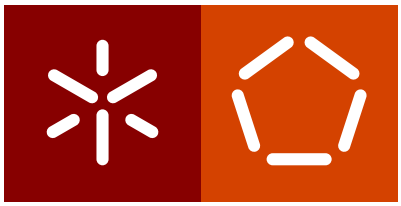


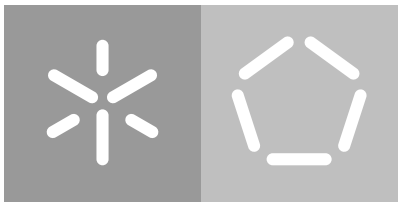
Universidade do Minho
Escola de Engenharia
Departamento de Electrónica Industrial

Ana Rita Fernandes Martins

Management of Planned Value

Métricas para Monitorizar Globalmente o Desempenho de Projetos de Desenvolvimento de Software

Maio de 2018



Universidade do Minho
Escola de Engenharia
Departamento de Electrónica Industrial

Ana Rita Fernandes Martins

Management of Planned Value

Métricas para Monitorizar Globalmente o Desempenho de Projetos de Desenvolvimento de Software

Dissertação de Mestrado

Desenvolvido sob a orientação de
Professor Doutor Pedro Ribeiro
Professor Doutor Jorge Cabral

Maio de 2018

AGRADECIMENTOS

Em primeiro lugar quero agradecer ao meu orientador, o Professor Pedro Ribeiro, por toda a orientação, discussões, acompanhamento e sugestões feitas que levaram à conclusão do meu trabalho. Agradeço também ao meu co-orientador, o Professor Jorge Cabral, por todas as dicas e sugestões.

Quero agradecer aos meus amigos que me acompanham há anos, realçando a Tânia Pereira, a Carla Gonçalves, a Ana Antunes, o Diogo Gonçalves, o Ricardo Guimarães, o Paulo Félix, o João Alves, e o Luís Almeida, os técnicos das oficinas do Departamento de Eletrónica Industrial, nomeadamente a D^{a} Ângela, o Sr. Joel, o Sr. Carlos, os meus colegas PMOs, a Joana Peixoto, o David Silva, o Pedro Magalhães, a Clarisse Pessoa, o Rui Ribeiro, o Hugo Sousa e o Bruno Lemos. A todos, um grande obrigada pela vossa amizade!

Não podia deixar de agradecer à minha família, em especial à minha Mãe e ao meu Pai, pois sem todo o seu apoio nada teria sido possível. Fico-vos eternamente grata. À minha irmã, um obrigada pela paciência e compreensão.

Por último quero agradecer ao David, que acompanhou todo o percurso de perto e nunca me deixou baixar os braços, mesmo nos momentos mais difíceis. Sem o seu apoio nunca teria iniciado esta dissertação. Obrigada por todo o carinho.

RESUMO

De forma a garantir que um projeto é entregue com sucesso, não é apenas necessário garantir o bom planeamento do mesmo. Depois de iniciados os trabalhos de desenvolvimento, é também necessário assegurar que o plano é concretizado ou, no caso de não se poder concretizar, encontrar alternativas de forma a se realize sem diminuir o seu valor. Assim torna-se importante garantir que as mudanças são monitorizadas, de forma a que apenas as mudanças aprovadas são implementadas.

As métricas permitem monitorizar e analisar de forma simples o desempenho de projetos, criando também alertas de situações que devem ser corrigidas. O conceito e uso de métricas não é recente nem desconhecido da maioria das organizações, sendo que uma técnica bastante conhecida é o *Earned Value Management* (EVM). No entanto, o EVM apenas apresenta métricas relativas ao triângulo âmbito, tempo e custo, não monitorizando outros fatores relevantes para o sucesso.

Existe um crescente número de trabalhos de investigação sobre métricas na área da engenharia de software, dado que a boa gestão das mesmas aumenta a probabilidade de sucesso do projeto. No início dos anos 50, a métrica mais utilizada no desenvolvimento de software era *Lines of Code* (LOC). LOC era extremamente eficiente nessa data, pois a fase de implementação constituía cerca de metade do esforço total. LOC estava relacionado com medidas de produtividade e de qualidade. No entanto, a engenharia de software evoluiu e estudos recentes indicam que o risco de insucesso cresce exponencialmente com o aumento do tamanho e da complexidade do software. Sabe-se hoje que grande parte dos projetos de software falhados são resultado de métricas imprecisas e de técnicas de medição inadequadas.

Assim, o objetivo desta dissertação é encontrar um conjunto de métricas que permitam monitorizar e fornecer informação relevante para o sucesso dos projetos, dentro do contexto da Engenharia de Software.

Palavras-chave: Gestão de Projetos, Engenharia de *Software*, Métricas, Sucesso, Controlo e Monitorização, Scrum

ABSTRACT

In order to ensure that a project is successfully delivered, it is not only necessary to ensure proper project planning. Once the development work has begun, it is also necessary to guarantee that the plan is implemented or, if it is not possible, to find alternatives in order to achieve it without diminishing its value. Thus, it becomes important to assure that changes are monitored, so that only approved changes are implemented.

Metrics allow you to easily monitor and analyze project performance, creating alerts for situations that need to be corrected. The concept and use of metrics is neither recent nor unknown to most organizations, and a well-known technique is Earned Value Management (EVM). However, EVM only presents metrics related to the scope, time, and cost triangle, not tracking other factors relevant to success.

There is a growing number of research studies on metrics in the Software Engineering field, since a good metrics management increases the probability of success. In the early 1950s, the most used metric in software development was Lines of Code (LOC). LOC was extremely efficient at that time, since the implementation phase consisted in about half of the total effort. LOC was related to productivity and quality measures. However, software engineering has evolved and recent studies indicate that the risk of failure grows exponentially with increasing software size and complexity. It is now known that most failed software projects are the result of inaccurate metrics and inadequate measurement techniques.

Therefore, this dissertation's objetive is to find a set of metrics that allow to monitor and provide information relevant to the success of the projects, within the context of Software Engineering.

Keywords: Project Management, Software Engineering, Metrics, Success, Control and Monitoring, Scrum

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÓNIMOS

AC *Actual Cost*

APM *Association for Project Management*

APMBoK *Association for Project Management Body of Knowledge*

CMMI *Capability Maturity Model Integration*

CoSQ *Cost of Software Quality*

CPI *Cost Performance Index*

CV *Cost Variance*

DSR *Design Science Research*

EV *Earned Value*

EVM *Earned Value Management*

IEEE *Institute of Electrical and Electronics Engineers*

ISO *International Organization for Standardization*

KLOC *Thousands Lines of Code*

KPI *Key Performance Indicator*

LOC *Lines of Code*

PMBoK *Project Management Body of Knowledge*

PMI Project Management Institute

PV *Planned Value*

QA *Quality Assurance*

RH Recursos Humanos

SEI *Software Engineering Institute*

SQuaRE *System and Software Quality Requirements and Evaluation*

SPI *Schedule Performance Index*

SPICE *Software Process Improvement and Capability dEtermination*

SV *Schedule Variance*

SWEBoK *Software Engineering Body of Knowledge*

WP *Work Package*

WBS *Work Breakdown Structure*

XP *eXtreme Programming*

1

INTRODUÇÃO E FUNDAMENTOS TEÓRICOS

1.1 MOTIVAÇÃO E CONTEXTUALIZAÇÃO

De forma a garantir que um projeto é entregue com sucesso, não é apenas necessário garantir o bom planeamento do mesmo. Depois de iniciados os trabalhos de desenvolvimento, é também necessário assegurar que o plano é concretizado ou, no caso de não se poder concretizar, encontrar alternativas de forma a se realize sem diminuir o seu valor [10]. O *Project Management Body of Knowledge* (PMBoK) apresenta cinco grupos de processos, sendo que um dos quais é reservado para a monitorização e controlo [11]. Este grupo de processos consiste em controlar as mudanças e recomendar ações corretivas ou preventivas, antecipando assim possíveis problemas e monitorizando as atividades em curso contra o plano de gestão do projeto e a *baseline* da medição do desempenho do projeto, garantindo desta forma que apenas as mudanças aprovadas são implementadas [11]. A monitorização contínua tem como vantagens fornecer à equipa uma introspeção sobre a saúde do projeto e identificação das áreas que necessitam de atenção adicional, sendo que esta análise deve ser realizada dentro de intervalos regulares. O grupo de processos de monitorização e controlo pode tomar ações preventivas no caso de existirem riscos para o projeto, sendo que esta revisão pode resultar em atualizações ao plano de gestão do projeto.

As métricas permitem monitorizar e analisar de forma simples o desempenho de projetos, criando também alertas de situações que devem ser corrigidas. O conceito e uso de métricas não é recente nem desconhecido da maioria das organizações. Uma das técnicas mais populares é EVM, que combina métricas de âmbito, tempo e custos para avaliar o progresso e a performance do projeto [12]. Dada a sua simplicidade de implementação, esta técnica pode ser utilizada em projetos independentemente da indústria em que este se encontra inserido [11]. Apesar de o EVM ter sido usado fielmente por muitos gestores e organizações durante décadas, muitos dos projetos que foram monitorizados de acordo com esta técnica não foram entregues com sucesso. Existem vários fatores por que podem causar o insucesso, mas o que se verificou na grande maioria dos casos foi que os clientes não esperam apenas que o projeto siga o triângulo âmbito, tempo e custo. É também esperado, por exemplo, que o projeto apresente requisitos de qualidade. Assim, é torna-se imprescindível que a definição de sucesso seja

acordada junto de todos os *stakeholders*, assim como todas as alterações quer ao planeamento do projeto, quer aos seus objetivos [13].

O objetivo final das métricas e *dashboards* para monitorização das mesmas não é fornecer mais informações, mas fornecer as informações corretas e fidedignas à pessoa certa, no momento certo. Hoje em dia, existe especial preocupação com a sobrecarga de informações. No entanto, o problema real é a sobrecarga de informação irrelevante [14]. Por outras palavras, há muitos relatórios inúteis que não podem ser facilmente lidos e que fornecem ao leitor demasiada informação, muitos dos quais podem não ter relevância, e funcionam apenas como distração das questões reais. As métricas irrelevantes, ineficazes, ou insuficientes dificultam a perceção de quais decisões realmente necessitam de ser feitas. Kerzner [14] afirma que a gestão de métricas deve ser abordada em todas as áreas de conhecimento do PMBoK, especialmente na gestão de comunicações. Várias empresas que criam as próprias metodologias de gestão de projetos negligenciam as métricas necessárias para a gestão eficiente das relações entre *stakeholders* [15].

Existe um crescente número de trabalhos de investigação sobre métricas na área da engenharia de software, dado que a boa gestão das mesmas aumenta a probabilidade de sucesso do projeto. No início dos anos 50, a métrica mais utilizada no desenvolvimento de software era *Lines of Code* (LOC). LOC era extremamente eficiente nessa data, pois a fase de implementação do software constituía cerca de 50% do esforço total para construção do software [16]. O número de linhas de código estava relacionado com medidas de produtividade e de qualidade. No entanto, a engenharia de software evoluiu e estudos recentes indicam que o risco de insucesso cresce exponencialmente com o aumento do tamanho e da complexidade do software [17]. Jones [18] afirma que a maioria dos projetos de software falhados são resultado de métricas imprecisas e de técnicas de medição inadequadas. Como exemplo de métricas de software geralmente utilizadas podemos referir o tamanho do software, a sua complexidade, o número de erros, e métricas de estimação de recursos [19].

1.2 Engenharia de Software

O *Institute of Electrical and Electronics Engineers* (IEEE) define engenharia de software como

> "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)"[20].

Isto é, a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software.

Os primeiros computadores digitais surgiram na década de 1940 [21]. No entanto, só na década de 1950 surgiram as primeiras linguagens de programação [22], de forma a criar mais abstração entre o *hardware* e o software, diminuindo assim a complexidade da computação.

Várias linguagens de programação, como Fortran, foram lançadas no final da mesmo década para permitirem a implementação de algoritmos mais complexos, de forma eficiente. Em 1972, David Parnas [23] apresentou o conceito de modularidade para ajudar os programadores a lidar com a crescente complexidade dos sistemas de software.

Em 1984, o *Software Engineering Institute* (SEI) foi estabelecido como um centro de pesquisa e desenvolvimento financiado pelo governo com sede no campus da Universidade Carnegie Mellon em Pittsburgh, Pensilvânia, Estados Unidos da América. Watts Humphrey fundou o SEI *Software Process Program*, com o objetivo de entender e gerir o processo de engenharia de software. Os Níveis de Maturidade do Processo (*Process Maturity Levels*) tornar-se-iam no *Capability Maturity Model Integration for Development*, que passou a definir como o governo dos Estados Unidos da América avalia as capacidades de uma equipa de desenvolvimento de software [24].

As práticas modernas recomendadas e geralmente aceites para engenharia de software foram publicadas como o *Software Engineering Body of Knowledge* (SWEBoK). Segundo o SWEBoK, a engenharia de software pode ser dividida em várias sub-disciplinas:

- Engenharia de requisitos - A elicitação, análise, especificação, e validação dos requisitos do software.
- *Design* de software - O processo de definição da arquitetura, componentes, interfaces, e outras características do sistema ou componente. Também pode ser definido como os resultados do mesmo processo.
- Implementação do software - A criação detalhada de software operacional e interessante, através da combinação de atividades de programação, verificação, testes unitários, testes de integração e *debugging*.
- Testes - A investigação empírica e técnica conduzida para fornecer aos *stakeholders* informação sobre a qualidade do produto ou serviço a ser testado.
- Manutenção do software - O conjunto de atividades necessárias para fornecer suporte ao software.
- Gestão da configuração - A identificação da configuração de um sistema em pontos temporais distintos, com o objetivo de controlar sistematicamente as mudanças na configuração e manter a integridade e rastreabilidade da configuração ao longo do ciclo de vida do sistema. Os processos modernos usam o controlo de versões do software.
- Gestão da engenharia de software - A aplicação de atividades de gestão software planeamento, coordenação, medição, monitorização, controlo e documentação software para garantir que o desenvolvimento e a manutenção de software sejam sistemáticos, disciplinados e quantificados.

- Processo de desenvolvimento de software - A definição, implementação, avaliação, medição, gestão, mudanças e melhorias do próprio processo do ciclo de vida do software.
- Modelos e métodos - A imposição de estrutura sobre engenharia de software com o objetivo de tornar essa atividade sistemática, repetível, e mais orientada para o sucesso.
- Gestão da qualidade do software - Atividades que garantem que o software tem as características desejadas, verifica a extensão em que um produto de software em particular possui essas características, e os processos, ferramentas e técnicas que são usadas para garantir essas características.
- Prática profissional da engenharia de software - Trata de questões relacionadas com o conhecimento, habilidades e atitudes que os engenheiros de software devem possuir para praticar engenharia de software de forma profissional, responsável e ética.
- Questões económicas - Tomada de decisões relacionadas com a engenharia de software em contexto organizacional.
- Fundamentos Computacionais,
- Fundamentos Matemáticos, e
- Fundamentos de Engenharia [7].

1.3 DEFINIÇÃO DE PROJETO

Um projeto é definido pelo Project Management Institute (PMI) no documento *A Guide to Project Management Body of Knowledge* como um esforço temporário empreendido para criar um produto ou serviço único. É definido como temporário no sentido em que cada projeto tem um início e um fim definitivos, ou seja, não se repetem no tempo. É ainda definido como único pois deve gerar um produto ou serviço que se destaque outros produtos ou serviços similares, isto é, deve gerar resultados interessantes e inovadores [11].

1.4 DEFINIÇÃO DE SUCESSO

Boas práticas de gestão de projetos são essenciais para o sucesso dos mesmos [25, 26]. Apesar da atenção que nos últimos anos tem sido dedicada à gestão de projetos, em muitos casos os projetos ainda não oferecem o sucesso esperado [27]. Por exemplo, no caso particular de projetos de software, os estes continuam a mostrar baixas taxas de sucesso [28, 29, 30]. O sucesso dos projetos ainda está longe do desejável e o estabelecimento de práticas de gestão de projetos eficientes continua um desafio [29].

O sucesso do projeto tem duas componentes distintas: sucesso na gestão de projetos e sucesso na entrega do projeto [31]. O sucesso de gestão de projetos concentra-se nos processos de gestão e a sua realização bem sucedida em relação ao âmbito, tempo e custos, sendo que estas 3 dimensões indicam o grau de eficiência da execução do projeto. O sucesso dos entregáveis concentra-se nos efeitos dos resultados do projeto. Existe relação entre o projeto e os seus resultados [32]. No entanto, a relação causa-efeito entre eles é fraca [33], isto é, o projeto pode ser bem sucedido do ponto de vista da gestão, mas ser mal sucedido do ponto de vista dos resultados. A garantia do sucesso dos resultados do projeto é mais difícil de garantir do que a garantia do sucesso da gestão do projeto [34].

De acordo com Kerzner [35], a definição correta de sucesso é aquela que avalia fatores primários e secundários. Os fatores primários incluem os limites orçamentais, cumprimento dos prazos estabelecidos e o nível esperado de qualidade. Os fatores secundários consistem, por exemplo, nas competências dos Recursos Humanos (RH) chave e a complexidade do projeto [30]. Durante o desenvolvimento do projeto, é muito difícil cumprir todos os requisitos, permanecer dentro do orçamento, cumprir os prazos e as expectativas dos clientes. Para que um projeto seja bem sucedido, é necessário gerir todas as atividades e aspetos que envolvam requisitos, custos, riscos e tempo, especialmente os que são críticos para o sucesso [30].

Dada a complexidade e ambiguidade desta questão [31], a comunidade científica tem reunido esforços para entender melhor o fenómeno [36, 37].

1.4.1 *Sucesso em Projetos de Software*

Existe uma grande variedade de interpretações do significado de sucesso dos projetos de software [38], o que se traduz numa diversidade de variáveis que o descrevem. Gerir um projeto de desenvolvimento de software implica lidar com várias adversidades, visto que a maioria dos projetos não são completamente desenvolvidos, ultrapassam o orçamento previsto, e alguns chegam mesmo a ser cancelados [39, 17]. Uma das principais causas dessa falha é falta de eficiência da gestão [40]. Aspetos como ”Envolvimento do cliente”, ”Competência das equipas”, ”Requisitos bem definidos”e ”Planeamento adequado”são tidos como importantes fatores de sucesso [17]. Um critério base para a avaliação do sucesso é a capacidade do projeto alcançar os seus objetivos [41].

No entanto, é comum existirem falhas na clareza das especificações e definições ambíguas dos objetivos [42], mesmo sendo a fase de engenharia de requisitos uma das fases de maior relevância para o desenvolvimento de projetos de software [43]. A modificação dos requisitos e objetivos durante a execução do projeto pode influenciar o sucesso dos resultados. Estas alterações podem surgir derivadas de mudanças no ambiente no qual o projeto é desenvolvido e podem aumentar os esforços necessários para a conclusão do sucesso. Os atrasos e interrupções, quando frequentes, podem também contribuir para o aumento do esforço [44]. Assim, é

possível afirmar que um aspeto crítico do sucesso da gestão é a caraterização clara, correta e inequívoca do âmbito e do ambiente do projeto.

Outro aspeto que limita o sucesso do projeto são as falhas na gestão dos custos. Entre as atividades mais exigentes na gestão de projetos de software é a estimação dos custos dos mesmo. As empresas que desenvolvem software ainda não estimam com precisão os custos do mesmo [45], que leva ao mau planeamento do projeto. O mau planeamento resulta em atrasos na execução dos trabalhos, falta de RH, baixo nível de qualidade dos resultados, ou até mesmo ao cancelamento do projeto.

A qualidade desempenha também um papel essencial para o sucesso do projeto. O controlo eficiente da qualidade é um dos fatores mais importantes que separa um projeto concluído com sucesso de um falhado [46]. Assim, é importante que o projeto atenda os requisitos estabelecidos, bem como as necessidades do cliente. O controlo a qualidade é fundamental, bem como a revisão do software, de forma a detetar e remover os defeitos do mesmo [47].

Gerir a equipa do projeto com base num planeamento incorreto pode comprometer o sucesso do mesmo. A falta de responsabilidade, apoio e foco dos membros da equipa podem também prejudicar os resultados. Assim, é necessária a gestão correta dos RH para conseguir que o nível comprometimento de cada membro da equipa seja o máximo. Um ambiente onde muitas pessoas interagem é propenso à ocorrência de conflitos. Estes devem ser resolvidos antes de afetarem o projeto [48].

É importante envolver o cliente desde o arranque do projeto, de forma a não criar expectativas erradas sobre o resultado final. Os clientes precisam estar cientes do plano do projeto, dos recursos financeiros envolvidos e dos benefícios gerais do projeto [49]. De acordo com Day [50], o sucesso depende da habilidade humana de comunicar. Se não existir troca de informação com frequência entre todos os *stakeholders*, o sucesso do projeto pode não ser alcançado.

1.5 Métricas na Gestão de Projetos

Uma métrica é algo que pode ser medido. É possível utilizar métricas para analisar o desempenho dos projetos e de organizações. Durante muitos anos, os gestores das mais variadas indústrias utilizaram o EVM para recolher informações relevantes acerca do desenvolvimento dos projetos [12]. No entanto, o estado real do projeto não pode ser determinado apenas por métricas de âmbito, tempo e custo. Apesar de estes aspetos serem relevantes para a análise da saúde do projeto, outras métricas, que fornecem informação sobre aspetos importantes para o sucesso do projeto devem também ser monitorizadas. Todavia, se não forem selecionadas métricas confiáveis, o resultado da análise pode não ser fidedigno [14]. É importante ter em mente a seguinte premissa bastante popular durante o processo de seleção das métricas:

> *Not everything that counts can be counted, and not everything that can be counted counts* [51].

As métricas devem ser definidas desde o início do projeto e partem com a identificação dos *stakeholders*. Estes devem participar na definição das métricas. No entanto, poderá ser necessário ter métricas específicas para cada *stakeholder*, isto é, selecionar as métricas mais interessantes do ponto de vista do *stakeholder* para este conseguir mais facilmente focar-se nos aspetos mais interessantes e relevantes do seu ponto de vista, pois pode não ser possível obter um conjunto de métricas interessante para todos [14]. O gestor de projetos deve analisar cada *stakeholder* para perceber quais os mais influentes e os que podem oferecer maior suporte ao projeto. Se existirem alterações na lista de *stakeholders* durante o ciclo de vida do projeto, as métricas devem também ser alteradas para corresponder às suas necessidades. A gestão eficiente dos *stakeholders* pode ser a diferença entre um sucesso extraordinário ou um terrível fracasso [52].

Quando o gestor de projetos promove a utilização de métricas, deve ter em conta que o uso destas requer mudanças, algo que nem sempre é bem aceite por parte das equipas. Os RH não aceitam que o esforço colocado na gestão das métricas seja utilizado para os monitorizar. Algumas pessoas são muito sensíveis quando sabem que o seu desempenho está a ser medido. Assim, as métricas devem ser utilizadas com cuidado e para monitorização da performance do projeto, não como uma base para punição [14].

De acordo com Head [53], ao estabelecer os processos a serem usados na seleção de métricas, é importante priorizar a lista por ordem de importância e evitar processar qualquer uma que não seja verdadeiramente necessária. É sempre possível aumentar o número de métricas à medida que o projeto avança, mas tentar assumir demasiado no início pode roubar tempo e atenção de outras atividades críticas do projeto.

Parmenter [54] define 3 categorias de métricas:

1. *Results indicators* (RIs): O que é que foi conseguido até agora?
2. *Performance indicators* (PIs): O que é que se pode fazer para aumentar ou cumprir a performance?
3. *Key performance indicators* (KPIs): Quais são os indicadores críticos de desempenho que podem aumentar drasticamente o desempenho ou que permitem a realização dos objetivos?

Não é possível gerir uma métrica que não possa ser medida. Existem várias técnicas que permitem a medição, por exemplo, tabelas com dados, simulações, estatísticas, técnicas de amostragem, intervalos de dados, julgamento humano, percentagens, entre outros. Para cada métrica, deve ser selecionada pelo menos uma técnica de medição. Se necessário, podem ser escolhidas várias técnicas. Os objetivos intangíveis também são mensuráveis, apesar de serem mais difíceis de quantificar. Como exemplo de objetivos intangíveis é possível referir a colaboração, compromisso, satisfação do cliente, maturidade dos funcionários, eficácia da liderança, motivação, nível de *stress*, e trabalho em equipa [14].

1.5.1 *Métricas de Software*

Muitos dos melhores engenheiros de software medem as características deste para perceber se os requisitos são consistentes e completos, se o *design* é de alta qualidade ou se o código está pronto para ser lançado. Os gestores de projetos eficientes medem atributos de processos e produtos para saber quando o software estará pronto para entrega ou se o orçamento será excedido. As organizações usam medidas de avaliação de processos para selecionar fornecedores. Os clientes informados medem as características do produto final para determinar se ele atende aos requisitos e se apresenta qualidade suficiente [55].

Existem inúmeros usos para métricas nos projetos de software. As métricas podem ser utilizadas para prever recursos em fases mais avançadas do projeto. Esta é uma das áreas com menos robustez nas metodologias atuais, sendo possível realçar a falta de eficiência nas técnicas utilizadas para estimar os recursos do projeto. As estimativas não são só importantes durante a fase de planeamento, mas também são de extrema importância durante a fase de execução. A maioria dos projetos de software sofrem de alterações aos requisitos durante o seu ciclo de vida, pelo que recalcular o efeito destas alterações utilizando as técnicas convencionais se torna um processo demorado. [56]

As métricas podem ser usadas como mecanismos de *Quality Assurance* (QA). Se o gestor de projetos está confiante de que existe relação direta entre uma métrica preditiva de um produto de software e uma métrica relevante para os resultados, pode garantir que os valores da métrica do produto são mantidos dentro dos limites que levam a valores aceitáveis dos resultados [57].

Em projetos de software, as métricas podem também ser utilizadas como mecanismos de avaliação da performance da equipa. Se apenas for obtida informação sobre a performance em estágios avançados do projeto, não é possível aplicar as devidas ações corretivas. Por exemplo, um fraco *design* é apenas detetado durante as fases de integração dos módulos, de testes ou de validação do sistema. Recorrendo a métricas, o gestor de projetos pode monitorizar o trabalho produzido, de forma a ajudar a equipa a atingir melhores resultados [58] e estimar a qualidade do trabalho produzido [59].

Um outro uso para as métricas é como forma de avaliação de métodos de desenvolvimento concorrentes, de formas de trabalho individual e das estruturas organizacionais. Ao comparar os vários métodos de desenvolvimento, é possível inferir conclusões sobre qual o método que permite resultados mais rápidos, mais económicos ou até qual o método que facilita as atividades de suporte do software [56, 59].

Finalmente, as métricas podem também ser usadas como base para ferramentas de desenvolvimento de software inteligente e semi-inteligente. Uma área de pesquisa tem como objetivo desenvolver ferramentas que proporcionem grande ajuda na especificação e *design* do sistema ou mesmo que automatizem totalmente estes processos [60].

1.6 PERGUNTAS DE INVESTIGAÇÃO

Nesta dissertação pretende-se responder a duas Perguntas de Investigação:

- Quais são as principais métricas a monitorizar durante o desenvolvimento de software de forma a concluir o mesmo com sucesso?
- Quais os processos e/ou atividades que devem ser utilizados para acompanhar as métricas selecionadas?

1.7 METODOLOGIA DE INVESTIGAÇÃO

Para o desenvolvimento desta dissertação a metodologia utilizada será a *Design Science Research* (DSR). A DSR é uma metodologia de investigação que, através da construção de artefactos, procura responder a questões relevantes, contribuindo para o conhecimento científico da comunidade da qual se insere [1].

DSR promove a compreensão de sistemas avançados de informação de ponta através da construção e avaliação desses sistemas e dos seus componentes. Como esta metodologia pode produzir resultados rigorosos e significativos na ausência de uma base teórica forte, esta destaca-se na investigação de tecnologias novas e até mesmo especulativas [1].

Esta metodologia divide-se nas seguintes fases apresentadas em esquema na Figura 1.

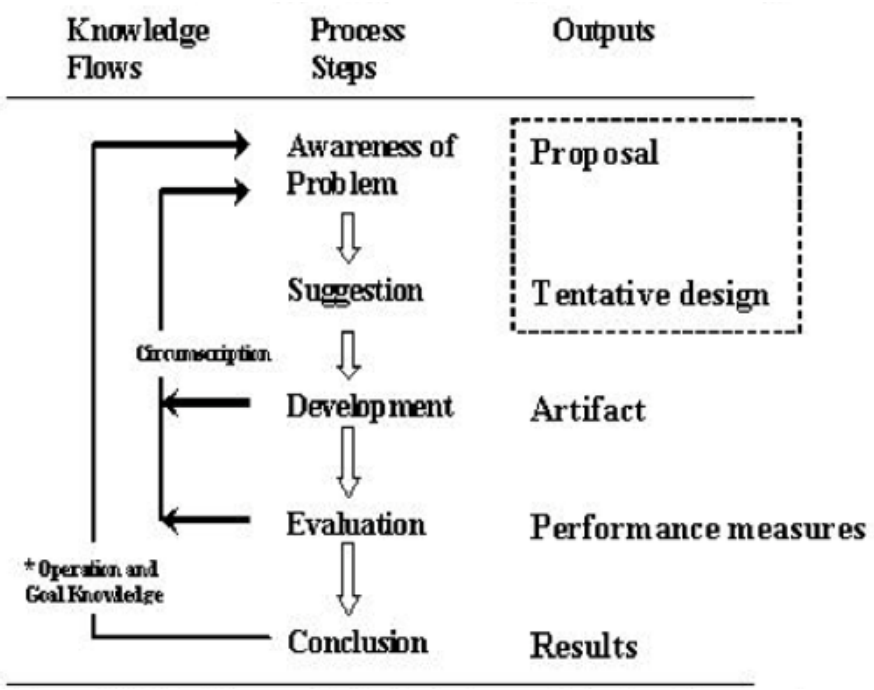


Figura 1: Metodologia geral de *Design Science Research* (DSR) [1].

1.8 ESTRUTURA DO DOCUMENTO

No primeiro capítulo desta dissertação é feita uma introdução ao tema escolhido, assim como uma apresentação de conceitos teóricos e definições relevantes para o assunto em discussão. Ainda no primeiro capítulo são ainda apresentadas as perguntas e a metodologia de investigação.

No segundo e terceiro capítulos é feito o levantamento do estado da arte, abordando assuntos como *frameworks* para gestão de projetos, gestão da qualidade, gestão de risco e gestão de requisitos, sendo que no segundo capítulo são abordados assuntos mais gerais e introdutórios sobre a disciplina da gestão de projetos e o terceiro capítulo foca-se em responder a questões relacionadas com a gestão de projetos dentro do contexto de engenharia de software.

No quarto capítulo são apresentados os resultados obtidos nesta dissertação, isto é, as métricas identificadas, assim como os procedimentos a seguir de forma a monitorizar as mesmas.

Por fim, no quinto capítulo são apresentadas as conclusões e a proposta de trabalho futuro.

2

GESTÃO DE PROJETOS

Tal como definido anteriormente, um projeto é um esforço temporário, empreendido para criar um produto ou serviço único. Com a singularidade do projeto, surge um desafio: como é que um projeto deve ser gerido?

Os métodos utilizados para gerir projetos também evoluíram. Desde o passado até hoje, a Gestão de Projetos é uma disciplina em constante evolução. Esta disciplina tornou-se uma realidade no início do século XX, pela mão de Frederick Taylor (1856-1915). Em *The Principles of Scientific Management*, Taylor analisa e sintetiza fluxos de trabalho, introduz o conceito de alocação de recursos e princípios de *workbreakdown*. Esta teoria é a base dos futuros conceitos de gestão. A abordagem de Taylor é frequentemente referida como Princípios de Taylor ou Taylorism [61].

Os princípios de Taylor inspiraram muitos outros autores, como o caso de Henri Fayol (1841-1925) e Henry Gantt (1861-1919). Fayol publicou em 1916 o livro *"Administration industrielle et générale"* ("Administração industrial e geral", em português), no qual descreve as funções básicas de gestão: planeamento, organização, comando, coordenação e controlo. O seu trabalho é de grande relevância para a gestão de projetos moderna.

Gantt extendeu o trabalho de Fayol, sendo que o seu trabalho se focou em técnicas de planeamento e controlo e criou uma das ferramentas de gestão de projetos mais populares: o diagrama de Gantt. O uso dessa ferramenta é reportado desde a Primeira Guerra Mundial, onde foi utilizada para planear e controlar a construção de navios de guerra [62].

Durante a Revolução Industrial, a procura pelo aumento da produtividade trouxe imenso sucesso à ferramenta desenvolvida por Gantt. Os seus diagramas mostraram-se uma ferramenta poderosa para os gestores, sendo que não foram feitas alterações aos mesmos durante décadas. Apenas do final do século XX foram propostas alterações e novas funcionalidades, como o caso da dependência entre tarefas, sendo que esta técnica é utilizada como base da maioria do software de planeamento de projetos.

Apesar desta disciplina ter sofrido alterações durante a Revolução Industrial e durante a Segunda Guerra Mundial, a Gestão de Projetos tal como a conhecemos apenas surgiu em meados do século XX. As necessidades mudaram e tornou-se óbvio que os projetos mais complexos deviam ser geridos de forma mais eficiente. A Gestão de Projetos evoluiu durante os tempos de guerra, onde atingir o *time-to-market* é crucial, pelo que surge a necessidade de aumentar

o número de máquinas e armas que são enviadas para o campo de batalha. Os benefícios da gestão foram facilmente compreendidos por várias indústrias, as quais espelharam estas boas práticas. A rápida mudança leva os líderes das organizações a procurar novas formas de lidar com a mudança e crescimento do mercado, num mundo cada vez mais competitivo.

Durante os anos 70, a Gestão de Projetos foi disseminada em vários jornais, conferências e seminários, revelando os resultados obtidos e conjuntos de boas práticas. O próximo passo lógico foi a criação de *standards*, de forma a garantir a utilização e expansão global da disciplina.

A uniformização da Gestão de Projetos deu os seus primeiros passos durante os anos 80, com o Project Management Institute (PMI), e mais tarde com o *Association for Project Management* (APM). Ambas as instituições publicaram um *Body of Knowledge* sobre a disciplina, revelando as boas práticas e *guidelines* (*Project Management Body of Knowledge* (PMBoK) e *Association for Project Management Body of Knowledge* (APMBoK), respetivamente). Estas práticas são transversais a todas as indústrias, o que provoca que algum conteúdo não seja aplicável a todos os projetos, independentemente da área [63].

O *Project Management Body of Knowledge* (PMBoK) [11] é um conjunto de terminologia e procedimentos *standard* para a gestão de projetos. O corpo de conhecimento (*Body of Knowledge* em inglês) evolui ao longo do tempo e é apresentado num guia (*A Guide to Project Management Body of Knowledge* ou *PMBoK Guide*), um livro cuja sexta edição foi lançada em 2017. A edição mais recente fornece também *guidelines* sobre métodos ágeis (*Agile Practice Guide*). O *PMBoK Guide* é um documento resultante do trabalho do Project Management Institute (PMI).

O PMBoK é baseado em processos. Os processos sobrepõem-se e interagem ao longo de um do ciclo de vida do projeto [11]. Esta abordagem é consistente com outros *standards*, como o ISO 9000 [64] e o *Capability Maturity Model Integration* (CMMI) [65]. O PMBoK reconhece 49 processos, sendo que cada um deles cai num dos 5 grupos básicos de processos. Os 5 grupos de processos são os seguintes:

1. Iniciação - processos executados para definição de um novo projeto ou nova fase de um projeto já existente, e obtenção da autorização para iniciar o mesmo.

2. Planeamento - processos necessários para estabelecer o âmbito do projeto, refinar os seus objetivos, e definir o plano para atingir os objetivos do projeto.

3. Execução - processos realizados para completar o o trabalho definido no plano de gestão e satisfazer as especificações do projeto.

4. Monitorização e Controlo - processos necessários para acompanhar, analisar e regular o progresso e o desempenho do projeto; identificação de todas as áreas em que são necessárias alterações do plano; e iniciar as mudanças correspondentes.

5. Fecho - processos executados para finalizar todas as atividades em todos os Grupos de Processo para fechar formalmente o projeto ou a fase [11].

Cada processo consiste num conjunto de *Inputs* (documentos, planos, etc.), Ferramentas e Técnicas (mecanismos aplicados às entradas), e *Outputs* (documentos, planos, etc.). Os *outputs* de um processo são normalmente *inputs* de outro.

O PMBoK engloba 10 áreas de conhecimento, sendo que cada uma delas contém vários processos de diferentes grupos:

1. Integração - processos e atividades necessários para identificar, definir, combinar, unificar e coordenar os vários processos e atividades de gestão de projetos dentro dos grupos de processos.
2. Âmbito - processos necessários para garantir que o projeto inclua todo o trabalho necessário, e apenas o trabalho necessário, para completar o projeto com sucesso.
3. Tempo - processos necessários para gerir a conclusão atempada do projeto.
4. Custos - processos envolvidos no planeamento, estimativa, orçamentação, financiamento, gestão e controlo de custos para que o projeto possa ser concluído dentro do orçamento aprovado.
5. Qualidade - processos e atividades que determinam políticas, objetivos e responsabilidades de qualidade para que o projeto atenda as necessidades para as quais foi realizada.
6. Recursos - processos que organizam, gerem e lideram a equipa do projeto.
7. Comunicações - processos necessários para assegurar o planeamento, coleta, criação, distribuição, armazenamento, recuperação, gestão, controlo e monitorização apropriados, e também a disposição final da informação do projeto.
8. Risco - processos de identificação, análise, planeamento das respostas e controlo dos riscos do projeto.
9. Aquisições - processos necessários para comprar ou adquirir produtos, serviços ou resultados necessários externos à equipa do projeto.
10. *Stakeholders* - processos necessários para identificar todas as pessoas ou organizações impactadas pelo projeto, analisando as expectativas e impactos dos *stakeholders* no projeto e desenvolvendo estratégias de gestão adequadas para o seu envolvimento efetivo nas decisões e execução do projeto [11].

De acordo com Schwalbe [66],

> *The PMBoK Guide is a standard that describes best practices for what should be done to manage a project. A methodology describes how things should be done [...]*

Isto é, o PMBoK é um *standard* que responde ao "o quê?", enquanto que uma metodologia dita o "como?". Assim, é possível conjugar as orientações do PMBoK com diferentes metodologias de gestão de projetos. Um outro *standard* bastante popular é o ISO 21500 *Guidance on Project Management*. Este é um *standard* internacional, desenvolvido pela *International Organization for Standardization* (ISO), e estabelecido em 2012. O seu objetivo é fornecer uma descrição de alto nível de conceitos, processos e boas práticas para gestão de projetos, que pode ser utilizado em projetos de qualquer indústria, independentemente do seu tamanho, complexidade ou duração. Os projetos estão normalmente compreendidos em programas e portefólios. No entanto, o ISO 21500 não fornece indicação sobre a gestão destes [67].

A ISO 21500 assemelha-se ao PMBoK, mas sem as ferramentas e as técnicas. Contem 10 grupos de assuntos que refletem as 10 áreas de conhecimento do PMBoK. Tem ainda 40 processos divididos entre este 10 grupos, sendo que cada um dos processos pertence também a um dos cinco grupos de processos: Iniciação, Planeamento, Implementação, Controlo e Fecho [67].

É estruturada da seguinte forma:

1. Âmbito - justificação da existência da ISO 21500, onde se aplica, quem poderá estar interessado e quem beneficia da sua utilização e conhecimento.

2. Terminologia - definição e identificação de termos específicos à prática da gestão de projetos.

3. Conceitos da Gestão de Projetos - definição de conceitos-chave da execução da gestão de determinado projeto, como o conceito de projeto, operações, etc.

4. Processos da Gestão de Projetos - as 10 áreas de conhecimento de um projeto: Integração, *Stakeholders*, Âmbito, Recursos, Tempo, Custo, Risco, Aquisições, Comunicações.

A ISO 21500 segue a abordagem caracterizada pelo PMBoK, como é possível verificar tendo por base os processos que define, que são compatíveis com as áreas de conhecimento definidas no PMBoK.

Os planos da ISO são que este seja o primeiro de uma série de *standards* sobre gestão de projetos. A ISO também desenvolveu esta norma para ser alinhada com outras normas relacionadas, como ISO 10005:2005 *Quality management systems – Guidelines for quality plans*, ISO 10006:2003 *Quality management systems – Guidelines for quality management in projects*, ISO 10007:2003 *Quality management systems – Guidelines for configuration management*, ISO 31000:2009 *Risk management – Principles and guidelines*.

Não existem apenas standards publicados no âmbito da gestão de projetos, mas também metodologias, tal como o PRNCE2 (*Projects in Controlled Environments*). PRINCE2 é uma

metodologia estruturada de gestão de projetos criada com base em contribuições de profissionais experientes na disciplina. Dada a generalidade dos seus princípios e estratégias, estes podem virtualmente ser aplicados a qualquer projeto [68].

PRINCE2 possui uma abordagem estruturada onde são definidos e explicados os processos, responsabilidades e mecanismos a utilizar no projeto, respondendo à questão "como?".

A sua abordagem baseia-se em 7 princípios, 7 temas e 7 processos. Os princípios fornecem uma *framework* de boas práticas, sendo genéricos o suficiente para aplicar a qualquer tipo de projeto e totalmente compatíveis com o PMBoK. Os princípios de PRINCE2 são os seguintes:

1. *Continued Business Justification* – Deve existir expectativa de retorno de valor em relação ao tempo, esforço e recursos despendidos no projeto durante a sua execução.

2. *Learn from Experience* – A existência de *background* sobre fatores semelhantes e resultados obtidos e problemas encontrados, é fundamental e tem influência sobre o sucesso de um projeto.

3. *Define Roles and Responsibilities* – A existência de uma hierarquia estruturada e de responsabilidades bem definidas e distribuídas é fundamental.

4. *Manage by Stages* – Atividades ou problemas complexos são geridos mais facilmente se forem fracionados em pequenos fragmentos.

5. *Manage by Exception* – PRINCE2 assume que os supervisores do projeto têm pouco tempo disponível e muito trabalho para executar. Apenas quando existe um desvio fora da área de tolerância definida para qualquer um dos fatores restritivos do projeto é que devem ser notificados os supervisores.

6. *Focus on products* – Definição de requisitos, entregáveis e padrões de qualidade. É necessário saber o que é esperado do produto para que se possam definir as atividades do projeto.

7. *Tailor to the environment* – É essencial adaptar a metodologia às necessidades do projeto. Todos os projetos são únicos e possuem necessidades diferentes no que se refere à sua gestão [68].

Os temas tratam de questões mais práticas e podem ser vistos como áreas de conhecimento, onde é descrito como proceder dentro de cada área. Devem ser abordados no início do projeto e deve ser feita a sua monitorização e controlo ao longo do ciclo de vida do projeto. Os temas propostos no PRINCE2 são os seguintes:

1. *Business Case* – Justificação do projeto, determinando a razão da sua existência, o seu impacto, demonstrando que o projeto deve continuar a ser desenvolvido e é possível concluir o mesmo com sucesso.

2. *Organisation* – Definição de papéis e responsabilidades dos envolvidos no projeto, ou seja, definir a organização para que seja claro quem é responsável por dada tarefa.

3. *Quality* – Definição dos padrões de qualidade a cumprir para que exista foco no desenvolvimento do projeto e seja possível verificar se o produto é adequado às necessidades.

4. *Plans* – Esquematização de como os objetivos vão ser atingidos, tratando das áreas de âmbito, calendarização, custo, qualidade, e possíveis benefícios.

5. *Risk* – Identificar, avaliação e controlo de eventos associados ao projeto, quer sejam ameaças, quer sejam oportunidades.

6. *Change* – Aceitação de mudanças quando justificadas.

7. *Progress* – O controlo e monitorização do projeto permite identificar desvios em relação ao plano e reagir atempadamente a essas alterações [68].

Os processos são enquadrados no ciclo de vida de um projeto. Para cada fase, oferecem a definição do que deve ser feito pelo gestor de projeto, mantendo a abordagem prática do característica PRINCE2. Os processos propostos são os seguintes:

1. *Starting Up a Project* – Justificação formal do projeto, definição da equipa e distribuição responsabilidades, e planificação como o projeto vai ser realizado.

2. *Initiating a Project* – Identificação do caderno de encargos e como vão ser tratadas as áreas de tempo, custo, qualidade, âmbito, benefícios e risco.

3. *Directing a Project* – Gestão dos recursos, monitorização do progresso, fornecimento de direções e conclusão o projeto são algumas das tarefas deste processo.

4. *Controlling a Stage* – O gestor de projeto deve monitorizar e redigir relatórios sobre o progresso das atividades e corrigir os desvios ou problemas que surjam durante o desenvolvimento do projeto. O gestor de equipa é responsável pela coordenação das atividades de trabalho diárias e de intermediário para a comunicação entre equipa e gestor de projeto.

5. *Managing Product Delivery* – Definição da comunicação entre gestor de projeto e gestor de equipa, que normalmente consiste na aceitação de tarefas, facultação da execução da tarefa e entrega dos resultados.

6. *Managing Stage Boundaries* – O gestor de projeto deve proceder à marcação de reuniões com a equipa para registar lições aprendidas para a próxima etapa do projeto. Os supervisores do projeto decidem se o projeto continua ou não a cada ponto de análise (*Stage gate*). Desta forma, é necessário planificar a etapa seguinte do projeto, atualizar

o plano e o modelo de negócio e reportar aos supervisores o final de cada etapa ou de um desvio fora da tolerância estabelecida.

7. *Closing a Project* – Identificação do cumprimento dos objetivos e o caderno de encargos, definição de que ações vão ser necessárias após a sua conclusão, libertação de recursos e entrega do produto ao cliente [68].

É de realçar que PRINCE2 é compatível com o PMBoK, sendo que podem ser utilizados em sintonia no mesmo projeto. A metodologia PRINCE2 continua sem fornecer um método prático de gestão de projetos, isto é, não fornece técnicas e planos detalhados sobre como gerir o projeto, precisamente para poder ser aplicado a projetos cuja área não está restrita a grupos específicos.

3

GESTÃO DE PROJETOS DE SOFTWARE

Selecionar a abordagem certa para gestão de projetos de software depende de vários fatores, tais como a qualidade, os custos e calendarização de atividades. Brooks, na sua obra mais popular, *"The Mythical Man-month"*, resume a importância da gestão de projetos de forma simples:

> The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers [69].

3.1 ABORDAGENS TRADICIONAIS

As abordagens tradicionais são aconselhadas quando os projetos são baseados em processos que devem ser seguidos rigorosamente para garantir a qualidade dos resultados. No entanto, esses métodos não permitem que sejam efetuadas mudanças até que todo o ciclo esteja completo. Perante um mundo dependente de novas tecnologias, com mercados cada vez mais exigentes e consumidores mais atentos, as organizações têm de se adaptar à mudança constante e à imprevisibilidade imposta pelos mercados. A solução para tornar sustentável este ambiente em constante transformação, passa pela adoção de abordagens ágeis. Assim, é possível responder rápida e eficazmente às exigências impostas pelos clientes e adaptar às mudanças do mercado, reduzindo drasticamente o tempo de desenvolvimento do software e aumentando a flexibilidade do mesmo. As abordagens ágeis serão apresentadas com maior detalhe posteriormente.

Tradicionalmente, o processo de desenvolvimento de software passava pela utilização de modelos de desenvolvimento bastante rígidos. Os processos mantinham uma estrutura inflexível que reunia vários passos desde a recolha de requisitos até aos testes e entrega do produto final. O objetivo era a criação de um produto sem falhas, de acordo com as normas de qualidade de software. Realçam-se duas formas de realizar um projeto de sucesso: *Software Development Life Cycle* e *Project Development Life Cycle*. *Project Development Life Cycle* é o conjunto de atividades dependentes umas das outras e que controlam o projeto. *Software Development Life Cycle* inclui detalhes sobre cada estágio de desenvolvimento e tem como objetivo

a normalização dos processos, de forma a que seja entregue um produto com a qualidade pretendida.

Este ciclo passa por 6 etapas: 1) Planeamento e análise, 2) Levantamento de requisitos, 3) Desenho da arquitetura do produto, 4) Implementação, 5) Testes e integração, e 6) Suporte.

Cada organização escolhe o método de gestão para o desenvolvimento de software mais adequado, sendo que entre os mais populares encontram-se: Waterfall, V-Model, e Spiral.

O modelo *Waterfall* foi apresentado por Royce [2], pioneiro na área de engenharia de software, no início dos anos 70. O progresso flui normalmente numa direção (para baixo, em cascata), através das fases de levantamento de requisitos, análise, *design*, implementação, testes e manutenção (ver Figura 2).

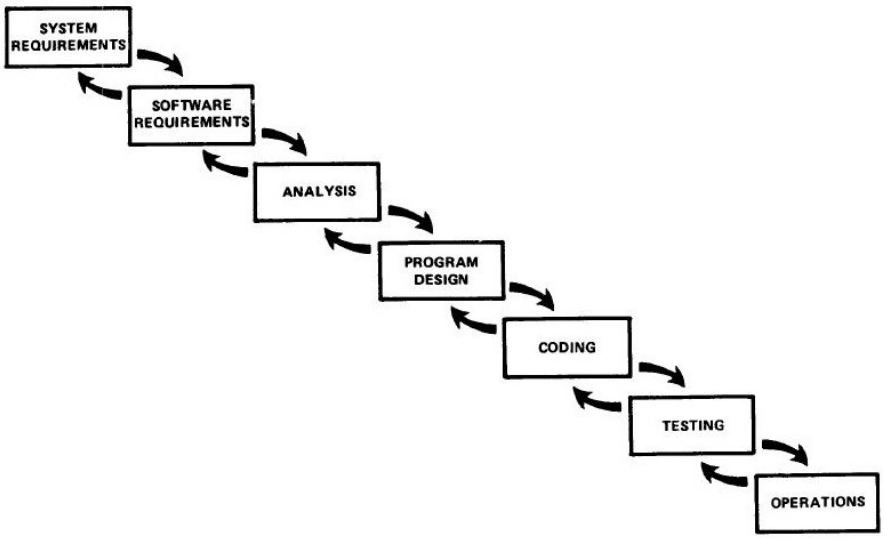


Figura 2: A interação iterativa das várias fases do modelo *Waterfall*, representada graficamente [2].

No modelo *Waterfall* é exigida a conclusão de cada estágio de desenvolvimento antes do início do próximo. Esta rigidez faz com que a revisão do produto seja um pré-requisito em cada estágio, de forma a cumprir os requisitos do cliente. No entanto, não existe flexibilidade e novos requisitos impostos pelo cliente não podem ser acomodados em nenhum estágio. Isso requer que trabalho deva voltar a ser desenvolvido desde o início. Neste modelo, a criação do protótipo requer a conclusão do ciclo de vida. O próprio autor, Winston W. Royce, apresenta falhas deste modelo:

> The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the

> software requirements upon which the design is based and which provides the rationale for everything are violated [...] [2]

V-Model representa um processo de desenvolvimento que pode ser considerado como uma extensão do modelo *Waterfall*. No entanto, em vez do fluxo das tarefas fluir de forma linear, as etapas do processo são dobradas para cima após a fase de implementação, para formar a forma em "V"típica (ver Figura 3). O V-Model demonstra as relações entre cada fase do ciclo de vida do desenvolvimento e sua fase de teste associada. Os eixos horizontais e verticais representam o tempo ou a completude do projeto (do lado esquerdo) e o nível de abstração (a maior parte da zona superior), respetivamente.

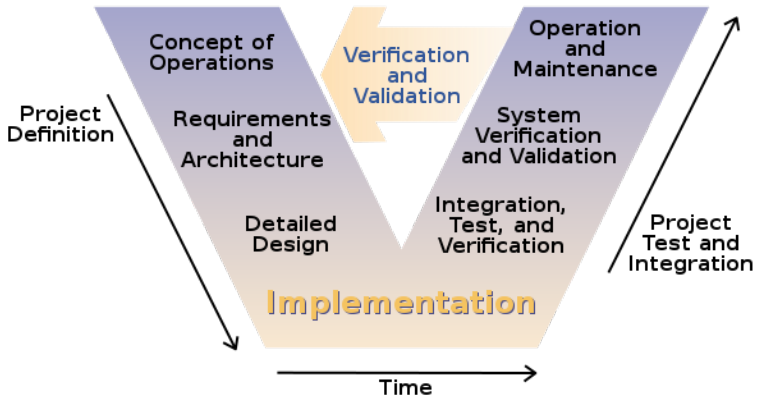


Figura 3: Representação gráfica do modelo V-Model [3].

V-Model está dividido em duas fases: verificação e validação. Na fase de verificação é feita a análise de requisitos e *design* do sistema, da arquitetura e dos módulos. Na fase de validação são feitos os testes ao sistema , unitários, de integração e de aceitação.

Este modelo tem recebido críticas [70], entre as quais se podem enumerar: (1) a sua inflexibilidade e fraca capacidade de resposta a alterações/mudanças, (2) como é uma variação do modelo Waterfall, apresenta os mesmos problemas, (3) encoraja os *testers* a procurar o que estão à espera de encontrar, em vez de procurarem realizar testes exploratórios, e (4) é pouco coerente e preciso.

Um outro método bastante popular é o *Spiral model*, sendo que este se foca na análise dos riscos. Este modelo apresenta 4 fases: planeamento, análise dos riscos, engenharia (desenvolvimento) e validação. Um projeto de software passa repetidamente por estas fases em iterações, chamadas *spirals* (ver Figura 4). A *spiral* tida como *baseline* é iniciada na fase de planeamento, onde existe levantamento de requisitos e avaliação dos riscos. Cada *spiral* consequente é construída tendo por base a *baseline* .

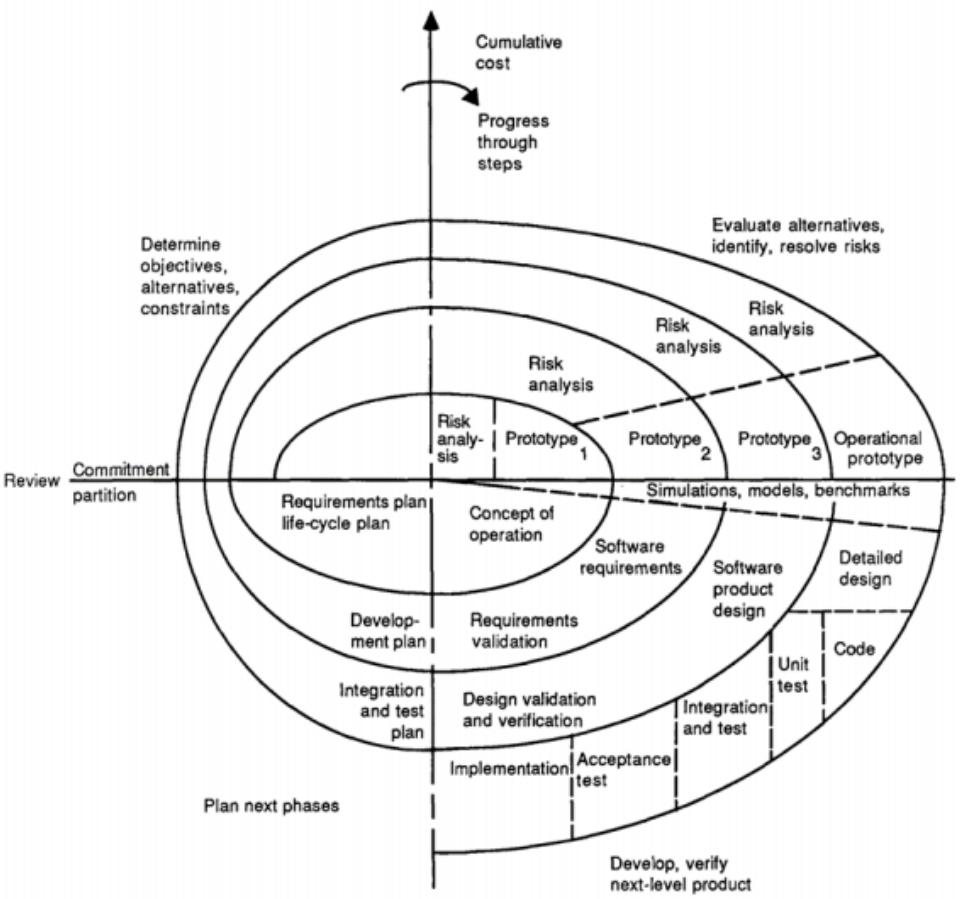


Figura 4: Diagrama do *Spiral model* [4].

Este modelo pode ser utilizado quando fazer a avaliação dos custos e dos riscos é fundamental, quando os clientes não têm uma clara definição dos requisitos ou estes são muito complexos, ou quando se esperam mudanças significativas durante o ciclo de vida. No entanto, tem também desvantagens: pode trazer vários custos, a análise dos riscos deve ser feita por alguém experiente, o sucesso do projeto é altamente dependente da fase de análise dos riscos e não funciona bem com projetos de pequena dimensão.

3.2 ABORDAGENS ÁGEIS

As abordagens ágeis são menos burocráticas, focadas no cliente, mais flexíveis e independentes. Em vez de serem baseadas em processos, documentação e um plano rígido, focam-se em indivíduos, interações e colaboração com o cliente. A utilização de métodos ágeis facilita a resposta a mudanças do mercado e novas oportunidades. Nas metodologias ágeis existem vários papeis distintos:

- *Team Leader* - Responsável por facilitar a equipa, obter recursos para a mesma e protegê-la de problemas. Requer as *soft skills* associadas à gestão de projetos, mas não as técnicas (como planeamento), que devem envolver toda a equipa.
- *Team member* - Responsável pela criação e entrega do sistema. Normalmente são incluídas tarefas como modelação, programação e teste, entre outras.
- *Product Owner* - Representa os *stakeholders*. É o responsável por priorizar o as tarefas, por tomar decisões e por fornecer informações.
- *Stakeholder* - Alguém que tenha direta ou indireta influência no desenvolvimento do projeto.

As metodologias ágeis seguem uma abordagem única:

- *Project Initiation* - Nesta fase é conduzida uma reunião para fazer um levantamento de alto nível dos requisitos. O *Product Owner* trabalha junto dos *stakeholders* para definição dos objetivos. O resultado deve ser escrito sob a forma de *User Stories*, que são apresentadas no seguinte formato: "Como um (ator) eu quero que (ação) para que (funcionalidade)".
- *Sprints*[1] *Planning* - os níveis de aceitação e o âmbito precisam ser entendidos desde o início. O*Product Owner* é envolvido em discussões sobre o âmbito, e o *Team Lead* e a equipa de *Quality Assurance* trabalham diretamente com a equipa de desenvolvimento.
- *Demo* - As demonstrações são apresentadas aos clientes e utilizadores finais. Nesta fase, a equipa decide se o produto atende aos níveis de aceitação.

Podemos ver claramente que as metodologias ágeis se concentram mais nos resultados do que em planeamento e documentação.

O *Agile Manifesto*, criado em Fevereiro de 2001, definiu formalmente o avanço dos métodos ágeis no desenvolvimento de software [71]. Este manifesto é constituído por 12 princípios e 4 valores (ver Apêndice A). O objetivo do *Agile Manifesto* é fornecer uma alternativa ao conceito de projetos desenvolvidos sobre estruturas e documentos estáticos e necessidades inalteráveis. É defendido que a existência de incerteza inicial é normal num projeto e que a capacidade de adaptação às necessidades do cliente fornece uma flexibilidade essencial para o sucesso do projeto.

1 Um *sprint* é um período de tempo definido durante o qual o trabalho específico deve ser concluído e preparado para revisão. Cada *sprint* começa com uma reunião de planeamento. Durante a reunião, o *Product Owner* (responsável por solicitar o trabalho) e a equipa de desenvolvimento concordam exatamente com o trabalho a realizar durante o *sprint*. A equipa de desenvolvimento tem a última palavra quando se trata de determinar a quantidade de trabalho que pode ser realizado realisticamente durante o *sprint*, e o *Product Owner* tem a última palavra sobre quais os critérios que devem ser cumpridos para que o trabalho seja aprovado e aceite.

A estrutura típica de uma metodologia ágil é estabelecida por iterações (*sprints*). Estas repetem-se ao longo do desenvolvimento do projeto até que este esteja completo. É comum iniciarem-se os trabalhos de desenvolvimento com uma fase de levantamento de requisitos, seguida de uma fase de planeamento e outra de execução, terminando com uma última de avaliação e retrospeção sobre o que foi realizado. É promovida a constante reavaliação do trabalho desenvolvido, o *feedback* constante entre *stakeholders*, a aceitação de alterações devidamente justificadas, a comunicação e cooperação, e o estabelecimento de entregas fracionadas.

As metodologias ágeis apresentam claras vantagens em projetos com incerteza elevada e que possam sofrer alterações no futuro. No entanto, quando existe necessidade de maior certeza e risco elevado, como no caso do desenvolvimento de software para a indústria aeroespacial, as metodologias ágeis podem não ser adequadas [72]. Quanto maior o risco, maior a necessidade de documentação detalhada, pelo que as vantagens das entregas iterativas se tornam menos evidentes. Nestas situações existe a necessidade de testes globais e extensos ao produto completo, desrespeitando o conceito de equipas de menores dimensões para promoção da comunicação.

Alguns dos métodos ágeis mais populares são *Scrum*, *Kanban* e *eXtreme Programming* (XP).

Scrum [73] é um método de desenvolvimento iterativo e incremental que fornece flexibilidade em relação a alterações relacionadas com o projeto e maior rapidez nas entregas ao cliente. Os elementos constituintes de uma equipa *Scrum* são:

- *Product Owner* - responsável por fazer a interface entre a equipa e os *stakeholders*. Deve possuir conhecimentos no domínio no qual o problema se insere, de forma a auxiliar a equipa na tomada de decisões. É ainda da sua responsabilidade validar o trabalho produzido e atribuir prioridades, especificando as *User Stories*.

- *Scrum Master* - responsável por liderar a equipa de desenvolvimento. Não interfere nas decisões das especificações de implementação, mas aconselha os elementos da equipa e sugere possíveis soluções, garantindo que os princípios do *Scrum* são respeitados. Deve ter conhecimento técnicos no domínio da solução para guiar a equipa de forma a que esta encontre as melhores soluções de acordo com os problemas que possam surgir.

- Equipa de Desenvolvimento - responsável pelo desenvolvimento da solução. Deve ser auto-organizada e independente [73].

A Figura 5 apresenta um exemplo de um projeto desenvolvido utilizando *Scrum*. Na primeira fase, é construído o *Product Backlog* do projeto, priorizado pelo *Product Owner*. Neste são definidos os objetivos do projeto, sendo que estes são reavaliados e priorizados a cada *sprint* [73].

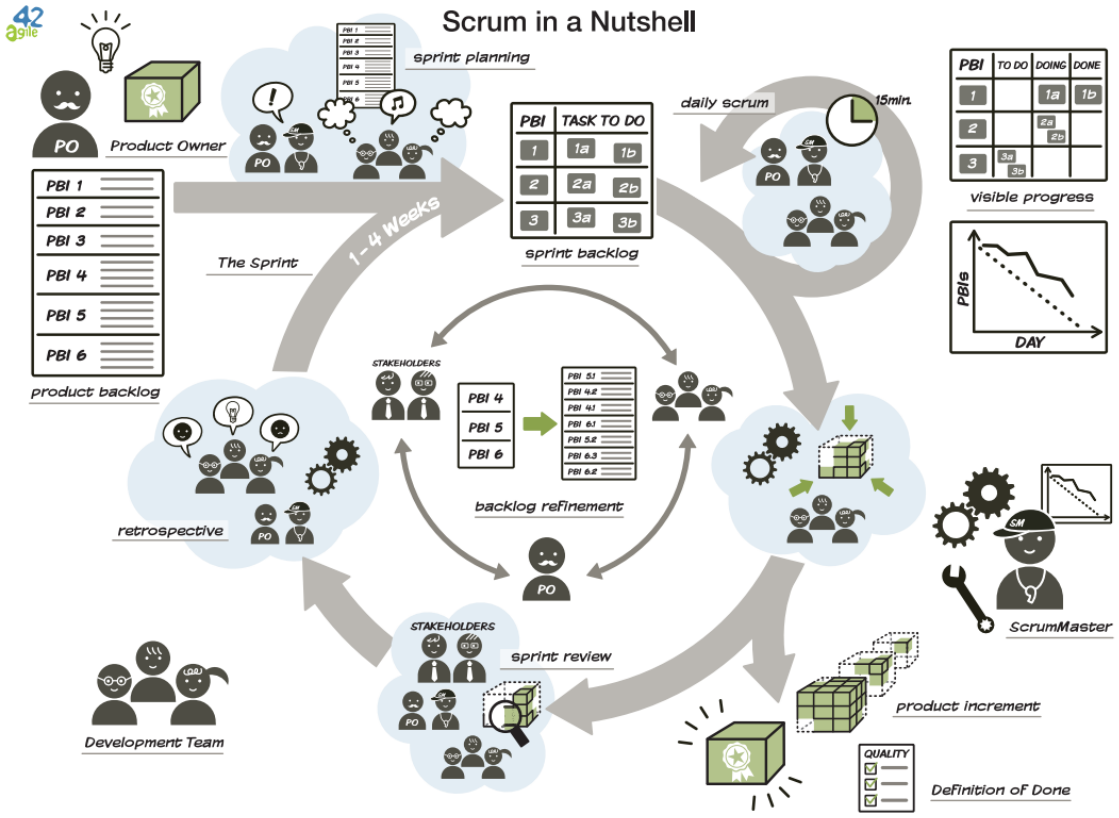


Figura 5: Representação gráfica de um projeto desenvolvido utilizando *Scrum* [5].

Cada *sprint* deve conter os seguintes eventos:

- *Sprint Planning* - reunião realizada no início do *sprint*, na qual estão presentes todos os intervenientes do projeto. É constituído o *Sprint Backlog*[2], tendo em conta que a capacidade de execução da equipa. Isto corresponde a tirar a parte do topo do *Product Backlog*[3] que, estando priorizado, corresponde a iniciar os desenvolvimentos das funcionalidades mais relevantes do projeto naquele momento.

- *Daily Scrum* - todos os dias a equipa é reunida, durante cerca de 15 minutos, para que cada membro explique o tem estado a desenvolver, os próximos passos, e se existe algum problema que não o permita cumprir o seu objetivo.

- *Sprint Review* - reunião que ocorre no final do *sprint*, com o objetivo de rever o trabalho desenvolvido e o que ficou por realizar. É também entregue aos *stakeholders* o produto obtido após o *sprint*, para que o resultado final possa ser avaliado.

2 Conjunto de *User Stories* que devem ser desenvolvidas durante o *sprint*.
3 Conjunto total de *User Stories* que devem ser desenvolvidas durante os trabalhos de desenvolvimento.

- *Sprint Retrospective* - tal como o *Sprint Review*, é realizada no final do *sprint* onde é feito o levantamento das lições aprendidas durante o *sprint*, de forma a aplicar as mesmas nos *sprints* seguintes [73].

Outra metodologia bastante popular e simples de aplicar é *Kanban*. A metodologia *Kanban* promove a utilização de um *kanban board* (ver Figura 6) para que seja possível visualizar de forma simples o *workflow* do projeto. Comparativamente ao *Scrum*, este método não fornece uma estruturação temporal em termos de processo de desenvolvimento, assim como não existem papéis e responsabilidades. No entanto, permite que sejam realizadas entregas e alterações em qualquer fase do projeto [74, 75].

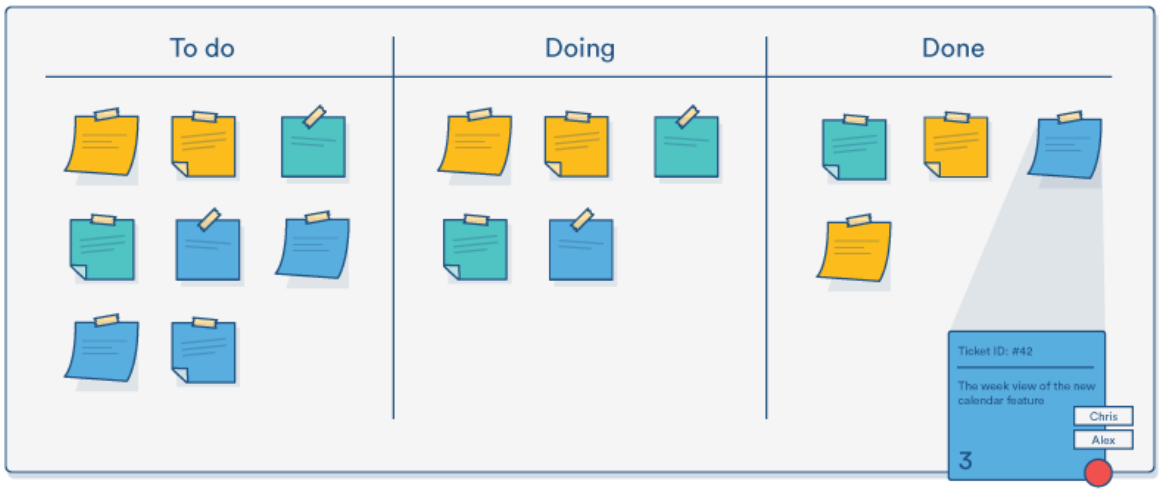


Figura 6: Exemplo simples de um *Kanban board* [6].

Kanban baseia-se nos seguintes princípios:

1. *Start with what you do know* - como não é definida uma estrutura detalhada para o desenvolvimento, pode ser utilizado aquilo que a equipa já conhece, não afetando o desempenho dos elementos da equipa.

2. *Agree to pursue incremental, evolutionary change* - devem ser introduzidas pequenas alterações em vez de alterações radicais que afetem o processo a que os trabalhadores estão acostumados.

3. *Respect the current process, roles, responsabilities and titles* - os papéis, responsabilidades e processos aplicados antes da introdução do *kanban* são dotados de valor. As alterações devem ser feitas incrementalmente, dado que alterações radicais podem trazer desconforto à equipa, afetando a sua produtividade.

4. *Encourage acts of leadership at all levels* - o encorajamento de atitudes de liderança entre os membros da equipa ajuda a que cada membro se sinta confortável na partilha de ideias, permitindo a melhoria da metodologia utilizada [6].

São ainda consideradas as seguintes propriedades:

1. *Visualize the workflow* - a criação de uma visão comum das tarefas e atividades a cumprir permite uma gestão mais eficaz do trabalho a realizar por parte dos membros da equipa de desenvolvimento.

2. *Limit Work-In-Progress* - é estabelecido um sistema de desenvolvimento incremental e progressivo. Limitando a quantidade de trabalho em desenvolvimento, é facilitada a visualização do *kanban board* e são evitados problemas com a troca de tarefas e constante priorização.

3. *Manage Flow* - cada tarefa deve ser monitorizada, podendo assim definir-se a rapidez de desenvolvimento e identificar problemas existentes no *workflow*.

4. *Make Process Policies Explicit* - todos os processos devem ser explícitos para os membros da equipa. Assim, é possível que todos os elementos partilhem a sua opinião sobre determinado tema e é facilitada a identificação de problemas em relação à metodologia e processos em uso.

5. *Improve Collaboratively* - utilizando modelos e métodos científicos, podem-se identificar problemas na metodologia e sugerir possíveis soluções. É permitida a comparação de resultados previstos com os obtidos e, face a esta análise, retirar conclusões sobre os processos e ajustar procedimentos [74].

Kanban apresenta vários benefícios: 1) pode ser utilizado por equipas de tamanho variável; 2) permite o replaneamento e a repriorização de tarefas de forma simples, pois as mudanças aplicadas ao trabalho que não está a ser desenvolvido no momento não têm impacto na equipa; 3) o tempo dispensado em cada ciclo é mais reduzido; 4) oferece facilidade na análise de várias métricas, pois podem ser analisadas visualmente; 5) reduz o número de tarefas a ser realizadas ao mesmo tempo, aumentado a eficiência; 6) permite entregas mais rápidas, o que torna mais fácil a integração de novas funcionalidades [6].

Scrum e *Kanban* apresentam os mesmos conceitos, mas seguem abordagens diferentes. A comparação entre os dois métodos é apresentada na Tabela 1.

	Scrum	***Kanban***
Cadência	Normalmente os *sprints* têm duração fixa	Fluxo contínuo
Entregas	Feitas no fim de cada *sprint*, se aprovadas pelo *Product Owner*	Entrega contínua ou à discrição da equipa
Papéis	*Product Owner*, *Scrum Master*, Equipa de Desenvolvimento	Sem papéis definidos
Métricas chave	Velocidade	Tempo de ciclo
Mudanças e alterações	A equipa deve evitar mudanças aos trabalhos do *sprint* durante o mesmo	Podem acontecer a qualquer momento

Tabela 1: Análise comparativa entre *Scrum* e *Kanban*.

É comum que as equipas juntem as duas metodologias, à qual se dá o nome de *"Scrumban"*, aproveitando os papéis e o tempo fixo dos *sprints* definidos pelo *Scrum* e o foco no trabalho em progresso e o tempo de ciclo do *Kanban*.

eXtreme Programming (XP) está mais focado no como desenvolver software enquanto que *Scrum* está mais focado na gestão de projetos de software. O foco principal de XP é organizar a equipa de forma a produzir software de maior qualidade, de forma mais eficiente. XP defende que se deve apenas desenvolver e entregar o que foi pedido. Deste modo, existe espaço para melhorias que acrescentem qualidade e valor ao produto, depois de todas as funcionalidades propostas estarem desenvolvidas e testadas.

Existem 4 entidades distintas:

1. *Developer* - o programador, quem desenvolve o software;
2. *Customer* - o cliente do projeto;
3. *Manager* - responsável pelo planeamento do projeto, monitorização do desenvolvimento, identificação de problemas, entre outras atividades, podendo ser considerado um gestor de desenvolvimento;
4. *Coach* - equivalente a um *Scrum Master*, deve garantir que os princípios e valores da metodologia são respeitados [76].

Esta metodologia define 4 atividades base, 5 valores e 29 regras. As 4 atividades base são:

1. *Coding* - é considerada a atividade vital para o desenvolvimento. Estabelece que entregável de maior importância é o produto de software;

2. *Testing* - é realçada a importância da realização de testes unitários e de integração, para garantir a qualidade técnica da solução, e de testes de qualidade e satisfação perante os *stakeholders*, para verificar se as suas expectativas e necessidades são cumpridas;

3. *Listening* - a comunicação com os clientes e utilizadores do produto final é crucial, pois só desta forma se percebem as suas necessidades. A equipa deve-se mostrar disponível para esclarecer questões técnicas, em linguagem familiar para os os clientes e utilizadores;

4. *Designing* - sem esta atividade, a probabilidade de se encontrem problemas durante o desenvolvimento é maior. A necessidade de conceção de soluções está diretamente relacionada com a complexidade do problema. Assim, quanto mais complexo o projeto, maior a vantagem em realizar um planeamento e estudo inicial [77].

XP melhora um projeto de software promovendo 5 valores essenciais:

1. *Commnunication* - a comunicação frequente permite a troca de conhecimentos e a entre-ajuda entre os colegas da equipa. A comunicação frequente com o clientes e utilizadores promove maior empatia para com as suas necessidades e o desenvolvimento de soluções adequadas;

2. *Simplicity* - os esforços devem incidir apenas no que é pedido. Isto aumenta a rapidez do desenvolvimento e utiliza os recursos do projeto de forma mais eficiente;

3. *Feedback* - receber e/ou fornecer *feedback* a cada iteração é importante para entender que alterações devem ser realizadas, de forma a construir uma solução apropriada às necessidades dos clientes;

4. *Respect* - todos os elementos devem ser tratados com o devido respeito e o papel de cada um deve ser valorizado. Tal contribui para aumento da motivação da equipa e, consequentemente, maior produtividade e melhor desempenho;

5. *Courage* - todos falham, pelo que as falhas devem ser vistas como algo normal e que não devem ser escondido do resto da equipa. Este reconhecimento permite que a equipa se sinta mais confiante na procura de ajuda e comunicação de problemas [78].

São ainda definidas 29 regras a seguir que estabelecem o processo de desenvolvimento: *Planning*, *Managing*, *Designing*, *Coding* e *Testing*, que respeitam tanto o definido nas atividades, como os valores propostos [79].

3.3 Análise Comparativa entre Abordagens

Os métodos ágeis são métodos adaptativos e os métodos tradicionais são baseados em métodos preditivos. Enquanto que os métodos tradicionais têm maior foco no planeamento, os métodos

ágeis requerem apenas os requisitos básicos para dar início ao projeto. Os métodos ágeis aceitam alterações em qualquer estágio de desenvolvimento, ao contrário dos métodos tradicionais, que requerem um processo de controlo de mudanças rigoroso uma vez que o desenvolvimento do software tenha sido iniciado. Os métodos ágeis reduzem significativamente a documentação e aumentam o número de interações com o cliente, fazendo com que o projeto seja um esforço colaborativo, aumentando assim a satisfação do mesmo.

Embora os métodos ágeis sejam adequados para projetos de pequenas e médias dimensões, ou onde as mudanças contínuas são necessárias, os métodos tradicionais ainda são muito utilizados para o desenvolvimento de software em grandes empresas, onde os requisitos e especificações devem ser claros para se iniciar o projeto.

Nas metodologias tradicionais, apesar do controlo de qualidade ser realizado frequentemente, é necessário aguardar o desenvolvimento do produto para executar um teste. Os métodos ágeis permitem testes após cada *sprint*.

Em conclusão, selecionar a abordagem certa para um determinado projeto de desenvolvimento de software depende dos requisitos. A abordagem depende de vários fatores, tais como: tamanho do projeto, riscos, clareza dos requisitos, complexidade, disponibilidade do cliente e da localização da equipa. Selecionar a abordagem certa levará a uma implementação e conclusão do projeto bem-sucedidas.

3.4 Engenharia de Requisitos

Os engenheiros de software deparam-se muitas vezes com problemas de grande complexidade. Compreender a sua natureza pode ser muito difícil, sendo que a dificuldade aumenta quando o problema é inovador [80]. Assim, definir com exatidão o que o software deve ser capaz de realizar é uma atividade complexa. Tal como referido por Brooks,

> *Adjusting to the requirement for perfection is, I think, the most difficult part of learning to program* [69].

Ainda de acordo com o mesmo autor, a parte mais difícil do desenvolvimento de software é decidir exatamente o que será desenvolvido. Nenhuma outra atividade é tão difícil como o estabelecimento dos detalhes técnicos necessários, incluindo interfaces para máquinas, pessoas e outros sistemas. Nenhuma outra atividade pode provocar tantos erros no software e é tão difícil de ser posteriormente corrigida [69]. A própria natureza humana percebe melhor o problema quando é iniciado o processo de desenvolvimento.

> *For the human makers of things, the incompletenesses and inconsistencies of our ideas become clear only during implementation* [69].

Os requisitos podem ser definidos como qualquer característica que o cliente deseje. Estes são normalmente características do produto, tais como especificações técnicas, prazos para as

entregas, garantia, isto é, o que o cliente espera do produto final. No contexto do desenvolvimento de sistemas, os requisitos são vistos como propriedades que o sistema deve possuir depois de construído. Os requisitos expressam as necessidades básicas do cliente, assim como as restrições que devem ser consideradas durante o desenvolvimento [43].

Apesar de existirem imensas definições, nesta dissertação considera-se a definição de requisito de acordo com a definição do IEEE:

> *1. a condition or a capacity that someone needs to solve a problem or to achieve an objective;*
> *2. a condition or a capacity that must be verified or possessed by a system or by a system component to satisfy a contract, standard, specification, or other formally imposed documents;*
> *3. a documented representation of a condition or capacity, the in (1) or (2).* [20]

O fluxo de requisitos reúne as atividades que visam a obter a definição precisa, clara e completa dos requisitos de um produto de software. Os requisitos devem ser levantados pela equipa responsável pelo desenvolvimento do software, em conjunto com o cliente, utilizadores do software e outros especialistas da área [81].

Os requisitos são normalmente classificados como funcionais ou não funcionais. Requisitos funcionais indicam funcionalidades que o software deve fornecer, como este deve reagir de acordo com determinados *inputs* e como se deve comportar em determinadas situações. Em certos casos, os requisitos funcionais podem também declarar o que o software não deve fazer. Os requisitos não funcionais são restrições sobre os serviços ou funções oferecidas pelo software. Entre eles destacam-se restrições a nível temporal, sobre o processo de desenvolvimento, *standards* e normas, entre outros [80, 43].

Assim, o processo de descobrir, analisar, documentar e verificar as funcionalidades e restrições do software, é chamado de engenharia de requisitos [80]. Esta é uma sub-área da engenharia de software (ver Secção 1.2), cujo objetivo é tratar o processo de definição dos requisitos de software. Para isso, é estabelecido um processo pelo qual o que deve ser feito é elicitado, modelado e analisado. Este processo lida com diferentes pontos de vista, visto que parte dos requisitos são impostos pelos *stakeholders*, e usa uma combinação de métodos, ferramentas e pessoal. Como resultado deste processo tem-se um modelo, a partir do qual se produz o documento com a definição dos requisitos [82].

A engenharia de requisitos pode ser descrita em 6 passos distintos: 1) Elicitação de requisitos, 2) Análise e negociação de requisitos, 3) Especificação de requisitos, 4) Modelação do sistema, 5) Validação de requisitos e 6) Gestão de requisitos [83].

A implementação da engenharia de requisitos é fator crítico para o sucesso dos projetos de software, dada a sua relevância e confiança que traz ao produto de software desenvolvido. Com a sua implementação, aumenta-se a probabilidade do software atender às necessidades dos clientes [43].

Existem já vários *standards* publicados no âmbito da engenharia de requisitos. O SWEBoK dispõe todo um capítulo sobre a área, no qual são abordados processos como elicitação de requisitos, validação de requisitos e análise de requisitos [7]. Os tópicos abordados neste capítulo são apresentados na Figura 7.

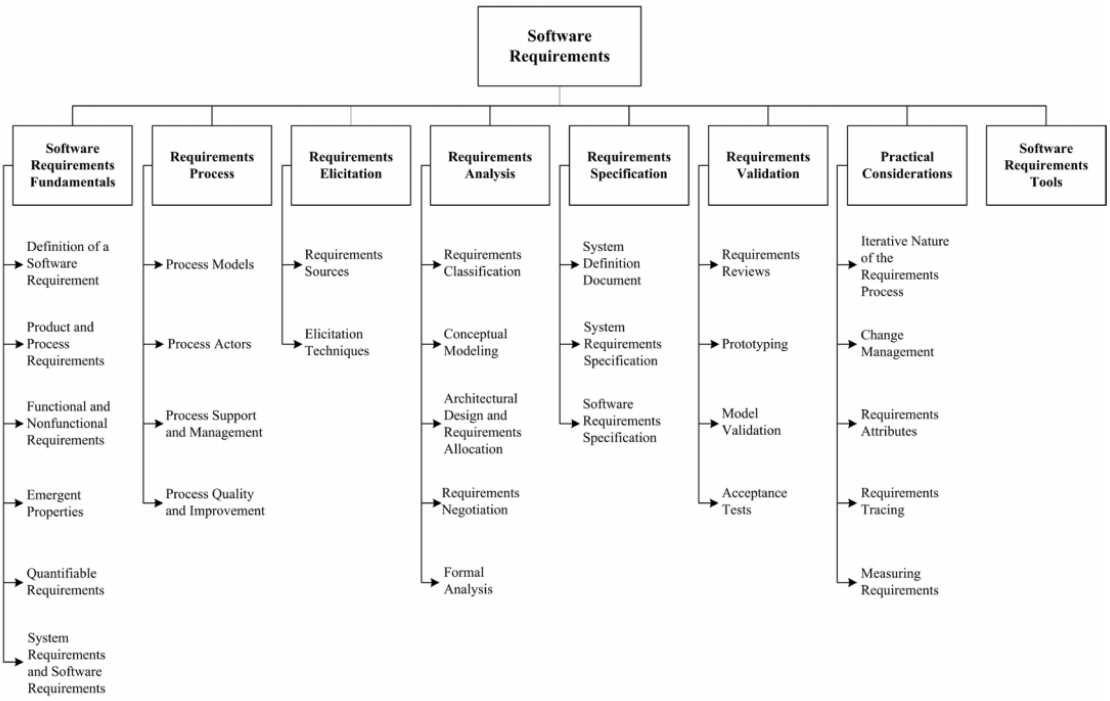


Figura 7: *Breakdown* dos tópicos da Engenharia de Requisitos [7].

A ISO tem também vários trabalhos neste domínio, entre os quais se realça o *ISO/IEC/IEEE 29148 Systems and Software Engineering - Life Cycle Processes - Requirements Engineering*. Este especifica os processos necessários que serão implementados para engenharia de requisitos para sistemas e produtos de software (incluindo serviços) ao longo do ciclo de vida, fornece diretrizes para a aplicação dos requisitos e processos relacionados aos requisitos descritos na ISO/IEC 12207:2008 [84] e ISO/IEC 15288:2008 [85], especifica os itens de informação necessários que devem ser produzidos através da implementação dos processos de requisitos, especifica o conteúdo exigido dos itens de informações exigidos e fornece diretrizes para o formato dos itens de informação necessários e relacionados [86].

3.5 Gestão de Risco

Um risco afeta acontecimentos futuros. Acontecimentos passados ou presentes não devem constituir uma preocupação, pois os resultados obtidos hoje já foram influenciados por ações anteriores. A verdadeira questão é, mudando as nossas ações hoje, podemos criar oportunida-

des para situações diferentes, e idealmente melhores, amanhã? Isto significa que risco envolve mudança, que pode ser a nível de pensamento, opinião, ou ações. O risco envolve tomada de decisão, e a incerteza associada à mesma [87].

O PMI [11] define risco como

> *[...] an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives.*

Riscos que podem afetar negativamente o projeto são denominados ameaças, enquanto que riscos que podem causar efeitos positivos no projeto não denominados oportunidades [11].

De acordo com a expressão anterior, podemos caracterizar o risco em duas dimensões diferentes: Impacto (I) e Probabilidade (P) de ocorrência. É possível quantificar um risco utilizando a seguinte expressão:

$$Risco = I \times P \tag{1}$$

Um risco 100% provável de acontecer é considerado uma restrição para o projeto.

A gestão de riscos inclui os processos de planeamento da gestão de riscos, identificação, análise, planeamento e implementação de respostas e monitorização de riscos no projeto. Os objetivos da gestão de risco são aumentar a probabilidade e/ou o impacto dos riscos positivos e diminuir a probabilidade e/ou impacto dos riscos negativos, de forma a otimizar a probabilidade de sucesso do projeto [11].

Os riscos podem ameaçar o projeto, o software em desenvolvimento ou a própria organização. Sommerville [80] define as seguintes categorias de riscos:

- Riscos relacionados ao Projeto - afetam a programação ou os recursos do projeto.
- Riscos relacionados ao Produto - afetam a qualidade ou o desempenho do software em desenvolvimento.
- Riscos para os negócios - afetam a organização que desenvolve ou adquire o software.

Quando o risco é considerado no domínio da engenharia de software, 3 conceitos devem ser evidenciados:

- Quais os riscos podem causar o insucesso do projeto?
- Como é que as mudanças nos requisitos afetam o sucesso em geral?
- Que métodos e ferramentas devemos usar, quantas pessoas devem ser envolvidas, quanta ênfase em qualidade é suficiente? [83]

Sommerville [80] descreve ainda os tipos de riscos que podem afetar o projeto e o ambiente organizacional em que o software é desenvolvido. No entanto, vários riscos são considerados universais e envolvem as seguintes áreas: tecnologia, pessoal, organizacional, ferramentas,

requisitos e estimativa. Entre os fatores de riscos que devem ser considerados podem ser incluídos os fatores legais, tecnológicos, tamanho e complexidade do produto, relativos a RH e aceitação pelos utilizadores.

Durante as várias fases do ciclo de vida do projeto, os riscos podem sofrer alterações. Alguns riscos permanecem constantes, outros desaparecem devido a fatores externos ou pois apenas podem afetar o projeto durante determinada fase, e outros sofrem alterações na probabilidade de ocorrência e impacto, mudando, consequentemente, de prioridade. Assim, é necessário monitorizar os riscos e planear as respostas aos mesmos para agir da forma mais adequada [88].

As seguintes questões foram derivadas de dados de riscos obtidos por levantamento feito com gestores de projeto de software experientes [89]. Estas estão ordenadas em relação à sua importância para o sucesso do projeto:

1. A administração do software e do cliente empenhou-se formalmente em apoiar o projeto?
2. Os utilizadores estão empenhados com relação ao projeto?
3. Os requisitos estão plenamente entendidos?
4. Os clientes envolveram-se totalmente na especificação dos requisitos?
5. Os utilizadores têm expectativas realistas?
6. O âmbito do projeto é estável?
7. A equipa de desenvolvimento tem a combinação de aptidões adequadas?
8. Os requisitos do projeto são estáveis?
9. A equipa de desenvolvimento tem experiência com a tecnologia a ser implementada?
10. A quantidade de pessoal é adequada ao projeto?
11. Todos os membros da equipa e utilizadores envolvidos no projeto concordam com a importância do projeto e com os requisitos do sistema?

Em caso de resposta negativa a qualquer uma das questões acima apresentadas, a monitorização e gestão devem ser instituídas imediatamente. O grau de risco do projeto é diretamente proporcional ao número de respostas negativas a estas questões [89].

Existem já vários esforços e trabalhos sobre a gestão de risco. O SWEBoK [7] apresenta parte de um capítulo totalmente dedicada ao risco e incerteza. São apresentados assuntos como objetivos, estimativas e planos, como priorizar riscos, e como tomar decisões afetas pela incerteza.

O ISO/IEC 16085:2006 *Systems and software engineering - Life cycle processes - Risk management* [90] define um processo para a gestão do risco durante o ciclo de vida. Pode ser

adicionado ao conjunto de processos do ciclo de vida do software definidos por outras normas, como o ISO/IEC 12207 [84] ou ISO/IEC 15288 [85], ou utilizado independentemente. Pode ser aplicado tanto a sistemas como a software.

3.6 GESTÃO DA CONFIGURAÇÃO

A gestão da configuração é a disciplina responsável pela identificação da configuração de um sistema em pontos distintos no tempo, com o objetivo de controlar sistematicamente as mudanças na sua configuração, e manter a integridade e rastreabilidade da mesma ao longo do ciclo de vida do sistema [7]. A definição formal de gestão da configuração, escrita pelo IEEE é a seguinte:

> *A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a confguration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specifed requirements* [20].

O SWEBoK apresenta um capítulo sobre gestão da configuração, no qual são abordadas questões sobre controlo da configuração do software, identificação da configuração do software, gestão do processo de gestão de configuração, entre outros [7]. A Figura 8 apresenta os tópicos abordados neste capítulo.

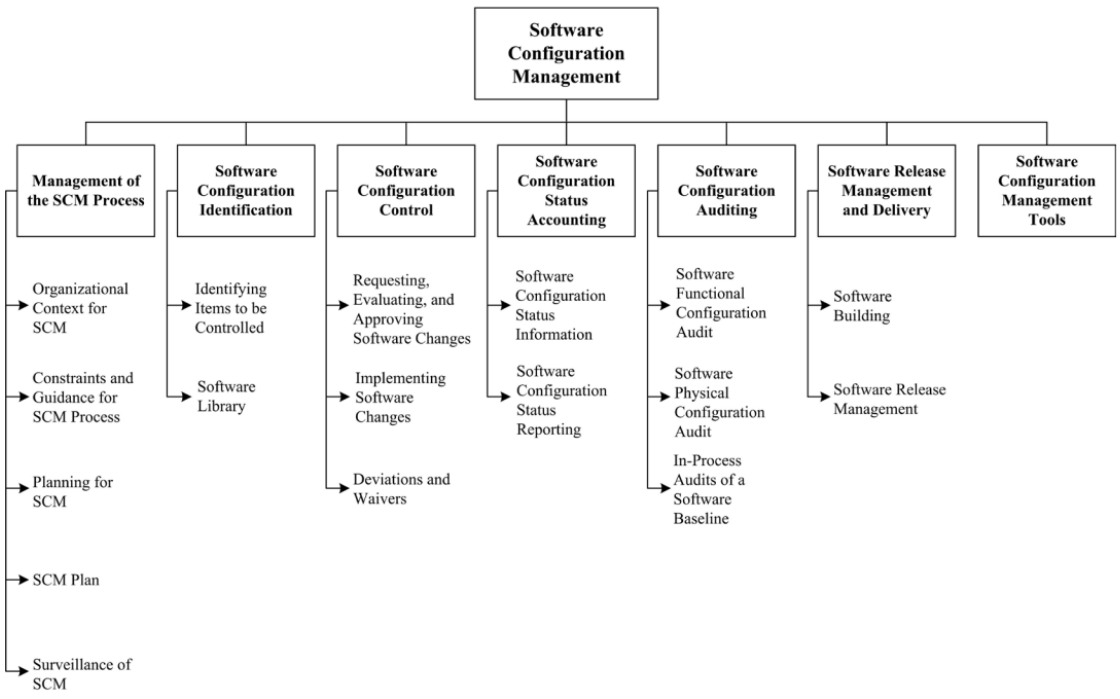


Figura 8: *Breakdown* dos tópicos da Gestão da Configuração [7].

De acordo com Sommerville [80], a gestão da configuração é o desenvolvimento e aplicação de padrões e procedimentos para gerir um produto de software em desenvolvimento. É necessário gerir os sistemas em desenvolvimento pois, à medida que estes são desenvolvidos, são criadas muitas versões diferentes de software. As novas versões do software apresentam propostas de mudanças, correções de *defeitos* e adaptação para diferentes plataformas de hardware e sistemas operativos. É possível que existam várias versões a serem desenvolvidas ou utilizadas ao mesmo tempo. Assim, é necessário manter o controlo sobre as mudanças que foram implementadas e como estas foram incluídas no software.

O processo de gestão da configuração aplica processos administrativos e técnicos durante todo o ciclo de vida do software. Os seus objetivos são identificar e definir os itens de software no sistema e estabelecer suas linhas básicas (*baselines*); controlar as modificações desses itens; registar e apresentar a situação dos itens e dos pedidos de modificação; garantir a conclusão, a consistência e a correção dos itens; e controlar o armazenamento, a manipulação e a distribuição dos itens [84].

Na gestão da configuração do software, não deve apenas ser monitorizado o código fonte, mas também outros artefactos relevantes, como a documentação e modelos do software. Ao realizar o controlo de versões, é também possível reaproveitar código, evitando desperdícios.

No entanto, a gestão da configuração do software implica a adoção de procedimentos que afetam todos os setores de uma organização, dado ser um processo complexo que envolve aspetos técnicos, de gestão e também culturais. O seu sucesso é mais dependente de fatores culturais do que dos fatores técnicos e de gestão. Apesar de apresentar conceitos simples de serem entendidos, estes são difíceis de colocar em prática [80].

Existem trabalhos, guias e *standards* publicados na área da gestão da configuração de software. Destacam-se o IEEE 828 [91] e ISO/IEC TR 18018 [92].

O IEEE 828 (*IEEE Standard for Configuration Management in Systems and Software Engineering*) [91] estabelece os requisitos mínimos para processos de gestão de configuração na engenharia de software e de sistemas. A aplicação desta norma aplica-se a qualquer forma, classe ou tipo de software ou sistema. A versão mais recente expande a versão anterior para explicar melhor a gestão da configuração, incluindo a identificação e aquisição de itens de configuração, controlo de alterações, relatórios sobre o status de itens de configuração, bem como versões de software. São abordadas quais as atividades que devem ser executadas, quando ocorrem durante o ciclo de vida, e quais os planos e recursos necessários. Descreve também os conteúdos de um plano de gestão da configuração. O padrão suporta ISO/IEC/IEEE 12207:2008 [84] e ISO/IEC/IEEE 15288:2008 [85] e adere à terminologia do ISO/IEC/IEEE 24765 [93] e aos requisitos de item de informação do IEEE 15939 [94].

O ISO/IEC TR 18018:2010 (*Information technology - Systems and software engineering - Guide for configuration management tool capabilities*) [92] fornece um guia para seleção e avaliação de ferramentas de gestão de configuração. A avaliação da ferramenta de gestão da

configuração pelos utilizadores pode ser complexa, demorada e cara. A ISO/IEC TR 18018: 2010 ajuda a caracterizar o que uma ferramenta de gestão da configuração pode e não pode fazer no processo de gestão de configuração. Fornece também orientação para os fabricantes das ferramentas na implementação de um conjunto mínimo de recursos. Os recursos definidos estão vinculados à ISO/IEC 12207:2008 [84] e à ISO/IEC 15288: 2008 [85] e fornecem aos fabricantes de ferramentas orientações sobre as características que suas ferramentas devem suportar para atender a essas normas.

3.7 NORMAS E MODELOS DA QUALIDADE

Tal como referido anteriormente, um projeto pode ser entregue dentro do prazo estabelecido e do orçamento estipulado, cumprindo todo o âmbito proposto e mesmo assim não ser do agrado do cliente. Brooks mostra a importância da qualidade de uma forma simples:

> *An omelette, promised in two minutes, may appear to be progressing nicely. But when it has not set in two minutes, the customer has two choices — wait or eat it raw. Software customers have had the same choices. The cook has another choice; he can turn up the heat. The result is often an omelette nothing can save — burned in one part, raw in another* [69].

Quando nos referimos a qualidade do software, podemos referir-nos às características desejáveis dos produtos de software, na medida em que um produto de software específico possui essas características, e aos processos, ferramentas e técnicas utilizadas para atingir essas características [7].

O SWEBoK apresenta um capítulo sobre qualidade do software, para além de incluir este tema em várias áreas de conhecimento. Os tópicos abordados neste capítulo são apresentados na Figura 9.

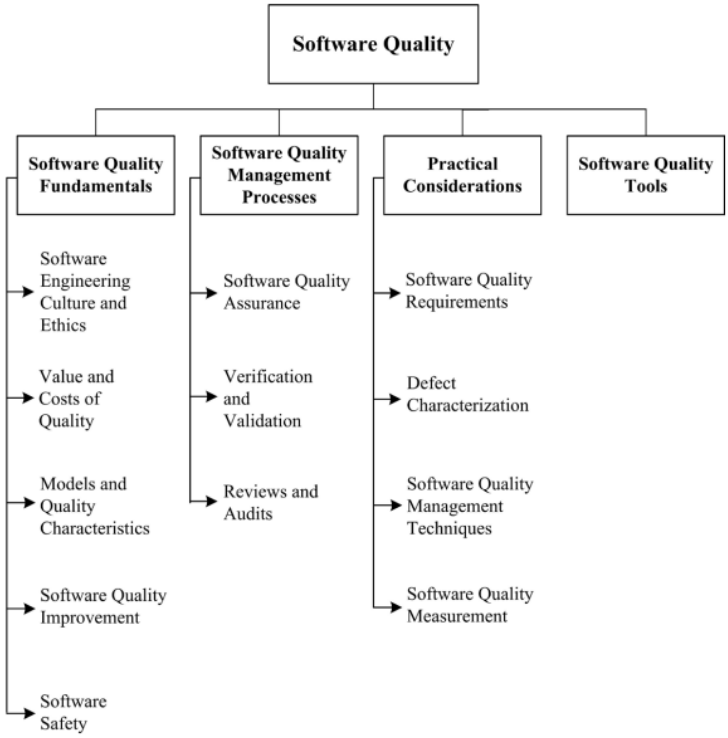


Figura 9: *Breakdown* dos tópicos da Qualidade do Software [7].

Medições de qualidade de software são usadas para apoiar a tomada de decisões. Com a crescente sofisticação e complexidade do software, questões sobre a qualidade vão além de saber se o software funciona ou até o ponto em que atinge objetivos de qualidade mensuráveis [7]. Mais adiante serão apresentadas técnicas de medição da qualidade do software.

Existem inúmeras normas e modelos de qualidade de software. Neste documento serão apenas abordadas as que se consideram de maior relevância: SPICE (ISO 15504), *System and Software Quality Requirements and Evaluation* (SQuaRE) (ISO 25000), e ISO 9000.

A família ISO 9000 é uma série de *standards* internacionais sobre gestão e garantia da qualidade, desenvolvidos para ajudar as organizações a garantir que atingem as necessidades dos clientes e dos *stakeholders*, enquanto cumprem com os requisitos legais e regulamentares. O ISO 9000 (ISO 9000:2015 *Quality management systems - Fundamentals and vocabulary*) trata dos sistemas de gestão da qualidade, incluindo os 7 princípios de gestão de qualidade:

1. Foco no cliente: as organizações dependem dos seus clientes e, portanto, devem entender as necessidades atuais e futuras dos mesmos, atender aos seus requisitos e se esforçarem-se para exceder as suas expetativas.

2. Liderança: os líderes estabelecem a direção da organização. Eles devem criar e manter o ambiente interno, no qual as pessoas podem se envolver plenamente na conquista dos objetivos da organização.

3. Envolvimento dos colaboradores: pessoas de todos os níveis da hierarquia da organização são a essência da mesma, e o seu envolvimento total permite que suas capacidades sejam utilizadas para benefício da organização.

4. Abordagem do processo: um resultado desejado é alcançado de forma mais eficiente quando as atividades e recursos relacionados são geridos como um processo.

5. Melhoria: a melhoria do desempenho geral da organização deve ser um objetivo permanente da mesma.

6. Tomada de decisão baseada em evidências: as decisões efetivas são baseadas na análise de dados e informações.

7. Gestão das relações: uma organização e os seus fornecedores externos (fornecedores, colaboradores, prestadores de serviços, etc) são interdependentes, pelo que um relacionamento mutuamente benéfico aumenta a capacidade de ambos criarem valor [64].

São reconhecidas 4 categorias genéricas de produtos. Para cada uma delas, foram publicadas diretrizes para implementação de sistemas da qualidade: hardware (ISO 9004-1), serviços (ISO 9004-2), materiais processados (ISO 9004-3), e software (ISO 9000-3).

Dadas as dificuldades na implementação dos requisitos do ISO 9001 em software, é fundamental o uso da ISO 9000-3 para auxílio à implementação do sistema de gestão da qualidade [95]. Este *standard* não adiciona ou altera os requisitos da ISO 9001; apenas reescreve a mesma de forma a ser mais facilmente percebida pelas equipas que desenvolvem software.

A família de *standards* ISO/IEC 25000, também conhecida como SQuaRE, tem como objetivo a criação de uma *framework* para avaliação da qualidade dos produtos de software [8]. Esta família é o resultado da evolução natural de vários outros *standards*, podendo-se realçar o ISO/IEC 9126, que define um modelo de qualidade para avaliação de produtos de software, e ISO/IEC 14598, que define o processo de avaliação do produto de software.

A SQuaRE consiste em 5 divisões: 1) ISO/IEC 2500n – *Quality Management Division*, 2) ISO/IEC 2501n – *Quality Model Division*, 3) ISO/IEC 2502n – *Quality Measurement Division*, 4) ISO/IEC 2503n – *Quality Requirements Division* e 5) ISO/IEC 2504n – *Quality Evaluation Division*.

Dentro desta família realça-se o ISO/IEC 25010, que apresenta uma *framework* de qualidade do software. A qualidade do software é dividida em duas categorias: 1) *Quality in Use* (qualidade da utilização) e 2) *System/Software Product Quality* (qualidade do produto).

A qualidade do produto está relacionada às propriedades estáticas e dinâmicas do software propriamente dito. Está dividido em 8 características: 1) *Functional Suitability* (adequação funcional), 2) *Performance Efficiency* (eficiência do desempenho), 3) *Compatibility* (compatibilidade), 4) *Usbaility* (usabilidade), 5) *Reliability* (confiança), 6) Security (segurança), 7) *Maintainability* (manutenção), e 8) *Portability* (portabilidade).

A qualidade na utilização está relacionada ao resultado da interação humana com o software. Está dividida em 5 características: 1) *Effectiveness* (eficácia), 2) *Efficiency* (eficiência), 3) *Satisfaction* (satisfação), 4) *Freedom fromn risk* (livre de riscos), e 5) *Context coverage* (cobertura de contexto).

As características listadas acima são apresentadas em alto nível, mas a ISO/IEC 25010 aumenta o nível de definição, especificando um conjunto de sub-características. Estas são apresentadas nas Figuras 10 e 11.

Figura 10: *Quality in Use Model* [8].

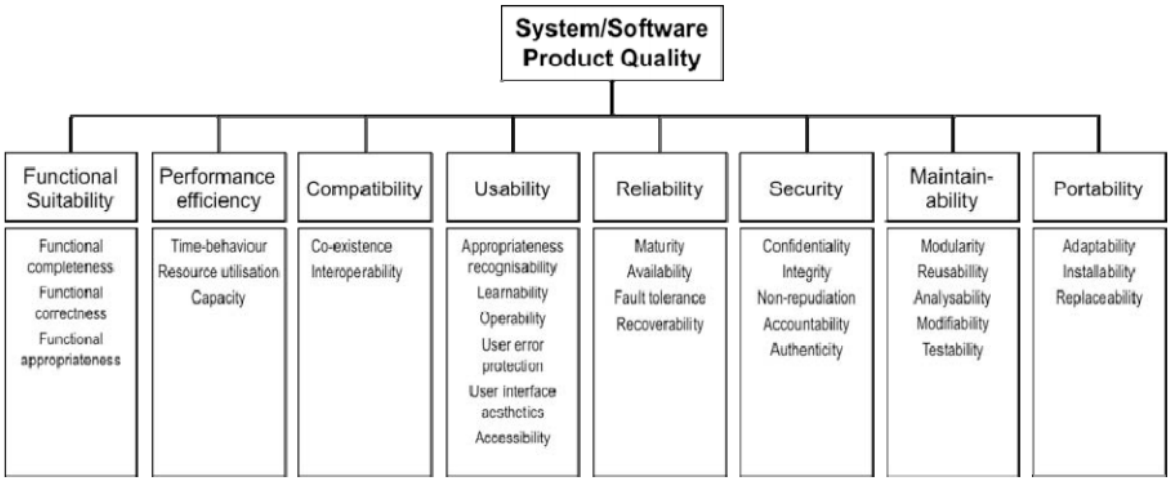


Figura 11: *System/Software Product Quality Model* [8].

A ISO/IEC 25010 é uma ótima adição para equipas responsáveis pelo o desenvolvimento de software, que desejam uma *framework* para definir o software. Ao decompor as características de qualidade em sub-características, os *developers* podem definir métricas de software que façam sentido para o projeto. No entanto, não fornece um mapeamento de sub-características para métricas de software. O contexto individual é muito importante: a qualidade do código no contexto de um *site* de notícias *on-line* é uma coisa, enquanto que a qualidade do código para um software de negociação automatizado de alta frequência tem um leque completamente diferente de consequências. A qualidade do código do software embebido em aviões pode ser uma questão de vida ou morte. Em suma, as apostas em cada um dos projetos de desenvolvimento são diferentes e exigem diferentes prioridades, métricas e planos de contingência [96].

A ISO/IEC 15504, também conhecida como *Software Process Improvement and Capability dEtermination* (SPICE), presta-se à realização de avaliações de processos de software com os objetivos de melhoria dos processos e a determinação da capacidade de processos de uma organização. SPICE baseia-se em modelos existentes, como o CMMI, tendo entre os objetivos

básicos criar harmonia entre modelos existentes e fornecer uma *framework* para estes e outros que possam ser desenvolvidos [97].

Este modelo descreve um conjunto de processos e boas práticas de engenharia de software, tidos como universais e fundamentais. Existem 5 categorias de processos: Cliente-fornecedor, Engenharia, Suporte, Gestão e Organização. Estes processos são um conjunto dos processos definidos no *standard* ISO/IEC 12207. [84]

Para cada processo, o SPICE define um nível de capacidade (*capability level*) na seguinte escala: 0 - Processo incompleto, 1 - Processo realizado, 2 - Processo gerido, 3 - Processo estabelecido, 4 - Processo previsível e 5 - Processo otimizado.

A capacidade dos processo é medida utilizando atributos de processos. O *standard* define 9 atributos de processos: 1.1) Desempenho do processo, 2.1) Gestão do desempenho, 2.2) Gestão de produtos do trabalho, 3.1) Definição do processo, 3.2) Implementação do processo, 4.1) Medição do processo, 4.2) Controlo do processo, 5.1) Inovação do processo e 5.2) Otimização do processo. Cada atributo do processo consiste em uma ou mais práticas genéricas, posteriormente elaboradas em indicadores, para auxilio à avaliação do desempenho [97].

Cada atributo do processo é avaliado em uma escala de classificação de quatro pontos (N-P-L-F):

N. *Not achieved* (Não alcançado) - 0 - 15%

P. *Partially achieved* (Parcialmente atingido) - 16% - 50%

L. *Largely achieved* (Em grande parte alcançado) - 51% - 85%

F. *Fully achieved* (Totalmente alcançado) - 86% - 100% [97]

SPICE fornece também um guia para realizar uma auditoria, o que inclui o processo, o modelo e as ferramentas utilizadas para auditar. O *standard* pode ser utilizado em 2 contextos diferentes: na melhoria de processos e para determinação de capacidade (ou seja, avaliação da capacidade de processamento do fornecedor).

3.8 Medição e estimativas

Saber o que medir e que método utilizar para efetuar medições e estimativas, é crítico para as atividades de engenharia. É de elevada importância que todos os *stakeholders* compreendam os métodos utilizados para as medições e os resultados das mesmas [7].

Estimar ajuda a determinar quantos custos, esforço, recursos e tempo são necessários para construir determinado sistema ou produto. As estimativas pode ser baseadas em experiência e dados passados, documentos e conhecimentos existentes, assunções e riscos identificados. Deve ser estimado o tamanho do produto a desenvolver, o esforço (em pessoa/mês ou pessoa/hora), o tempo necessário para a concretização do ciclo de vida e o orçamento total. Estimar não

deve ser uma atividade única: deve ser repetida durante as fases de Iniciação, Planeamento e Monitorização e Controlo, as vezes necessárias [98].

As medições começam com a conceptualização de conceitos abstratos, passando de seguida para a definição do método de medição e aplicação real desse método para obter um resultado de medição. Cada etapa deve ser bem entendida, comunicada e empregue, para gerar dados utilizáveis. Na engenharia tradicional, medidas diretas são frequentemente utilizadas. Na engenharia de software, é necessária a combinação de medidas diretas e indiretas (derivadas de medidas diretas) [99].

A medição pode ser realizada em quatro escalas diferentes: nominal, ordinal, em intervalos e rácio. A escala nominal apresenta o nível mais baixo de medição e representa a atribuição mais irrestrita de numerais. Os numerais são utilizados apenas como rótulos, sendo que as palavras ou letras serviriam também o mesmo propósito. A escala nominal envolve apenas a classificação e as unidades de amostragem observadas são colocadas em qualquer uma das categorias mutuamente exclusivas e coletivamente exaustivas (classes) [7].

A escala ordinal refere-se à escala em que diferentes valores obtido através do processo de medição têm uma ordenação implícita. Como exemplos de medições efetuadas nesta escala tem-se o nível de capacidade (baixo, médio, alto), os níveis de maturidade do software (CMMI) ou a satisfação do consumidor. No entanto, se medirmos a satisfação do consumidor numa escala de 1 a 5, sendo que 1 significa o nível de satisfação mais baixo e 5 o mais alto, não podemos considerar que o nível 2 significa metade da satisfação em relação ao nível 4. Assim, deve-se utilizar uma escala não numérica e preferir terminologia como excelente, acima da média, médio, abaixo da média e fraco, de forma a evitar o erro comum de se tratar uma escala ordinal como uma nominal [99].

A escala de intervalos, atinge-se uma forma quantitativa, no sentido mais comum da palavra. Quase todas as medidas estatísticas são aplicáveis aqui, a menos que requeiram conhecimento de um verdadeiro ponto zero. O ponto zero, numa escala de intervalos, é uma questão de convenção. As proporções podem não fazer sentido, mas a diferença entre os níveis pode ser calculada e é significativa [7].

As escalas de rácio são geralmente utilizadas nas ciências. Estas são caracterizadas pelo fato de existirem operações para determinar todas as quatro relações: igualdade, ordem de classificação, igualdade de intervalos e igualdade de rácio. Uma vez disponível, os seus valores numéricos podem ser transformados de uma unidade para outra, simplesmente multiplicando por uma constante, por exemplo, conversão de polegadas para centímetros. Quando as medições são feitas numa escala de rácio, a existência de um zero não arbitrário é obrigatório. Todas as medidas estatísticas são aplicáveis à escala de rácio; o uso do logaritmo é válido somente quando essas escalas são usadas, como no caso de decibéis. Como exemplos tem-se o número de *statements* num programa de software ou a temperatura medida em Kelvin ou Fahrenheit [7].

As medições podem ser diretas ou derivadas (indiretas). Como medição direta pode ser dado o exemplo da contagem do númeor de vezes que um evento ocorreu, ou o número de *bugs* no software. Uma medição derivada ou indireta combina medições diretas consistentes. Um exemplo de uma medição indireta é o cálculo da produtividade da equipa, que pode ser calculada dividindo o número de linhas de código produzidas por cada membro da equipa, por mês.

Uma questão que deve ser colocada sobre qualquer método de medição é se este está realmente a medir o conceito com boa qualidade. Confiança e validade são os dois critérios mais importantes para resolver essa questão. A confiança de um método de medição é a medida em que a aplicação do método de medição produz resultados de medição consistentes. Essencialmente, confiança refere-se à consistência dos valores obtidos quando o mesmo item é medido várias vezes. Quando os resultados estão de acordo um com o outro, o método de medição é considerado confiável. Confiança pode ser quantificada usando o índice de variação, que é calculado como a razão entre o desvio padrão e a média. Quanto menor o índice, mais confiáveis serão os resultados da medição. Validade refere-se a se o método de medição realmente mede o que é pretendido medir. A validade de um método de medição pode ser analisada a partir de três perspetivas diferentes: validade do construto, validade de critério e validade de conteúdo [7].

A importância da medição e o seu papel nas práticas de gestão e de engenharia são amplamente reconhecidas. A medição pode ser aplicada a organizações, projetos, processos e produtos. No Apêndice B o foco está na aplicação da medição ao nível dos projetos, processos e produtos, seguindo-se por base a norma ISO/IEC/IEEE 15939 [94]. Esta descreve um processo para definir as atividades e tarefas necessárias para implementar um processo de medição de software, incluindo também um modelo de medição da informação.

Antes de se dar início aos trabalhos de desenvolvimento, é necessário estimar o esforço, tempo e custos associados ao projeto. O esforço estimado necessário para um projeto, ou parte do mesmo, pode ser determinado utilizando um modelo de estimação calibrado com base no tamanho histórico e dados de esforço (quando disponíveis) e outros métodos relevantes, como o julgamento de especialistas e analogia. Podem ser estabelecidas dependências entre tarefas, identificadas tarefas que podem ser concluídas simultaneamente, e documentar as mesmas utilizando, por exemplo, diagramas de Gantt. Em projetos preditivos, o cronograma das tarefas é tipicamente produzido durante o planeamento. Para projetos adaptativos, é geralmente desenvolvida uma estimativa geral do esforço a partir do entendimento inicial dos requisitos [7].

As estimativas de recursos podem ser traduzidas em estimativas de custos. A estimativa inicial de esforço, cronograma e custo é uma atividade iterativa que deve ser negociada e revista junto dos *stakeholders* até que se atinja o consenso sobre os recursos e o tempo disponível para a conclusão do projeto.

Podem também ser feitas medições da qualidade do software para ajudar no processo de tomada de decisão. As decisões suportadas pela medição da qualidade do software incluem a determinação dos níveis de qualidade do software (nomeadamente porque os modelos de qualidade do produto do software incluem medidas para determinar o grau em que o produto do software atinge os objetivos de qualidade); perguntas de gestão sobre esforço, custo e cronograma; determinar quando parar de testar e terminar o desenvolvimento de um produto; e determinar a eficácia dos esforços de melhoria de processos [7].

O custo dos processos de medição da qualidade do software interfere na decisão de como um projeto deve ser organizado. É comum a utilização de modelos genéricos de custo, baseados em quando um defeito é encontrado e quanto esforço leva para corrigir o defeito em relação ao encontrar o defeito no início do processo de desenvolvimento. Apesar de os dados resultantes da medição da qualidade do software possam ser úteis por si só (por exemplo, o número ou proporção de requisitos defeituosos), técnicas matemáticas e gráficas podem ser aplicadas para auxiliar na interpretação dos resultados das medições. Essas técnicas incluem testes e análises estatísticas, análise de tendências e previsões. A partir destas medições, é possível direcionar os esforços para onde os problemas são mais prováveis [7].

A qualidade, para além de acrescentar valor ao projeto, também acrescenta custos. Para ajudar a determinar o nível de qualidade do software, foi criado o *Cost of Software Quality* (CoSQ) [100]. CoSQ é um conjunto de medidas derivadas da avaliação económica dos processos de desenvolvimento e manutenção da qualidade do software, que podem ser usadas para inferir as características de um produto. A premissa subjacente ao CoSQ é de que o nível de qualidade de um produto de software pode ser inferido do custo das atividades relacionadas a lidar com as consequências da má qualidade.

Existem quatro categorias de custo de qualidade: prevenção, avaliação, falha interna e falha externa. Os custos de prevenção incluem investimentos em esforços de melhoria de processos de software, infraestrutura de qualidade, ferramentas de qualidade, treino qualificado, auditorias e revisões de gestão. Esses custos geralmente não são específicos para o projeto; abrangem toda a organização. Os custos de avaliação surgem de atividades de projeto que apresentam defeitos. Essas atividades de avaliação podem ser categorizadas em custos das revisões e custos dos testes; os custos de avaliação seriam estendidos aos fornecedores de software subcontratados. Custos de falhas internas são aqueles que são incorridos em defeitos encontrados durante as atividades de avaliação e descobertos antes da entrega do produto com defeito. Os custos de falha externa incluem atividades para responder a problemas de software descobertos após a entrega ao cliente [100].

Os engenheiros de software devem ser capazes de usar CoSQ para verificar os níveis de qualidade de software e também devem apresentar alternativas de qualidade e seus custos para que as compensações entre custo, cronograma e valor possam ser feitas [7].

Dentro do tema das medições, é também pertinente incluir o EVM, dado que é possível utilizar esta técnica para medições relacionas com o triângulo âmbito, tempo e custos, em qualquer projeto de qualquer indústria. O EVM é uma técnica bastante popular utilizada para avaliar a performance e o progresso de um projeto. As métricas monitorizadas pelo EVM servem como sinais de aviso antecipadas para detetar problemas ou para explorar oportunidades para o projeto, e permitem medir e avaliar a saúde geral de um projeto. São monitorizadas 3 métricas chave: *Planned Value* (PV), *Earned Value* (EV) e *Actual Cost* (AC). PV é o orçamento total atribuído e autorizado para que o projeto possa ser concluído com sucesso. EV são os custos associados ao trabalho que está completo. AC é o custo realizado incorrido pelo trabalho realizado em uma atividade durante um período de tempo específico. EVM monitoriza também as variação à *baseline*, tanto a nível temporal (*Schedule Variance* (SV)) como a nível de custos (*Cost Variance* (CV)). SV e CV podem ser convertidos em indicadores de eficiência que permitem analisar de forma mais simples a performance do projeto em relação ao tempo (*Schedule Performance Index* (SPI)) e custo (*Cost Performance Index* (CPI)) [11].

4

MANAGEMENT OF PLANNED VALUE

Nesta primeira parte, será explicado o processo de definição e seleção das métricas. Posteriormente, será apresentado um conjunto de métricas relevantes para a gestão de projetos de software.

É importante escolher uma metodologia de gestão de projetos para se poder indicar com maior detalhe como a gestão das métricas deve ser feita. Dada a proximidade e facilidade de acesso a equipas que utilizam *Scrum* para gestão de projetos de desenvolvimento de software, e sendo necessária a posterior validação do estudo, optou-se por esta metodologia.

Scrum é um método ágil com fundamento na teoria empírica de controlo de processos. Nesta teoria é afirmado que o conhecimento vem da experiência e tomada de decisões com base no que é conhecido. Nesta teoria são promovidas a transparência, a inspeção e a adaptação [73].

Scrum é amplamente utilizado em equipas de desenvolvimento de software, dada a volatilidade dos requisitos existente em projetos deste tipo. As alterações no código podem ser realizadas e testadas rapidamente, e os erros podem facilmente ser corrigidos. É um método levemente controlado, insistindo na atualização frequente do progresso por meio de reuniões regulares. Assim, há clara visibilidade do desenvolvimento do projeto. Sendo um método ágil, apresenta natureza iterativa, o que permite a monitorização de métricas com mais frequência, e requer *feedback* do cliente, o que permite entregar um produto de maior qualidade e valor. A natureza iterativa e *feedback* constante também permitem que seja mais fácil lidar com as mudanças. As reuniões diárias (*Daily Scrum*) permitem medir a produtividade individual, levando a melhorias na produtividade de cada *developer*. O *Daily Scrum* permite também a identificação de problemas quando estes estão a acontecer ou a previsão dos mesmos. Assim, é possível encontrar soluções para os mesmos e definir planos de contingência com a devida antecedência.

No entanto, *Scrum* apenas deve ser utilizado em projetos com equipas pequenas e com pouco risco associado. Nesta metodologia também não deve ser aplicado um controlo e monitorização muito rigoroso, dada a natureza ágil do mesmo.

Escolhida a metodologia, segue-se para a apresentação de um conjunto de passos a seguir de forma a efetuar uma correta gestão das métricas.

4.1 passos para a gestão das métricas

São apresentados quatros passos para a correta gestão das métricas. O primeiro consiste em definir os critérios de sucesso. O segundo passo consiste em identificar e selecionar as métricas a monitorizar. Depois de selecionadas as métricas, é possível medir o progresso e retirar conclusões provenientes das medições. O último passo consiste em refinar e melhorar a gestão das métricas.

4.1.1 *Definição de sucesso*

O primeiro passo não é perceber o que medir. Antes de se poder fazer isso, é necessário perceber o que realmente traz valor para os *stakeholders*. Nesta fase, deve-se reunir com os *stakeholders* externos aos trabalhos de desenvolvimento e com a equipa de desenvolvimento as vezes necessárias até que o que é realmente importante seja acordado por todas as partes.

Sucesso é relativo a cada projeto: não existe uma definição geral e correta do que é concluir um projeto com sucesso. Podem-se colocar várias questões simples para ajudar a compreender o que realmente é importante. Sucesso é definido por:

- Um projeto que entrega dentro do prazo estipulado?
- Um projeto cujos custos estão dentro do orçamentado?
- Um projeto que resulta num produto que apresenta todas as funcionalidades estipuladas?
- Um projeto que resulta num produto interessante do ponto de vista do utilizador?
- Um projeto que atende a todas as metas de qualidade?
- ...

A definição do sucesso será tida como a combinação de todas as restrições importantes. No entanto, algumas restrições são consideradas mais importantes do que as outras. Assim, é necessário criar uma lista de prioridades. Por exemplo, num projeto com restrições a nível de âmbito, tempo, custos, segurança e qualidade, foi definido que as restrições a priorizar são a segurança e a qualidade. O projeto pode não apresentar todas as funcionalidades definidas inicialmente, ter atrasos na entrega dos produtos e ultrapassar o orçamento, mas atender a todas as restrições de segurança e de qualidade. Este projeto é concluído com sucesso, mas não atendeu a todas as restrições definidas inicialmente. É importante referir que as alterações ao planeado devem discutidas com cliente logo que identificadas.

Utilizando *Scrum*, os critérios de sucesso devem ser espelhados no *Product Backlog*. Este artefacto é uma lista ordenada de tudo o que pode ser necessário ter em conta no desenvolvimento do produto (as *User Stories*) e das alterações que devem ser efetuadas ao mesmo. O *Product Backlog* não deve ser estático: deverá ser constantemente atualizado de forma a manter o produto competitivo, atual, útil e apropriado para os contextos de utilização. Os itens do *Product Backlog* devem ser priorizados pelo *Product Owner* de acordo com a definição de sucesso acordada.

É importante realçar que a definição de sucesso só deve ser fechada quando todos os *stakeholders* concordarem e estiverem confortáveis com a mesma.

4.1.2 *Identificação e seleção de métricas*

Depois de se acordar a definição de sucesso, é necessário criar métricas que ajudem a medir o mesmo. É importante deixar claro junto dos *stakeholders* qual a função das métricas, qual o seu propósito e qual a vantagem associada à sua gestão.

Kerzner [14] refere que as métricas requerem:

- Uma necessidade ou propósito - não existe interesse em gerir métricas que não apresentam valor para o projeto,
- Um objetivo, *baseline*, ou ponto de referência - permite compreender se estamos próximos de alcançar os objetivos e os desvios em relação ao planeado,
- Uma forma de medição - mesmo que pareça intangível, existe sempre forma de efetuar medições [101],
- Uma forma de interpretação - deve ser interpretada da mesma forma por todos os *stakeholders* para evitar conflitos que possam surgir em fases mais avançadas,
- Uma estrutura de relatório - a métrica deve ser colocada de forma a que possa ser analisada e comparada com a *baseline*, assim como apresentada a todos os interessados.

Mesmo com a correta definição das métricas, a gestão das mesmas pode falhar. De acordo com o mesmo autor, as causa mais comuns são: a) má governação, especialmente por parte dos *stakeholders*, b) processo de tomada de decisão são demasiado lentos, c) planeamento do projeto excessivamente otimista, d) tentativa de realização de vários objetivos em tempo inferior ao necessário, e) metodologias e práticas de gestão de projeto pouco satisfatórias e f) pouca perceção de como as métricas serão utilizadas [14]. A gestão das métricas requer compreensão do comportamento humano, pelo que estas devem ser utilizadas com cuidado, de forma a não encorajar comportamentos não intencionais. Tal como referido anteriormente neste trabalho, as métricas não devem ser utilizadas como mecanismo de punição.

Para proceder à seleção das métricas, é importante responder a algumas questões, tidas como críticas neste processo. Sobre as medições, devem existir respostas para: a) o que deve ser medido, b) quando deve ser medido, c) como deve ser medido, e d) quem irá realizar as medições. Sobre a recolha de informação e elaboração de relatórios, é importante responder às seguintes questões: a) quem irá recolher a informação, b) quando é que a informação será recolhida, c) quando e como será feito o relatório [14].

As métricas devem reunir um conjunto de características, podendo-se enumerar:

- Surgem de uma necessidade ou têm um propósito,
- Fornecem informação útil,
- Focam-se num objetivo concreto,
- Podem ser medidas com precisão razoável,
- Refletem o estado real do projeto,
- Suportam gestão pro-ativa,
- Auxiliam na avaliação da probabilidade de sucesso ou fracasso,
- Aceites pelos *stakeholders* como uma ferramenta de tomada de decisão [14].

Três métricas bastante populares são *Schedule Variance* (SV), *Cost Variance* (CV) e Utilização de Recursos. CV, a variação dos custos, é uma métrica importante a monitorizar quando é prioritário manter-se dentro do orçamento. CV (ver Apêndice C para mais informação) rastreia as diferenças entre as despesas planeadas e as despesas reais. Ao perceber se as despesas estão a acontecer mais depressa do que o esperado, é possível também reajustar o orçamento. Mesmo assim, é improvável que os gastos excessivos consigam ser recuperados.

Se for importante atingir o prazo estipulado pois o produto precisa de ser lançado dentro de uma determinada data, deve-se considerar a métrica *Schedule Variance* (SV) (ver Apêndice C para mais informação), a variação em relação ao calendarizado. SV permite perceber se o projeto está, ou não, a progredir conforme o planeado. Esta métrica está relacionada com a diferença entre o que o que foi feito e o que deveria ter sido concluído até determinado ponto do projeto. Ao comparar o trabalho planeado com o trabalho concluído, é possível determinar quanto tempo está disponível para compensar a variação se é possível entregar dentro do prazo.

Perceber quanto tempo é que cada membro da equipa dispensa com o projeto é importante para perceber se os membros não estão sobrecarregados com tarefas do projeto. A Utilização de Recursos pode ser calculada comparando o esforço total planeado e o esforço real, por cada membro da equipa. Esta métrica também permite perceber quem dispensa mais tempo no projeto em relação ao planeado e quem dispensa menos tempo. Assim, é possível distribuir

as tarefas de forma mais eficiente, isto é, o esforço é distribuído de igual forma por todos os membros.

No entanto, na análise desta última métrica não podem ser considerados apenas os números. Muitos outros fatores podem afetar os resultados. Não é apenas relevante saber que determinado membro da equipa dispensou menos tempo do que o esperado. É preciso perceber o motivo. Possíveis razões podem ser a) doença ou necessidade de baixa, b) o membro delegou esse trabalho a outro membro, c) o membro ainda não terminou o trabalho, d) o membro termina o seu trabalho em menos tempo do que o esperado, entre outras razões. Compreender a história por detrás do problema ajuda a tomar as decisões certas. Este é um exemplo de uma métrica que precisa de dados provenientes de várias fontes para poder ser analisada corretamente.

Muitas outras métricas podem ser identificadas. No entanto, é importante compreender quais são relevantes e apresentam interesse no contexto do projeto. As métricas devem estar de acordo com os critérios de sucesso definidos no passo anterior. O número de métricas pode também variar de acordo com a dimensão e complexidade do projeto.

4.1.3 *Medir o progresso*

Com o avanço de projeto é possível acompanhar o seu desempenho utilizando as métricas escolhidas, analisando as mesmas para perceber se o projeto avança no caminho certo para alcançar os seus objetivos. Na definição das métricas, deve-se definir a frequência com que estas devem ser medidas e analisadas. No entanto, durante o cursos dos trabalho de desenvolvimento, pode ser necessário passar a efetuar medições com maior ou menor frequência. Em caso de projetos que seguem a metodologia *Scrum*, o ideal é que as métricas sejam medidas e analisadas, pelo menos, no final do *sprint*, podendo vários dados e informações ser recolhidos durante o os eventos *Sprint Review* e o *Sprint Retrospective*. No *Sprint Review* é possível obter dados sobre métricas relacionadas com o produto, enquanto que no *Sprint Retrospective* é possível obter dados relativos à equipa e aos trabalhos de desenvolvimento efetuados durante o *sprint*. Podem existir métricas que devam ser monitorizadas todos os dias, pelo que é possível utilizar o *Daily Scrum* para recolha de dados e informação.

Para ser possível perceber o progresso, é necessário estabelecer uma *baseline* para cada métrica antes de se iniciarem as medições. A *baseline* pode ser estabelecida com base em estimativas, dados provenientes de projetos anteriores (se existentes), opinião de especialistas, entre outros métodos.

Existem 3 métricas base que devem ser estimadas no início de cada projeto de desenvolvimento de software: o tamanho, o tempo e o esforço. O tamanho indica quão grande é o projeto. Existem várias formas de medir o tamanho, mas as mais utilizadas são a métrica LOC (ou *Thousands Lines of Code* (KLOC)) ou os *Function Points*. A métrica tempo indica

quanto tempo é necessário para concluir os trabalhos de desenvolvimento. É a medição do intervalo de tempo desde do início do projeto até à sua conclusão. O esforço indica o esforço total necessário para a conclusão do projeto.

Outras métricas podem também ser estimadas. No entanto, estas 3 métricas são transversais a qualquer projeto de software. Outras métricas dependem do contexto no qual o software se insere, dos critérios de sucesso, e são escolhidas em conjunto com a equipa.

De seguida apresenta-se uma abordagem que pode ser seguida para estimar as métricas esforço, tempo e tamanho. O primeiro passo consiste em analisar o âmbito do software, o segundo passo consiste na estimação do tamanho do software, e o terceiro na estimação o esforço e do tempo necessários para a conclusão do projeto.

1. Analisar o âmbito do software a desenvolver

Para efetuar estimativas é necessário compreender o âmbito do projeto. Esta análise passa por dividir o projeto em vários componentes mais pequenos, isto é, criar uma *Work Breakdown Structure* (WBS). Para dividir o produto em componentes menores, é possível utilizar duas técnicas: análise de especialistas ou a técnica de decomposição [11]. A análise de especialistas pode ser efetuada por indivíduos ou grupos com conhecimento e experiência em projetos similares, enquanto que na técnica de decomposição não é necessário experiência prévia em projetos similares.

Na técnica de decomposição é possível decompor o produto em vários *Work Packages* (WPs) (as tarefas a realizar). Estes componentes deverão ser último nível da WBS, sendo que é possível estimar os custos e duração a partir destes. Apesar do nível de decomposição ser frequentemente guiado pelo nível de controlo necessário para gerir eficientemente o projeto, este pode também variar de acordo com a complexidade e tamanho do projeto. A decomposição do projeto, de acordo com o PMBoK [11], envolve geralmente as seguintes atividades:

- Identificação e análise dos entregáveis e trabalhos relacionados,
- Estruturação e organização da WBS,
- Decomposição dos níveis superiores da WBS em componentes de nível inferior, aumentando o detalhe,
- Associação de códigos ou numeração a cada componente da WBS, e
- Verificar que o grau de decomposição dos entregáveis é apropriado ao projeto.

A estrutura de uma WBS pode ser criada seguindo a abordagem que melhor se adeque ao problema. Apesar de ser comum a utilização de um abordagem *top-down*, a abordagem *bottom-up* também pode ser utilizada para agrupar componentes. A estrutura pode ser representada de formas variadas:

- Utilizando as fases do ciclo de vida do projeto como segundo nível da decomposição, sendo que os entregáveis são representados no terceiro nível,
- Utilizar os entregáveis mais relevantes como segundo nível da decomposição, ou
- Incorporar subcomponentes que podem ser desenvolvidos por organizações externas à equipa do projeto, como o trabalho contratado [11].

Utilizando *Scrum*, pode-se construir uma WBS com uma estrutura semelhante à da Figura 12. Nesta estrutura considera-se um projeto constituído por vários *sprints*. Cada *sprint* pode conter 1 ou mais *User Stories*, constituindo o *Sprint Backlog*. Por sua vez, cada *User Story* pode ser decomposta em várias tarefas.

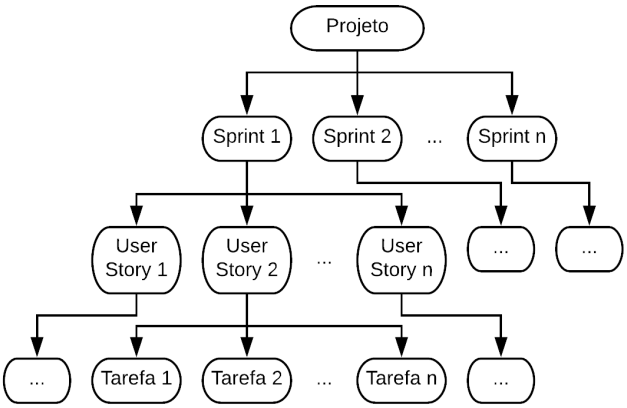


Figura 12: Exemplo de uma estrutura de uma WBS para projetos que usem *Scrum*.

Os requisitos funcionais do projeto podem ser traduzidos em *User Stories* (note-se o formato das mesmas, apresentado na Secção 3.2). *User Stories* escritas tendo em conta os requisitos funcionais do software permitem a estimação do tamanho do mesmo.

2. Estimar o tamanho do software

Estimar o tamanho do software não é uma tarefa simples, pois é estimado o tamanho de um produto intangível. Note-se que estimar o tamanho do software não é igual a estimar o esforço do mesmo. A estimativa do esforço será apresentada posteriormente.

Para estimar o tamanho do software é necessário partir primeiro com a declaração do âmbito do mesmo. Para tal, parte-se da WBS realizada no passo anterior. Para estimar o tamanho de uma aplicação de software, tendo como base a visão funcional do mesmo, sugere-se a utilização de *Function Points*. O tamanho pode ser determinado fazendo a contagem das entradas, saídas, *queries*, e ficheiros internos e externos ao sistema no qual o software se insere, e ajustando o valor resultante de acordo com a complexidade funcional.

Existem vários *standards* publicados sobre estimação do tamanho do software utilizando *Function Points*:

- COSMIC - *ISO/IEC 19761:2011 Software engineering. A functional size measurement method* [102];
- FiSMA - *ISO/IEC 29881:2010 Information technology – Systems and software engineering – FiSMA 1.1 functional size measurement method* [103];
- IFPUG - *ISO/IEC 20926:2009 Software and systems engineering – Software measurement – IFPUG functional size measurement method* [104];
- Mark-II - *ISO/IEC 20968:2002 Software engineering – Ml II Function Point Analysis – Counting Practices Manual* [105];
- NESMA - *ISO/IEC 24570:2005 Software engineering – NESMA function size measurement method version 2.1 – Definitions and counting guidelines for the application of Function Point Analysis* [106].

Estes métodos não serão abordados em detalhe nesta dissertação, pois não existe um método melhor do que outro. Estes são dependentes do contexto do software e da própria escolha da organização.

3. Estimar o esforço e o tempo

As equipas tradicionais fornecem estimativas em formato temporal, isto é, em horas, dias, semanas ou meses. Muitas equipas ágeis, no entanto, fizeram a transição para *Story Points*. *Story Points* são uma unidade de medida utilizada para expressar uma estimativa do esforço geral necessário para implementar o *Product Backlog* [107]. Os *Story Points* classificam o esforço relativo do trabalho num formato semelhante à sequência de Fibonacci: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100, etc. Esta abstração é útil pois leva a equipa a tomar decisões mais complicadas em torno da complexidade do trabalho [108].

O seguinte método (adaptado de apppm [9]) é um exemplo da aplicação de *Story Points* para estimar o esforço do projeto. No entanto, cada equipa pode decidir qual a melhor forma de estimação utilizando *Story Points*, de acordo com o projeto.

1. **Identificar as *Base Stories*** - geralmente, as equipas de desenvolvimento de software ágeis têm *User Stories* gerais ou técnicas disponíveis desde o início do projeto. No *backlog* das *User Stories*, identificam-se uma ou mais *User Stories* base, as *Base Stories*, para servir como referência para ajudar a estimar o esforço relativo do *backlog*.
2. **Atribuir *Story Points* individualmente a cada *Base Story*** - cada membro da equipa anota uma proposta de *Story Points* razoável para cada *Base Story* , sem discutir

o valor com os outros membros da equipa. A proposta deve representar a estimativa pessoal de cada membro do esforço necessário para completar determinada *Base Story*. Cada membro da equipa deve considerar não só o número de horas que eles acreditam que a tarefa levará para ser concluída, mas também considerar a complexidade, os riscos, o tamanho da tarefa e outros aspetos que considerem relevantes.

3. **Explicar os *Story Points* atribuídos** - depois de todos os membros atribuírem *Story Points* a cada *Base Story*, cada membro revela e explica o valor proposto ao resto da equipa.

4. **Chegar a um consenso para a estimativa** - depois de apresentadas todas as propostas, a equipa deve discutir qual o valor mais apropriado e chegar a um consenso. No início do projeto os passos 2, 3 e 4 são complexos. No entanto, com o avanço do projeto, caso seja necessário estimar outra vez, existem mais dados e informações que facilitam a tarefa da estimação de esforço.

5. **Dividir as *Base Stories*** - por vezes podem existir *Base Stories* que requerem mais esforço. A equipa pode dividir as *Base Stories* em histórias mais pequenas. Neste caso, deve voltar ao passo 2 e repetir o processo para cada *Base Story* nova.

6. **Usar as *Base Stories* para estimar o esforço do *Backlog*** - para cada *User Story* no *Backlog*, fazer uma estimativa do esforço, utilizando as *Base Stories* já estimadas como referência. Assim, é possível perceber o esforço relativo do *Product Backlog*, assim como de cada *Sprint Backlog*, evitando que existam *sprints* que apresentem esforço desproporcional em relação aos outros.

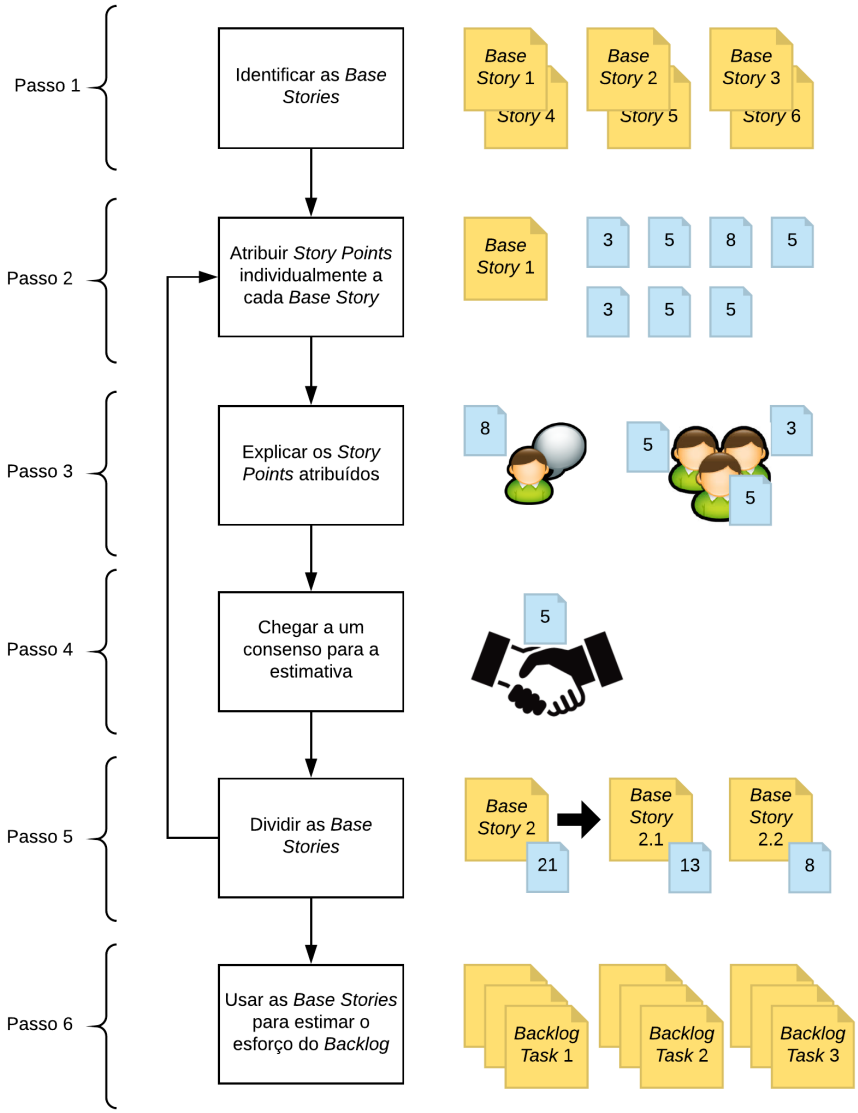


Figura 13: Exemplo de um guia passo a passo para estimar o esforço utilizando *Story Points*. Adaptado de Ian Thobias Jacobsen [9].

Depois de estimado o esforço para cada *User Story*, é possível estimar o tempo necessário para a conclusão do projeto. A partir das *User Story* com *Story Points* mais baixos, podem-se estimar o número de horas/dias/semanas necessário para a conclusão da mesma. Como os valores dos *Story Points* apresentam relação entre si, depois de proposto o tempo necessário à conclusão das *User Stories* de menor esforço, pode-se também atribuir um tempo a todas as outras histórias, até se encontrar uma estimativa para o *Product Backlog*. Toda a equipa deve estar envolvida nesta tarefa, tal como na estimação do esforço.

4.1.4 *Refinar e melhorar*

A forma como se planeia monitorizar o projeto no início pode não ser a mais adequada. No entanto, só com os avanços efetuados e durante o ciclo de vida do projeto, é possível compreender se os *stakeholders* estão ou não satisfeitos com o processo e como os relatórios entregues. Os requisitos e fatores de sucesso do projeto podem também mudar drasticamente e, de um momento para o outro, o que era considerado importante, deixou de o ser ou assuntos que eram considerados menores, passaram a influenciar criticamente o sucesso. Podem também surgir novas questões relevantes à medida que se acompanha o projeto, pelo que o gestor de projetos deve-se manter aberto e confortável à ideia de que possa ser necessário alterar, aperfeiçoar e melhorar as métricas escolhidas.

À medida que o projeto avança, existem dois conjuntos de melhorias ou mudanças a serem consideradas. A primeira questão que deve ser colocada é se as métricas são as mais corretas e adequadas. Depois de passar algum tempo a acompanhar as mesmas, é possível perceber se elas informam sobre os dados necessários para tomada de decisão. Se esta questão for respondida negativamente, pode deixar de ser necessário monitorizar determinada métrica ou alterar a mesma de forma a recolher dados interessantes para o projeto em questão. No entanto, a métrica e toda a informação relativa à mesma, deve ser arquivada pois pode ser útil para projetos futuros.

A segunda questão a ser colocada é se é fácil chegar aos dados. Se esta tarefa for muito complexa ou demorada, isto é, se for gasto muito tempo a calcular resultados e na apresentação dos mesmos à equipa, é necessário ter em atenção se todo o esforço compensa. A gestão das métricas deve favorecer o projeto e não provocar atrasos no mesmo.

É mais fácil perceber o progresso se as métricas forem reportadas de forma visual. Assim, sugere-se a utilização de *dashboards*. As *dashboards* são mecanismos visuais para apresentação de dados, que apresentam um conjunto de métricas. Devem ser desenhados de forma clara e concisa, reduzindo a quantidade de informação apenas ao que verdadeiramente é essencial [14]. Tal como as métricas, as *dashboards* também podem ser melhoradas e refinadas conforme necessário.

4.2 MÉTRICAS

Depois de apresentados os passos para a gestão das métricas, são então propostas métricas que podem ser monitorizadas, para além das já apresentadas. São apresentadas métricas relativas ao processo de gestão, neste caso relativas à metodologia de desenvolvimento *Scrum*, um adaptação do EVM para projetos que usam *Scrum* e métricas relativas aos modelos de qualidade propostos pelo ISO 25010, *Quality in Use Model* e *System/Software Product Quality Model*.

4.2.1 *Métricas da Metodologia Scrum*

É possível gerir métricas relativas à metodologia escolhida para gestão dos trabalhos de desenvolvimento e obter informação sobre a qualidade do processo de gestão. Dado que nesta dissertação foi escolhida a metodologia *Scrum*, e esta é uma metodologia que segue os princípios ágeis, é possível analisar o *Agile Manifesto* (ver Apêndice A) de forma a encontrar assuntos pertinentes e métricas que possam advir destes assuntos. A análise foi feita em dois passos. Para cada 1 do 12 princípios presentes no *Agile Manifesto*, levantaram-se as seguintes questões: a) O que deve ser medido? e b) Como podemos medir?

Ao contrário das abordagens tradicionais, onde é comum existir apenas uma entrega (a do produto final), nas abordagens ágeis existem entregas frequentes ao cliente para que este o possa testar e dar o seu *feedback*, aumento o valor do produto final.

> *1. Our highest priority is to satisfy the customer through* ***early and continuous delivery*** *of* ***valuable software****.*

Dado a existência de várias entregas de diferentes versões do produto, o cliente pode pedir a alteração de requisitos impostos inicialmente em qualquer estágio de desenvolvimento.

> *2.* ***Welcome changing requirements****, even late in development. Agile processes harness change for the customer's competitive advantage.*

Continuando com a questão das entregas, estas devem acontecer frequentemente, de forma a promover a interação com o cliente e evitar que sejam feitos muitos pedidos de alterações de requisitos por parte do cliente.

> *3.* ***Deliver working software frequently****, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*

Scrum promove a colaboração entre todos os *stakeholders*, fomentando a troca de ideias e cooperação, sempre de forma a atingir o sucesso.

> *4. Business people and developers must* ***work together*** *daily throughout the project.*

Equipas motivadas produzem resultados de maior qualidade. Suportando um ambiente colaborativo, a troca de ideias e entre-ajuda acontece naturalmente. Assim, é possível entregar um produto de maior valor ao cliente.

> *5. Build projects around* ***motivated individuals****. Give them the environment and support they need, and trust them to get the job done.*

Nos métodos ágeis é estimulada a interação frequente entre o cliente e a equipa de desenvolvimento, sempre com o intuito de entregar produtos de alta qualidade e com valor para o cliente. A comunicação é crucial nesta situação, e deve ser sempre o mais clara possível. O sexto princípio refere que o método de comunicação mais efetivo são as reuniões presenciais.

> *6. The most efficient and effective method of conveying information to and within a development team is* ***face-to-face conversation****.*

O software só tem valor para o cliente e para o utilizador se corresponder aos requisitos propostos. Apenas com entregas frequentes de vários versões do produto, é possível perceber e medir o valor introduzido pelo produto, assim como o progresso do mesmo.

> *7.* ***Working software is the primary measure of progress****.*

Os processos utilizados nos métodos ágeis é promovido o desenvolvimento sustentável. Apesar de poderem existir alterações ao requisitos em qualquer fase do desenvolvimento, o ritmo da equipa deverá manter-se constante para evitar atrasos.

> *8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a* ***constant pace*** *indefinitely.*

O nono princípio indica que para se garantir a qualidade do produto final, deve-se manter a atenção na excelência técnica e garantir um bom *design* do produto. O *design* do produto é fase crítica ao seu sucesso: um mau *design* normalmente só é detetado durante a implementação do software.

> *9. Continuous attention to* ***technical excellence*** *and good design enhances agility.*

Deve-se manter o foco em fazer apenas o que é solicitado pelo cliente, mas que esse trabalho seja bem feito. Em vez de se gastar tempo a desenvolver funcionalidades que o cliente não solicitou, deve-se manter o foco no que realmente foi solicitado e entregar esse trabalho com o máximod e qualidade possível.

> *10.* ***Simplicity****–the art of maximizing the amount of work not done–is essential.*

O décimo primeiro princípio refere que as melhores arquiteturas, requisitos e *designs* de produtos de software, resultam de equipas que se conseguem organizar sozinhas. Equipas que se adaptam à mudança, que mantêm o ritmo de trabalho e que apresentam espírito colaborativo, consegume obter melhores resultados.

> *11. The best architectures, requirements, and designs emerge from* ***self-organizing teams****.*

Por fim, é importante que a equipa reveja o seu trabalho com a devida frequência, para compreender que problemas ocorreram, como se deve adaptar às novas situação e fazer recolha de lições aprendidas.

> *12. At regular intervals, the team reflects on how to become more effective, then* ***tunes and adjusts*** *its behavior accordingly.*

Depois de respondida à primeira questão, e percebendo o que se deve medir, é importante responder à questão de como é que estes aspetos devem ser medidos.

Primeiro, os assuntos destacados a negrito foram agrupados em 4 categorias, conforme fez mais sentido. Repare-se que um princípio pode ser incluído em mais do que uma categoria. Essas categorias ajudam a encontrar as métricas que devem ser monitorizadas.

A primeira categoria é a colaboração. A colaboração é referida em 6 dos 12 princípios do *Agile Manifesto*:

4. (...) work together (...)
5. (...) motivated individuals (...)
6. (...) face-to-face conversation.
8. (...) constant pace (...)
11. (...) self-organizing teams.
12. (...) tunes and adjusts (...)

A colaboração pode ser medida com recurso a um *Kanban board*. A partir a utilização de *Kanban boards*, é possível desenhar o *Burndown Chart*, isto é, uma representação gráfica do trabalho que falta realizar em relação ao tempo restante.

Também é possível medir a satisfação da equipa com ajuda de questionários. Estes questionários devem ser desenvolvidos pelo *Scrum Master* e podem incluir questões como:

- Faço parte de uma equipa dinâmica e orientada aos resultados;
- Existe espírito de equipa e colaboração entre os elementos da equipa;
- Sinto-me valorizado como membro da equipa;
- Dentro da equipa, todos são tratados com respeito;
- Percebo o que a equipa deve realizar para atingir os objetivos do projeto;
- Os membros da equipa colaboram entre si para concretizar o trabalho;
- A comunicação entre os membros da equipa é eficiente;
- A comunicação entre os membros da equipa e o cliente é eficiente;
- Satisfação geral com a experiência na equipa;
- Entre outras questões que se considerem relevantes.

Estas questões podem ser respondidas numa escala de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 "Muito Satisfeito", ou outra escala que se considere mais apropriada. Também deve ser incluído no questionário um campo para resposta livre, para que possam ser escritas críticas e sugestões. Com o questionário, é possível perceber ou confirmar aspetos que devem ser melhorados, e proceder à resolução de problemas.

A segunda categoria identificada são as entregas. Estas são referidas em 4 dos 12 princípios:

1. (...) early and continuous delivery (...)
2. (...) Welcome changing requirements (...)
3. Deliver working software frequently (...)
7. Working software is the primary measure of progress.

As métricas relacionadas com as entregas podem ser analisadas, permitindo compreender o progresso do trabalho e a performance da equipa.

A primeira métrica a monitorizar é a velocidade. A velocidade pode ser representada graficamente, demonstrando a quantidade de trabalho ou funcionalidades completas durante um determinado período. A velocidade deve ser medida no final de cada *sprint*. Pode ser calculada pela relação entre o número de *User Stories* completos por *sprint* e utilizar o valores da velocidade de *sprints* anteriores para perceber se a equipa mantém ou altera o seu ritmo e se será possível entregar o software dentro do prazo.

Mais uma vez, é possível utilizar um *Kanban board* para desenhar o *Burndown Chart*. O *Burndown Chart*, tal como referido anteriormente, permite analisar a quantidade de *User Stories* que faltam completar por *sprint*.

Também é possível perceber o trabalho total realizado pela equipa durante o *sprint*. Parte do trabalho realizado não é apenas a concluir *User Stories*, mas também a corrigir *bugs*, entre outras tarefas que possam surgir. Assim, uma métrica importante a monitorizar é o *throughput*. O *throughput* pode ser representado graficamente e indica a quantidade total de trabalho realizado durante o *sprint*. Pode também ser comparado com valores de *sprints* anteriores, permitindo compreender o progresso dos trabalhos de desenvolvimento.

A terceira categoria identificada é a qualidade. Esta foi identificada em 3 dos 12 princípios:

> *7. Working software is the primary measure of progress.*
> *9. (...) technical excellence (...)*
> *10. Simplicity (...)*

A qualidade nesta situação refere-se à qualidade dos trabalhos de desenvolvimento. Mais à frente, serão abordadas mais métricas de qualidade, referentes aos modelos de qualidade do ISO 25010.

Foram identificadas 2 métricas chave:

- Número de *bugs* detetados por *sprint*; e
- Percentagem de cobertura dos testes realizados por *sprint*.

As primeira métrica, "Número de *bugs* detetados por *sprint*", indica a qualidade dos teste feitos ao software. Se continuarem a existir vários *bugs*, é bastante provável que os testes não cubram todos os casos de uso, o que nos leva à segunda métrica apresentada. A "Percentagem de cobertura dos testes realizados por *sprint*"indica o esforço que está a ser colocado nos testes às várias funcionalidades. Todas as funcionalidades devem ser testadas nos vários contextos de utilização, para reduzir o número de *bugs* da versão do software entregue.

A quarta categoria identificada é o valor. Esta pode ser identifica em 2 dos 12 princípios:

> *1. (...) valuable software.*
> *7. Working software is the primary measure of progress.*

No caso do valor, este pode ser representar o valor que acrescenta ao cliente ou o valor adicionado para o negócio. O valor que a solução de software tem para o cliente está relacionado com a satisfação do mesmo. Por sua vez, a satisfação está relacionada com a qualidade de utilização do produto, que será apresentada em maior detalhe posteriormente neste capítulo. No entanto, a satisfação do cliente também pode ser medida de forma simples, em casos esporádicos, recorrendo a um questionário, que pode ser desenvolvido pelo *Product Owner*, e que pode conter questões como:

- Nível de satisfação com a última entrega,
- Quão satisfeito está com a última versão do produto?
- Quão satisfeito está com a evolução do produto?
- Em que nível a última entrega atende às suas necessidades?
- Qual o nível de qualidade do produto?
- Como classificaria a relação preço-qualidade do produto?
- Como classificaria o interesse da equipa em responder às suas questões?
- Como classificaria o interesse da equipa em resolver os problemas identificados no produto?
- Que adjetivos utilizaria para descrever a última entrega?
  - Selecionar todos os que se aplicam: De confiança, De alta qualidade, Útil, Único, Boa relação preço-qualidade, Demasiado caro, Pouco prático, Ineficiente, Transmite pouca confiança, Fraca qualidade.

Excetuando a última questão, que podem ser de escolha múltipla e permite realizar uma análise qualitativa, todas as outras questões podem ser respondidas numa escala de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”, ou outra escala que se considere mais apropriada. Também deve ser incluído no questionário um campo para resposta livre, para que possam ser escritas críticas e sugestões. Com o questionário, é possível construir uma *dashboard* para fácil análise e visualização das métricas, perceber que aspetos que devem ser melhorados, e proceder à resolução de problemas.

4.2.2 *Métricas EVM em Projetos que Usam Scrum*

Tal como referido anteriormente, EVM apresenta um conjunto de métricas baseadas no triângulo âmbito-tempo-custos. O conjunto de métricas apresentado pelo EVM pode ser

consultado no Apêndice C. Para calcular as métricas do *Earned Value* utilizando a metodologia *Scrum*, podem-se utilizar *User Stories* como medida do trabalho planeado e do executado, sendo possível obter a base para todos os cálculos. A percentagem do trabalho completo pode ser obtida dividindo o número de *User Stories* completos pelo número total de *User Stories* planeados. Para calcular a *baseline* inicial, pode-se partir de:

- O tamanho de cada *sprint*;
- A data de início do projeto;
- O número de *sprints* planeados;
- O orçamento planeado;
- O número de *User Stories* planeadas.

É possível construir uma *dashboard* simples para a utilização do EVM em projetos geridos segundo a metodologia *Scrum*. Uma solução das métricas que podem ser apresentadas numa *dashboard* deste tipo é apresentada na Tabela 2.

Métrica	***Definição***
Budget At Complete (BAC)	O orçamento total do projeto
Actual Cost (AC)	O custo real do projeto
Planned User Stories per Release (PUSR)	Número de *User Stories* planeadas para cada entrega
Expected Percentage Completed (EPC)	Número total de *sprints* concluídos / Número total de *sprints* planeados
Actual Percentage Completed (APC)	Número de *User Stories* completas / Número total de *User Stories* planeadas
Planned Value (PV)	PV = BAC * EPC
Earned Value (EV)	EV = BAC * APC
Cost Performance Index (CPI)	CPI = EV / AC
Schedule Performance Index (SPI)	SPI = EV / PV

Tabela 2: *Dashboard* EVM utilizando *Scrum*.

4.2.3 *Métricas Relativas aos Modelos de Qualidade*

Nesta secção são apresentadas as métricas relativas aos modelos *Quality in Use Model* e *System/Software Product Quality Model*, presentes na norma ISO 25010.

Qualidade é uma área em que os métodos ágeis oferecem maior oportunidade de discernimento do que as abordagens tradicionais tendem a permitir. O foco na entrega frequente de

software envolve o cliente a testar várias versões do software, em vez de analisar produtos de trabalho intermediários, como requisitos e documentação do projeto. O foco nos critérios de verificação (frequentemente chamado de *Definition of Done*) permite a compreensão das funcionalidades necessárias e os atributos do produto que são importantes para o cliente.

As métricas podem ser identificadas para três tipos de utilizadores:

- Utilizador direto primário - o utilizador *target* do software;
- Utilizador direto secundário - todo aquele que pode utilizar o software, mas que não é o utilizador *target*, por exemplo, o programador;
- Utilizador indireto - alguém que recebe um *output* do produto, mas não interage com este [8].

Dado que o utilizador indireto pode nem sempre existir e depende do contexto de utilização, não será considerado nesta dissertação. As métricas selecionadas irão incidir sobre os utilizadores diretos.

Considera-se que o software é parte integrante de um sistema de informação, tendo este uma interface gráfica que o utilizador utiliza para interação com o software.

4.2.3.1 *Quality in Use Model*

A qualidade na utilização é a qualidade no ponto de vista do utilizador, e é medida sempre de acordo com o contexto de utilização. As propriedades externas do produto, como a adequação ou a tolerância a falhas, influenciam a qualidade observada na utilização. As falhas na qualidade na utilização, por exemplo, se o utilizador não consegue terminar uma tarefa, podem ser rastreadas para atributos de qualidade externa, como a operabilidade, e os seus atributos internos associados terão de ser alterados [109].

O objetivo da qualidade na utilização é que o produto possa ser utilizado por utilizadores reais, para que possam realizar tarefas no mundo real. O *design* de software focado no utilizador acrescenta valor ao produto, tal como demonstrado por um estudo recente. Esse estudo, conduzido pelo projeto ESSI PET [110], mostra que 60% dos defeitos surgem de erros de usabilidade, enquanto que apenas 15% são relacionados com a funcionalidade. Esse mesmo estudo também demonstrou que os defeitos mais significativos em termos de custo de reparação são os relacionados com a incorreta definição dos requisitos. O valor dos métodos centrados no consumidor para resolução destes problemas é confirmado num estudo de Keil e Carmel [111]. Esse estudo demonstra também que a probabilidade de sucesso aumenta com a utilização de *design* centrado no cliente.

Existem vários benefícios documentados [112, 113] associados à qualidade na utilização:

- Maior eficiência. Um sistema que incorpora um bom design ergonómico e adaptado à maneira preferida de trabalhar, permitirá que o utilizador opere de forma eficaz e

eficiente, em vez de gastar tempo vital lutando com uma interface gráfica mal desenhada e uma funcionalidade mal pensada.

- Melhor produtividade. Uma boa interface para um produto bem desenhado permitirá que o utilizador se concentre na tarefa em vez da ferramenta que, se projetada inadequadamente, pode estender ao invés de reduzir o tempo para realizar uma tarefa, bem como afetar diretamente outros aspetos de desempenho ou qualidade.

- Erros reduzidos. Uma proporção significativa do chamado ”erro humano”pode ser atribuída a um produto com uma interface mal desenhada para uma funcionalidade que não corresponde às necessidades de tarefas do utilizador. Evitar inconsistências, ambiguidades ou outras falhas no *design* da interface reduz o erro do utilizador.

- Treino reduzido. Uma interface e diálogo mal desenhados podem-se tornar uma barreira para um sistema tecnicamente sólido. Um sistema bem desenhado, com foco no utilizador final, pode reforçar o aprendido, reduzindo o tempo e o esforço de treino.

- Melhor aceitação. Particularmente importante quando o uso é facultativo. Os utilizadores preferem usar e provavelmente confiariam em sistemas bem desenhados, com funcionalidades que tornam a informação fácil de encontrar e que fornecem a informação num formato simples de interpretar.

Para melhor compreensão, volta-se a apresentar a representação do *Quality in Use Model*, na Figura 14. No primeiro nível são apresentadas as características deste modelo, e no segundo nível as sub características.

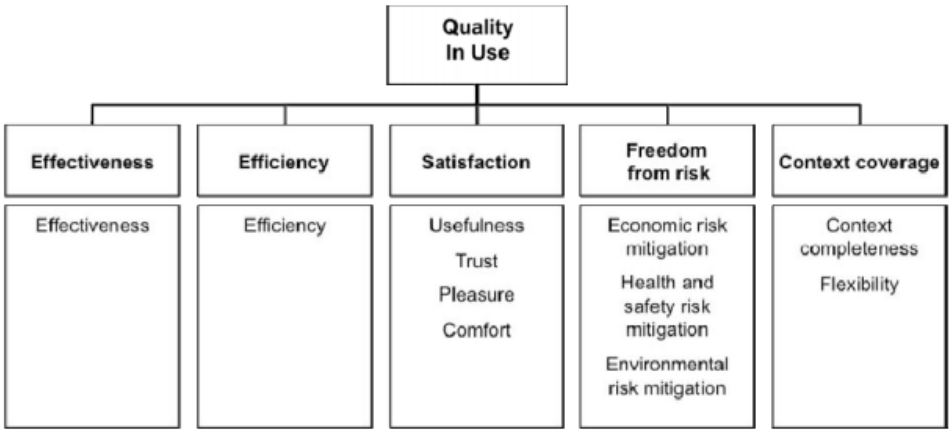


Figura 14: *Quality in Use Model* [8].

De seguida é apresentada uma breve descrição de cada característica e sub características correspondentes, assim como, sempre que se aplicar, as métricas que modem ser utilizadas para medir cada uma delas.

A característica ***Effectiveness*** é definida pela precisão e integridade com as quais os utilizadores atingem metas específicas. As métricas relativas a esta característica são apresentadas na Tabela 3.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação	1. Tempo de observação
2. Tempo de operação	2. Tempo de operação
3. Número de *bugs* detetados	3. Número de *bugs* por KLOC
4. Percentagem de requisitos implementados	4. Número de *bugs* por *test case*;
5. Número de vezes que é apresentada uma mensagem de erro	5. Número de vezes que é apresentada uma mensagem de erro
6. Nível de satisfação em relação à precisão com que as tarefas são executadas (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	
7. Nível de satisfação em relação à integridade dos resultados (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	

Tabela 3: Métricas relativas à característica *Effectiveness*.

A característica ***Efficiency*** indica os recursos gastos em relação à precisão e completude com que os usuários alcançam metas (por exemplo, o software pode demorar muito tempo a concluir a tarefa, a sua utilização pode ser muito cara a nível energético, etc). As métricas relativas a esta característica são apresentadas na Tabela 4.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação	1. Tempo de observação
2. Tempo de operação	2. Tempo de operação
3. Tempo médio de resposta	3. Tempo por tarefa
4. Nível de satisfação em relação ao custo por funcionalidade (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	4. Média de horas de trabalho por KLOC
	5. KLOC por requisito
	6. Número de *bugs* por KLOC
5. Nível de satisfação com a percentagem de bateria utilizada durante execução da tarefa (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	7. Número de *bugs* reabertos
	8. Número de *bugs* por entrega
	9. Percentagem de horas de trabalho utilizadas para corrigir *bugs*
	10. Cobertura dos testes
6. Número de *bugs* recorrentes	11. Transações de dados por segundo
7. Nível de satisfação com a velocidade das transações de dados (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	12. Tempo médio de reparação

Tabela 4: Métricas relativas à característica *Efficiency*.

A característica ***Satisfaction*** é o nível em que as necessidades do utilizador são satisfeitas quando um produto ou sistema é usado em um contexto especificado de uso.

- ***Usefulness*** indica o nível em que um utilizador está satisfeito com sua perceção de realização de objetivos pragmáticos, incluindo os resultados e as consequências do uso. As métricas relativas a esta sub característica são apresentadas na Tabela 5.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação em relação ao cumprimento dos objetivos (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito") 4. Nível geral de satisfação com o produto (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	Não se aplica

Tabela 5: Métricas relativas à sub característica *Usefulness*.

- ***Trust*** é o nível em que um utilizador ou *stakeholder* tem confiança de que um produto ou sistema se comporta como pretendido. As métricas relativas a esta sub característica são apresentadas na Tabela 6.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Número de vezes em que é apresentada uma mensagem de erro 4. Número de *bugs* detetados 5. Nível de confiança no comportamento do produto (resposta de 1 a 5, sendo que 1 indica "Desconfio Completamente"e 5 indica "Confio Completamente")	1. Tempo de observação 2. Tempo de operação 3. Número de *bugs* detetados 4. Média de *bugs* por entrega 5. Número de *bugs* por KLOC 6. Número de *bugs* reabertos 7. Número de vezes que é apresentada uma mensagem de erro 8. Percentagem de funcionalidades testadas

Tabela 6: Métricas relativas à sub característica *Trust*.

- ***Pleasure*** é o nível em que um utilizador obtém prazer pela satisfação das suas necessidades pessoais. As necessidades pessoais podem incluir necessidades para adquirir novos conhecimentos e habilidades, para comunicar identidade pessoal e provocar memórias agradáveis. As métricas relativas a esta sub característica são apresentadas na Tabela 7.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível em que o utilizador obtém prazer pela realização das suas necessidades pessoais (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	Não se aplica

Tabela 7: Métricas relativas à sub característica *Pleasure*.

- ***Comfort*** é o nível em que o utilizador está satisfeito com o conforto físico. No caso do software em si, não existe conforto físico. No entanto, estando o software inserido num sistema, pode existir conforto associado à sua utilização. As métricas relativas a esta sub característica são apresentadas na Tabela 8.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação em relação ao conforto físico associado à utilização (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos a nível de conforto atendidos

Tabela 8: Métricas relativas à sub característica *Comfort*.

A característica ***Freedom from Risk*** indica o nível em que um produto ou sistema reduz o risco potencial à situação económica, à vida humana, à saúde ou ao meio ambiente. O risco é uma função da probabilidade de ocorrência de uma determinada ameaça e das potenciais consequências adversas da ocorrência dessa ameaça.

- ***Economic Risk Mitigation*** é o nível em que um produto ou sistema mitiga o risco potencial à situação financeira, operação eficiente, propriedade comercial, reputação ou outros recursos, nos contextos de uso pretendidos. As métricas relativas a esta característica são apresentadas na Tabela 9.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Número de riscos económicos associados à utilização do software 4. Nível satisfação em relação à mitigação de riscos económicos (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos a nível económico que sofreram mudanças desde o início do projeto 4. Percentagem de requisitos a nível económico que foram atendidos

Tabela 9: Métricas relativas à sub característica *Economic Risk Mitigation.*

- ***Health and Safety Risk Mitigation*** é o nível em que um produto ou sistema mitiga o risco potencial para as pessoas nos contextos de uso pretendidos. As métricas relativas a esta característica são apresentadas na Tabela 10.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Número de riscos para a segurança e saúde associados à utilização do software 4. Nível de satisfação com a mitigação de riscos para a segurança e saúde (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos a nível de saúde e segurança que sofreram mudanças desde o início do projeto 4. Percentagem de requisitos a nível de saúde e segurança que foram atendidos

Tabela 10: Métricas relativas à sub característica *Health and Safety Risk Mitigation.*

- ***Environmental Risk Mitigation*** é o nível em que um produto ou sistema mitiga o risco potencial à propriedade ou ao meio ambiente, nos contextos de uso pretendidos. As métricas relativas a esta característica são apresentadas na Tabela 11.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Número de riscos ambientais associados à utilização do software 4. Nível de satisfação em relação à mitigação de riscos ambientais (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos a nível ambiental que sofreram mudanças desde o início do projeto 4. Percentagem de requisitos a nível ambiental que foram atendidos

Tabela 11: Métricas relativas à sub característica *Environmental Risk Mitigation*.

A característica ***Context coverage*** indica o nível em que um produto ou sistema pode ser usado com eficácia, eficiência, livre de riscos e satisfação em ambos os contextos específicos de uso e em contextos além daqueles inicialmente explicitamente identificados. O contexto de uso é relevante tanto para a qualidade em uso como para algumas sub características da qualidade do produto (onde são referidas como "condições especificadas").

- ***Context completeness*** é o nível em que um produto ou sistema pode ser usado com eficácia, eficiência, isenção de riscos e satisfação em todos os contextos de uso especificados. As métricas relativas a esta sub característica são apresentadas na Tabela 12.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Percentagem de casos de uso previamente definidos a funcionar corretamente 4. Nível de satisfação em relação à execução nos casos de uso especificados (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de casos de uso previamente definidos que passaram nos testes 4. Percentagem de requisitos que sofreram mudanças desde o início do projeto

Tabela 12: Métricas relativas à sub característica *Context Completeness*.

- ***Flexibility*** é o nível em que um produto ou sistema pode ser usado com eficácia, eficiência, livre de riscos e satisfação em contextos além daqueles especificados inicialmente nos requisitos. As métricas relativas a esta sub característica são apresentadas na Tabela 13.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Número de casos de uso não identificados inicialmente no qual o software funciona corretamente 4. Nível de satisfação em relação à execução nos casos de uso não especificados (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Número de casos de uso não identificados inicialmente no qual o software passou os testes

Tabela 13: Métricas relativas à sub característica *Flexibility*.

4.2.3.2 *System/software Product Quality Model*

Tal como referido anteriormente, falhas na qualidade da utilização podem ser rastreadas para atributos de qualidade externa. Esses atributos (ou características), são apresentados no *System/Software Product Quality Model* (ver Figura 15). No primeiro nível são apresentadas as características deste modelo, e no segundo nível as sub características.

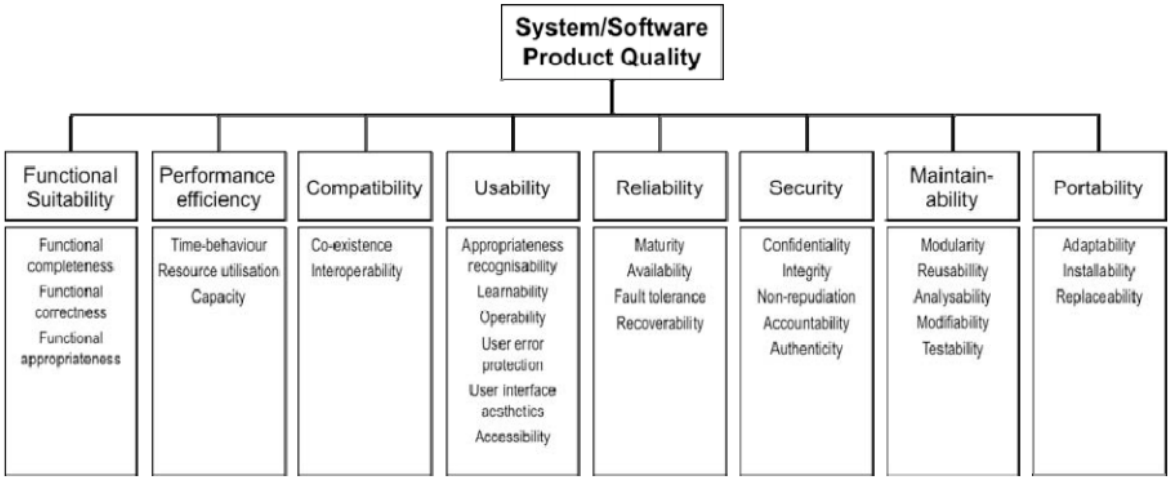


Figura 15: *System/Software Product Quality Model* [8].

De seguida é apresentada uma breve descrição de cada característica e sub características correspondentes, assim como, sempre que se aplicar, as métricas que modem ser utilizadas para medir cada uma delas.

A característica ***Functional Suitability*** indica o nível em que um produto ou sistema fornece funcionalidades que atendem às necessidades declaradas e implícitas quando usado nas condições especificadas.

- ***Functional Completeness*** é o nível em que o conjunto de funcionalidades abrange todas as tarefas especificadas e objetivos do utilizador. As métricas relativas a esta sub característica são apresentadas na Tabela 14.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Percentagem de funcionalidades implementadas 4. Percentagem de objetivos cumpridos 5. Nível de satisfação com o cumprimento de objetivos (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”) 6. Nível de satisfação com as funcionalidades implementadas (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	1. Tempo de observação 2. Tempo de operação 3. Percentagem de funcionalidades implementadas 4. Percentagem de objetivos cumpridos 5. Número de testes unitários por funcionalidade

Tabela 14: Métricas relativas à sub característica *Functional Completeness*.

- ***Functional Correctness*** é o nível em que as funcionalidades fornecem os resultados corretos com o nível de precisão necessário. As métricas relativas a esta sub característica são apresentadas na Tabela 15.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Percentagem de funcionalidades implementadas que fornecem resultados corretos 4. Número de *bugs* por funcionalidade 5. Número de vezes que o sistema apresenta um mensagem de erro 6. Nível de satisfação com a precisão dos resultados (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	1. Tempo de observação 2. Tempo de operação 3. Percentagem de funcionalidades implementadas que fornecem resultados corretos 4. Número de vezes que o sistema apresenta um mensagem de erro 5. Percentagem de *bugs* reabertos 6. Número de testes por funcionalidade 7. Tempo médio necessário para encontrar erros 8. Número de *bugs* por funcionalidade

Tabela 15: Métricas relativas à sub característica *Functional Correctness*.

- ***Functional Appropriateness*** é o nível em que as funcionalidades facilitam a realização de tarefas e objetivos específicos. As métricas relativas a esta sub característica são apresentadas na Tabela 16.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação em relação à adequação funcional (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito") 4. Nível de satisfação em relação à simplicidade da interface gráfica (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de funcionalidades que podem ser executadas de forma direta na interface gráfica

Tabela 16: Métricas relativas à sub característica *Functional Appropriateness*.

A característica ***Performance Efficiency*** refere-se ao desempenho em relação à quantidade de recursos usados sob as condições estabelecidas. Os recursos podem incluir outros produtos de software, a configuração de software e hardware do sistema e materiais (por exemplo, papel de impressão, *media* de armazenamento).

- ***Time Behaviour*** é o nível em que os tempos de resposta e processamento e as taxas de rendimento de um produto ou sistema, ao executar suas funcionalidades, atendem aos requisitos. As métricas relativas a esta sub característica são apresentadas na Tabela 17.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível em que o utilizador está satisfeito com o tempo de resposta do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. *Throughput* do sistema 4. Percentagem de funcionalidades que atendem aos requisitos temporais 5. Tempo médio de resposta 6. Tempo máximo de resposta 7. Tempo médio de utilização por recurso 8. Tempo máximo de utilização por recurso

Tabela 17: Métricas relativas à sub característica *Time Behaviour*.

- ***Resource Utilization*** é o nível em que as quantidades e tipos de recursos utilizados por um produto ou sistema, ao executar suas funções, atendem aos requisitos. As métricas relativas a esta sub característica são apresentadas na Tabela 18.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível em que o utilizador está satisfeito com os custos associados à utilização de recursos (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Tempo de resposta 4. Tempo médio de utilização por recurso 5. Tempo máximo de utilização por recurso

Tabela 18: Métricas relativas à sub característica *Resource Utilization*.

- ***Capacity*** é o nível em que os limites máximos do produto ou sistema atendem aos requisitos. As métricas relativas a esta sub característica são apresentadas na Tabela 19.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação com a capacidade de resposta do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito") 4. Nível de satisfação com a capacidade de armazenamento do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. *Throughput* do sistema 4. Tempo máximo de resposta 5. Percentagem de funcionalidades que atendem aos requisitos temporais 6. Percentagem de funcionalidades que atendem aos requisitos de armazenamento do sistema 7. Percentagem de funcionalidades que atendem aos requisitos limitadores do sistema 8. Número de testes de validação por funcionalidade

Tabela 19: Métricas relativas à sub característica *Capacity*.

A característica ***Compatibility*** indica o nível em que um produto, sistema ou componente pode trocar informações com outros produtos, sistemas ou componentes, e/ou executar suas funções necessárias, enquanto partilha o mesmo ambiente de hardware ou software.

- ***Co-Existence*** é o nível em que um produto pode desempenhar suas funcionalidades necessárias de forma eficiente, partilhando um ambiente e recursos comuns com outros produtos, sem afetar negativamente qualquer outro produto. As métricas relativas a esta sub característica são apresentadas na Tabela 20.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Número de erros detetados em recursos partilhados durante a execução 4. Nível de satisfação com a co-existência do sistema (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	1. Tempo de observação 2. Tempo de operação 3. Número de erros detetados em recursos partilhados durante a execução 4. Número de recursos partilhados 5. Número de ambientes partilhados 3. Número de erros detetados em ambientes partilhados durante a execução 4. Número de testes de integração por funcionalidade

Tabela 20: Métricas relativas à sub característica *Co-Existence*.

- ***Interoperability*** é o nível em que dois ou mais sistemas, produtos ou componentes podem trocar informações e usar as informações que foram trocadas. As métricas relativas a esta sub característica são apresentadas na Tabela 21.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação em relação à partilha de informações corretas com outro sistema (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	1. Tempo de observação 2. Tempo de operação 3. Eficiência da transmissão de dados 4. Taxa de ocupação do canal de transmissão de dados 5. Número de sistemas, produtos ou componentes que interagem com o sistema

Tabela 21: Métricas relativas à sub característica *Interoperability*.

A característica ***Usability*** indica o nível em que um produto ou sistema pode ser usado pelos utilizadores para alcançar objetivos específicos com eficácia, eficiência e satisfação em contextos especificados de uso. A usabilidade pode ser medida como uma característica de qualidade do produto em termos de suas sub-características, ou medida diretamente por medidas que são um subconjunto de qualidade em uso.

- ***Appropriateness Recognizability*** é o nível em que os utilizadores podem reconhecer se um produto ou sistema é adequado às suas necessidades. As métricas relativas a esta sub característica são apresentadas na Tabela 22.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação com adequação funcional do produto (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos atendidos

Tabela 22: Métricas relativas à sub característica *Appropriateness Recognizability*.

- ***Learnability*** é o nível em que um produto ou sistema permite ao utilizador aprender a usar o mesmos. As métricas relativas a esta sub característica são apresentadas na Tabela 23.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação com a aprendizagem sobre o sistema que o mesmo oferece (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de tutoriais implementados em relação ao número de funcionalidades implementadas 4. Número de sugestões de utilização em relação ao número de mensagens de erro

Tabela 23: Métricas relativas à sub característica *Learnability*.

- ***Operability*** é o nível em que um produto ou sistema é fácil de operar, controlar e apropriado para uso. As métricas relativas a esta sub característica são apresentadas na Tabela 24.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação relacionado com a operação do sistema (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”)	1. Tempo de observação 2. Tempo de operação 3. Percentagem de funcionalidades que podem ser executadas de forma direta na interface gráfica

Tabela 24: Métricas relativas à sub característica *Operability*.

- ***User Error Protection*** é o nível em que um produto ou sistema protege os utilizadores contra erros na utilização. As métricas relativas a esta sub característica são apresentadas na Tabela 25.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Número de sugestões de utilização em relação ao número de mensagens de erro 4. Nível de satisfação em relação à proteção contra erros causados pelo utilizador (resposta de 1 a 5, sendo que 1 indica ”Muito Insatisfeito”e 5 indica ”Muito Satisfeito”) 5. Número de ações proibidas que são assinaladas como tal	1. Tempo de observação 2. Tempo de operação 3. Número de sugestões de utilização em relação ao número de mensagens de erro 4. Número de ações proibidas que são assinaladas como tal

Tabela 25: Métricas relativas à sub característica *User Error Protection*.

- ***User Interface Aesthetics*** é o nível em que uma interface permite uma interação agradável e satisfatória para o utilizador. As métricas relativas a esta sub característica são apresentadas na Tabela 26.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação em relação à iteração com a interface gráfica (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito") 4. Nível de satisfação com o aspeto visual da interface gráfica (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos relativos à interface gráfica atendidos

Tabela 26: Métricas relativas à sub característica *User Interface Aesthetics*.

- ***Accessibility*** é o nível em que um produto ou sistema pode ser usado por pessoas com a mais ampla gama de características e capacidades para atingir um objetivo especificado num contexto específico de uso. As métricas relativas a esta sub característica são apresentadas na Tabela 27.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação em relação à acessibilidade da utilização do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito") 4. Nível de satisfação com a simplicidade ao acesso à ferramentas de acessibilidade do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de *layout* da interface gráfica com características responsivas

Tabela 27: Métricas relativas à sub característica *Accessibility*.

A característica ***Reliability*** indica o nível em que um sistema, produto ou componente executa funcionalidades especificadas, sob condições especificadas, por um período de tempo especificado. Limitações na confiança (*reliability*) surgem devido a falhas nos requisitos, *design* e implementação, ou devido a mudanças contextuais. As características de confiança incluem disponibilidade e seus fatores de influência inerentes ou externos, como disponibili-

dade, tolerância a falhas e recuperação das mesmas, segurança, manutenção, durabilidade e suporte.

- ***Maturity*** é o nível em que um sistema, produto ou componente atende às necessidades de confiança em operação normal. As métricas relativas a esta sub característica são apresentadas na Tabela 28.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação com maturidade do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Falhas evitadas 4. Falhas detetadas 5. Falhas previstas 6. Falhas resolvidas 7. Média de tempo em baixo 8. Número de vezes em que o sistema foi abaixo 9. Tempo de resposta 10. Tempo de recuperação de falhas 11. Número de testes unitários realizados por funcionalidade 12. Número de testes de validação realizados 13. Número de testes de integração realizados

Tabela 28: Métricas relativas à sub característica *Maturity*.

- ***Availability*** é o nível em que um produto ou sistema está operacional e acessível quando necessário para utilização. As métricas relativas a esta sub característica são apresentadas na Tabela 29.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação	1. Tempo de observação
2. Tempo de operação	2. Tempo de operação
3. Nível de satisfação com disponibilidade do sistema para utilização (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	3. Tarefas executadas
	4. Tarefas reiniciadas
	5. Tarefas restauradas
	8. Tempo de resposta
4. Número de vezes que o sistema se mostrou ocupado	9. Número de testes de validação realizados ao sistema
	10.Número de testes de validação falhados

Tabela 29: Métricas relativas à sub característica *Availability*.

- ***Fault Tolerance*** é o nível em que um sistema, produto ou componente opera como pretendido, apesar da presença de falhas de hardware ou software. As métricas relativas a esta sub característica são apresentadas na Tabela 30.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação	1. Tempo de observação
2. Tempo de operação	2. Tempo de operação
3. Nível de satisfação com a tolerância a falhas do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	3. Falhas evitadas
	4. Falhas detetadas
	5. Falhas previstas
	6. Falhas resolvidas

Tabela 30: Métricas relativas à sub característica *Fault Tolerance*.

- ***Recoverability*** é o nível em que, no caso de uma interrupção ou falha, um produto ou sistema pode recuperar os dados diretamente afetados e restabelecer o estado desejado do sistema. As métricas relativas a esta sub característica são apresentadas na Tabela 31.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação com a recuperação automática do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito") 3. Nível de satisfação com o tempo de recuperação automática do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo em operação 3. Tempo máximo de resposta 4. Tempo médio de recuperação automática 5. Percentagem de falhas recuperadas automaticamente 6. Número de medidas de recuperação automática implementadas

Tabela 31: Métricas relativas à sub característica *Recoverability*.

A característica ***Security*** indica o grau em que um produto ou sistema protege as informações e os dados para que as pessoas ou outros produtos ou sistemas tenham o grau de acesso aos dados adequado aos seus tipos e níveis de autorização. Assim como os dados armazenados num produto ou sistema, a segurança também se aplica à transmissão de dados. A sobrevivência, isto é, o nível em que um produto ou sistema continua a cumprir sua missão, fornecendo serviços essenciais em tempo real, apesar da presença de ataques, é coberto pela possibilidade de recuperação. A imunidade, isto é, o nível em que um produto ou sistema é resistente a ataques, é coberto pela integridade. A segurança contribui para a confiança no sistema ou produto.

- ***Confidentiality*** é o nível em que o protótipo garante que os dados sejam acessíveis apenas aos utilizadores com acesso autorizado. As métricas relativas a esta sub característica são apresentadas na Tabela 32.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Número de *bugs* detetados relativos a questões de confidencialidade dos dados do utilizador 4. Nível de satisfação em relação à confidencialidade dos dados do utilizador (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos relativos à identificação do utilizador atendidos 4. Percentagem de requisitos relativos à autenticação do utilizador atendidos 5. Percentagem de requisitos relativos à privacidade do utilizador atendidos 6. Percentagem de ficheiros protegidos 7. Poder computacional necessário para quebrar a encriptação

Tabela 32: Métricas relativas à sub característica *Confidentiality*.

- ***Integrity*** é o nível em que um sistema, produto ou componente impede o acesso não autorizado ou a modificação de programas ou dados na plataforma. As métricas relativas a esta sub característica são apresentadas na Tabela 33.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação 2. Tempo de operação 3. Número de *bugs* detetados relativos a questões de integridade do sistema 4. Nível de satisfação em relação à integridade do sistema (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos relativos à identificação do utilizador atendidos 4. Percentagem de requisitos relativos à autenticação do utilizador atendidos 5. Percentagem de requisitos relativos à privacidade do utilizador atendidos 6. Percentagem de requisitos relativos à integridade atendidos 7. Percentagem de ficheiros protegidos 8. Poder computacional necessário para quebrar a encriptação

Tabela 33: Métricas relativas à sub característica *Integrity*.

- ***Non-repudiation*** é o nível em que ações ou eventos podem ser provados como derivadas da utilização de determinada entidade, de modo que os eventos ou ações não possam ser repudiados mais tarde. As métricas relativas a esta sub característica são apresentadas na Tabela 34.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Número de *bugs* detetados relativos a questões de repudio do sistema 4. Nível de satisfação em relação ao não-repudio (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos relativos à deteção de atividades maliciosas atendidos 4. Percentagem de requisitos relativos ao não-repudio atendidos

Tabela 34: Métricas relativas à sub característica *Non-repudiation*.

- ***Accountability*** é o nível em que as ações de uma entidade podem ser rastreadas exclusivamente para essa entidade. As métricas relativas a esta sub característica são apresentadas na Tabela 35.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Número de *bugs* detetados relativos a questões de rastreio das ações da entidade 4. Nível de satisfação em relação ao rastreio das ações da entidade (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos relativos à deteção de atividades maliciosas atendidos 4. Percentagem de requisitos relativos à responsabilidade sobre as ações atendidos

Tabela 35: Métricas relativas à sub característica *Accountability*.

- ***Authenticity*** é o nível em que a identidade de um assunto ou recurso pode ser provada como sendo aquela reivindicada. As métricas relativas a esta sub característica são apresentadas na Tabela 36.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Número de *bugs* detetados relativos a questões de autenticidade 4. Nível de satisfação em relação à autenticidade reivindicada (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Percentagem de requisitos relativos à identificação do utilizador atendidos 4. Percentagem de requisitos relativos à autenticação do utilizador atendidos 5. Percentagem de requisitos relativos à privacidade do utilizador atendidos 6. Percentagem de requisitos relativos à autenticidade dos dados atendidos 7. Percentagem de ficheiros protegidos 8. Poder computacional necessário para quebrar a encriptação

Tabela 36: Métricas relativas à sub característica *Authenticity*.

A característica ***Maintainability*** indica o nível de eficácia e eficiência com o qual um produto ou sistema pode ser modificado pela equipa de suporte. As modificações podem incluir correções, melhorias ou adaptação do software a mudanças no ambiente e nos requisitos e especificações funcionais. As modificações incluem aquelas realizadas por pessoal de suporte especializado e aquelas executadas por funcionários comerciais ou operacionais, ou usuários finais. A manutenção inclui a instalação de atualizações e melhorias. A manutenção pode ser interpretada como uma capacidade inerente do produto ou sistema para facilitar atividades de manutenção, ou a qualidade em uso experimentada pela equipa de suporte para o objetivo de manter o produto ou sistema.

- ***Modularity*** é o nível em que um sistema ou software é composto por componentes discretos, de forma que uma alteração em um componente tenha impacto mínimo em outros componentes. As métricas relativas a esta sub característica são apresentadas na Tabela 37.

Utilizador direto primário	*Utilizador direto secundário*
Não se aplica	1. Tempo de observação 2. Tempo de operação 3. Número de módulos por funcionalidade do sistema 4. KLOC por módulo 5. Número de testes de integração 6. Percentagem de testes de integração passados

Tabela 37: Métricas relativas à sub característica *Modularity*.

- ***Reusability*** é o nível em que um ativo pode ser usado em mais de um sistema ou na construção de outros ativos. As métricas relativas a esta sub característica são apresentadas na Tabela 38.

Utilizador direto primário	*Utilizador direto secundário*
Não se aplica	1. Tempo de observação 2. Tempo de operação 3. Número de módulos por funcionalidade do sistema 4. KLOC por módulo 5. Número de entradas por módulo 6. Número de saídas por módulo

Tabela 38: Métricas relativas à sub característica *Reusability*.

- ***Analysability*** é o nível de eficácia e eficiência com o qual é possível avaliar o impacto num produto ou sistema de uma alteração pretendida para uma ou mais das suas partes, ou para diagnosticar um produto em relação a deficiências ou causas de falhas, ou para identificar componentes para serem modificadas. As métricas relativas a esta sub característica são apresentadas na Tabela 39.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação 2. Tempo de operação 3. Nível de satisfação com a disponibilidade da documentação (arquitetura do sistema, requisitos, análise, especificações da interface, entre outros) (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	1. Tempo de observação 2. Tempo de operação 3. Disponibilidade da documentação (arquitetura do sistema, requisitos, análise, especificações da interface, entre outros) 4. Percentagem de comentários por KLOC

Tabela 39: Métricas relativas à sub característica *Analysability*.

- ***Modifiability*** é o nível em que um produto ou sistema pode ser modificado com eficácia e eficiência sem introduzir defeitos ou degradar a qualidade do produto existente. As métricas relativas a esta sub característica são apresentadas na Tabela 40.

Utilizador direto primário	***Utilizador direto secundário***
Não se aplica	1. Tempo de observação 2. Tempo de operação 3. Número de módulos do sistema 4. KLOC 5. KLOC por funcionalidade 6. Número de *bugs* detetados depois de fazer alterações no código 7. Número de *bugs* reabertos depois de fazer alterações no código 8. Número de funcionalidades com *bugs* depois de serem feitas alterações no código

Tabela 40: Métricas relativas à sub característica *Modifiability*.

- ***Testability*** é o nível de eficácia e eficiência com o qual critérios de teste podem ser estabelecidos para um sistema, produto ou componente, e testes podem ser realizados para determinar se esses critérios foram atendidos. As métricas relativas a esta sub característica são apresentadas na Tabela 41.

Utilizador direto primário	*Utilizador direto secundário*
1. Tempo de observação	1. Tempo de observação
2. Tempo de operação	2. Tempo de operação
3. Percentagem de testes passados	3. Número de testes unitários
4. Número de testes efetuados	4. Número de testes de validação
	5. Número de testes de integração
	6. Percentagem funcionalidades testadas
	7. Percentagem de testes planeados realizados

Tabela 41: Métricas relativas à sub característica *Testability*.

A característica ***Portability*** indica o grau de eficácia e eficiência com o qual um sistema, produto ou componente pode ser transferido de um hardware, software ou outro ambiente operacional ou de uso para outro. A portabilidade pode ser interpretada como uma capacidade inerente do produto ou sistema para facilitar as atividades de portabilidade, ou a qualidade em uso experimentada para o objetivo de portar o produto ou sistema.

- ***Adaptability*** é o nível em que um produto ou sistema pode ser adaptado de forma eficaz e eficiente para hardware, software ou outros ambientes operacionais ou de uso diferente ou em evolução. As métricas relativas a esta sub característica são apresentadas na Tabela 42.

Utilizador direto primário	*Utilizador direto secundário*
Não se aplica	1. Tempo de observação
	2. Tempo de operação
	3. Número de pontos de configuração do sistema

Tabela 42: Métricas relativas à sub característica *Adaptability*.

- ***Instalability*** é o nível de eficácia e eficiência em que um produto ou sistema pode ser instalado e/ou desinstalado com êxito em um ambiente especificado. As métricas relativas a esta sub característica são apresentadas na Tabela 43.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação	1. Tempo de observação
2. Tempo de operação	2. Tempo de operação
3. Percentagem de instalações concluídas com sucesso por plataforma	3. Percentagem de instalações concluídas com sucesso por plataforma
4. Percentagem de desinstalações concluídas com sucesso por plataforma	4. Percentagem de desinstalações concluídas com sucesso por plataforma
5. Nível de satisfação com a eficácia e eficiência da instalação (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	5. Percentagem de testes unitários realizados ao software de instalação concluídos com sucesso
6. Nível de satisfação com a eficácia e eficiência da desinstalação (resposta de 1 a 5, sendo que 1 indica "Muito Insatisfeito"e 5 indica "Muito Satisfeito")	6. Tamanho do software instalador
7. Tempo médio de instalação	7. Tempo médio de instalação
8. Número de passos necessários para instalação	8. Número de passos necessários para instalação
9. Número de passos necessários para desinstalação	9. Número de passos necessários para desinstalação
10. Tempo médio de desinstalação	10. Tempo médio de desinstalação
	11. Percentagem de ficheiros eliminados da plataforma depois da desinstalação

Tabela 43: Métricas relativas à sub característica *Instalability*.

- ***Replaceability*** é o nível em que um produto pode substituir outro produto de software especificado para o mesmo propósito, no mesmo ambiente. As métricas relativas a esta sub característica são apresentadas na Tabela 44.

Utilizador direto primário	***Utilizador direto secundário***
1. Tempo de observação	1. Tempo de observação
2. Tempo de operação	2. Tempo de operação
3. Número de produtos similares existentes no mercado	3. Número de produtos similares existentes no mercado

Tabela 44: Métricas relativas à sub característica *Replaceability*.

4.3 Guia e Sugestões de Utilização

Nesta secção é apresentado um guia e algumas sugestões de utilização do *Management of Planned Value* que as equipas podem seguir para gestão eficiente de métricas em projetos de desenvolvimento de software.

1. A cada entrega de cada versão do produto, estipular um tempo necessário para o teste por parte do utilizador final ao mesmo e recolher o *feedback* sobre a qualidade de utilização.

2. Em caso de projetos de grande complexidade, risco e com grande número de métricas para serem monitorizadas, sugere-se a criação de um PMO para monitorização das métricas.

3. Algumas métricas devem ser tratadas como *Key Performance Indicators* (KPIs). Em caso de dúvida, ou mesmo para confirmação, se determinada métrica deve ser tratada como KPI, devem-se efetuar dois passos: 1) perceber se a métrica segue objetivos *SMART* (ver Secção 1.5) e 2) responder, para cada métrica, positiva ou negativamente às seguintes questões, propostas por Kerzner [14]:

   - É preditiva?
   - É quantificável?
   - É contestável/discutível?
   - É relevante?
   - É automática?

   Se a maioria das respostas para determinada métrica for positiva, é provável estarmos perante um KPI.

4. O gestor de projeto deve-se reunir com todos os *stakeholders* que considere necessário no início do projeto e chegar a um acordo sobre o que constitui o sucesso do projeto.

5. O gestor de projetos deve chegar a um acordo sobre que métricas devem ser monitorizadas para verificar se o sucesso foi atingido. Algumas métricas deverão ser tratadas como KPIs.

6. O gestor de projetos (em certos casos, juntamente com o PMO) deverá preparar *dashboards* para entregar a cada *stakeholder*.

7. No fecho do projeto, deverá ser mantido um registo das métricas utilizadas, para que possam ser reutilizadas em projetos similares no futuro.

8. É importante que o responsável pela métrica cumpra os seguintes requisitos:

- Deve compreender a cultura e valores da organização,
- Deve funcionar como mentor a todos os que pretendam utilizar ou analisar a métrica,
- Deve ter respeito por toda a equipa afeta aos trabalhos relacionados com o projeto,
- Deve ser capaz de executar melhorias contínuas na métrica, incluindo melhorias nas técnicas utilizadas para medição,
- Deve ser capaz de suportar a promoção do uso de métricas,
- Deve suportar o trabalho do PMO, caso este exista, ajudando a determinar se a métrica mantém a sua validade.

9. É comum selecionarem-se métricas em demasia, pois tudo parece importante. Tentar encontrar um balanço e acordar quais as métricas relevantes.

10. Durante a definição da métrica (nas fases de iniciação e planeamento), incluir os stakeholders de forma a que estes possam perceber o processo e torna-lo o mais adequado possível para que seja atingido o sucesso.

11. Pedir a opinião ao pessoal mais experiente.

12. Se forem selecionadas métricas relativas a RH, assumir que estes apenas vão ser produtivos durante cerca de 80% do tempo. Assim, considerar uma alocação inferior a 80% para cada RH.

13. RH que estão alocados a múltiplos projetos levam mais tempo a completar tarefas devido ao tempo necessário para a troca de contexto.

14. Ter sempre planos de contingência para eventos inesperados e resolução de problemas.

15. Reservar tempo suficiente para fazer uma estimativa adequada do projeto. As estimativas apressadas são estimativas imprecisas e de alto risco. Para grandes projetos de desenvolvimento, a etapa de estimativa deve ser considerada como um mini projeto.

16. Sempre que possível, utilizar dados de projetos anteriores similares, realizados dentro da mesma organização. Isto resultará numa estimativa com mais precisão.

17. Se a organização não recolhe dados de projetos anteriores, será uma boa ocasião para iniciar a arquivar os mesmos e tornar este procedimento habitual.

18. Considerar as estimativas dos membros da equipa, pois as estimativas efetuadas por quem efetua o trabalho de desenvolvimento são mais precisas em comparação às estimativas efetuadas por quem não está incluído na equipa de desenvolvimento.

19. Considerar as estimativas de várias pessoas e diferentes técnicas para garantir o menor erro possível.

20. Observar e relacionar as estimativas e perceber a convergência ou divergência entre as mesmas.

21. Re-estimar o projeto várias vezes durante o seu ciclo de vida de forma a manter os valores atuais e efetuar as devidas correções o mais depressa possível.

22. De forma a que se obtenham estimativas confiáveis, devem-se considerar os seguintes passos:
    - Estimar tendo por base projetos semelhantes já concluídos;
    - Utilizar técnicas de decomposição simples e a partir do resultado desta gerar estimativas de custo e esforço do projeto;
    - Utilizar várias técnicas de estimativa diferentes para poder analisar e comparar os resultados.

Apesar de serem propostas dezenas de métricas que podem ser monitorizadas em projetos de desenvolvimento de software, não é de todo viável monitorizar todas as métricas apresentadas nesta dissertação, dado o esforço necessário para tal atividade. Tal como referido anteriormente nesta dissertação, um número elevado de métricas pode provocar atrasos e atrapalhar o progresso, levando ao insucesso do projeto.

Como o maior número de métricas apresentadas são relativas aos modelos de qualidade, aconselha-se a reunir com o cliente e perceber quais as características dos modelos que têm maior relevância para o sistema ou software a desenvolver e que vão acrescentar valor para o utilizador final.

Para perceber quais as características mais relevantes, é proposto o seguinte método:

1. Classificação da relevância - solicitar ao *Product Owner* que classifique cada um dos fatores de qualidade do software de 1 a 5, sendo 1 menos aplicável e 5 mais aplicável, em termos do produto de software.

2. Identificação de 3 fatores de qualidade chave - perguntar tanto aos *stakeholders* externos, como à equipa de desenvolvimento, quais são os 3 fatores de qualidade de software que eles mais desejam e que priorizem os mesmos de 1 a 3, sendo o 1 o mais prioritário e 3 o menos prioritário. Anotar os aspetos que levaram à decisão, para compreender as necessidades de cada um.

3. Encontrar o consenso - reunir o *Product Owner*, os *stakeholders* externos e os membros da equipa de desenvolvimento de software para discutir e chegar a um consenso sobre quais as característica de qualidade de software que serão focadas no projeto.

Este método não é rígido, pelo que podem ser identificadas mais do que 3 características de qualidade de software tidas como chave para o sucesso do produto. No entanto, deve-se manter o foco no que realmente é essencial.

5

CONCLUSÃO

Terminado o estudo, segue-se um conjunto de conclusões, identificação de limitações e sugestões para trabalhos futuros.

Tal como apresentado no Capítulo 1, esta dissertação pretende responder a duas perguntas de investigação:

- Quais são as principais métricas a monitorizar durante o desenvolvimento de software de forma a concluir o mesmo com sucesso?
- Quais os processos e/ou atividades que devem ser utilizados para acompanhar as métricas selecionadas?

Para responder às perguntas de investigação propostas, primeiro foi feito um levantamento do estado da arte e fundamentos teóricos sobre a disciplina da Gestão de Projetos e da Engenharia de Software. Foram estudadas abordagens ágeis e tradicionais, tendo-se optado pela metodologia *Scrum* para este estudo. Dentro do âmbito da Engenharia de Software, foi percebida a importância da qualidade do software para a entrega do software com sucesso, razão pela qual a qualidade tem especial destaque nesta dissertação.

Assim, no quarto capítulo, *Management of Planned Value*, foi apresentada uma sugestão de um processo que se deve seguir para a gestão das métricas, respondendo à segunda pergunta de investigação, assim como métricas relativas à metodologia *Scrum*, um exemplo de uma *dashboard* EVM em contexto ágil, e métricas relativas aos modelos de qualidade apresentados pela ISO 25010 (SQuaRE), respondendo à primeira pergunta de investigação. Foi ainda apresentado um conjunto de sugestões e um guia que ajudam na monitorização das métricas em projetos de desenvolvimento de software.

O processo apresentado para gestão das métricas consiste em quatro passos: definir o sucesso, identificar e selecionar as métricas, medir o progresso e refinar e melhorar. Apesar de ser sugerida a utilização deste processo em projetos *Scrum*, pode também ser aplicado a projetos tradicionais ou com abordagens mistas. Neste caso poderão ser feitas algumas alterações ao processo, de forma a torná-lo o mais adequado possível. No decorrer do estudo, levantou-se a seguinte questão: "Adicionando o processo a abordagens ágeis, estará a flexibi-

lidade da abordagem comprometida?". Em resposta afirmativa, a diminuição da flexibilidade da abordagem é vista como uma limitação do *Management of Planned Value*.

O maior desafio foi a identificação de métricas gerais relativas às caraterísticas qualidade, pois várias características dos modelos *Quality In Use Model* e *System/Software Product Quality Model* são relativas a um produto concreto. Mesmo partindo de um produto ou ideia concreta do produto, existem várias barreiras à medição da qualidade, tais como requisitos mal definidos ou que não são levados em consideração, a qualidade provoca custos adicionais e o cliente nem sempre tem isso em conta, existem barreiras resultantes da definição pouco clara de quem são os utilizadores do sistema e quais os contextos de uso, entre outras desvantagens.

Analisando as métricas de qualidade propostas, existem duas métricas que surgem associadas à grande maioria das sub-características e que podem parecer repetidas: o "Tempo de observação"e o "Tempo de operação". No entanto, podem ser medidas versões diferentes de cada métrica, dependendo da sub característica ou da necessidade. Por exemplo, para determinada sub característica, pode ser interessante registar o tempo de operação total do sistema e o tempo de operação de determinada funcionalidade. Estas duas métricas fornecem também informação sobre qual foi o tempo dispensado a testar o sistema. Se o tempo de observação for muito reduzido, podem não ser sido feitos testes suficientes, e a qualidade das medições pode ser afetada ou estas não apresentarem a devida confiança.

Na grande maioria das sub caraterísticas é apresentada uma métrica que mede o "Nível de satisfação em relação a (...)"determinado aspeto de qualidade. Estas métricas informam sobre a experiência do utilizador final em relação ao uso da aplicação. Utilizando estas métricas, é possível receber *feedback* de forma mais completa, garantindo que nenhum aspeto crucial para o sucesso é esquecido. O "Nível de satisfação"depende também do "Tempo de observação". Isto é, os níveis de precisão e de confiança na medição do "Nível de satisfação"aumentam conforme o tempo dispensado pelo utilizador a testar o sistema e a observar o seu comportamento.

A lista de métricas propostas não é final, apesar da sua extensão. Certamente que, no futuro, muitas outras métricas poderão ser identificadas e adicionadas a esta lista. Adicionando contexto ao estudo, muitas outras métricas dependentes do contexto e dos critérios de sucesso do projeto poderão ser identificadas. Apesar de terem sido propostas dezenas de métricas, espera-se que, aplicando a um contexto específico, o número seja bastante mais reduzido e que algumas das métricas selecionadas sejam tratadas como KPIs.

Este estudo pode ainda ser continuado e melhorado. Como sugestões de trabalhos futuros, aponta-se a 1) validação prática desta dissertação num projeto real, quer este siga uma abordagem ágil ou um abordagem tradicional na sua gestão, 2) comparação dos resultados e conclusões obtidas da aplicação do *Management of Planned Value* nas diferentes abordagens, 3) identificação de *Key Performance Indicators* (KPIs), 4) a criação de um mecanismo automático de gestão de métricas, 5) o desenvolvimento de um sistema de informação capaz de

fornecer informação em tempo real acerca das métricas selecionadas e que permita a seleção de métricas a visualizar por parte dos stakeholders, podendo estes escolher que métricas desejam analisar, ou ainda 6) identificar novas métricas para outras áreas de conhecimento do PMBoK, tais como métricas relativas à comunicação ou ao risco.

BIBLIOGRAFIA

[1] Vijay K. Vaishnavi e William Kuechler, Jr., *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Boston, MA, USA: Auerbach Publications, 1st ed., 2007.

[2] Winston W. Royce, "Managing the Development of large Software Systems," *IEEE Wescon*, no. August, pp. 1–9, 1970.

[3] Federal Highway Administration, *Clarus Concept of Operations*. 2005.

[4] Barry Boehm, "A Spiral model of software development and enhancement," in *Software Management, Seventh Edition*, pp. 37–48, 2007.

[5] "The agile42 Scrum Cheat Sheet." https://www.agile42.com/en/agile-info-center/scrum/scrum-cheat-sheet/. Acedido em: 27 de Fevereiro de 2018.

[6] "What is Kanban." https://sites.google.com/site/wcfpandu/what-is-kanban. Acedido em: 27 de Fevereiro de 2018.

[7] IEEE Computer Society, Pierre Bourque e Richard E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. Los Alamitos, CA, USA: IEEE Computer Society Press, 3rd ed., 2014.

[8] International Organization for Standardization e International Electrotechnical Commission, "ISO/IEC 25000:2014 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE," 2014.

[9] apppm, "Story Points Estimation." http://apppm.man.dtu.dk/index.php/Story_Points_Estimation. Acedido em: 31 de Março de 2018.

[10] A K Munns e B F Bjeirmi, "The role of project management in achieving project success," *International Journal of Project Management*, vol. 14, no. 2, pp. 81–87, 1996.

[11] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBoK Guide)*. Project Management Institute, 6ª ed., 2017.

[12] Wayne Abba, "How earned value got to prime time: A short look back and glance ahead," *The Measurable News*, vol. 2001, 2001.

[13] Jan Terje Karlsen, "Project stakeholder management," *Engineering Management Journal*, vol. 14, no. 4, pp. 19–24, 2002.

[14] Harold Kerzner, *Project Management Metrics, KPIs, and Dashboards: A Guide to Measuring and Monitoring Project Performance*. John Wiley & Sons, 2011.

[15] Francesco Perrini e Antonio Tencati, "Sustainability and stakeholder management: The need for new corporate performance evaluation and reporting systems," *Business Strategy and the Environment*, vol. 15, no. 5, pp. 296–308, 2006.

[16] Capers Jones, *Applied Software Measurement: Assuring Productivity and Quality*. New York, NY, USA: McGraw-Hill, Inc., 1991.

[17] Standish Group, "CHAOS Report," 1995.

[18] Capers Jones, *Assessment and Control of Software Risks*. Yourdon Press Series, Yourdon Press, 1994.

[19] Michael L. Cook, "Software Metrics: An Introduction and Annotated Bibliography," *SIGSOFT Softw. Eng. Notes*, vol. 7, pp. 41–60, Apr.

[20] IEEE, "IEEE 610.12-1990 Standard Glossary of Software Engineering Terminology," 1990.

[21] Cornelius T. Leondes, *Intelligent Systems: Technology and Applications, Six Volume Set*. Taylor & Francis, 2002.

[22] Martin Campbell-Kelly, "The Development of Computer Programming in Britain (1945 to 1955)," *Annals of the History of Computing*, vol. 4, no. 2, pp. 121–139, 1982.

[23] David Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, pp. 1053–1058, Dec. 1972.

[24] "Software Engineering." https://en.wikipedia.org/wiki/Software_engineering. Acedido em: 17 de Janeiro de 2018.

[25] Farzana Asad Mir e Ashly H. Pinnington, "Exploring the value of project management: Linking Project Management Performance and Project Success," *International Journal of Project Management*, 2014.

[26] Kam Jugdev e Ralf Moller, "A retrospective look at our evolving understanding of project success," *IEEE Engineering Management Review*, vol. 34, no. 3, pp. 110–127, 2006.

[27] João Varajão, "Success Management as a PM Knowledge Area - Work-in-Progress," *Procedia Computer Science*, vol. 100, pp. 1095–1102, 2016.

[28] João Varajão, Caroline Dominguez, Pedro Ribeiro e Anabela Paiva, "Failures in software project management – are we alone? a comparison with construction industry," *The Journal of Modern Project Management*, pp. 22–27, 2014.

[29] Marco Liberato, João Varajão e Paulo Martins, "CMMI Implementation and Results: The Case of a Software Company," *Modern Techniques for Successful IT Project Management*, 2015.

[30] João Varajão, Caroline Dominguez, Pedro Ribeiro e Anabela Paiva, "Critical Success Aspects in Project Management : Similarities and Differences Between the Construction and the Software Industry," *Tehnicki Vjesnik-Technical Gazette*, vol. 21, no. 2, pp. 583–589, 2014.

[31] Adam Collins e David Baccarini, "Project Success - A Survey," *Journal of Construction Research*, vol. 5, pp. 211–231, 2004.

[32] William J. Pinkerton, *Project Management : Achieving Project Bottom-Line Success*. McGraw-Hill Education, 1st ed., 2003.

[33] Danie Van Der Westhuizen e Edmond P. Fitzgerald, "Defining and measuring project success," in *Defining and measuring project success*, 2005.

[34] Terry Cooke-Davies, "The "real" success factors on projects," *International Journal of Project Management*, vol. 20, no. 3, pp. 185–190, 2002.

[35] Harold Kerzner, *Project Management: a Systems Approach to Planning, Scheduling, and Controlling*. John Wiley & Sons, 2009.

[36] Jeffrey K. Pinto e Dennis P. Slevin, "Critical factors in successful project implementation," *IEEE Transactions on Engineering Management*, vol. EM-34, no. 1, pp. 22–27, 1987.

[37] Jeffrey K. Pinto e John E. Prescott, "Variations in Critical Success Factors Over the Stages in the Project Life Cycle," *Journal of Management*, vol. 14, no. 1, pp. 5–18, 1988.

[38] Nitin Agarwal e Urvashi Rathod, "Defining 'success' for software projects: An exploratory revelation," *International Journal of Project Management*, vol. 24, no. 4, pp. 358–370, 2006.

[39] Gabriela Fernandes, Stephen Ward e Madalena Araújo, "Identifying useful project management practices : A mixed methodology approach," *International Journal of Information Systems and Project Management*, vol. 1, no. 4, pp. 5–21, 2013.

[40] Terry L. Fox e J. Wayne Spence, "The effect of decision style on the use of a project management tool," *ACM SIGMIS Database*, vol. 36, no. 2, pp. 28–42, 2005.

[41] Karlos Artto, Miia Martinsuo, Perttu Dietrich e Jaakko Kujala, "Project strategy: strategy types and their contents in innovation projects," *International Journal of Managing Projects in Business*, vol. 1, no. 1, pp. 49–70, 2008.

[42] Andreas Auinger e Alexander Hochmeier, "An Enterprise 2.0 project management approach to facilitate participation, transparency, and communication," *International Journal of Information Systems and Project Management*, vol. 1, no. 2, pp. 43–60, 2013.

[43] João M. Fernandes e Ricardo J. Machado, *Requirements in Engineering Projects*. Lecture Notes in Management and Industrial Engineering, Springer International Publishing, 2015.

[44] Evelyn J. Barry, Tridas Mukhopadhyay e Sandra A. Slaughter, "Software Project Duration and Effort: An Empirical Study," *Information Technology and Management*, vol. 3, no. 1-2, pp. 113–136, 2002.

[45] Anish Mittal, Kamal Parkash e Harish Mittal, "Software cost estimation using fuzzy logic," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 1, p. 1, 2010.

[46] Capers Jones, "Software project management practices: Failure versus success," *CrossTalk: The Journal of Defense Software ...*, vol. 17, no. 10, pp. 5–9, 2004.

[47] Kunal Jamsutkar, Viki Patil e P. M. Chawan, "Software Project Quality Management," *International Journal of Engineering Research and Applications*, vol. 2, no. 3, 2012.

[48] Meghann L. Drury-Grogan, "Performance on agile teams: Relating iteration objectives and critical decisions to project management success factors," *Information and Software Technology*, vol. 56, no. 5, pp. 506–515, 2014.

[49] Norizan Ahmad, Faridah Ismail, Syarifah N. A. S. Alwi e Rahasnan A. Rashid, "Important client attributes that influence project success: A focus on the briefing process," in *ISBEIA 2011 - 2011 IEEE Symposium on Business, Engineering and Industrial Applications*, pp. 314–319, 2011.

[50] Julian Day, "Software development as organizational conversation: analogy as a systems intervention," *Systems Research and Behavioral Science*, vol. 17, no. 4, pp. 349–358, 2000.

[51] William Bruce Cameron, *Informal sociology: a casual introduction to sociological thinking*. Studies in sociology, Random House, 1963.

[52] Tom Gilb, "Adding stakeholder metrics to agile projects," 2004.

[53] "A Minimalist's Approach to Project Metrics." https://pmhut.com/a-minimalists-approach-to-project-metrics. Acedido em: 17 de Janeiro de 2018.

[54] David Parmenter, *Key Performance Indicators (KPI)*. 2ª ed., 2010.

[55] Norman Fenton e James Bieman, *Software Metrics: A Rigorous and Practical Approach*. 3ª ed., 2014.

[56] D. Ince, "Software metrics: introduction," *Information and Software Technology*, vol. 32, no. 4, pp. 297–303, 1990.

[57] Stephen H. Kan, *Metrics and Models in Software Quality Engineering*. Addison-Wesley Professional, 2ª ed., 2014.

[58] Luiz Fernando Capretz, "Bringing the human factor to software engineering," *IEEE Software*, vol. 31, no. 2, pp. 0–2, 2014.

[59] Gary S. Lynn e Richard R. Reilly, "Measuring Team Performance," *Research-Technology Management*, vol. 43, no. 2, pp. 48–56, 2000.

[60] B. A. Kitchenham e Littlewood, *Measurement for Software Control and Assurance*. Springer Netherlands, 1989.

[61] Frederick Winslow Taylor, *The Principles of Scientific Management*. Library of American civilization, Harper & Brothers, 1911.

[62] Chiu, Y.C., *An Introduction to the History of Project Management: From the Earliest Times to A.D. 1900*. Uitgeverij Eburon, 2010.

[63] Jason Charvat, *Project Management Methodologies: Selecting, Implementing, and Supporting Methodologies and Processes for Projects*. Wiley & Sons, 2003.

[64] *ISO 9000: Quality Management*. Geneva: International Organization for Standardization, 2015.

[65] Mary Beth Chrissis, Michael Konrad e Sandra Shrum, *CMMI for Development: Guidelines for Process Integration and Product Improvement*. Addison-Wesley Professional, 3ª ed., 2011.

[66] Kathy Schwalbe, "Information Technology Project Management," *Course Tecnhnology*, pp. 1–527, 2014.

[67] *ISO 21500: Guidance on Project Management*. Geneva: International Organization for Standardization, 2012.

[68] Office of Government Commerce, *Managing Successful Project with PRINCE2*. 2009.

[69] Frederick P. Brooks Jr., *The Mythical Man-month*. Addison-Wesley Longman Publishing Co., Inc., Anniversary ed., 1995.

[70] Brian Marick, "New Models for Test Development," 2000.

[71] Kent Beck, Mike Beedle Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland e Dave Thomas, "Manifesto for Agile Software Development." http://www.agilemanifesto.org/, 2001.

[72] Dan Turk, Robert France e Bernhard Rumpe, "Limitations of agile software processes," *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP 2002)*, p. 4, 2002.

[73] Ken Schwaber e Jeff Sutherland, "The Scrum Guide^{TM}," 2017.

[74] Julia Wester, "What is Kanban?." http://www.everydaykanban.com/what-is-kanban/. Acedido em: 27 de Fevereiro de 2018.

[75] David J. Anderson e Andy Carmichael, *Essential Kanban Condensed*. D. J. Anderson and A. Carmichael, Essential Kanban Condensed, 1st Edition ed.: Lean Kanban University Press, 2016., 2016.

[76] Tutorials Point, "Extreme Programming - Roles." https://www.tutorialspoint.com/extreme_programming/extreme_programming_roles.htm. Acedido em: 27 de Fevereiro de 2018.

[77] Tutorials Point, "Extreme Programming - Activities & Artifacts." https://www.tutorialspoint.com/extreme_programming/extreme_programming_activities_artifacts.htm. Acedido em: 27 de Fevereiro de 2018.

[78] Don Wells, "The Values of Extreme Programming." http://www.extremeprogramming.org/values.html. Acedido em: 27 de Fevereiro de 2018.

[79] Don Wells, "The Rules of Extreme Programming." http://www.extremeprogramming.org/rules.html. Acedido em: 27 de Fevereiro de 2018.

[80] Ian Sommerville, *Software Engineering*. USA: Addison-Wesley Publishing Company, 9th ed., 2010.

[81] Wilson de Pádua Paula Filho, *Engenharia de software: fundamentos, métodos e padrões.* Livros Técnicos e Científicos, 2003.

[82] Ana R. C. da Rocha, José C. Maldonado e Kival C. Weber, *Qualidade de software: teoria e prática.* Prentice Hall, 2001.

[83] Roger S. Pressman, *Software Engineering: A Practitioner's Approach.* McGraw-Hill Higher Education, 5th ed., 2001.

[84] Institute of Electrical and Electronics Engineers, "IEEE STD 12207-2008 - Systems and software engineering - Software life cycle processes," 2011.

[85] International Organization for Standardization e International Electrotechnical Commission e Institute of Electrical e Electronics Engineers and IEEE-SA Standards Board, "ISO/IEC 15288:2008(E) IEEE Std 15288-2008 (Revision of IEEE Std 15288-2004) - ISO/IEC/IEEE International Standard - Systems and software engineering System life cycle processes," 2008.

[86] International Organization for Standardization e International Electrotechnical Commission and Institute of Electrical e Electronics Engineers e IEEE-SA Standards Board, "ISO/IEC/IEEE 29148 Systems and Software Engineering - Life Cycle Processes - Requirements Engineering," 2011.

[87] Robert N. Charette, *Software Engineering Risk Analysis and Management.* New York, NY, USA: McGraw-Hill, Inc., 1989.

[88] Gabriela Fernandes, Ana R. Martins, Eduardo B. Pinto, Madalena Araújo e Ricardo J. Machado, "Risk Response Strategies for Collaborative University-Industry R&D Funded Programs." A ser apresentado na Regional HELIX 2018 Conference. Para publicação em *Springer Lecture Notes in Electrical Engineering.*, 2018.

[89] Mark Keil, Paul E. Cule, Kalle Lyytinen e Roy C. Schmidt, "A framework for identifying software project risks," *Communications of the ACM*, vol. 41, no. 11, pp. 76–83, 1998.

[90] International Organization for Standardization e International Electrotechnical Commission and Institute of Electrical e Electronics Engineers e IEEE-SA Standards Board, "ISO/IEC 16085:2006 Systems and software engineering - Life cycle processes - Risk management," 2006.

[91] Institute of Electrical and Electronics Engineers, "IEEE Std 828-2012 (Revision of IEEE Std 828-2005) - IEEE Standard for Configuration Management in Systems and Software Engineering," 2012.

[92] International Electrotechnical Commission e Institute of Electrical e Electronics Engineers, "ISO/IEC TR 18018:2010 Information technology - Systems and software engineering - Guide for configuration management tool capabilities," 2010.

[93] International Organization for Standardization e International Electrotechnical Commission and Institute of Electrical e Electronics Engineers e IEEE-SA Standards Board, "ISO/IEC/IEEE 24765:2017(E) - ISO/IEC/IEEE International Standard - Systems and software engineering - Vocabulary," 2017.

[94] International Organization for Standardization e International Electrotechnical Commission and Institute of Electrical e Electronics Engineers e IEEE-SA Standards Board, "ISO/IEC/IEEE 15939:2017(E) - ISO/IEC/IEEE International Standard - Systems and software engineering - Measurement process," 2017.

[95] International Organization for Standardization, "ISO/IEC 9000-3:2014 Software engineering - Guidelines for the application of ISO 9001:2008 to computer software," 2014.

[96] Codacy, "ISO 25010 Software Quality Model." https://blog.codacy.com/enterprise-software-a-summary-of-iso-25010-software-quality-model-7100575d6f6. Acedido em: 2 de Março de 2018.

[97] International Organization for Standardization e International Electrotechnical Commission, "ISO/IEC 15504 Information technology – Process assessment," 2014.

[98] TutorialsPoint, "Estimation Techniques - Overview." https://www.tutorialspoint.com/estimation_techniques/estimation_techniques_overview.htm. Acedido em: 29 de Março de 2018.

[99] Richard E. Fairley, *Managing and Leading Software Projects*. 2008.

[100] Josehph M. Juran, Frank M. Gryna e R. S. Bingham, *Quality Control Handbook*. 3ª ed., 1974.

[101] Douglas W. Hubbard, *How to Measure Anything: Finding the Value of Intangibles in Business*. John Wiley & Sons, 2. auflage ed., 2010.

[102] International Organization for Standardization e International Electrotechnical Commission, "COSMIC - ISO/IEC 19761:2011 Software engineering. A functional size measurement method," 2011.

[103] International Organization for Standardization e International Electrotechnical Commission, "FiSMA - ISO/IEC 29881:2010 Information technology – Systems and software engineering – FiSMA 1.1 functional size measurement method," 2010.

[104] International Organization for Standardization e International Electrotechnical Commission, "IFPUG - ISO/IEC 20926:2009 Software and systems engineering – Software measurement – IFPUG functional size measurement method," 2009.

[105] International Organization for Standardization e International Electrotechnical Commission, "Mark-II - ISO/IEC 20968:2002 Software engineering – Ml II Function Point Analysis – Counting Practices Manual," 2002.

[106] International Organization for Standardization e International Electrotechnical Commission, "NESMA - ISO/IEC 24570:2005 Software engineering – NESMA function size measurement method version 2.1 – Definitions and counting guidelines for the application of Function Point Analysis," 2005.

[107] Mountain Goat Software, "What Are Story Points?." https://www.mountaingoatsoftware.com/blog/what-are-story-points. Acedido em: 31 de Março de 2018.

[108] Dan Radigan em Atlassian Agile Coach, "The secrets behind story points and agile estimation." https://www.atlassian.com/agile/project-management/estimation. Acedido em: 31 de Março de 2018.

[109] Nigel Bevan, "Quality in use: Meeting user needs for quality," *Journal of Systems and Software*, vol. 49, no. 1, pp. 89–96, 1999.

[110] O. Vinter, P.M. Poulsen e S. Lauesen, "Experience driven software process improvement," *Software Process Improvement '96*, vol. 49, 1996.

[111] Mark Keil e Erran Carmel, "Customer-developer Links in Software Development," *Commun. ACM*, vol. 38, pp. 33–44, Maio 1995.

[112] Randolph G. Bias e Deborah J. Mayhew, *Cost-Justifying Usability: An Update for the Internet Age, Second Edition*. Interactive Technologies, Elsevier Science, 2005.

[113] C.M. Karat, "Cost-justifying human factors support on development projects," *Human Factors Society Bulletin*, vol. 35, no. 11, pp. 1–8, 1992.

A

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

Neste apêndice é apresentado o Manifesto *Agile*, assim como os princípios por detrás do manifesto. Este foi copiado integralmente do *website* http://agilemanifesto.org/ [71].

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Principles behind the Agile Manifesto

We follow these principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

B

SOFTWARE ENGINEERING MANAGEMENT - EXCERTO DO SOFTWARE ENGINEERING BODY OF KNOWLEDGE

Neste apêndice é apresentada a secção 6, *Software Engineering Management*, do Capítulo 7, *Software Engineering Management*, do *Software Engineering Body of Knowledge* (SWEBoK) [7].

6. SOFTWARE ENGINEERING MEASUREMENT

The importance of measurement and its role in better management and engineering practices is widely acknowledged (see Measurement in the Engineering Foundations KA). Effective measurement has become one of the cornerstones of organizational maturity. Measurement can be applied to organizations, projects, processes, and work products. In this section the focus is on the application of measurement at the levels of projects, processes, and work products.

This section follows the IEEE 15939:2008 standard [6], which describes a process to define the activities and tasks necessary to implement a software measurement process. The standard also includes a measurement information model.

6.1. ESTABLISH AND SUSTAIN MEASUREMENT COMMITMENT

- Requirements for measurement. Each measurement endeavor should be guided by organizational objectives and driven by a set of measurement requirements established by the organization and the project (for example, an organizational objective might be "first-to-market with new products").
- Scope of measurement. The organizational unit to which each measurement requirement is to be applied should be established. This may consist of a functional area, a single project, a single site, or an entire enterprise. The temporal scope of the measurement effort should also be considered because time series of some measurements may be

required; for example, to calibrate estimation models (see section 2.3, Effort, Schedule, and Cost Estimation).

- Team commitment to measurement. The commitment should be formally established, communicated, and supported by resources (see next item).

- Resources for measurement. An organization's commitment to measurement is an essential factor for success, as evidenced by the assignment of resources for implementing the measurement process. Assigning resources includes allocation of responsibility for the various tasks of the measurement process (such as analyst and librarian). Adequate funding, training, tools, and support to conduct the process should also be allocated.

6.2. PLAN THE MEASUREMENT PROCESS

- Characterize the organizational unit. The organizational unit provides the context for measurement, so the organizational context should be made explicit, including the constraints that the organization imposes on the measurement process. The characterization can be stated in terms of organizational processes, application domains, technology, organizational interfaces, and organizational structure.

- Identify information needs. Information needs are based on the goals, constraints, risks, and problems of the organizational unit. They may be derived from business, organizational, regulatory, and/or product objectives. They should be identified and prioritized. Then a subset of objectives to be addressed can be selected, documented, communicated, and reviewed by stakeholders.

- Select measures. Candidate measures should be selected, with clear links to the information needs. Measures should be selected based on the priorities of the information needs and other criteria such as cost of collection, degree of process disruption during collection, ease of obtaining accurate, consistent data, and ease of analysis and reporting. Because internal quality characteristics (see Models and Quality Characteristics in the Software Quality KA) are often not contained in the contractually binding software requirements, it is important to consider measuring the internal quality of the software to provide an early indicator of potential issues that may impact external stakeholders.

- Define data collection, analysis, and reporting procedures. This encompasses collection procedures and schedules, storage, verification, analysis, reporting, and configuration management of data.

- Select criteria for evaluating the information products. Criteria for evaluation are influenced by the technical and business objectives of the organizational unit. Information

products include those associated with the product being produced, as well as those associated with the processes being used to manage and measure the project.

- Provide resources for measurement tasks. The measurement plan should be reviewed and approved by the appropriate stakeholders to include all data collection procedures; storage, analysis, and reporting procedures; evaluation criteria; schedules; and responsibilities. Criteria for reviewing these artifacts should have been established at the organizational-unit level or higher and should be used as the basis for these reviews. Such criteria should take into consideration previous experience, availability of resources, and potential disruptions to projects when changes from current practices are proposed. Approval demonstrates commitment to the measurement process.

- Identify resources to be made available for implementing the planned and approved measurement tasks. Resource availability may be staged in cases where changes are to be piloted before widespread deployment. Consideration should be paid to the resources necessary for successful deployment of new procedures or measures.

- Acquire and deploy supporting technologies. This includes evaluation of available supporting technologies, selection of the most appropriate technologies, acquisition of those technologies, and deployment of those technologies.

6.3. PERFORM THE MEASUREMENT PROCESS

- Integrate measurement procedures with relevant software processes. The measurement procedures, such as data collection, should be integrated into the software processes they are measuring. This may involve changing current software processes to accommodate data collection or generation activities. It may also involve analysis of current software processes to minimize additional effort and evaluation of the effect on employees to ensure that the measurement procedures will be accepted. Morale issues and other human factors should be considered. In addition, the measurement procedures should be communicated to those providing the data. Training and support may also need to be provided. Data analysis and reporting procedures are typically integrated into organizational and/or project processes in a similar manner.

- Collect data. Data should be collected, verified, and stored. Collection can sometimes be automated by using software engineering management tools (see topic 7, Software Engineering Management Tools) to analyze data and develop reports. Data may be aggregated, transformed, or recoded as part of the analysis process, using a degree of rigor appropriate to the nature of the data and the information needs. The results of this analysis are typically indicators such as graphs, numbers, or other indications that

will be interpreted, resulting in conclusions and recommendations to be presented to stakeholders (see Statistical Analysis in the Engineering Foundations KA). The results and conclusions are usually reviewed, using a process defined by the organization (which may be formal or informal). Data providers and measurement users should participate in reviewing the data to ensure that they are meaningful and accurate and that they can result in reasonable actions.

- Communicate results. Information products should be documented and communicated to users and stakeholders.

6.4. EVALUATE MEASUREMENT

- Evaluate information products and the measurement process against specifed evaluation criteria and determine strengths and weaknesses of the information products or process, respectively. Evaluation may be performed by an internal process or an external audit; it should include feedback from measurement users. Lessons learned should be recorded in an appropriate database.

- Identify potential improvements. Such improvements may be changes in the format of indicators, changes in units measured, or reclassifcation of measurement categories. The costs and benefts of potential improvements should be determined and appropriate improvement actions should be reported.

- Communicate proposed improvements to the measurement process owner and stakeholders for review and approval. Also, lack of potential improvements should be communicated if the analysis fails to identify any improvements.

C

EARNED VALUE MANAGEMENT

Este apêndice apresenta uma tabela (*"Table 7-1. Earned Value Calculations Summary Table"*), copiada integralmente da página 267 da 6ª edição do PMBoK [11]. Nesta tabela são apresentadas as métricas utilizadas nesta análise, a sua definição, como podem ser calculadas e como interpretar as mesmas.

Earned Value Analysis

Abbreviation	Name	Lexicon Definition	How Used	Equation	Interpretation of Result
PV	Planned Value	The authorized budget assigned to scheduled work.	The value of the work planned to be completed to a point in time, usually the data date, or project completion.		
EV	Earned Value	The measure of work performed expressed in terms of the budget authorized for that work.	The planned value of all the work completed (earned) to a point in time, usually the data date, without reference to actual costs.	EV = sum of the planned value of completed work	
AC	Actual Cost	The realized cost incurred for the work performed on an activity during a specific time period.	The actual cost of all the work completed to a point in time, usually the data date.		
BAC	Budget at Completion	The sum of all budgets established for the work to be performed.	The value of total planned work, the project cost baseline.		
CV	Cost Variance	The amount of budget deficit or surplus at a given point in time, expressed as the difference between the earned value and the actual cost.	The difference between the value of work completed to a point in time, usually the data date, and the actual costs to the same point in time.	CV = EV – AC	Positive = Under planned cost Neutral = On planned cost Negative = Over planned cost
SV	Schedule Variance	The amount by which the project is ahead or behind the planned delivery date, at a given point in time, expressed as the difference between the earned value and the planned value.	The difference between the work completed to a point in time, usually the data date, and the work planned to be completed to the same point in time.	SV = EV – PV	Positive = Ahead of Schedule Neutral = On schedule Negative = Behind Schedule
VAC	Variance at Completion	A projection of the amount of budget deficit or surplus, expressed as the difference between the budget at completion and the estimate at completion.	The estimated difference in cost at the completion of the project.	VAC = BAC – EAC	Positive = Under planned cost Neutral = On planned cost Negative = Over planned cost
CPI	Cost Performance Index	A measure of the cost efficiency of budgeted resources expressed as the ratio of earned value to actual cost.	A CPI of 1.0 means the project is exactly on budget, that the work actually done so far is exactly the same as the cost so far. Other values show the percentage of how much costs are over or under the budgeted amount for work accomplished.	CPI = EV/AC	Greater than 1.0 = Under planned cost Exactly 1.0 = On planned cost Less than 1.0 = Over planned cost
SPI	Schedule Performance Index	A measure of schedule efficiency expressed as the ratio of earned value to planned value.	An SPI of 1.0 means that the project is exactly on schedule, that the work actually done so far is exactly the same as the work planned to be done so far. Other values show the percentage of how much costs are over or under the budgeted amount for work planned.	SPI = EV/PV	Greater than 1.0 = Ahead of schedule Exactly 1.0 = On schedule Less than 1.0 = Behind schedule
EAC	Estimate At Completion	The expected total cost of completing all work expressed as the sum of the actual cost to date and the estimate to complete.	If the CPI is expected to be the same for the remainder of the project, EAC can be calculated using: If future work will be accomplished at the planned rate, use: If the initial plan is no longer valid, use: If both the CPI and SPI influence the remaining work, use:	EAC = BAC/CPI EAC = AC + BAC – EV EAC = AC + Bottom-up ETC EAC = AC + [(BAC – EV)/(CPI x SPI)]	
ETC	Estimate to Complete	The expected cost to finish all the remaining project work.	Assuming work is proceeding on plan, the cost of completing the remaining authorized work can be calculated using: Reestimate the remaining work from the bottom up.	ETC = EAC – AC ETC = Reestimate	
TCPI	To Complete Performance Index	A measure of the cost performance that must be achieved with the remaining resources in order to meet a specified management goal, expressed as the ratio of the cost to finish the outstanding work to the budget available.	The efficiency that must be maintained in order to complete on plan. The efficiency that must be maintained in order to complete the current EAC.	TCPI = (BAC–EV)/(BAC–AC) TCPI = (BAC–EV)/(EAC–AC)	Greater than 1.0 = Harder to complete Exactly 1.0 = Same to complete Less than 1.0 = Easier to complete Greater than 1.0 = Harder to complete Exactly 1.0 = Same to complete Less than 1.0 = Easier to complete