguns casos, *Crystal Yellow* para equipas de 10-20 pessoas, *Crystal Orange* de 20-50, *Crystal Red* 50-100.

O modelo *Crystal* possui também causas potenciais de falhas de um sistema. Estas podem classificar-se como *Confort* (C), *Discretionary Money* (D), *Essential Money* (E) e *Life* (L), , como é possível observar na figura 6. Esta divisão das causas permite saber a severidade de um projeto em caso de falha. Assim, quando temos um projeto com nível crítico ao sistema de L, é indicativo que, em caso de falha do sistema pode causar, literalmente, a perda de vidas humanas. Todos os projetos *Crystal* utilizam ciclos de desenvolvimento incremental entre um e três meses ou um máximo de quatro meses (Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002).

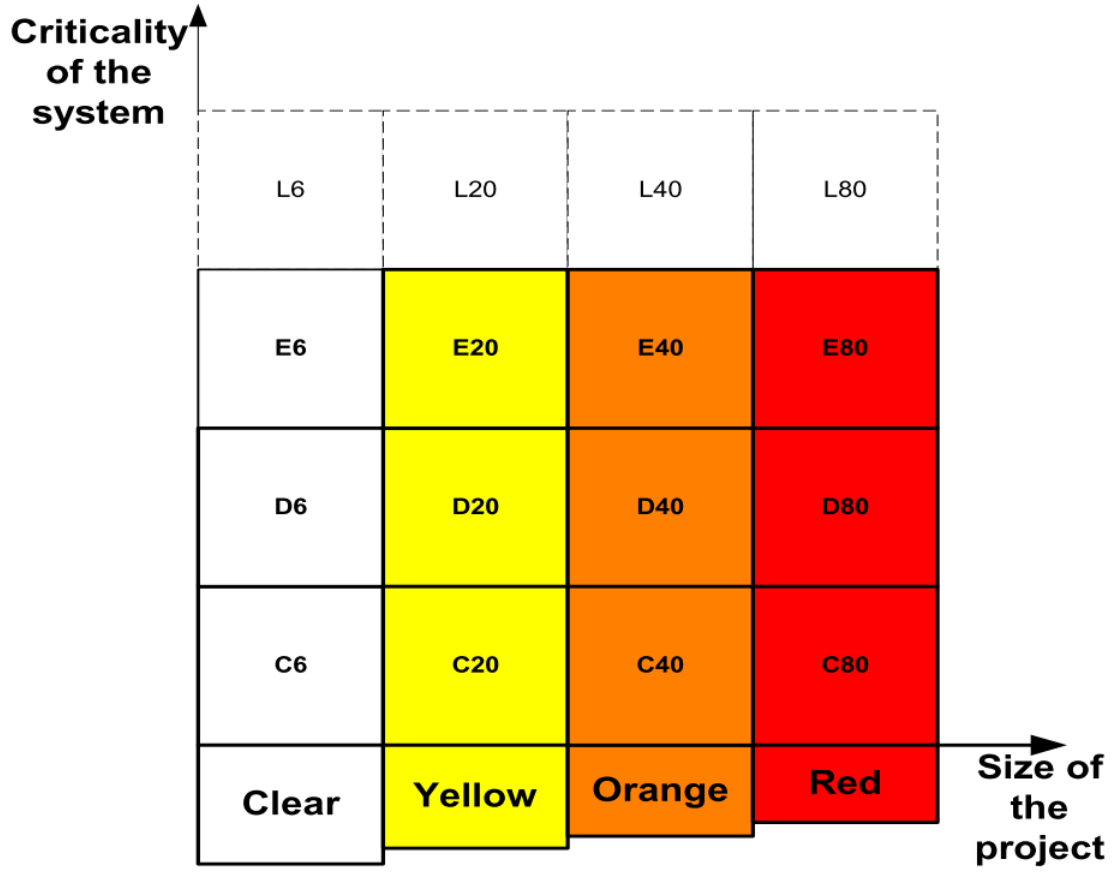

Figura 6. Dimensão das metodologias Crystal retirado de (Cockburn & Alistair, 2002)

3.5.2 Feature Driven Development

A metodologia *Feature Driven Development* (FDD) é também considerada uma abordagem para o desenvolvimento de sistemas. Esta metodologia não cobre

todo o desenvolvimento do software, focando mais sobre as fases de design e construção.

Como é possível constatar na figura 7, a FDD consiste em cinco processos sequenciais, durante os quais o design e a construção do sistema são desenvolvidos (Palmer & Felsing, 2002).

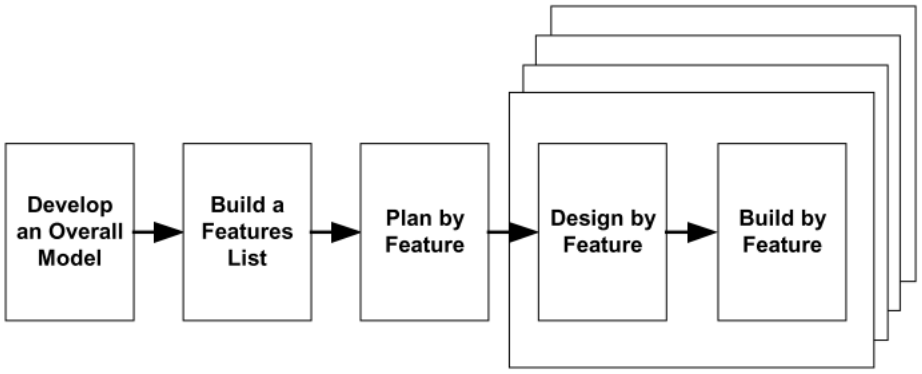

Figura 7. Processos da metodologia FDD retirado de (Palmer & Felsing, 2002)

***Develop an Overall Model*:** apesar de não ser explicitamente necessário a recolha de requisitos, nesta fase podem ser documentados requisitos como casos de uso ou especificações que devem existir no sistema. Especialistas fazem uma descrição global do sistema, usando um "*walktrough* de alto nível", os membros das equipas e o arquiteto são informados da descrição do sistema. De seguida são divididos em pequenas equipas e é feito um "*walktrough*" mais detalhado de cada subsistema. Cada equipa discute e decide entre si os modelos mais apropriados para cada domínio.

***Build a Features List*:** depois de feita a descrição do sistema, é feita uma lista das funcionalidades para o sistema ser desenvolvido. A equipa apresenta ao cliente as funções valiosas incluídas no sistema. Este grupo de funções de cada domínio consiste num conjunto de funcionalidades principais, funcionalidades essas que depois se dividem em diferentes atividades de cada área do domínio. A lista de funcionalidade é revista pelos utilizadores e patrocinadores, de acordo com a sua validade e a sua integridade.

***Plan by feature*:** é criado um plano de alto nível, no qual o conjunto de funcionalidades são sequenciadas e atribuídas ao chefe dos programadores de acordo com a sua prioridade e dependências. É feita também uma calendarização das *milestones* mais relevantes para o conjunto de funcionalidades.

***Design by Features and Build by Feature*:** é selecionado um conjunto de funcionalidades e é formada uma equipa necessária para desenvolver as funcionalidades escolhidas pelos *class owners*. Esta equipa fica responsável por tarefas como inspeção do design, programação, teste aos módulos, integração e inspeção do código. Depois de a iteração ter sido concluída, as funcionalidades são colocadas na aplicação final, como podemos ver na figura 8, começando de seguida uma nova iteração de funcionalidades retiradas do conjunto (Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002).

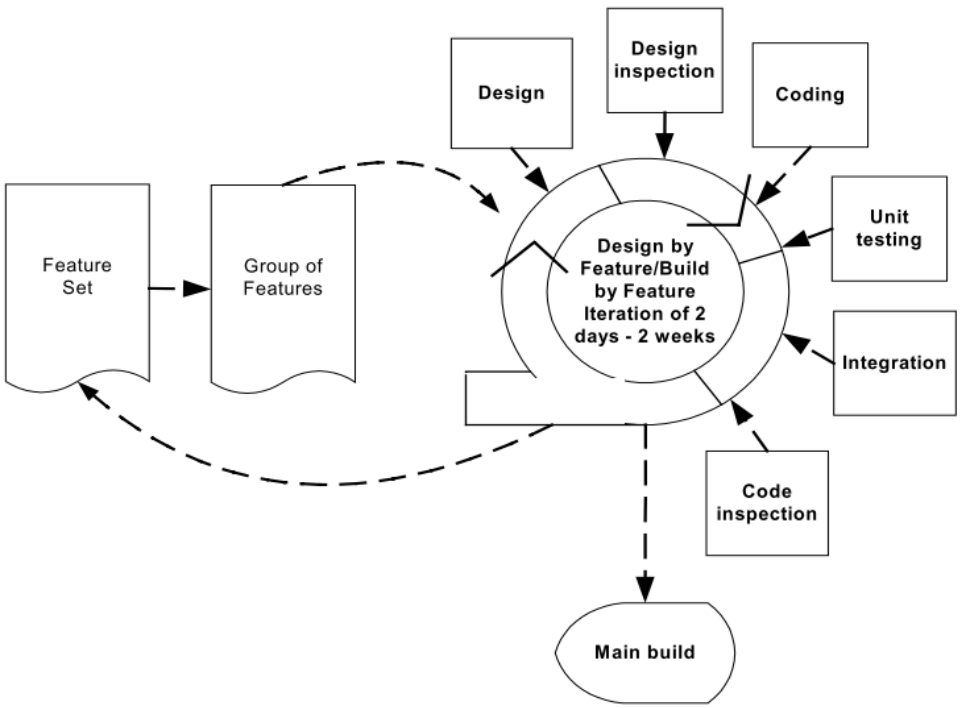

Figura 8. Design por funcionalidade e construção por funcionalidade da FDD retirado de (Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002)

Há vários cargos e responsabilidades associadas na metodologia FDD. Eles podem dividir-se em três categorias: cargos cruciais, cargos de suporte, e cargos

adicionais. Cada categoria possui, ela própria, cargos a desempenhar (Palmer & Felsing, 2002).

1. Cargos cruciais
    a. Gestor de projeto;
    b. Arquiteto chefe;
    c. Gestor de desenvolvimento;
    d. Chefe de programadores;
    e. Responsável de uma classe;
    f. Especialista do domínio.
2. Cargos de suporte
    a. Gestor;
    b. Guru da linguagem;
    c. Engenheiro de construção;
    d. Responsável pelas ferramentas (*toolsmith*)
    e. Administrador de sistemas.
3. Cargos adicionais
    a. Profissionais de testes;
    b. Técnicos de transição;
    c. Escritores de material técnico.

De realçar, que um membro de uma equipa pode desempenhar vários papéis, e um cargo pode ser partilhado por vários membros.

3.5.3 Lean Software Development

Apresentam-se em seguida algumas ideias chave do desenvolvimento *lean* de software:

"*Lean Software Development provides a wealth of information about applying lean techniques from an industrial setting to software development. In particular, it presents a toolkit for project managers, team leaders, and technology managers who want to add value rather than become roadblocks to their project teams*" (Poppendieck, Mary; Cusumano, 2003).

"*Lean Software Development is the application of the principles of the Toyota Product Development System to software development*" (Poppendieck, 2007).

"*When correctly applied, lean software development results in high quality software that is developed quickly and at the lowest possible cost*" (Poppendieck, 2007).

O termo *lean development* foi aplicado na gestão de processos de produção no *Massachusetts Institute of Technology* (MIT) durante os anos 80. Contudo esta aplicação já era feita na sede da Toyota onde os elementos necessários à produção eram produzidos em pequenos lotes "*Just In Time*" (JIT) para montagem e envio, eliminando assim processos ou inventários desnecessários. Mais tarde, Michael e Richard Selby (Poppendieck & Cusumano, 2012), repararam na filosofia semelhante à Toyota que era utilizada pela Microsoft onde os funcionários tinham de parar e corrigir bugs diariamente, assim, como a filosofia JIT da Toyota, implicava que os trabalhadores paravam a linha de montagem assim que detetassem problemas.

Hoje, estando já fortemente acolhida na comunidade da engenharia de software, quem utiliza a metodologia reforça a diferença entre trabalhos mais intensivos, burocráticos, métodos estilo *push*, com métodos novos, menos burocráticos, iterativos ou incrementais e flexíveis.

Na década de 1990 Robert Charette (Poppendieck & Cusumano, 2012) usou o termo "*lean development*" referindo-se à estratégia da gestão de risco que traz estabilidade dinâmica nas organizações, fazendo-as mais ágeis, flexíveis e tolerantes à mudança. Como o propósito de um negócio é servir o cliente, a *lean development* não foca apenas no desenvolvimento do processo, mas enfatiza sete princípios:

- Otimização como um todo;
- Eliminação do resíduo;
- Contruir qualidade interiormente;
- Aprender constantemente;
- Entregar rapidamente;
- Envolver toda a gente;
- Melhoria contínua.

3.5.4 Kanban

A metodologia adota os princípios anteriormente vistos na *Lean Software Development*, fornecendo uma ferramenta para otimizar o resultado, focando no fluxo da gestão. Os cinco princípios da metodologia Kanban são (Ahmad et al., 2017):

1. Visualizar o fluxo de trabalho: o trabalho passa por diferentes estados, (Planeado, Em progresso, Feito) à medida que atravessa a organização. São utilizados quadros físicos (figura 9) ou virtuais com cartões que representam itens de trabalho, que permitem aos colaboradores observarem o trabalho em progresso e para a equipa poder organizar-se sem necessitar da ajuda do gestor.

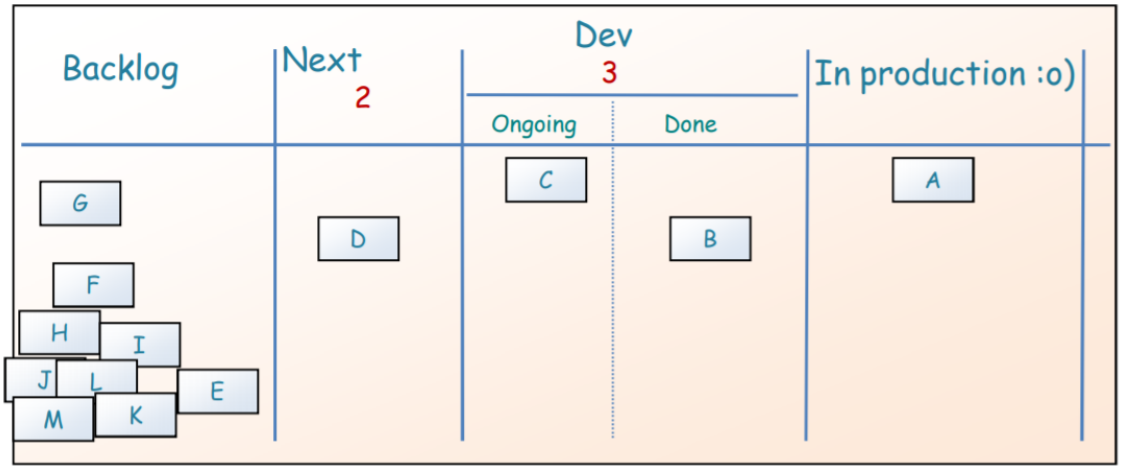

Figura 9. Exemplo de uma Kanban board retirado de (Kniberg, 2009)

2. Limitar o trabalho em progresso: limites WIP (*Work In Progress*) são usados para gerir a quantidade de progresso a ser feito a qualquer altura.
3. Medir e gerir o fluxo: existem cinco técnicas usadas para gerir o fluxo de gestão: (i) mapas de fluxo de valor; (ii) quadro *Kanban*; (iii) diagramas de fluxo cumulativos; (iv) gráficos *burndown*; (v) gráficos de *line balanced status*.
4. Fazer políticas de processo explícitas: é necessário estabelecer critérios de entrada e saída, "*entry*" e "*exit*" para determinar quando um item de trabalho pode ser arrastado de um estado para o outro. As políticas

permitem à organização observar causas e efeitos quando fazem mudanças no processo.

5. Utilizar modelos para reconhecer uma oportunidade de melhoria: oportunidade de melhoria contínua podem ser identificadas utilizando modelos como teoria de constrangimentos e pensamento sistémico.

3.5.5 eXtreme Programming (XP)

O primeiro projeto a surgir que utilizava esta metodologia foi em 1996. Atualmente, a XP é uma das metodologias ágeis com mais sucesso em companhias pelo mundo, independentemente do tamanho da organização (Wells, 1999). O termo extremo provém de levar ao extremo os princípios e as práticas pelo qual se rege.

O ciclo de vida de projeto XP, como podemos constatar na figura 10, passa por seis fases: Exploração, Planeamento, Iterações para lançamento, Produção, Manutenção e Morte (Beck, 2000).

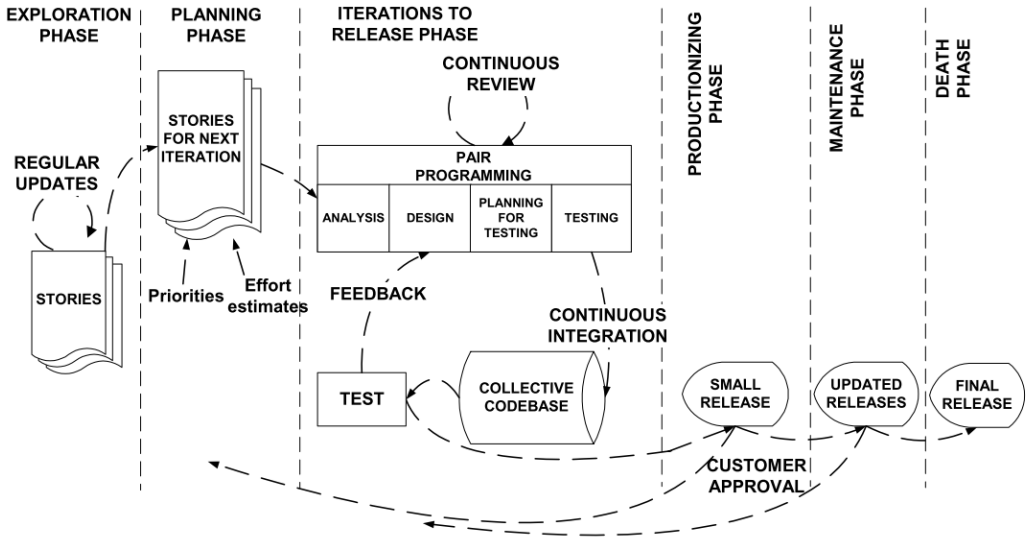

Figura 10. Ciclo de vida do processo XP retirado de (Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002)

1. Exploração – nesta fase o cliente escreve os "*story cards*" que deseja ver incluídos no primeiro lançamento. É também nesta fase que a tecnologia a ser usada é testada, bem como as possibilidades para a arquitetura são

exploradas, construindo-se um protótipo do sistema. A duração da fase é de algumas semanas a alguns meses, dependendo da dimensão do projeto.

2. Planeamento – é estabelecida a prioridade das *stories* e é feito um acordo para os conteúdos do primeiro lançamento. Os programadores estimam o esforço necessário e depois a calendarização é feita. Isto leva alguns dias a ficar concluído.
3. Iteração para lançamento – esta fase inclui bastantes iterações do sistema antes do lançamento. Assim, a lista feita na fase de planeamento é dividida em pequenas iterações que levam entre uma a quatro semanas para implementar. O cliente decide quais as *stories* a serem escolhidas a seguir. Os testes funcionais são criados pelo cliente e executados no fim de cada iteração.
4. Produção – é feita uma verificação do desempenho do sistema e são executados testes extra antes de ser enviado para o cliente. Podem ser encontradas mudanças necessárias e assim, tem de se decidir se elas são incluídas nesse lançamento ou posteriormente.
5. Manutenção – depois de feito o lançamento do sistema, a equipa tem de manter o sistema em produção e continuar a produzir novas iterações. Nesta fase é requerido um esforço para apoio ao cliente, o que pode abrandar a velocidade com que o sistema é desenvolvido.
6. Morte – quando não houverem mais *stories* a serem implementadas é produzida a documentação necessária do sistema, e não são feitas mais alterações na arquitetura, design ou código desenvolvido. Esta fase pode dever-se ao fato dos requisitos satisfazerem o cliente, ou o sistema não ir ao encontro do resultado esperado ou porque o cliente não consegue suportar os custos de desenvolvimento.

A metodologia tem diferentes papéis a serem desempenhados pelas pessoas envolvidas no projeto. Estes cargos podem passar por programador, cliente, *tester*, *tracker*, treinador, consultor e gestor.

A utilização de XP permite o desenvolvimento de software, apesar dos requisitos serem constantemente alterados. Das principais caraterísticas presentes na metodologia estão iterações curtas com pequenas implementações e rápido

feedback, participação do consumidor, comunicação e coordenação, documentação reduzida, e programação a pares (Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002).

O sucesso da boa implementação da metodologia *eXtreme Programming* num projeto depende de fatores como a resistência às práticas ou princípios por parte dos membros do projeto e cliente, ou a tecnologia a usar poder também fornecer obstáculos para o sucesso de um projeto XP. Uma tecnologia que não suporta elevadas mudanças ou obriga a grandes tempos de feedback não é adequada para um processo XP.

3.5.6 *Scrum*

Das metodologias com mais sucesso no desenvolvimento de software ágil encontra-se o *Scrum*. Desenvolvido na década de 1990, por Ken Schwaber e Jeff Sutherland, esta abordagem empírica aplica as ideias de processos de controlo industrial para o desenvolvimento de sistemas, reintroduzindo assim, ideias de flexibilidade, adaptabilidade e produtividade (Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002).

Scrum foca-se em como os membros de uma equipa devem funcionar de modo a produzir um sistema flexível num ambiente em constante mudança.

Em 2009, o cofundador do *Scrum*, Ken Schwaber, fundou o Scrum.org, uma organização mundial que se dedica a aperfeiçoar os profissionais de software. No Scrum.org, encontramos a documentação dos aspetos mais importantes da metodologia. O *Scrum* carateriza-se como sendo leve, fácil de compreender, mas ainda assim difícil de dominar. Os valores da metodologia listados são coragem, foco, compromisso, respeito e a transparência (Schwaber, 2009). Os três pilares que suportam qualquer implementação empírica de controlo de processos são: transparência, inspeção e adaptação (Jeff Schwaber, Ken; Sutherland, 2016).

- Transparência – aspetos importantes do processo devem ser visíveis e compreendidos para aqueles que são responsáveis pelo resultado. Esses

aspetos devem seguir um padrão comum, de forma que todos os observadores tenham um entendimento comum.

- Inspeção – os utilizadores devem inspecionar regularmente os artefactos *Scrum* e o progresso em relação ao objetivo do s*print* para detetar desvios indesejáveis. Este trabalho não deve afetar a execução normal do trabalho e deve, idealmente, ser realizado por inspetores qualificados.
- Adaptação – se na inspeção for detetada alguma anomalia de um processo, o inspetor deve ajustar o processo ou material em produção. Este ajustamento deve ser feito o mais rápido possível para minimizar danos adicionais.

Na figura 11 podemos observar o ciclo de vida que um projeto tem utilizando *Scrum*. Esta metodologia tem uma nomenclatura própria no que toca à designação de eventos e cargos desempenhados pela equipa. É feita de seguida uma explicação dos mesmos. (Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, 2002; Jeff Schwaber, Ken; Sutherland, 2016).

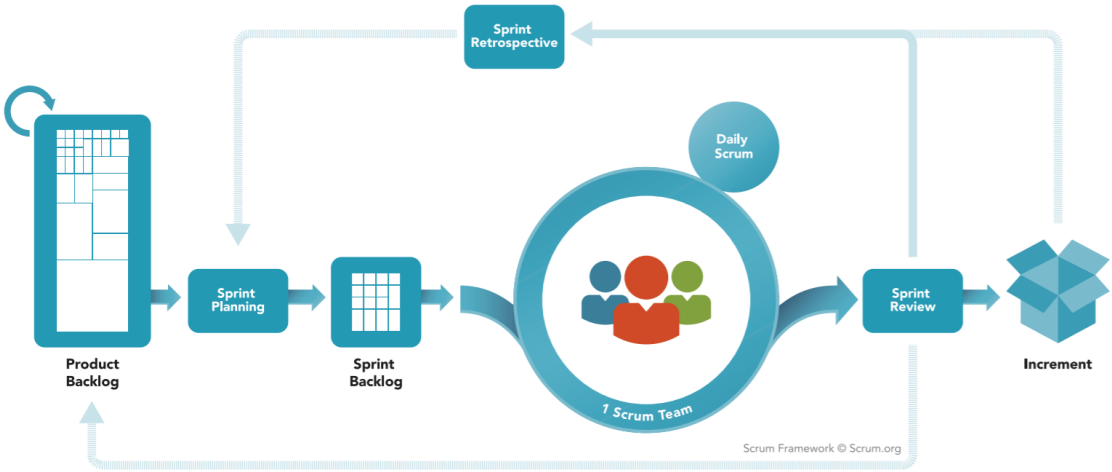

Figura 11. Ciclo de vida do processo Scrum retirado de (Schwaber, 2009)

3.5.6.1 Eventos

Sprint

Com a duração limitada a um mês, é o elemento no qual é feito um incremento potencialmente comerciável. Cada s*print* contém a reunião de planeamento do

sprint, as reuniões diárias, o trabalho desenvolvido, a revisão do s*print* e na retrospetiva do s*print*.

Durante a execução de uma s*print* não são feitas alterações que coloquem em causa o objetivo do s*print* ("*Sprint Goal*"), os objetivos de qualidade não diminuem. Os s*prints* têm sempre duração consistente durante todo o desenvolvimento. Um novo s*print* inicia-se assim que o anterior terminar.

Um s*print* pode ser cancelado antes do seu fecho, apesar de raramente acontecer. Apenas o *Product Owne*r pode cancelar um *Sprint*. Isto pode acontecer porque a empresa mudou de rumo ou porque as condições de mercado ou tecnologia alteraram-se. Devido à curta duração do s*print*, raramente é cancelado, mas quando este elemento ocorre, todos os itens do *Product Backlog* concluídos são revistos.

Sprint Planning

É feito o planeamento do trabalho a executar. Tem a duração máxima de 8 horas para um s*print* de um mês. Este valor pode variar com s*prints* de duração mais curta.

Devem ser utilizados como inputs o *Product Backlog*, o último incremento do produto, a capacidade estimada da equipa de desenvolvimento e o seu desempenho no passado. O número de itens selecionados é da responsabilidade da equipa de desenvolvimento. Depois de selecionados os itens, a equipa define um *Sprint Goal*.

O *Sprint Go*al é um objetivo que pode ser atingido através da implementação de itens do *Product Backlog*. Este *goal* dá uma linha de orientação à equipa de desenvolvimento acerca do desenvolvimento e oferece alguma flexibilidade no que diz que respeito à funcionalidade implementada durante o *sprint*.

Caso o trabalho desenvolvido resulte em algo diferente daquilo que era esperado, a equipa tem de colaborar com o *Product Ow*ner para negociar o âmbito do *Sprint Backlog*.

Daily Scrum

Reunião diária com limite máximo de 15 minutos para a equipa de desenvolvimento sincronizar atividades e criar um plano para as próximas 24 horas.

É inspecionado o trabalho desde a última *Daily Scrum Meeting*. Esta reunião serve para evitar complexidades e desencontros entre elementos da equipa de desenvolvimento e tem lugar sempre no mesmo sítio à mesma hora.

Sprint Review

Para *sprints* de um mês, a *Sprint Review* tem duração de 4 horas. Serve para inspecionar o incremento e adaptar o *Product Backlog* se necessário.

Durante a revisão a equipa *Scrum* e os *Stakeholders* colaboram sobre o que foi feito no *sprint*, apesar de esta ser informal, servindo assim para estimular a colaboração e feedback.

Sprint Retrospective

Reunião na qual a equipa tem a oportunidade para realizar uma introspeção e criar um plano de melhoramento. É delimitada a 3 horas para *sprints* de um mês.

Ocorre depois da *Sprint Revie*w e antes da próxima *Sprint Planning*.

3.5.6.2 Cargos

Product Owner

Pessoa responsável pelo projeto, por gerir e controlar o *Product Backlog*. Escolhido pelo *Scrum Master*, pelo cliente e pela gestão. Toma todas as decisões relacionadas com o *Product Backlog*, participa na estimativa de esforço necessário e transforma os assuntos do *Backlog* em funcionalidade a desenvolver.

Scrum Master

Responsável por assegurar que o projeto é levado de acordo com as práticas, valores e regras estabelecidas pela metodologia *Scrum* e que progride de acordo com o plano.

Interage com a equipa, cliente e com a gestão e está responsável por assegurar que os impedimentos no processo são removidos para manter a equipa o mais produtiva possível.

Development Team

Constituída pelos profissionais que fazem o *sprint* para entregar um incremento "*Done*". A equipa de desenvolvimento tem a responsabilidade de organizar e gerir o seu próprio trabalho.

A dimensão da equipa tem entre os três e os nove elementos para que o trabalho não tenha constrangimentos de produtividade e para não haver demasiada complexidade na gestão do processo.

3.5.6.3 Artefactos

Product Backlog

Lista ordenada de tudo o que é necessário no produto e única fonte de requisitos. A sua manutenção recai sobre o *Product Owner*, que tem como responsabilidade o conteúdo, disponibilidade e ordenação do mesmo.

Visto que este é um artefacto em constante alteração, nunca está completo, adquirindo valor e fornecendo feedback à medida que é produzido. Os itens presentes no *Product Backlog* contém atributos como descrição, prioridade, estimativa e valor. Os itens com maior prioridade são mais claros e detalhados ao contrário dos menos prioritários.

O *Product Owner* tem de monitorizar o *sprint* para que este cumpra com o *Sprint Goal* traçado no *Sprint Planning*. Compara a quantidade de trabalho conseguido com as *Sprints Reviews* anteriores e avalia o progresso em termos do trabalho

projetado face ao tempo desejado para atingir o objetivo, transmitindo aos *stakeholders* essa informação.

Para avaliar o progresso ou tendência que o projeto está a ter, o *Product Owner* pode recorrer a *burndowns*, *burnups* ou fluxos cumulativos. Este tipo de informação não substitui, no entanto, a importância do conhecimento empírico. Na secção 3.6.1 apresenta-se uma explicação para este tipo de gráficos utilizados, visto que alguns estão instanciados na *framework*.

Sprint Backlog

É o conjunto de itens do *Product Backlog* selecionados para o *sprint*, juntamente com o plano para os entregar e cumprir assim o *Sprint Goal*.

Contém o detalhe necessário para que sejam identificadas alterações necessárias durantes a *Daily Scrum* e assim, a equipa altera o *Sprint Backlog* durante o *sprint*.

Quando a equipa de desenvolvimento está a trabalhar num *sprint*, o trabalho que resta no *Sprint Backlog* pode ser somado de forma a entender a probabilidade de atingir o *Sprint Goal*.

Incremento

Este artefacto diz respeito à soma de todos os itens pertencentes ao *Product Backlog*, referentes ao *sprint* que foi terminado, juntamente com os incrementos concluídos anteriormente. No final do *sprint*, o incremento tem de estar num estado utilizável, "*Done*", caso contrário não é considerado concluído. O incremento tem de se encontrar pronto para ser passado ao cliente, independentemente do *Product Owner* decidir, ou não, entregá-lo.

Done

Numa equipa de *Scrum*, a definição de *Done* é atingida quando o trabalho feito sobre um incremento do *Product Backlog* está concluído. O objetivo de cada *sprint* é a produção de incrementos que possam ser lançados no mercado.

A definição de *Done* tem de ser clara e transparente, de modo a que todos os elementos da Equipa *Scrum* a sigam.

A adição de um novo incremento deve ser testada, de forma exaustiva, com os incrementos anteriores, para que funcionem em conjunto.

3.6 Métricas

Métricas são medidas que permitem aos responsáveis por um projeto saber como este está a ser desenvolvido e assim entender as causas que podem estar a prejudicar a equipa e o que deve ser feito para resolver esses problemas.

"*Measurement is used to assess situations, track progress, evaluate effectiveness, and more*" (Fenton, Norman; Bieman, 2014).

Uma das afirmações mais conhecidas no tópico relacionado com métricas é feita pelo físico matemático e engenheiro Lord Kelvin que diz "*if you cannot measure it, you cannot improve it*" (Kupiainen, Eetu; Mäntyla, Mika V.; Itkonen, 2015).

Existe muita literatura no que toca a métricas de software. Esta investigação é, no entanto, orientada maioritariamente para a metodologia tradicional. Como a investigação sobre métricas na metodologia ágil é escassa, as organizações que usam métodos ágeis utilizam métricas que provêm da metodologia tradicional ou métricas descritas na literatura ágil ou então criam métricas de acordo com aquilo que acham necessário para avaliar um parâmetro do projeto.

Pulford e Shirlaw dividem a motivação para as métricas em quatro categorias, são elas (Pulford et al., 1996):

- Planeamento e estimação do projeto;
- Gestão e controlo do projeto;
- Compreensão da qualidade e objetivos de negócio;

- Melhoraria da comunicação no desenvolvimento de software, processos e ferramentas.

Para contextualizar melhor algumas das métricas relevantes na *framework*, foi elaborada, de seguida, uma explicação do tipo de gráficos listados na secção 3.5.6.3, bem como, uma área muito utilizada denominada de *Earned Value Management* (EVM).

3.6.1 Gráficos

O *sprint burndown*, figura 12, mostra o progresso dentro do *sprint* para que a equipa chegue ao objetivo que definiu para esse *sprint* (*Sprint Goal*). À medida que itens do *backlog* são concluídos, o gráfico fornece o ritmo a que é feito, bem como a quantidade de progresso. Existe uma linha que mostra o ritmo ideal para a conclusão daquele *sprint* como é possível observar na figura com a linha a tracejado (Hayes et al., 2014).

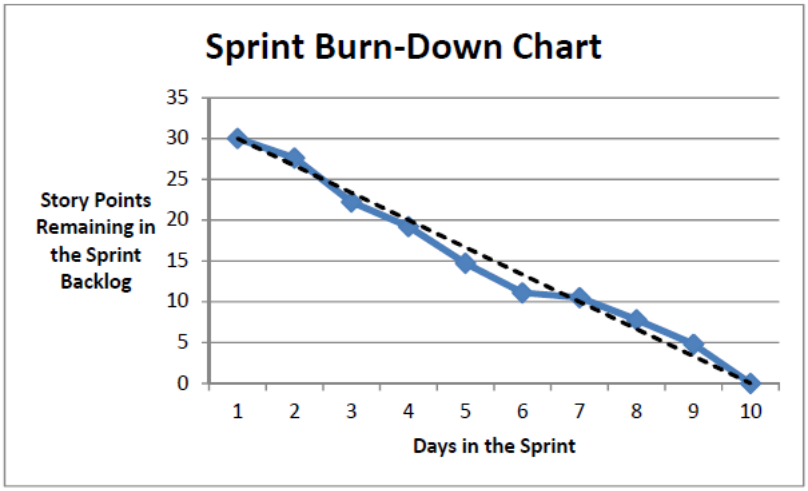

Figura 12. Exemplo de gráfico sprint burndown retirado de (Hayes et al., 2014)

O gráfico de *burnup* mostra o progresso feito para a conclusão do projeto. Este apresenta, normalmente, duas linhas, sendo que uma diz respeito a todo o âmbito do projeto e outra concerne ao trabalho concluído. O projeto é dado como concluído quando as duas linhas se encontram. Na figura 13 está representado um gráfico elucidativo dessa mesma situação (Hayes et al., 2014).

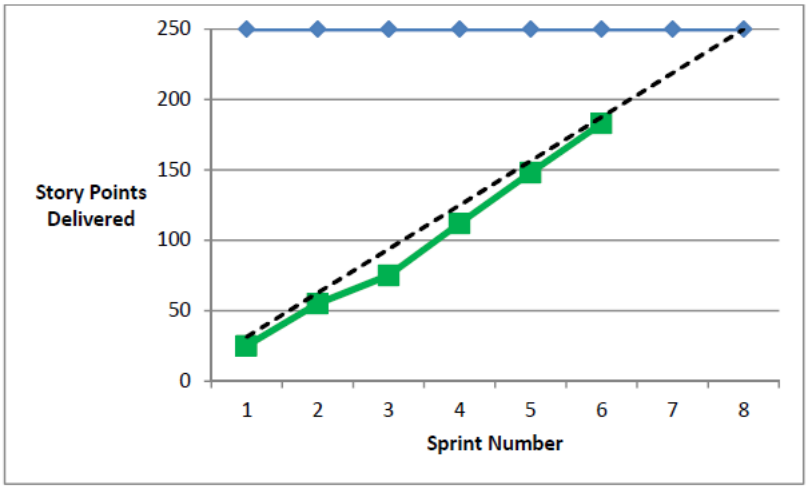

Figura 13. Exemplo de gráfico burnup retirado de (Hayes et al., 2014)

Os fluxos cumulativos, figura 14, servem o mesmo propósito que os gráficos *burndown* e *burnup*, mas têm a capacidade de compilar mais informação num mesmo gráfico. São, assim, gráficos que possuem várias áreas representativas da quantidade de trabalho em diferentes fases do desenvolvimento. Algumas dessas áreas são de inventário, iniciado, design, codificado e completado (Cabri & Griffiths, 2006).

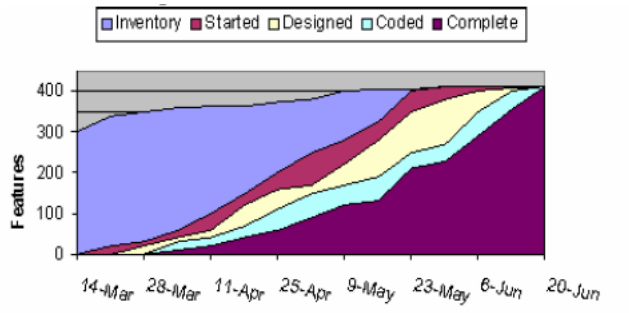

Figura 14. Exemplo de gráfico fluxo cumulativo retirado de (Cabri & Griffiths, 2006)

3.6.2 Earned Value Management

As técnicas de EVM surgiram inicialmente na América, nos anos 60, para projetos federais dos Estados Unidos; sendo aí, ainda hoje, obrigatório o seu uso por qualquer projeto federal com elevado custo (Ghosh, 2015).

Tradicionalmente, o EVM mede divergências financeiras em termos de tempo, custo e qualidade, podendo assim compreender o rumo do projeto e redirecionar para o rumo certo, visto que a aplicação do EVM tende a ser em projetos de

grande dimensão (Nkiwane1 et al., 2016). Estas medidas são observadas em valores de *Planned Value* (PV), *Earned Value* (EV) e *Actual Cost* (AC).

Medidas de cálculo como o *Schedule Variance* (SV) e *Cost Variance* (CV) são indicativas de desempenhos no passado. Estas medidas depois permitem-nos obter outras medidas como o *Schedule Performance Index* (SPI) e o *Cost Performance Index* (CVI) como podemos observar no caso de estudo na secção 4.3.2 (Nkiwane1 et al., 2016).

Para obtenção de algumas destas medidas é necessária a recolha de dados relativa ao orçamento do projeto (*Budget at Completion* – BAC) e à percentagem de trabalho concluída (*Actual Percentagem Complete* – APC).

3.7 ISO 25010 ou modelo de qualidade de software

Com base no IEEE (IEEE, 1997), apresenta-se em seguida uma definição, bastante consensual, de Qualidade de Software, "*(1) (A) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs, for example, conform to specifications. (B) The degree to which software possesses a desired combination of attributes. (C) The degree to which a consumer or user perceives that software meets his or her composite expectations. (D) The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer. (2) The ability of software to satisfy its specified requirements"*.

Todo o software produzido a um nível empresarial, tem de obedecer a um padrão responsável por medir vários aspetos da qualidade. Com este propósito foram desenvolvidos vários modelos (figura 15) que permitem compreender a qualidade que um software produzido tem, no contexto em que está inserido. Como podemos observar na imagem, há vários modelos que foram influenciados por modelos existentes no passado. É também possível constatar que até ao ano 2000 os modelos de qualidade eram básicos e com o passar do tempo foram sendo customizados para diferentes áreas de aplicação.

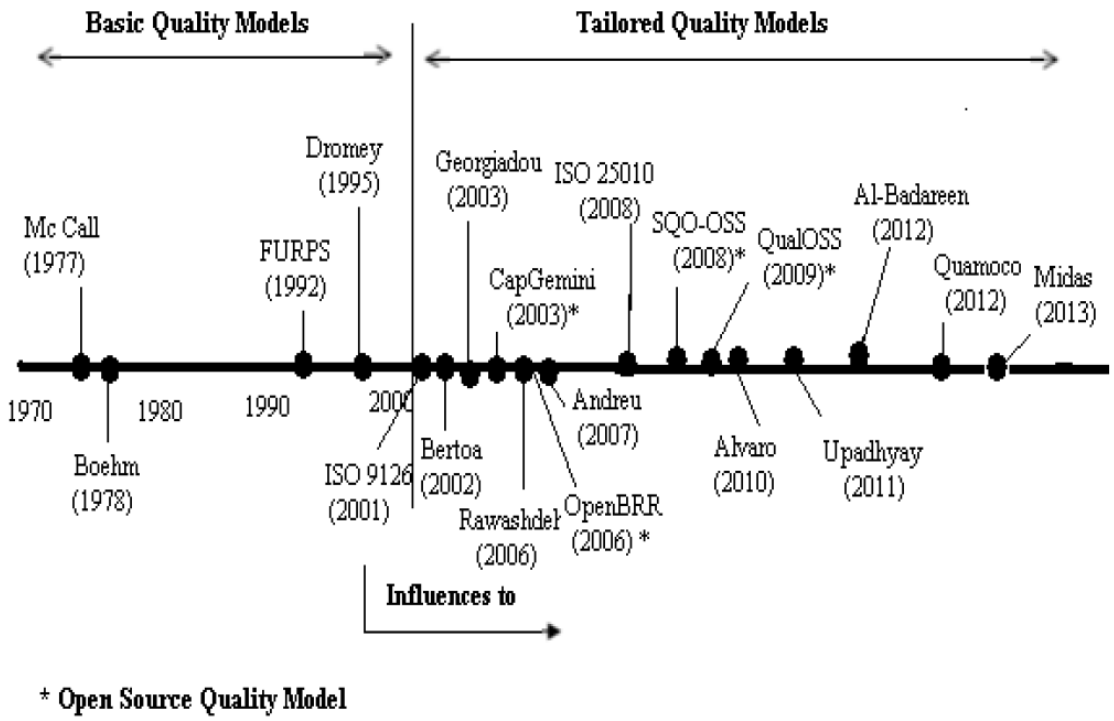

Figura 15. Modelos de qualidade existentes retirado de (Miguel, Mauricio, & Rodriguez, 2014)

Para este projeto será utilizado o modelo de qualidade desenvolvido pela *International Organization for Standardization* (ISO) com o modelo ISO 25010 que sucede à norma ISO 9126 (Hussain & Mkpojiogu, 2015). A norma ISO 25010, figura 16, está dividida em 8 categorias, algumas delas provêm da antiga norma (Miguel et al., 2014).

A ISO 25010 define qualidade de software em uso como o grau em que um produto usado por utilizadores específicos vai ao encontro das suas necessidades para alcançar objetivos específicos com eficácia, eficiência, segurança e satisfação, num determinado contexto de utilização (Hussain & Mkpojiogu, 2015).

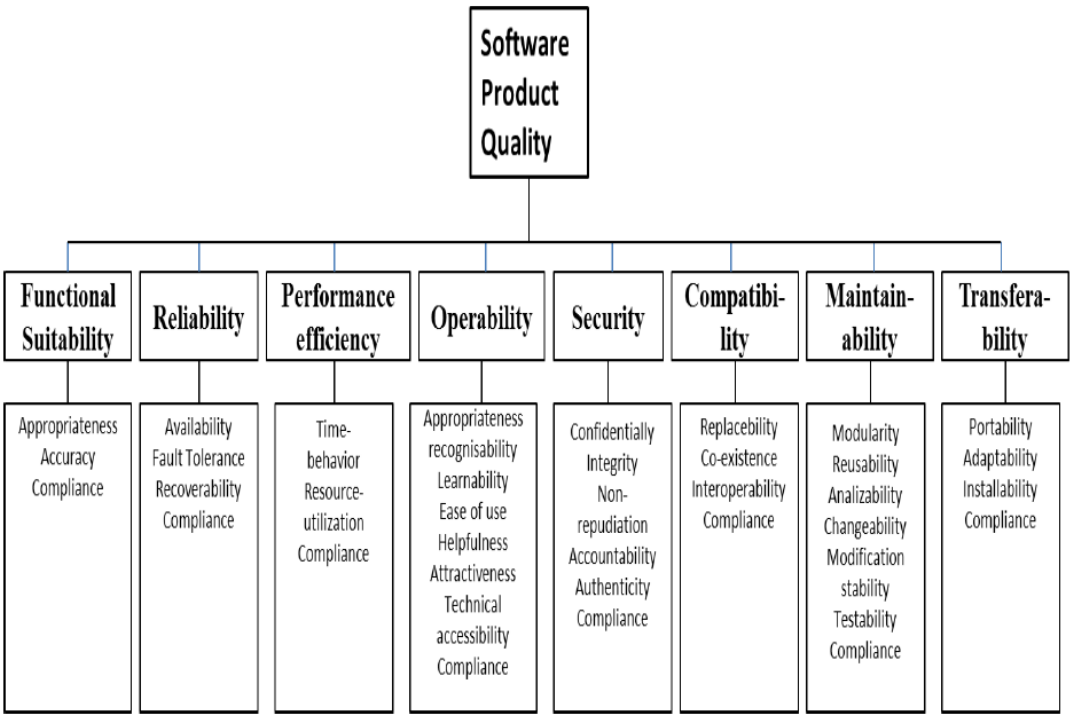

Figura 16. Modelo ISO 25010 retirado de (Miguel et al., 2014)

A ISO 25010 vê a usabilidade como um constituinte da qualidade em uso e como um atributo de qualidade de software composto por três elementos como na ISO 9241-11, mas com foco na qualidade em uso. As principais características aqui definidas são a eficácia, a eficiência e a satisfação em uso, sendo esta última caracterizada como a probabilidade de um software ser mais aprazível e assegurar maior conforto e confiança. À semelhança do seu antecessor 9126, o modelo de qualidade ISO 25010 define qualidade de um sistema como a extensão para a qual o sistema satisfaz as necessidades constadas e implícitas dos vários *stakeholders* (Hussain & Mkpojiogu, 2015).

4. Resultados obtidos

O trabalho elaborado nesta dissertação teve foco numa metodologia de desenvolvimento ágil, mais concretamente o *Scrum*. Esta metodologia permite rápidas entregas de produto ao cliente e compreende que todo o projeto está sujeito a alterações e que estas devem ser aceites e colocadas em futuras iterações do desenvolvimento do projeto.

Para perceber o desenvolvimento que o projeto está a ter numa determinada altura, os gestores de projeto recorrem a métricas, a maioria reconhecidas na comunidade de desenvolvimento de software. Estas métricas podem centrar-se em diferentes aspetos do desenvolvimento, e foi assim, necessário dividir as métricas em categorias para que não tornassem o modelo proposto muito extenso e também permitir que diferentes partes do desenvolvimento pudessem ser avaliadas.

Abaixo será apresentado o modelo preliminar, elaborado numa primeira fase da dissertação, onde foram escolhidas as métricas mais referenciadas em projetos de desenvolvimento de software (Kupiainen, Eetu; Mäntyla, Mika V.; Itkonen, 2015). Métricas essas que, numa fase posterior, sofreram um refinamento, juntamente com a construção da *framework*.

4.1 Modelo preliminar

Na tabela 4, estão listadas as métricas mais referenciadas em projetos industriais que apresentam potencial para o projeto em estudo. Ainda assim, foi possível refinar a seleção de métricas que irão servir para avaliação dos projetos na fase posterior da dissertação. A *framework* que forma estas métricas preliminares será reajustada com o tempo, à medida que os resultados forem analisados.

Tabela 4. Métricas selecionadas retirado de (Kupiainen, Eetu; Mäntyla, Mika V.; Itkonen, 2015)

ID	Métrica	Estudo
1	*Velocity*	Número de cenários desenvolvidos por programador por semana; Funcionalidades desenvolvidas por iteração
2	*Effort Estimate*	Esforço estimado por *story* num par de dias; Tarefas são divididas em kits de dois a cinco dias de trabalho de staff
3	*Customer Satisfaction*	Não é claramente definido no estudo primário.
4	*Defect count*	Número de defeitos depois da 1ª ronda de testes
5	*Technical debt*	Mostra o estado de cada categoria de dívida técnica por equipa; Dívida técnica na quantidade de horas necessárias à resolução de todos os problemas que aumentam a mesma, calculada por uma ferramenta de terceiros, nomeadamente SONAR
6	*Build status*	*Build* com erros (danificado) ou não
7	*Progress as working code*	Produto é demonstrado ao cliente que depois dá feedback
8	*Lead time*	O tempo em média que demora para um pedido atravessar o processo do início ao fim; Tempo que leva para um requisito atravessar um subprocesso ou o processo inteiro
9	*Story flow pecentage*	Tempo de implementação estimado por tempo de implementação real × 100
10	*Velocity of working features*	O tempo que demora para clarificar uma funcionalidade do cliente em requisitos que podem ser implementados
11	*Story percentage complete*	Visualização das *stories* completas até à data, com auxílio de uma *framework*
12	*Number of test cases*	Não é definido no estudo primário
13	*Queue time*	Tempo em média entre subprocessos que o pedido espera
14	*Processing time*	O tempo em que um pedido está a ser trabalhado por uma pessoa ou equipa
15	*Defect trend indicator*	Indica se o montante de defeitos na semana que se segue vai aumentar, diminuir ou manter-se em reação à semana atual

ID	Métrica	Estudo
16	*Work In Progress* (WIP)	Quantidade de *stories* por fase de trabalho; Quantidade de itens de trabalho por fase
17	*Number of unit tests*	Número de testes unitários
18	*Cost types*	Custo de distribuição de um requisito
19	*Variance in handovers*	Mudanças na quantidade de transferência de requisitos
20	*Defects deferred*	Quantidade de defeitos que são conhecidos, mas não são resolvidos para o lançamento
21	*Predicted number of defects in Backlog*	Todos os defeitos conhecidos e por resolver no projeto
22	*Test coverage*	Quanto código fonte é executado durante a fase de teste
23	*Test growth ratio*	Diferença do número de testes por diferença da quantidade de código fonte
24	*Check-ins per day*	Número de *commits* (código, testes automatizados, especificação) por dia
25	*Cycle time*	Tempo que demora para uma *story* de tamanho X a ser completada

4.2 Modelo proposto

Como podemos observar na *framework* (figura 17), este modelo encontra-se dividido em cinco áreas, são elas a *Earned Value Management* (EVM), o Planeamento, o Desenvolvimento, a Qualidade e os *Stakeholders*. Foi dada à *framework* o nome de *Agile Metric Framework* (AMF), visto que tem aplicação em projetos ágeis. A escolha do nome passou por ser em inglês pois, na comunidade científica a língua predominante é a inglesa.

Como já foi referido anteriormente, a divisão destas métricas centrou-se em dois aspetos: facilitar a medição em diferentes aspetos do desenvolvimento do software e empregar assim apenas um pequeno grupo de métricas e também para não agrupar todas numa só área o que tornaria a *framework* muito difícil utilizar.

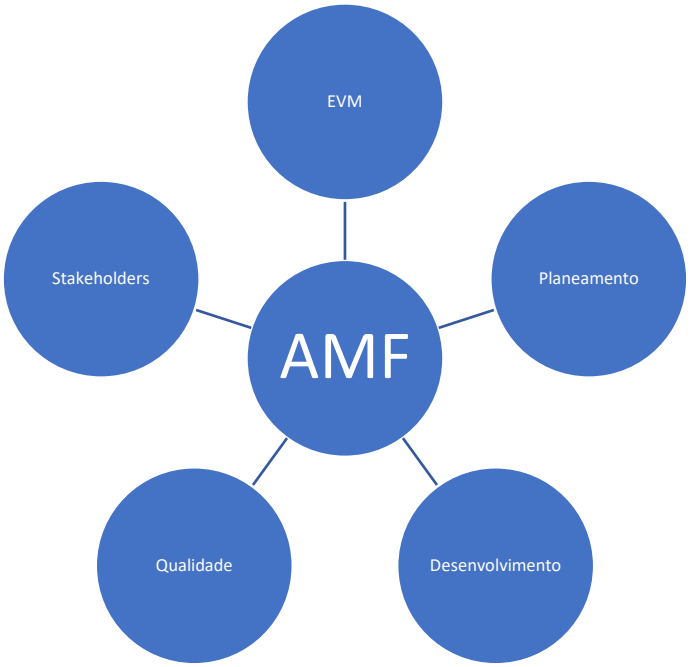

Figura 17. Estrutura da Agile Metric Framwork (AMF)

O aspeto da *framework* é o de uma estrela, pois todas as áreas, estão a convergir para o centro da *Agile Metric Framework*, podendo assim haver trocas de informação de uma área para a outra. Desta forma, p.e. uma métrica de EVM pode necessitar de uma medida disponível no Planeamento ou Desenvolvimento.

4.2.1 *Earned Value Management* (EVM)

Uma equipa de desenvolvimento tem como principal preocupação conseguir produzir valor com o investimento que um cliente faz nessa equipa. Com esse princípio em mente, os custos do projeto devem ser avaliados para poder determinar se os planos de *release* tem valor de negócio suficiente ou se é necessário repensá-los (Sulaiman et al., 2006).

As métricas EVM auxiliam o gestor a medir a performance do projeto com métricas associadas a custos (PMI, 2013; Sulaiman et al., 2006). Esta é uma área bastante conhecida entre gestores, pois é através dela que consegue ter uma perspetiva do custo que o projeto tem e do retorno que está a obter com o seu desenvolvimento.

Esta área é composta por 8 métricas, 6 são calculadas e 2 são valores definidos pela equipa (uma é o orçamento definido entre a equipa e o cliente para o projeto e outra corresponde aos custos orçamentados) como é possível constatar na tabela 5. Estas métricas pretendem avaliar o projeto numa perspetiva de custo, tempo, qualidade e índices de desempenho.

Tabela 5. Métricas da área de EVM/Financeira

Métrica	Medição
***Budget Cost At Completion* (BAC)** (Ghosh, 2015; Sulaiman et al., 2006)	Orçamento planeado para a *release*
***Actual Cost* (AC)** (Ghosh, 2015; Sulaiman et al., 2006)	Custos orçamentados no PV para a *release*
***Earned Value* (EV)** (Ghosh, 2015; Sulaiman et al., 2006)	EV = APC × BAC
***Planned Value* (PV)** (Ghosh, 2015; Sulaiman et al., 2006)	PV = PPC × BAC
***Cost Variance* (CV)** (Ghosh, 2015; Sulaiman et al., 2006)	CV = EV – AC
***Schedule Variance* (SV)** (Ghosh, 2015; Sulaiman et al., 2006)	SV = EV – PV
***Schedule Performance Index* (SPI)** (Ghosh, 2015; Sulaiman et al., 2006)	SPI = EV ÷ PV
***Cost Performance Index* (CPI)** (Ghosh, 2015; Sulaiman et al., 2006)	CPI = EV ÷ AC

4.2.2 Planeamento/Gestão

Para uma maior probabilidade de sucesso num projeto, este tem de ter uma boa base de planeamento. A capacidade de um gestor conseguir fazer um bom *roadmap* do projeto em mão, pode determinar o sucesso ou insucesso do mesmo. Contrariamente à área anterior, onde o foco é maioritariamente financeiro, aqui é importante compreender como a equipa está a conseguir cumprir o plano. Para isso, as métricas escolhidas são de suporte ao desenvolvimento, como podemos ver na tabela 6.

Esta é uma área onde começa a ser percetível à equipa de trabalho se o projeto está no rumo certo para ser finalizado. No entanto, serve, mais de apoio aos *Product Owners* (PO). Como algumas das métricas estão relacionadas com a terminologia ágil, esta é uma fase cuja compreensão não é tão linear para o cliente, servindo assim mais de suporte para o PO. O cliente pode, no entanto, auxiliar a equipa de desenvolvimento naquilo que é o valor pretendido no projeto para determinar o *Business Value Delivered*.

Tabela 6. Métricas da área Planeamento/Gestão

Métrica	Medição
Business Value Delivered **(BVD)** (Hartmann & Dymond, 2006)	Valor de negócio a ser entregue em cada *feature*
Release Burndown (Hayes et al., 2014)	Gráfico que permite observar o rumo do projeto e o progresso em cada *sprint*
Sprint Burndown (Hayes et al., 2014)	Gráfico que permite observar quantidade de trabalho existente no *sprint backlog*
Velocidade (Anderson, 2005; Hayes et al., 2014)	Velocidade de *story points* por *sprint*
Planned Percentage Complete **(PPC)** (Sulaiman et al., 2006)	Número de $sprints$ desenvolvidas ÷ Número de $sprints$ planeadas
Actual Percentage Complete **(APC)** (Sulaiman et al., 2006)	Número de $stories$ desenvolvidas ÷ Número de $stories$ planeadas

4.2.3 Desenvolvimento

Na área de Desenvolvimento são expostas métricas que estão diretamente relacionadas com o tempo de desenvolvimento da equipa. Estas métricas auxiliam o gestor a perceber se deve ou não realocar recursos para uma determinada área do desenvolvimento, de forma a aumentar a produção de produto pronto a ser expedido para o cliente. Na tabela 7 segue-se a enumeração das métricas aqui utilizadas.

Tabela 7. Métricas para área de Desenvolvimento

Métrica	Medição
Lead Time (Mujtaba et al., 2010)	Tempo total para um processo atravessar todo o desenvolvimento
Queue Time (Mujtaba et al., 2010)	Tempo de espera em que um processo está à espera de recursos
Processing Time (Mujtaba et al., 2010)	Tempo em que um processo está a ser trabalhado
Story Flow Percentage (Kupiainen, Eetu; Mäntyla, Mika V.; Itkonen, 2015)	Percentagem de trabalho efetuado da *story* em desenvolvimento
Work in Progress **(WIP)** (Anderson, 2005; Petersen & Wohlin, 2011; Polk, 2011; Seikola et al., 2011)	Número *stories* em desenvolvimento

4.2.4 *Stakeholders*

Uma área de importante relevo é o *Stakeholder*. É sobre este que se centram as atenções para que o produto esteja de acordo com os requisitos, depois do produto desenvolvido. Após entregar o produto ao cliente, este vai mostrar a sua satisfação ou insatisfação do mesmo, o que pode originar a reutilização dos serviços prestados pela equipa.

Apesar de tudo, esta área não tem foco apenas no cliente, mas também na comunicação entre a equipa de desenvolvimento. Uma boa comunicação entre membros de equipa de área de desenvolvimento diferentes, *developer vs tester*, pode por vezes ser crucial para um bom desenvolvimento.

As métricas relativas a esta área estão expostas na tabela 8.

Tabela 8. Métricas para área de Stakeholders

Métrica	Medição
Satisfação de cliente (Kohlbacher et al., 2011)	Questionário de satisfação fornecido ao cliente
Tempo de feedback	Tempo de feedback entre cliente e equipa e a comunicação dentro dos diferentes cargos da equipa

4.2.5 Qualidade

Como última área exposta no modelo temos a Qualidade. De forma a tentar combater a possível insatisfação do cliente, erros ou defeitos encontrados no produto devem ser resolvidos previamente à entrega ao cliente. Como tal, após o fecho de cada *sprint* e início de uma nova, deve observar-se os defeitos existentes no *Backlog* para proceder a um ajuste da equipa, se necessário. Estas métricas, tabela 9, servem para compreender, com base no código produzido, os testes que estão a ser feitos e quantos existem para ser corrigidos, o que não impossibilita que alguns escapem e que o próprio cliente dê por eles.

Tabela 9. Métricas para área de Qualidade

Métricas	Medição
Defect Backlog (Staron et al., 2010)	Conjunto de defeitos conhecidos que faltam ser resolvidos
Build Status (Janus, André; Schmietendorf, Andreas; Dumke, Reiner; Jäger, 2012)	Número de *builds* feitas num *sprint*
Test Coverage (Janus, André; Schmietendorf, Andreas; Dumke, Reiner; Jäger, 2012)	$\mathrm{codeCoverageByTests} \div \mathrm{CompleteCode}$
Test Growth Ratio (Janus, André; Schmietendorf, Andreas; Dumke, Reiner; Jäger, 2012)	Medido o tamanho do teste face ao tamanho do código fonte
Defeitos diferidos (Green, 2011)	Correlaciona os defeitos encontrados no sistema com os defeitos encontrados pelo cliente em *live*

4.3 Caso de estudo

A abordagem metodológica escolhida para esta dissertação tem como resultado esperado um artefacto (a *framework*), testado num caso de estudo real de um projeto na área das TI. Com este caso foi possível observar e validar a *framework* num contexto real.

A *framework* foi utilizada no projeto *Unified Hub for Smart Plants* (UH4SP), cujo objetivo passa pelo desenvolvimento de uma arquitetura de software orientada a serviços e soluções tecnológicas, incorporando também o paradigma de IoT e *Industry* 4.0.

Este projeto, tem duração de 2 anos e está a ser liderado pela Cachapuz, empresa que atua e detém referências internacionais no âmbito do desenvolvimento de soluções para indústrias, tendo outros associados ao projeto que exploram as várias linhas de investigação possíveis, designadamente, o CCG (Centro de Computação Gráfica), no âmbito de

arquiteturas de referência de software para sistemas industriais no domínio de *Industry* 4.0 (desenvolvido pelo grupo EPMQ – "*IT Enginneering Process Maturity and Quality*") e no desenvolvimento de técnicas de sensorização (desenvolvido pelo grupo cVIG – "*Computer Vision Interaction and Graphics*"); a Universidade do Minho, no âmbito de modelo de simulação; e a empresa Eurotux Informática S.A, no âmbito de plataformas computacionais e de armazenamento.

O projeto UH4SP tem como objetivo desenvolver uma plataforma para integrar informação de unidades industriais distribuídas, permitindo o uso de informação adquirida de sistemas IoT para gestão de produção a um nível do grupo corporativo e processos colaborativos entre unidades industriais, fornecedores, transportadores e clientes. O consórcio foi composto por cinco entidades diferentes para o desenvolvimento de software, onde cada uma tinha contributos específicos expetáveis, desde arquitetura *cloud* até serviços industriais de software e aplicações móveis. A solução é baseada no paradigma da *Industry* 4.0, e IoT e tecnologias de computação. Os serviços UH4SP foram validados dentro do projeto, fazendo um conjunto de cenários piloto, na maioria relacionados com o domínio da indústria cimenteira.

O sistema de produção da Cachapuz é responsável por fazer controlo de entrada/saída de camiões, bem como, as atividades de carga/descarga e de comunicação com o ERP da unidade e o hardware industrial.

A arquitetura do sistema é implementada em várias camadas (N-camadas) de nós entre a *cloud* UH4SP e a IoT ao nível industrial. É na camada *cloud* que uma arquitetura de micro serviços é implementada para suportar as funcionalidades de execução. Um nível intermediário, tipicamente localizado nas fronteiras de cada unidade industrial, implementa capacidades distribuídas, nomeadamente relacionadas com computação, rede e armazenamento. Finalmente, aplicações de negócio, aplicações *web* (*desktop*) ou aplicações móveis são as principais interações com os atores humanos. Eles utilizam micro-serviços na *cloud* para executar os seus processos (N. Santos, Rodrigues, et al., 2018).

O projeto UH4SP tem como objetivo as seguintes caraterísticas para a arquitetura do sistema:

- Otimizar o *upload* de informação da produção para a *cloud*;

- Gerir informação de múltiplas instalações industriais por parte do grupo corporativo;
- Permitir o uso de serviços por outras entidades.

A fase de requisitos foi conduzida por uma equipa de analistas composta por elementos de cada entidade, apontando para definir os requisitos iniciais. Só depois de começar os ciclos de entrega é que, cada equipa foi responsável por refinar os seus requisitos. O levantamento de requisitos começou por listar um conjunto de expetativas dos *stakeholders,* rumo ao *roadmap* do produto ("*product roadmap*") a desenvolver, abrangendo todo o projeto. Apenas as funcionalidades prioritárias ("*Just-enough*") foram detalhadas (N. Santos, Pereira, Ferreira, & Machado, 2018), compondo assim o "*Minimum Valuable Product*" (MVP). A lista de expetativas incluiu 25 expetativas, categorizados por ambiente, arquitetura, funcional e de integração. Elas estão relacionadas com as necessidades de negócio que depois permitiram descrever requisitos funcionais, modelados em casos de uso (*Unified Modeling Languange* – UML).

De seguida, a análise de requisitos incluiu a refinação os casos de uso e definição da árvore de decomposição funcional. O modelo de casos de uso foi composto por 37 casos de uso depois da refinação. De seguida foram introduzidas para o método *Four Step Rule Set* (4SRS) (Ferreira, Santos, Machado, Fernandes, & Gasević, 2014) de forma a obterá a arquitetura lógica da solução.

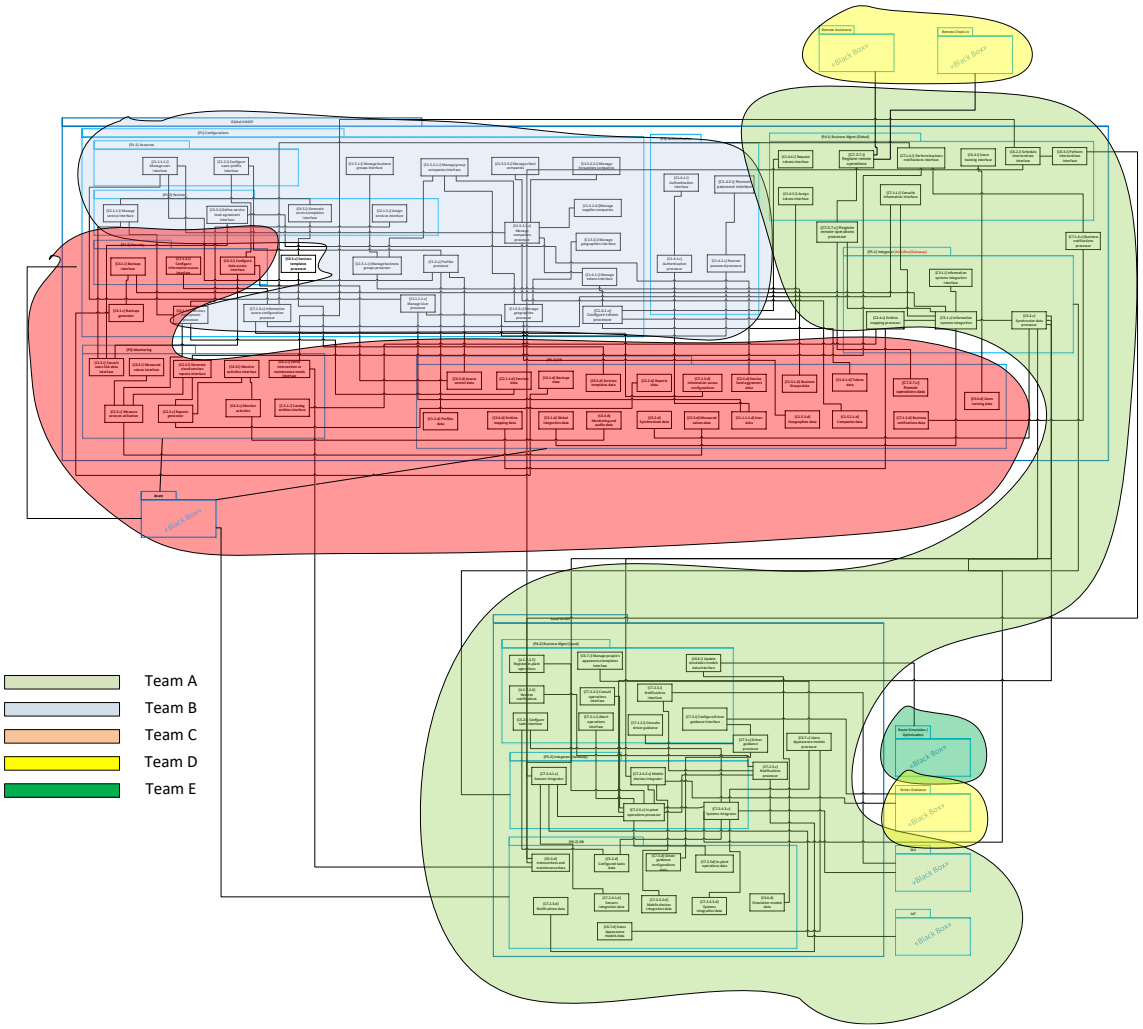

Figura 18. Modularização da arquitetura lógica retirado de (N. Santos, Rodrigues, et al., 2018)

Após obter a arquitetura, a mesma foi modularizada em 5 subsistemas, retratada na imagem da figura 18, que foram atribuídos pelas 5 equipas do projeto. A divisão foi feita com base nas contribuições que cada trazia para o consórcio, nomeadamente IoT, infraestrutura da *cloud*, aplicações *cloud* e sensores/sistemas embutidos. O foco deste estudo foi feito sobre a equipa CCG-EPMQ.

Durante a modelação "*Just-enough*", haviam 37 casos de uso e 77 componentes de arquitetura, depois de aplicar o método 4SRS. Depois de modularizar, 11 casos de uso dos 37 e 15 componentes dos 77, compuseram o módulo sobre análise. Finalmente, o refinamento dos requisitos resultara em 29 casos de uso, 18 funcionalidades refinadas, que depois derivaram 37 componentes de arquitetura do método 4SRS. Os requisitos foram assim, refinados, modelados

e validados com o consórcio (N. Santos, Pereira, Morais, et al., 2018) e só de seguida foram introduzidos de novo para o método 4SRS (Ferreira et al., 2014). No projeto, depois da modelação em casos de uso UML, o pacote de requisitos foi também composto por *wireframes*, para enriquecer a discussão.

Nas cerimónias de *Sprint Retrospective* da metodologia *Scrum*, os modelos eram objeto de feedback e, se necessário, os requisitos em falta eram incluídos. Depois do método 4SRS, esses 5 casos eram derivados de 11 componentes. O MVP que era composto por 94 componentes, era implementado no final desses *Sprints*. Estes componentes suportavam os cenários experimentais do projeto. Contudo as *releases* que se seguem devem ser desenvolvidas para que possam seguir o *roadmap* do produto.

Depois do feedback e consequente aprendizagem e reajustes, se necessário, a abordagem termina com a arquitetura lógica, que é depois utilizada como input para definir os itens do *backlog* necessários.

Para validação do modelo, este foi testado no âmbito da equipa do CCG, nomeadamente no grupo EPMQ. Trata-se de um projeto *greenfield*, cujo desenvolvimento é feito de raiz, que é normalmente adotado em projetos de desenvolvimento ágil.

Para este caso de estudo nem todas as métricas foram utilizadas, porque algumas delas, nomeadamente *test growth,* eram para ser utlizadas em projetos *brownfield*, que devem ser utilizadas em projetos já existentes. Outro ajuste feito foi o das métricas EVM, que foram ajustadas para serem utilizadas as horas em que trabalharam no projeto em vez da utilização dos custos que não puderam ser fornecidos.

O grupo EPQM teve, neste projeto, seis elemento na equipa, mas como é uma organização afiliada à universidade, acolhe alunos que estão a terminar os estudos ou que estão a trabalhar na tese de dissertação ou doutoramento e como tal, os recursos vão sendo reajustados com o passar do tempo. A juntar a esse fator, temos também a aprendizagem que a equipa se submete para determinada ferramenta ou metodologia de desenvolvimento.

De seguida serão expostos alguns exemplos das métricas presentes na *framework*. Devido ao contexto industrial do caso de estudo, alguns dados de medição tiveram de ser ajustados, visto que em algumas partes os dados já não existem, ou por confidencialidade poderem ser revelados.

Ao longo do desenvolvimento do projeto, a sua constituição variou. Os papéis existentes neste projeto foram os que constam na lista seguinte:

- *Business Owner* – BO;
- *Projet Manager* – PM;
- *Scrum Master* – SM;
- *Tech Leader* – TL;
- *Business Analyst* – BA;
- *Developer* – Dev;
- *Tester* – QA.

Como o software utilizado na gestão do trabalho da equipa utilizava as datas como eixo e outro o número do *sprint* foi feita uma atribuição do *sprint* com base no intervalo da data de início e fim. Isto para facilitar a apresentação de dados e tornar mais coerente a exibição dos mesmo. Como tal, na tabela 10, é listado o intervalo de tempo empregue em cada *sprint*.

Tabela 10. Intervalo de tempo de cada sprint

Sprint	**Data de início**	**Data de conclusão**
S#0	25/10/2017	10/11/2017
S#1	13/11/2017	17/01/2018
Sprint	**Data de início**	**Data de conclusão**
S#2	18/01/2018	16/02/2018
S#3	26/02/2018	28/03/2018
S#4	29/03/2018	19/04/2018
S#5	27/04/2018	17/05/2018

4.3.1 EVM/Financeira

Para as métricas de EVM, a equipa recorreu às horas empregues por cada membro do projeto em cada *sprint*. Na tabela 11, está uma contagem das horas que cada um despendeu até ao *sprint* final, *sprint* #5.

Tabela 11. Horas de trabalho de cada membro no projeto

	S#0	S#1	S#2	S#3	S#4	S#5
BO	3	10	0	6	1	1
PM	24	81	3	107	10	0
SM	8	35	0	0	0	0
TL	3	125,5	23	116	59	63
BA	60	185	88	100	60	12
Dev	0	87	166	395	148	128
QA	32	88	128	152	96	32
Total	130	611,5	408	876	374	236

Com os dados das horas, foi assim possível calcular as métricas alocadas à área de EVM. Alguns dos cálculos tiveram, no entanto, de ser ajustados para poder calcular cada um dos componentes apresentados na tabela 12.

Tabela 12 - Dados relativo às métricas EVM

	S#0	S#1	S#2	S#3	S#4	S#5
AC	130	611,5	408	876	374	236
EV	26,11	109,67	219,33	276,78	297,67	344,67
PV	78,33	156,67	235,00	313,33	391,67	470,00
CV	-103,89	-501,83	-188,67	-599,22	-76,33	108,67
SV	-52,22	-47,00	-15,67	-36,56	-94,00	-125,33
SPI	0,33	0,70	0,93	0,88	0,76	0,73
CPI	0,20	0,18	0,54	0,32	0,80	1,46

Como não foi possível obter os custos para este grupo de métricas, fica de seguida uma explicação de como foram obtidas cada uma das métricas para um dado *sprint*, neste caso o *sprint* #1.

***Actual Cost*:** foi utilizado o total número de horas empregues no *sprint* #1.

***Earned Value*:** foi utlizada a fórmula disponível na *framework* em que $EV = APC \times BAC$, onde APC é o número de *stories* existentes até o *sprint* atual a dividir pelo 90, que é o número total de *stories* para o projeto, e BAC foi atribuído como 470 horas (definido pela própria equipa da CCG-EPMQ).

Assim sendo, $EV = \frac{(5+16)}{90} \times 470 \Leftrightarrow \text{EV} \approx 109{,}67$.

***Planned Value*:** para obter este cálculo, a fórmula exposta na *framework* é de $PV = PPC \times BAC$, mas como a equipa não possuía o dado da percentagem do plano completa, esta utilizou a razão entre o *sprint* atual e o total de *sprints*.

Assim sendo, $PV = \frac{2}{6} \times 470 \Leftrightarrow PV \approx 156{,}67$.

***Cost Variance*:** é obtida pela diferença entre EV – AC.

Assim sendo, $CV = 109{,}67 - 611{,}5 \Leftrightarrow CV = -501{,}83$.

***Schedule Variance*:** é a diferença entre EV – PV.

Assim sendo, $SV = 109{,}67 - 156{,}67 \Leftrightarrow SV = -47$.

***Schedule Performance Index*:** o resultado da divisão entre EV por PV.

Assim sendo, $SPI = \frac{109{,}67}{156{,}67} \Leftrightarrow SPI \approx 0{,}70$.

***Cost Performance Index*:** o resultado da divisão entre EV por AC.

Assim sendo, $CPI = \frac{109{,}67}{611{,}5} \Leftrightarrow CPI \approx 0{,}18$.

Estas métricas da categoria EVM permitiram construir dois gráficos. No primeiro gráfico, figura 19, podemos constatar três curvas: a azul, a laranja e a cinza que correspondem à evolução ao longo do tempo do AC, EV e PV, respetivamente.

Com este gráfico pretendemos perceber se os custos, neste caso o número de horas que planeámos para o projeto (PV), não é inferior às horas que estamos realmente a despender (AC) e se com esse número de horas estamos a conseguir entregar algum valor ao cliente (EV). Idealmente, a curva de AC deve estar o mais próxima possível da PV, o que queria dizer que os custos, ou neste caso, as horas, estão a ser bem atribuídas e a curva de EV deve estar sempre acima de AC, sinal de que está a ser produzido algo positivo, caso contrário está

haver um esforço que não estará a resultar no máximo valor ganho possível e devem então ser feitos novos cálculos para ajustar essa situação.

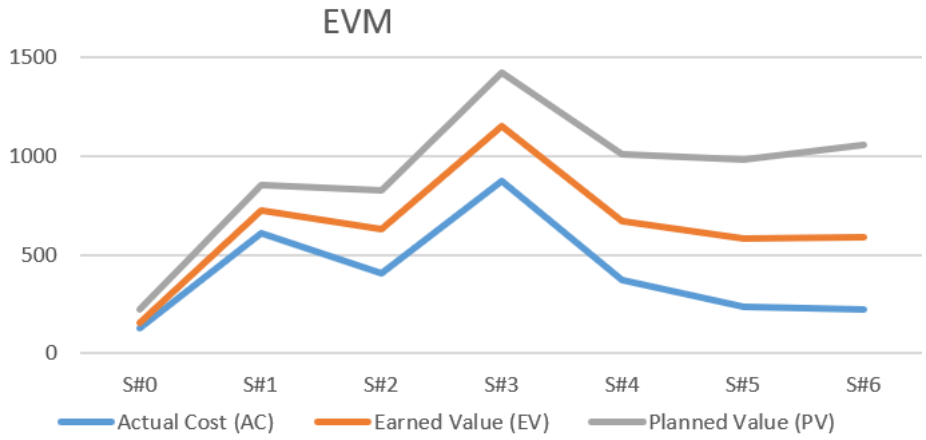

Figura 19. Gráfico das métricas AC, PV e EV

Já no gráfico da figura 20, fazemos um estudo sobre as últimas duas métricas recolhidas, SPI e CPI. Temos no eixo dos *xx*, o SPI e no eixo dos *yy*, o CPI e de seguida os valores de cada *sprint* para essas duas métricas são utilizados como coordenadas no gráfico, resultando assim 6 pontos. Após colocados os pontos é feito um estudo sobre onde cada ponto se encontra num dos quatro quadrantes possíveis e assim é retirado uma conclusão acerca do prazo e do custo que este *sprint* tem. Podemos observar de seguida as condições para um ponto estar em cada um dos quadrantes.

$1 \leq SPI \cap 1 \leq CPI \Leftrightarrow$ Bom prazo, bom custo

$1 \leq SPI \cap CPI < 1 \Leftrightarrow$ Bom prazo, mau custo

$SPI < 1 \cap 1 \leq CPI \Leftrightarrow$ Mau prazo, bom custo

$SPI < 1 \cap CPI \leq 1 \Leftrightarrow$ Mau prazo, mau custo

Com as regras que tem de obedecer cada um dos quadrantes conseguimos assim perceber onde os *sprints* se encaixam em termos de custo e prazo. Assim sendo:

- Bom prazo, bom custo -> nenhum *sprint*
- Bom prazo, mau custo -> *sprint* #2 e #3

- Mau prazo, bom custo -> *sprint* #5
- Mau prazo, mau custo -> *sprint* #0, #1 e #4

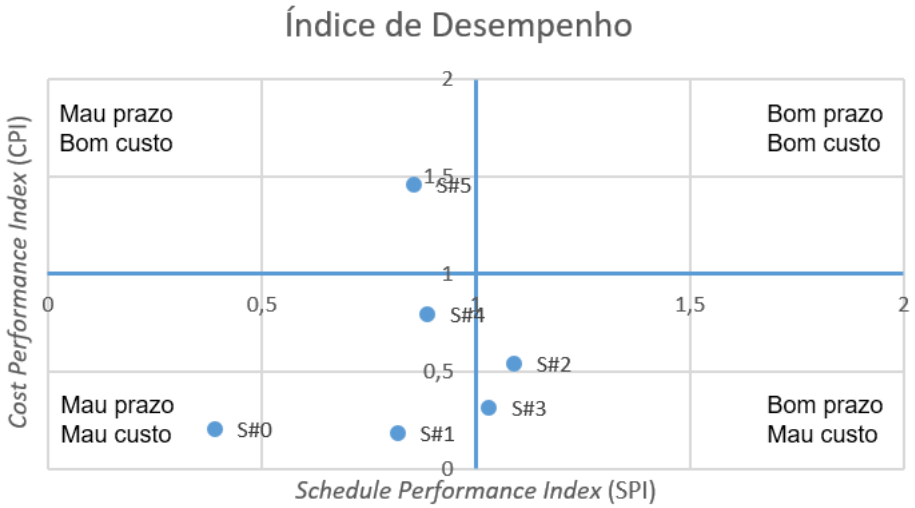

Figura 20. Gráfico das métricas CPI vs SPI

Ao olhar para a figura 20, consegue-se observar que, no início do projeto, no *sprint* 0, o projeto encontrava-se num mau cenário, apresentando mau custo e mau prazo. Com o passar do tempo houve uma evolução e apesar de ainda se encontrar no mesmo quadrante houve uma melhoria no prazo. Ao chegar à S#2 encontram-se novos valores positivos, visto que se consegue enquadrar no cenário de bom prazo e mau custo, havendo, no entanto, uma descida no *sprint* 3. Por fim nos últimos *sprints*, S#4 e S#5 houve uma grande melhoria em termos de custo, baixando, no entanto, os valores de prazo observados nos *sprints* 2 e 3. A S#4 e S#5 foram aquelas que mais se aproximaram do cenário ótimo de bom prazo e bom custo.

Desta forma, consegue-se perceber-se o quão complicado é para o PO fazer um balanço entre custo e prazo, mas fica a ideia de que, à medida que o projeto avança, há uma clara tendência para o melhor cenário possível de custo e prazo.

4.3.2 Planeamento

Como explicado antes, não conseguimos estudar todas as métricas, pois implicavam grandes mudanças na maneira como o projeto estava a ser monitorizado pelos gestores. De seguida são expostos os dados recolhidos

acerca da área de Planeamento/Gestão, onde a equipa de trabalho conseguiu fornecer dados relativos ao *Business Value Delivered* (BVD) e no que toca ao *Release* e *Sprint burndown*, a equipa trabalha com gráficos *burndown* e um gráfico cumulativo do fluxo de trabalho. O BVD foi considerado como as stories que iam sendo definidas como entregues ("*done*") ao longo dos *sprints*. Para efeitos deste trabalho, não foi calculado o valor "de negócio" que cada *story* fornecia – fosse o "valor" monetário (p.e., OKR – "*Objetives and Key Results*") – mas sim o valor "funcional" entregue, nomeadamente as funcionalidades de software do MVP relativas às *stories* implementadas.

As métricas de PPC e APC têm o mesmo propósito, mas com perspetivas diferentes. Como é possível observar na tabela 13, as duas mostram uma evolução do desenvolvimento do projeto ao longo do tempo.

Tabela 13. Métricas de PPC e APC

Sprint	**PPC**	***Stories***	**APC**
S#0	$^{1}/_{7} \cong 0{,}14$	S#0→5	$^{5}/_{90} \cong 0{,}06$
S#1	$^{2}/_{7} \cong 0{,}29$	S#0→5+16	$^{21}/_{90} \cong 0{,}23$
S#2	$^{3}/_{7} \cong 0{,}43$	S#0→5+16+21	$^{42}/_{90} \cong 0{,}47$
S#3	$^{4}/_{7} \cong 0{,}57$	S#0→5+16+21+11	$^{53}/_{90} \cong 0{,}59$
S#4	$^{5}/_{7} \cong 0{,}71$	S#0→5+16+21+11+4	$^{57}/_{90} \cong 0{,}63$
S#5	$^{6}/_{7} \cong 0{,}86$	S#0→5+16+21+11+4+9	$^{66}/_{90} \cong 0{,}73$

O PPC utiliza o *sprint* que se encontra em execução e divide pelo total de sprints planeadas para o projeto, obtendo assim o valor de trabalho completado até aquele *sprint*.

A métrica de APC tem resultado semelhante ao de PPC, mas enquanto no primeiro se trabalha com o sprint, neste caso com o número de *stories*. Assim para obter o APC de um dado sprint, dividimos o número de *stories* obtidas até aquela fase pelo número total de *stories* estimadas pela equipa e assim obtemos a percentagem de *stories* finalizadas.

Ambas as métricas mostram a evolução do projeto, mas a métrica de APC é mais fiável, pois trabalha com o número de *stories* enquanto que o PPC trabalha com o número de *sprints*, não tomando em consideração o fato de um *sprint* poder ter mais ou menos *stories* que outro *sprint*, fazendo assim o mesmo cálculo para todas os *sprints*.

No caso de estudo conseguimos perceber que ambas estão a aproximar-se de conclusão, mas que apesar de faltar uma sprint ainda existem 24 *stories* que a equipa estimou para a conclusão do projeto o que resulta em aproximadamente 27% do projeto.

Alguns gráficos que existem em projetos de metodologia ágil, os gráficos de *release* e *sprint burndown* permitem observar o rumo que um *sprint* está a ter no que toca ao tempo atribuído à mesma.

No gráfico da figura 21 podemos observar o *burndown* relativo ao *Sprint* #2 que iniciou a 18/01/18 e foi até 16/02/18. Este gráfico possui uma linha que é traçada desde o início do *sprint* relativa ao trabalho que falta ser produzido até à data de fim do *sprint*. Existe uma linha verde relativa aos recursos (total de horas de cada elemento) disponíveis e do desenvolvimento do projeto e por fim, existe a área relativa ao trabalho que existe que vai diminuindo consoante o avançar do tempo.

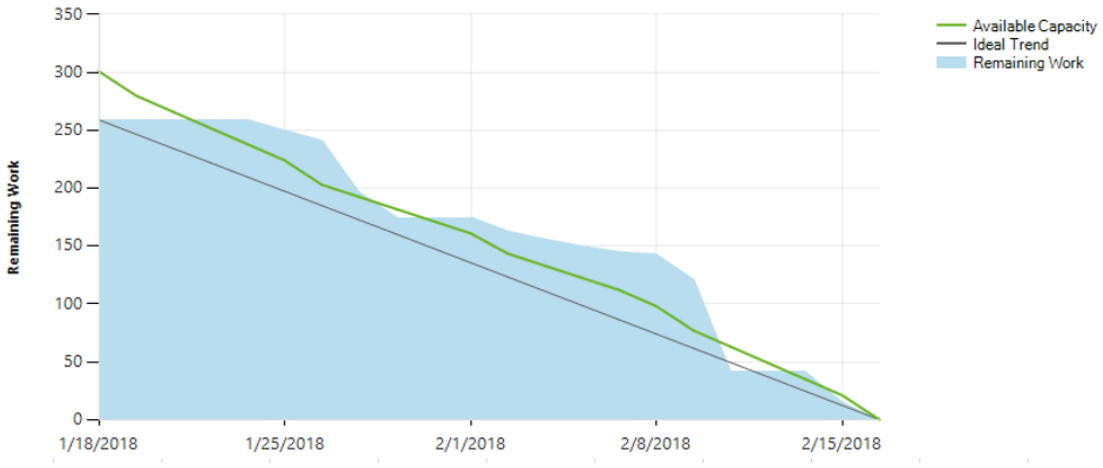

Figura 21. Sprint burndown da S#2

Por último, temos o gráfico cumulativo do trabalho. Este gráfico está dividido por áreas dos requisitos novos, aprovados, *committed* e completados, como é

possível observar na legenda. O gráfico da figura 22 tem a perspetiva evolutiva do trabalho e do tempo e ao analisarmos o gráfico conseguimos compreender pela área cinzenta que no fim do *Sprint* #2 e por toda o *Sprint* #3, 26/02/18 até 28/03/18, surgiram novos requisitos ao sistema. Pode também ver-se que, apesar de mais de metade do trabalho estar concluída, até à data mais recente, ainda existiam requisitos a serem aprovados e outros a surgirem como novos. Algo que acontece na metodologia ágil, visto que as equipas têm de estar preparadas para a alteração de requisitos ou adição de novos.

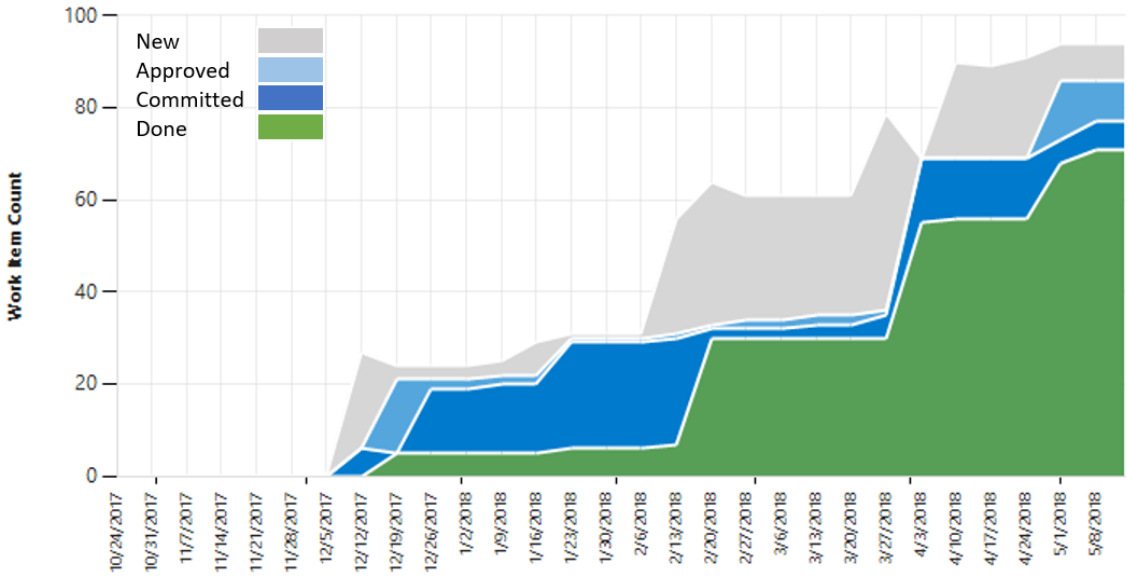

Figura 22. Gráfico cumulativo do trabalho

Os restantes gráficos relativos à métrica de *Sprint burndown* estão disponíveis no anexo A.

4.3.3 Desenvolvimento

Na área de Desenvolvimento, foi possível recolher dados relativos ao *Lead Time*, *Processing Time, Queue Time* e por fim o WIP. Os três primeiros estão, de certa forma, agrupados entre si, visto que, como referido na secção 4.2.3, o Lead Time entendemos que é a soma de *Processing Time* com o *Queue Time*. Já o *Processing* corresponde ao tempo em que este é trabalhado pela equipa de

desenvolvimento e o *Queue Time* é o tempo em que este se encontra em espera para ser desenvolvido.

Para este projeto, a especificação das *features* desde os casos de uso à arquitetura lógica, e posterior mapeamento para um *backlog* constituído por *user stories*, permitiu ao CCG-EPMQ deter os dados de cada *feature*, desde a sua especificação (antes da S#0 ou já com os *sprints* a decorrer) até à *user story* respetiva ser incluída no *Sprint Backlog* e posterior entrega ("*done*").

Como é possível observar na tabela 13, temos na primeira coluna as *features* encontradas pela equipa da CCG-EPMQ, na coluna seguinte encontramos quando é que uma dada *feature* foi especificada, isto é, reuniu toda a informação necessária para o seu desenvolvimento e foi aceite pela equipa. Na terceira coluna temos o número do *sprint* em que foi iniciado o desenvolvimento, tendo na coluna seguinte quando é que esta terminou. Nas última três colunas temos assim o *Lead Time* que resulta da soma do *Queue* com o *Processing Time*; o *Queue Time* que é obtido pela diferença de quando ele é iniciado e quando ele é especificado; e por fim, o *Processing Time* resulta da diferença entre o *sprint* em que uma *feature* termina e quando esta começa.

Para complementar os dados da tabela 14, foi feita uma média dos valores de *lead time*, *queue time* e *processing time*. Conseguiu-se assim, perceber que todas as *features* elaboradas estiveram, no total, um *sprint* em tempo de espera e pouco mais de um *sprint* em desenvolvimento, o que perfaz, em média, um total de mais de dois *sprints* para a conclusão de cada *feature*. Com isto é possível tentar descobrir locais específicos onde se pode diminuir o tempo, realocando recursos, de modo a conseguir entregar as *features* mais atempadamente ao cliente.

Tabela 14. Lead Time, Processing Time e Queue Time

Features	***Specified***	***Implemented Start***	***Implemented Finish***	***Lead Time***	***Queue Time***	***Processing Time***
Configure User profile	0	1	2	2	1	1
Configure User Account	0	1	2	2	1	1
Perform Authentiation	0	1	2	2	1	1
Manage Stakeholders	0	1	3	3	1	2
Manage Trucks	0	1	2	2	1	1
Manage Applications	1	2	3	2	1	1
Collaborative tool	2	3	5	3	1	2
Manage Work Tokens	4	5	5	1	1	0
Média				2,125	1	1,125

No que concerne a *Work In Progress*, foram contabilizadas o número de *stories* em que estavam a trabalhar num dado *sprint*. Podemos assim observar (tabela 15), que houve um maior esforço de trabalho a ser desenvolvido na primeira metade do ciclo de vida do projeto, que veio depois a diminuir à medida que se aproximavam da sua conclusão. Esta informação coincide com aquela observada anteriormente, onde se vê no gráfico cumulativo que a meio do projeto é quando há uma grande quantidade de trabalho, mais concretamente a partir do *sprint* 2 a ficar em estado de *commit* e que, de seguida, é aprovado.

Tabela 15. Work in Progress

	Sprint #0	***Sprint #1***	***Sprint #2***	***Sprint #3***	***Sprint #4***	***Sprint #5***
WIP	5	16	21	11	4	9

4.3.4 *Stakeholders*

Para a medição dos *stakeholders* foi escolhido o tempo de feedback entre cliente e equipa, bem como dentro da própria equipa nos diferentes departamentos e a satisfação do cliente no produto desenvolvido. Contudo, devido à

confidencialidade existente entre cliente e organização não foi possível apresentar as respostas sobre a satisfação do cliente. Apresenta-se apenas um exemplo do questionário utilizado pelo CCG-EPMQ para o cliente que pode servir de apoio para uso futuro. É possível observar questionário no anexo B.

4.3.5 Qualidade

Relativamente à área de qualidade, foi bastante complicado obter algum tipo de informação relevante para o estudo. Como o projeto já se encontrava num estado avançado de desenvolvimento era difícil obter informação acerca do *defect backlog*, entre outros. Assim sendo, a equipa conseguiu fornecer dados relativo ao número de bugs, tabela 16 , no fim de cada *sprint*, que, como já era de esperar, é mais elevado quando existem mais *stories* a serem trabalhadas. Um acompanhamento com a equipa desde a fase inicial e de forma contínua poderia permitir recolher dados como a quantidade de *builds* em cada *sprint* (possível de registar caso sejam utilizadas ferramentas de gestão de código como o JIRA ou Microsoft "Team Foundation Server" – TFS), mas tal não foi possível.

Tabela 16. Número de bugs em cada sprint

	***Sprint* #0**	***Sprint* #1**	***Sprint* #2**	***Sprint* #3**	***Sprint* #4**	***Sprint* #5**
Bugs	0	0	0	23	5	10

O *test coverage* era uma das métricas que parecia fornecer mais informação acerca de quantidade de testes que são executados face ao código produzido, mas provou ser algo difícil de alcançar com o tempo que havia para a entrega de resultados.

Como referido, sendo o UH4SP um projeto do tipo *greenfield*, não foi medido o *test growth*. De igual forma, os valores para defeitos diferidos não são apresentados, já que se referem a falhas reportadas pelo cliente após a entrega (habitualmente até ao espaço de 6 meses) e, no espaço temporal desta dissertação, esses valores não eram sequer recebidos pela equipa CCG-EPMQ.

5. Conclusão

Como fecho da dissertação, nesta secção apresentam-se todas as conclusões inerentes ao trabalho desta dissertação e à aprendizagem obtida durante todo o projeto.

O projeto tinha como pressuposto o absorvimento das metodologias associadas ao desenvolvimento ágil e partir depois para a escolha de algumas métricas que seriam usadas para o estudo de um projeto. Para isso foi definido um conjunto de atividades de forma a manter controlo sobre a *timeline* da dissertação.

Foi possível observar com o estudo levantado que as metodologias ágeis ocupam atualmente uma presença fundamental nos projetos de Sistemas de Informação. Numa altura em que existe grande incerteza acerca do rumo que a tecnologia tem, com o aparecimento de novas tecnologias que facilitam os processos de construção e implementação de produto ao cliente, ou com a constante mudança que o mercado em que determinado cliente está inserido e a competitividade que existe atualmente, uma equipa de desenvolvimento tem que estar constantemente pronta a responder aos novos requisitos do cliente ou à mudança de todo o paradigma do negócio.

Dentro das metodologias ágeis, há uma panóplia de opções para o gestor da equipa. Com o passar do tempo o mais observado é que uma equipa escolhe uma metodologia como base na gestão do seu projeto, mas depois certos detalhes são diferenciados em cada equipa. Dentro das metodologias estudadas, a mais registada é o *Scrum* que como foi dito anteriormente, é observada com detalhes de *Kanban*, formando assim uma metodologia híbrida, com os pontos fortes de cada uma.

Depois de algumas semanas para a recolha de toda a informação inerente ao tema e ao que o rodeia para melhor compreender o universo da gestão de projetos, bem como aquela que é a prática utilizada por muitas organizações de uma gestão ágil, houve um levantamento de métricas utilizas por organizações que ajudam a medir o sucesso/insucesso dos seus projetos.

Após esse levantamento de métricas, quer ágeis quer tradicionais, houve um *refinement* das métricas que melhor se adequavam num contexto ágil e como se

poderia agrupá-las de modo a não tornar a *framework* demasiado pesada aos olhos do gestor do projeto. Um projeto ágil deve ser monitorizado para perceber o seu sucesso, mas essa monitorização não deve tomar muito tempo do gestor, pois perde a essência de um projeto ágil.

A ideia por detrás da construção da *framework* passou por dividir por áreas a que um projeto de software está sujeito e com isso poder medir em diferentes atores do projeto e facilmente alocar uma pessoa a uma determinada área. Deste resultaram cinco áreas distintas, sendo elas, EVM, Planeamento, Desenvolvimento, *Stakeholders* e Qualidade.

Em relação à validação da *framework*, para algumas das métricas expostas no caso de estudo, foi complicado obter valores, pois a equipa já se encontrava num estado avançado de desenvolvimento e já tinha ela própria as suas métricas. Assim sendo, uma das melhorias a fazer numa próxima fase passava por deixar algumas métricas para a entrada de métricas que fossem utilizadas pela equipa em estudo ou utilizar a *framework* em projetos para medição desde o momento inicial para assim ter toda a informação necessária. Outra medida pode passar por consultar diferentes projetos e perceber quais as métricas mais utilizadas na comunidade de desenvolvimento de software com metodologias ágeis.

Para a obtenção de um bom valor de negócio acrescentado (*Business Value Delivered* – BVD) ao longo do projeto, este deve ser discutido fortemente com o cliente para compreender quais são as *key features* e que percentagem representam para o negócio. É necessário ter uma participação bastante ativa por parte do cliente, pois este é que está por dentro do setor e deve saber qual o valor que dada *feature* preenche no negócio. Outra razão para a participação do cliente é a de que este, no geral, nunca abandona o projeto. Ao contrário do gestor da equipa que por várias razões pode ser alterado e assim, toda a perspetiva de BVD pode mudar dentro do projeto.

Uma área da *Agile Metric Framework* que deve ser revista é a área de qualidade. A última área estudada foi das de mais difícil de compreensão, bem como, a mais difícil de aplicar no caso de estudo. Algumas das métricas implicam uma aprendizagem prévia ao arranque do projeto e requeriam que toda a equipa estivesse em conformidade na sua aplicabilidade. Apesar de ser uma área que

pode oferecer bastantes garantias de melhoria em cenário de boa implementação, deve ser revisto se realmente a sua aplicação é a melhor escolha na maioria dos projetos de software ágil.

De modo geral, todas as métricas são possíveis de obter se houver boa comunicação entre a equipa e o cliente. Os projetos ágeis alcançam bastante sucesso com base no feedback dado pelo cliente. Pode-se dizer assim que a comunicação é tida como pilar do sucesso de qualquer projeto. A comunicação, neste caso, não deve existir só com o cliente, mas sim com elementos de equipa, caso haja diferentes cargos a serem desempenhados pelos mesmos.

Por fim, acho importante realçar que, apesar de a dissertação não ter obtido todas as medições e ter sido necessário adaptar algumas, existe investigação que pode ser feita e melhorias na *framework* para poder servir de apoio a equipas que estejam a desenvolver software de forma ágil.

Com vista a trabalhos futuros, existe nesta dissertação algo passível de ser usado e que merece alguma atenção. Considerando este documento como a primeira fase, deve, em seguida, haver uma nova fase onde será enfatizada a nova versão da *framework,* olhando com especial cuidado para a área de Qualidade. Como abordado anteriormente, esta foi uma área bastante complicada para a obtenção de resultados, mas tem métricas bastante interessantes para as equipas de trabalho.

De forma a aumentar a eficiência do trabalho pode até ser utilizado o *input* de gestores de equipas de trabalho e conseguir perceber quais são as suas maiores preocupações enquanto gestores de equipa.

Bibliografia

1/SC7, I. C. S. I. J. (n.d.). SEVOCAB. Retrieved from https://pascal.computer.org/sev_display/index.action

Abbas, N., Gravell, A. M., & Wills, G. B. (2010). The Impact of Organization, Project and Governance Variables on Software Quality and Project Success. In *2010 Agile Conference* (pp. 77–86). IEEE. https://doi.org/10.1109/AGILE.2010.16

Abrahamsson, Pekka; Salo, Outti; Ronkainen, Jussi; Warsta, J. (2002). Agile software development methods Review and analysis. *VTT Publication*. Retrieved from http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf

Ahmad, M. O., Dennehy, D., Conboy, K., & Oivo, M. (2017). Kanban in software engineering: A systematic mapping study. *The Journal of Systems and Software*, *137*, 96–113. https://doi.org/10.1016/j.jss.2017.11.045

Aked, M. (2003). Risk reduction with the RUP phase plan. Retrieved from https://www.ibm.com/developerworks/rational/library/1826-pdf.pdf

Anderson, D. J. (2005). Stretching agile to fit CMMI level 3 - the story of creating MSF for CMMI/spl reg/ process improvement at Microsoft corporation. In *Agile Development Conference (ADC'05)* (pp. 193–201). IEEE Comput. Soc. https://doi.org/10.1109/ADC.2005.42

Atkinson, R. (1999). Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International Journal of Project Management*. Retrieved from https://ac.els-cdn.com/S0263786398000696/1-s2.0-S0263786398000696-main.pdf?_tid=80c781fe-f9f9-11e7-8185-00000aab0f6c&acdnat=1516023746_a6378dde82c5e7e274157ce571f80848

Bassil, Y. (2012). A Simulation Model for the Waterfall Software Development Life Cycle. *International Journal of Engineering & Technology*, *2*(5), 2049–3444. Retrieved from http://iet-journals.org/archive/2012/may_vol_2_no_5/255895133318216.pdf

Beck, K. (2000). *Extreme programming eXplained : embrace change*. Addison-Wesley. Retrieved from https://books.google.pt/books?hl=pt-PT&lr=&id=G8EL4H4vf7UC&oi=fnd&pg=PR13&dq=Extreme+programming+explained:+Embrace+change.+Reading,+Mass&ots=jatJxtnQBj&sig=slnC5xD6rGE95TNoZ0btsRyl_tw&redir_esc=y#v=onepage&q&f=false

Cabri, A., & Griffiths, M. (2006). Earned Value and Agile Reporting. In *AGILE 2006 (AGILE'06)* (pp. 17–22). IEEE. https://doi.org/10.1109/AGILE.2006.21

Cockburn, A. (2004). *Crystal clear : a human powered methodology for small teams*. *Agile software development series.* Addison-Wesley Professional.

Cockburn, A., & Alistair. (2002). *Agile software development*. Addison-Wesley. Retrieved from https://dl.acm.org/citation.cfm?id=502980

dos Santos, P. S. M., Varella, A., Dantas, C. R., & Borges, D. B. (2013). Visualizing and Managing Technical Debt in Agile Development: An Experience Report (pp. 121–134). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-38314-4_9

Fenton, Norman; Bieman, J. (2014). *Software Metrics: A Rigorous and Practical Approach, Third Edition*.

Fernandez, D. J. (2016). Agile Project Management - Agilism Versus Traditional Approaches. *Journal of Computer Information Systems*. Retrieved from http://www.tandfonline.com/doi/pdf/10.1080/08874417.2009.11646044?needAccess=true

Ferreira, L.; Lopes, N.; Ávila, P. S.; Castro, H.; Varela, M. L. R.; Putnik, G. D.; Martinho, R.; Rijo, R.; Miranda, I. M.; Cruz-Cunha, M. M. (2017). Virtual Enterprise integration on management based on a Meta-enterprise - a PMBoK approach. *ScienceDirect*. Retrieved from https://ac.els-cdn.com/S1877050917328880/1-s2.0-S1877050917328880-main.pdf?_tid=4ea430b8-f9e1-11e7-b57a-00000aab0f6b&acdnat=1516013354_37564734ab90febc4466efcdcf07da82

Ferreira, N., Santos, N., Machado, R. J., Fernandes, J. E., & Gasević, D. (2014). A V-Model Approach for Business Process Requirements Elicitation

in Cloud Design. In *Advanced Web Services* (pp. 551–578). New York, NY: Springer New York. https://doi.org/10.1007/978-1-4614-7535-4_23

Ghosh, S. (2015). Systemic Comparison of the Application of EVM in Traditional and Agile Software Project. *PM World Journal Systemic Comparison of the Application of EVM*, *IV*. Retrieved from http://pmworldjournal.net/wp-content/uploads/2015/08/pmwj37-Aug2015-Ghosh-comparison-of-evm-in-traditional-and-agile-projects-student-paper.pdf

Green, P. (2011). Measuring the Impact of Scrum on Product Development at Adobe Systems. In *2011 44th Hawaii International Conference on System Sciences* (pp. 1–10). IEEE. https://doi.org/10.1109/HICSS.2011.306

Hartmann, D., & Dymond, R. (2006). Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value. In *AGILE 2006 (AGILE'06)* (pp. 126–134). IEEE. https://doi.org/10.1109/AGILE.2006.17

Haugen, N. C. (2006). An Empirical Study of Using Planning Poker for User Story Estimation. In *AGILE 2006 (AGILE'06)* (pp. 23–34). IEEE. https://doi.org/10.1109/AGILE.2006.16

Hayes, W., Garcia-Miller, S., Lapham, M. A., Wrubel, E., & Chick, T. (2014). Agile Metrics: Progress Monitoring of Agile Contractors. *Software Engineering Institute*. Retrieved from http://repository.cmu.edu/sei/775

Henderson-Sellers, B., Collins, G., Due, R., & Graham, I. (2001). A qualitative comparison of two processes for object-oriented software development. *Information and Software Technology*. Retrieved from https://ac.els-cdn.com/S095058490100180X/1-s2.0-S095058490100180X-main.pdf?_tid=41fdc574-fad9-11e7-8342-00000aacb35e&acdnat=1516119848_8231fbcd404b9450313621eea06ac2bc

Hodgetts, P. (2004). Refactoring the Development Process: Experiences with the Incremental Adoption of Agile Practices. In *Agile Development Conference* (pp. 106–113). IEEE. https://doi.org/10.1109/ADEVC.2004.17

Hussain, A., & Mkpojiogu, E. O. C. (2015). Jurnal Teknologi AN APPLICATION OF THE ISO/IEC 25010 STANDARD IN THE QUALITY-IN-USE

ASSESSMENT OF AN ONLINE HEALTH AWARENESS SYSTEM. *Jurnal Teknologi*. Retrieved from https://s3.amazonaws.com/academia.edu.documents/44323992/6107-17059-1-SM.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1518624862&Signature=q%2Bj7bkBxMbR8kHeiF6tf4RwYZZU%3D&response-content-disposition=inline%3B filename%3DAN_APPLICATION_OF_THE_ISO_IEC_

IBM. (2001). Rational Unified Process Best Practices for Software Development Teams. *Rational Software White Paper*. Retrieved from https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

IEEE. (1997). *The IEEE Standard Dictionary of Electrical and Electronics Terms*.

IEEE. (2014). Guide to the Software Engineering Body of Knowledge SWEBOK ® A Project of the IEEE Computer Society. Retrieved from https://www.computer.org/ieeecs-swebokdelivery-portlet/swebok/SWEBOKv3.pdf?token=n627zzKnXRiLH7iYunQUdhbE5ZsvEYp3

Janus, André; Schmietendorf, Andreas; Dumke, Reiner; Jäger, J. (2012). *The 3C approach for agile quality assurance*. *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics*. IEEE. Retrieved from https://dl.acm.org/citation.cfm?id=2669382

Kniberg, H. (2009). *Kanban vs Scrum*. Retrieved from http://www.crisp.se/henrik.kniberg

Kohlbacher, M., Stelzmann, E., & Maierhofer, S. (2011). Do agile software development practices increase customer satisfaction in Systems Engineering projects? In *2011 IEEE International Systems Conference* (pp. 168–172). IEEE. https://doi.org/10.1109/SYSCON.2011.5929091

Kupiainen, Eetu; Mäntyla, Mika V.; Itkonen, J. (2015). Using metrics in Agile and Lean Software Development – A systematic literature review of

industrial studies. *Information and Software Technology*, *62*, 143–163. https://doi.org/10.1016/J.INFSOF.2015.02.005

Middleton, P., Taylor, P. S., Flaxel, A., & Cookson, A. (2007). Lean principles and techniques for improving the quality and productivity of software development projects: a case study. *International Journal of Productivity and Quality Management*, *2*(4), 387. https://doi.org/10.1504/IJPQM.2007.013334

Miguel, J. P., Mauricio, D., & Rodriguez, G. (2014). A Review of Software Quality Models for the Evaluation of Software Products. https://doi.org/10.5121/ijsea.2014.5603

Mujtaba, S., Feldt, R., & Petersen, K. (2010). Waste and Lead Time Reduction in a Software Product Customization Process with Value Stream Maps. In *2010 21st Australian Software Engineering Conference* (pp. 139–148). IEEE. https://doi.org/10.1109/ASWEC.2010.37

Nkiwane1, N. H., Meyer, W. G., & Steyn, H. (2016). THE USE OF EARNED VALUE MANAGEMENT FOR INITIATING DIRECTIVE PROJECT CONTROL DECISIONS: A CASE STUDY. *South African Journal of Industrial Engineering*, *27*(1), 192–203. https://doi.org/10.7166/27-1-1260

Palmer, S. R. (Stephen R., & Felsing, J. M. (2002). *A practical guide to feature-driven development*. Prentice Hall PTR. Retrieved from https://dl.acm.org/citation.cfm?id=600044

Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2006). THE DESIGN SCIENCE RESEARCH PROCESS: A MODEL FOR PRODUCING AND PRESENTING INFORMATION SYSTEMS RESEARCH. Retrieved from http://geni15.wrsc.org/sites/default/files/documents/000designscresearchproc_desrist_2006.pdf

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Petersen, K., & Wohlin, C. (2011). Measuring the flow in lean software development. *Software: Practice and Experience*, *41*(9), 975–996. https://doi.org/10.1002/spe.975

PMI. (2013). *A Guide to the Project Management Body of KnowledGe (PMBOK ® Guide)-Fifth Edition*. Retrieved from www.PMI.org

Polk, R. (2011). Agile and Kanban in Coordination. In *2011 AGILE Conference* (pp. 263–268). IEEE. https://doi.org/10.1109/AGILE.2011.10

Poppendieck, Mary; Cusumano, M. A. . (2003). *Lean Software Development*.

Poppendieck, M. (2007). Lean Software Development. *29th International Conference on Software Engineering*. Retrieved from http://delivery.acm.org/10.1145/1250000/1248986/28920165.pdf?ip=193.137.92.163&id=1248986&acc=ACTIVE SERVICE&key=2E5699D25B4FE09E.8B0EA5264B27B16E.4D4702B0C3E38B35.4D4702B0C3E38B35&__acm__=1516296359_1687e3c94cf37c0dee6e406087b29275

Poppendieck, M., & Cusumano, M. A. (2012). Lean Software Development: A Tutorial. *IEEE Software*, *29*(5), 26–32. https://doi.org/10.1109/MS.2012.107

Pulford, K., Kuntzmann-Combelles, A., & Shirlaw, S. (1996). *A quantitative approach to software management : the ami handbook*. Addison-Wesley. Retrieved from https://dl.acm.org/citation.cfm?id=217717

Radujković, Mladen; Sjekavica, M. (2017). Project Management Success Factors. *ScienceDirect*. Retrieved from https://ac.els-cdn.com/S1877705817331740/1-s2.0-S1877705817331740-main.pdf?_tid=4a189508-f9e0-11e7-b151-00000aab0f6b&acdnat=1516012917_c3ee932844eb694b37081437cb79b6b1

Santos, N., Pereira, J., Ferreira, N., & Machado, R. J. (2018). Modeling in agile software development: decomposing use cases towards (logical) architecture design. In *2nd International Workshop on Managing Quality in Agile and Rapid Software Development Processes (QuASD) in conjunction with 19th International Conference on Product-Focused Software Process*

Improvement (PROFES). Lecture Notes in Computer Science (in prin. Wolfsburg, Germany: Springer. (in print)

Santos, N., Pereira, J., Morais, F., Barros, J., Ferreira, N., & Machado, R. J. (2018). An agile modeling oriented process for logical architecture design. In *Enterprise, Business-Process and Information Systems Modeling. Proceedings of 23rd International Conference on Exploring Modeling Methods for Systems Analysis and development (EMMSAD) in conjunction with 30th International Conference on Advanced Informati*. Springer. (in print)

Santos, N., Rodrigues, H., Pereira, J., Morais, F., Abreu, R., Fernandes, N., … Machado, R. J. (2018). UH4SP: a software platform for integrated management of connected smart plants. In *9th IEEE International Conference on Intelligent Systems (IS) (in print)*. Funchal, Portugal: IEEE.

Schwaber, Ken; Sutherland, J. (2016). O Guia do Scrum™ O guia definitivo para o Scrum: As Regras do Jogo. Retrieved from https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Portuguese-European.pdf

Schwaber, Ken; Sutherland, J. (2017). What is Scrum development? Retrieved November 27, 2017, from https://www.scrum.org/resources/what-is-scrum?gclid=Cj0KCQiAjO_QBRC4ARIsAD2FsXPhXlhiJ2pQ_IB77_s-W5ILw9BA_9zuVHYcBx3fLNdDb1UUt2rUrwIaAkkZEALw_wcB

Schwaber, K. (2009). What is Scrum? Retrieved from https://www.scrum.org/resources/what-is-scrum

Seikola, M., Loisa, H.-M., & Jagos, A. (2011). Kanban Implementation in a Telecom Product Maintenance. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 321–329). IEEE. https://doi.org/10.1109/SEAA.2011.56

Špundak, M. (2014). ScienceDirect Mixed agile/traditional project management methodology – reality or illusion? *Procedia - Social and Behavioral Sciences*, *119*(119), 939–948. https://doi.org/10.1016/j.sbspro.2014.03.105

Staron, M., Meding, W., & Söderqvist, B. (2010). A method for forecasting

defect backlog in large streamline software development projects and its industrial evaluation. *Information and Software Technology*, *52*(10), 1069–1079. https://doi.org/10.1016/J.INFSOF.2010.05.005

Sulaiman, T., Barton, B., & Blackburn, T. (2006). AgileEVM - Earned Value Management in Scrum Projects. In *AGILE 2006 (AGILE'06)* (pp. 7–16). IEEE. https://doi.org/10.1109/AGILE.2006.15

Sutherland, Jeff; Schwaber, Ken; Cockburn, A. (2001). Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org/

Trimble, J., & Webster, C. (2013). From Traditional, to Lean, to Agile Development: Finding the Optimal Software Engineering Cycle. In *2013 46th Hawaii International Conference on System Sciences* (pp. 4826–4833). IEEE. https://doi.org/10.1109/HICSS.2013.237

Turk, D., France, R., & Rumpe, B. (2002). Limitations of Agile Software Processes. *Third International Conference on Extreme Programming and Flexible Processes in Software Engineering*. Retrieved from https://arxiv.org/ftp/arxiv/papers/1409/1409.6600.pdf

Varajão, J., & Cruz-Cunha, M. M. (2013). Using AHP and the IPMA Competence Baseline in the project managers selection process. *International Journal of Production Research*. https://doi.org/10.1080/00207543.2013.774473

Wells, D. (1999). Extreme Programming. Retrieved from http://www.extremeprogramming.org/

Wysocki, R. K. (2006). *Effective software project management*. Wiley Pub. Retrieved from https://www.wiley.com/en-pt/Effective+Software+Project+Management-p-9780764596360

Anexos

A. Sprint burndown

Figura 23. Sprint burndown S#0

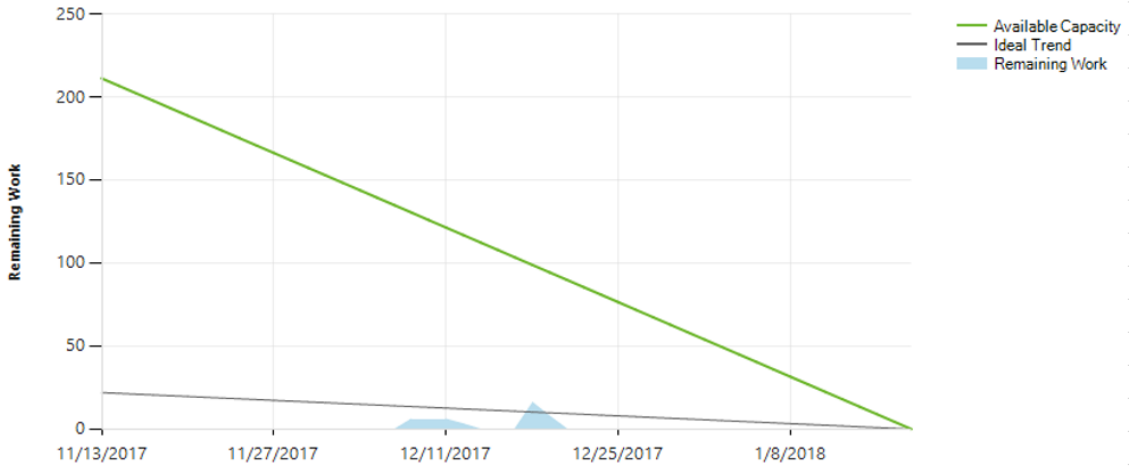

Figura 24. Sprint burndown S#1

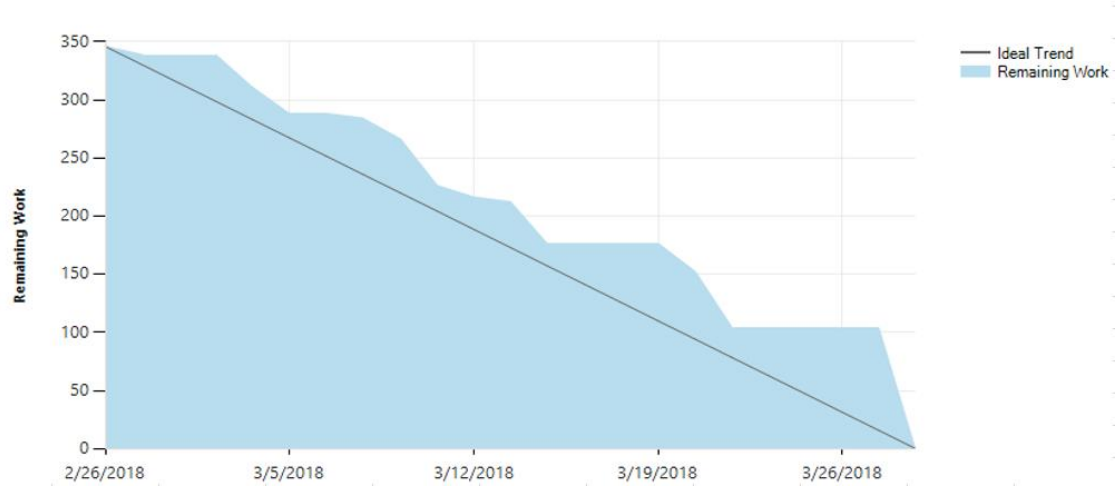

Figura 25. Sprint burndown S#3

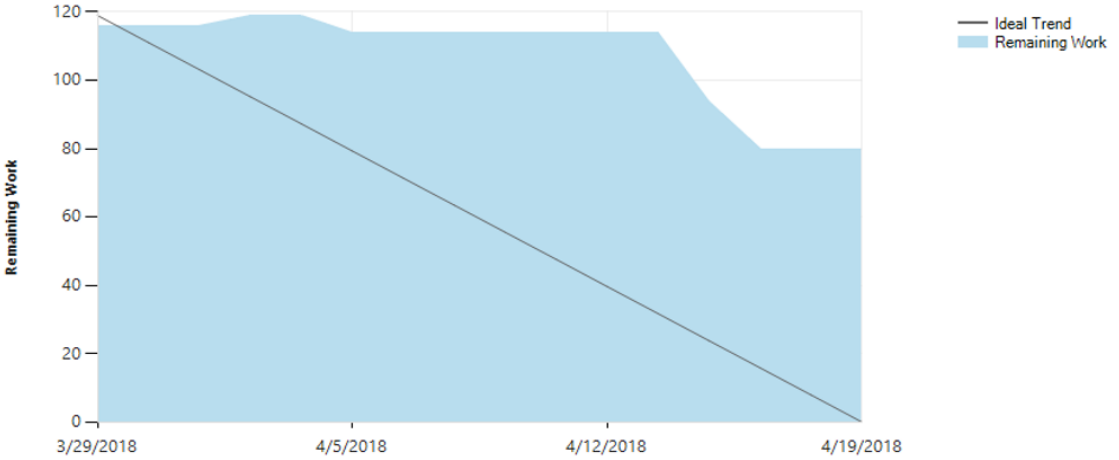

Figura 26. Sprint burndown S#4

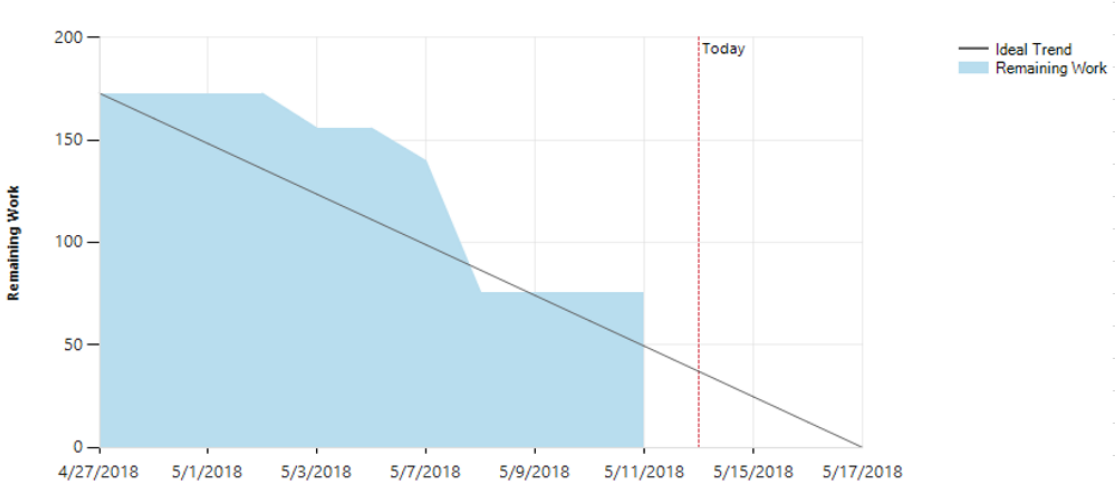

Figura 27. Sprint burndown S#5

B. Questionário

A procura da melhoria contínua, com vista a uma cada vez melhor prestação dos serviços, é o principal compromisso estabelecido na nossa organização.
Por conseguinte, a sua opinião é fundamental para que possamos criar novas alternativas e oferecer um atendimento cada vez mais eficaz.

Para cada uma das afirmações avalie de forma espontânea a sua concordância, desde « Insatisfeito»(1) «Razoável»(2) « Satisfeito »(3) « Muito Satisfeito »(4), preenchendo a respetiva quadrícula.

*Obrigatório

1. Empresa *

2. Designação do Projeto *

3. Data *

Desenvolvimento de projetos

Resposta relativas ao nosso desempenho no período de 2017

4. A) Avaliação do Serviço *

Marcar apenas uma oval por linha.

	1	2	3	4
Facilidade em contactar-nos				
Cumprimento dos prazos acordados/definidos				
Cumprimento das metodologias acordadas				
Apresentação de ideias, alternativas e cenários à resolução de problemas				
Adequação dos resultados às v/ expetativas				
Relação com os serviços administrativos e financeiros				

5. B) Avaliação da equipa de Coordenação e Desenvolvimento * Marcar apenas uma oval por linha.

	1	2	3	4
Articulação entre o [redacted] e a v/ empresa				
Práticas da Gestão de Projetos				
Atuação do Gestor de Projeto				
Envolvimento e disponibilidade				
Rapidez de resposta				
Cortesia e simpatia				
Eficácia na resolução de problemas				
Competência/conhecimentos técnicos				
Aconselhamento e esclarecimento técnico				

6. C) Impacto do projeto na empresa * Marcar apenas uma oval por linha.

	1	2	3	4
O projeto teve esperado o impacto na empresa				

7. D) Avaliação da evolução dos serviços prestados em relação a anos/projetos anteriores *

Marcar apenas uma oval por linha.

	1	2	3	4
Domínio Técnico				
Conhecimento do mercado onde a V/ empresa se insere				
Conhecimento da atividade da v/empresa				
Gestão de projetos				

8. E) O que distingue os nossos serviços de outras entidades similares? *
Pode selecionar mais de uma opção.

Marcar tudo que for aplicável.

- [ ] Competência dos colaboradores
- [ ] Disponibilidade dos colaboradores
- [ ] Criatividade & Inovação
- [ ] Resultados e expetativas do output final
- [ ] Preço
- [ ] Outra: _____

9. Globalmente, qual o grau de satisfação em relação à nossa empresa? * Marcar apenas uma oval.

	1	2	3	4	
Muito Insatisfeito					Muito Satisfeito

10. Consideraria voltar a trabalhar connosco? * Marcar apenas uma oval.

- Sim
- Não

11. Recomendaria os nossos serviços? * Marcar apenas uma oval.

- Sim
- Não

12. Sugestões para melhoria/Reclamações:

_____

_____

_____

_____

_____