UNIVERSIDADE
LUSÓFONA
Centro Universitário de Lisboa

Escola de Comunicação, Arquitetura, Artes e Tecnologias da Informação (ECATI)

Mestrado em Engenharia Informática e Sistemas de Informação

BENEFÍCIOS DA UTILIZAÇÃO DO AGILE EM LARGA ESCALA

Dissertação de Mestrado apresentada a provas públicas para a obtenção do grau de mestre em Engenharia Informática e Sistemas de Informação**,** orientada por Professor Doutor Paulo Jorge Tavares Guedes.

Rúben Teixeira Batista Tomazio

2024

UNIVERSIDADE
LUSÓFONA
Centro Universitário de Lisboa

Escola de Comunicação, Arquitetura, Artes e Tecnologias da Informação (ECATI)

Mestrado em Engenharia Informática e Sistemas de Informação

BENEFÍCIOS DA UTILIZAÇÃO DO AGILE EM LARGA ESCALA

Dissertação defendida em provas públicas na Universidade Lusófona, Centro Universitário de Lisboa no dia 14/11/2024, perante o júri, nomeado pelo Despacho de Nomeação n.º: 1002/2024, de 28 de outubro, com a seguinte composição:

Presidente: Professor Doutor João Carlos Palmela Pinheiro Caldeira

Arguente: Professor Doutor Rúben Filipe Pereira (ISCTE/IUL)

Orientador: Professor Doutor Paulo Jorge Tavares Guedes

Rúben Teixeira Batista Tomazio

2024

Epígrafe

"Conhecimento não é aquilo que você sabe, mas o que você faz com aquilo que você sabe."

Aldous Huxley

Agradecimentos

Agradeço ao meu orientador Professor Doutor Paulo Guedes pela sua orientação, apoio e dedicação na realização desta dissertação.

Resumo

Como resposta aos desafios que as empresas se deparavam no início dos anos 90 algumas organizações começaram a ajustar a metodologia tradicional de desenvolvimento de *software* que utilizavam de modo a atender às suas necessidades. Desta forma surgiu a metodologia Agile com o principal objetivo de ser mais produtivo e deste modo aumentar a satisfação do cliente, melhorar a capacidade de adaptação às mudanças, diminuir o desperdício, garantir uma boa comunicação entre equipas e garantir entrega rápida de *software*.

O foco deste trabalho está na análise de quatro empresas de grande dimensão, Spotify, Google, Cisco e Amazon, que implementaram Agile com sucesso nos seus projetos e na análise dos seus resultados. Em cada caso é descrito o enquadramento da empresa, a metodologia que foi aplicada e os resultados obtidos.

Alguns dos principais resultados obtidos dos casos de estudo foi a diminuição em 50% dos ciclos de entrega e a diminuição em 25% dos defeitos de garantia de qualidade. Em suma, a aplicação de metodologias Agile elevou o patamar das organizações ao aumentar a produtividade e diminuir o desperdício nos custos e no tempo.

Palavras-chave: Agile, Larga Escala, Gestão de Projeto, Sistemas de Informação

Abstract

In response to the challenges that companies faced in the early 90s, some organizations began to adjust the traditional software development methodology they used in order to meet their needs. In this way the Agile methodology emerged with the main objective of being more productive and thus increasing customer satisfaction, improving the ability to adapt to changes, reducing waste, ensuring good communication between teams and ensuring fast software delivery.

The focus of this work is on the analysis of four large companies, Spotify, Google, Cisco and Amazon, that have successfully implemented Agile in their projects and the analysis of their results. In each case, the company's framework, the methodology that was applied and the results obtained are described.

Some of the main results obtained from the case studies were a 50% reduction in delivery cycles and a 25% reduction in quality assurance defects. In short, the application of Agile methodologies has raised the bar for organizations by increasing productivity and reducing waste in costs and time.

Keywords: Agile, Large Scale, Project Management, Information Systems

Abreviaturas, siglas e símbolos

API - Application Programming Interface

ART - Agile Release Train

ATT - Agile Tiger Team

AWS - Amazon Web Services

CCG - Collaboration and Communications Group

CEO - Chief Executive Officer

CI/CD - Continuous Integration/Continuous Delivery

CSIT - Cloud and Software IT

DevOps - Development and Operations

DSDM - Dynamic Systems Development Method

EMI - Electric and Musical Industries

ESB - Enterprise Service Bus

FAQs - Frequently Asked Questions

FDD - Feature Driven Development

IC - Integração Contínua

Inc - Incorporated

LLC - Limited Liability Company

Ltd - Limited

MVP - Minimum Viable Product

ONA - Organizational Network Analysis

PI - Program Increment

R&D - Research and Development

RAD - Rapid Application Development

SaaS - Software as a Service

SAFe - Scaled Agile Framework

STO - Single-Threaded Owner

TAP - Test Automation Platform

TBD - Trunk-Based Development

TDD - Test Driven Development

TI - Tecnologia da Informação

UI - User Interface

XP - Extreme Programming

Índice

Índice de Tabelas

Índice de Figuras

Capítulo 1 - Introdução

Algumas das alterações mais significativas no desenvolvimento de *software* surgiram quando os programadores começaram a trabalhar em aplicações, principalmente nas *start-ups*, isso levantou muitas questões em relação às metodologias utilizadas, por essa razão procuraram-se maneiras de se ser mais produtivo. A metodologia Agile surgiu em 2001, quando um grupo de 17 programadores de *software*, entre eles Jeff Sutherland, Martin Fowler, Ken Schwaber, Kent Beck e Ron Jeffries, se apercebeu que estava a aplicar nos seus projetos uma metodologia diferente das metodologias tradicionais (Beck et al., 2001) (Sommerville, 2011). Este grupo criou o manifesto Agile, que consiste em 4 valores e 12 princípios, que documenta como um processo de desenvolvimento de *software* moderno deve ser realizado (Beck et al., 2001). A metodologia Agile é uma *framework* que divide o projeto em várias fases e ajuda as equipas pelos ciclos de planeamento, execução e avaliação, essas fases são chamadas de *sprints*.

Agile começou por ser um método Lean de desenvolvimento de *software* com o objetivo de desenvolver código de melhor qualidade em menos tempo. Ao longo dos anos tem sido usado por todo o tipo de organizações, desde universidades a empresas de marketing. Desde a sua criação até aos dias de hoje, a metodologia Agile evolui bastante tornando-se no método de desenvolvimento de *software* mais usado no mundo (Hoda et al., 2018).

Agile tem sido cada vez mais adotado por empresas de todas as dimensões, principalmente em momentos mais complicados. Em 2009, devido à crise económica várias empresas implementaram Agile nos seus projetos de maneira a serem mais produtivas. Em 2020, a pandemia COVID-19 obrigou muitas empresas a adotarem o teletrabalho e a arquitetura em nuvem o que originou um aumento significativo na necessidade de desenvolvimento rápido e flexível. Segundo os dados do *15th State of Agile report*, entre 2020 e 2021, a utilização de Agile em desenvolvimento de *software* aumentou de 37% para 84% (Knaster, 2021).

As metodologias Agile são os processos que suportam a filosofia Agile, cada metodologia tem os seus princípios, práticas, funções, ciclo de vida, vantagens e desvantagens. Segundo Organize Agile, os principais benefícios da utilização do Agile reportados pelas empresas são a maior agilidade e flexibilidade com 83%, os resultados financeiros com 68% e a cultura organizacional mais produtiva com 61% (Taylor, 2023).

Todas as metodologias desenvolvem os seus produtos em iterações e processos incrementais. Existem muitas metodologias Agile como Scrum, Extreme Programming, Crystal e Feature Driven Development (Al-Saqqa et al., 2020). Estas metodologias já existiam antes do aparecimento do Agile, mas só depois da criação do manifesto Agile é que se tornaram coesas e fundamentadas.

Nos primeiros anos do Agile, XP era a metodologia mais utilizada com práticas que ainda hoje se utilizam como Test Driven Development (TDD), *user stories* e programação em pares, mas com o passar dos anos foi perdendo protagonismo (Sommerville, 2011). Atualmente, Scrum é a metodologia mais popular devido ao seu principal foco ser a gestão de projetos (Al-Saqqa et al., 2020). As melhores equipas usam as duas metodologias em conjunto, escolhem técnicas de cada metodologia e usam-nas de acordo com as necessidades dos projetos. XP complementa Scrum de modo a obter um melhor desempenho e Scrum complementa o XP a escalar melhor. Em organizações de grande escala Scrum, SAFe e modelo Spotify são as principais metodologias (Edison et al., 2022).

Implementar Agile em grandes organizações pode, por vezes, ser um grande desafio por causa da resistência à mudança dos *stakeholders* e da necessidade de todas as equipas dos vários departamentos estarem alinhadas e compreendam bem os princípios e práticas Agile. Para além disso, é importante que os funcionários procurem sempre melhorar no seu trabalho o que pode implicar correrem alguns riscos, e recebam feedback com uma mentalidade aberta (Edison et al., 2022). Apesar das dificuldades que possam ocorrer, as empresas reconhecem os ótimos resultados da utilização desta metodologia como o aumento da rapidez, a resiliência e a colaboração das equipas (Edison et al., 2022).

Num mundo onde as TI estão em constante evolução com o propósito de cumprir os objetivos das organizações e os sistemas de TI que suportam os negócios tornaram-se essenciais e mais complexos é cada vez mais importante que as empresas façam uma escolha educada sobre que metodologia é a mais indicada para o seu negócio (Verwijs & Russo, 2023). Cada empresa deve escolher a metodologia tendo em conta a sua estrutura, tarefas e produtos. Em alguns casos é preferível usar várias metodologias ou criar uma metodologia personalizada que seja adequada especificamente às suas necessidades (Al-Saqqa et al., 2020) (Edison et al., 2022). A satisfação dos *stakeholders* e a eficácia das equipas devem ser regularmente controladas para identificar as áreas de melhoria e fornecer formação de modo a expandir a experiência das equipas com as metodologias (Verwijs & Russo, 2023).

Atualmente, Agile é uma metodologia de desenvolvimento muito conhecida e é a abordagem escolhida por muitas equipas de desenvolvimento, especialmente aquelas que querem criar um ambiente de entrega contínuo. Segundo o Zippia, em 2022, pelo menos 71% das empresas nos EUA usam Agile, os projetos Agile têm uma taxa de sucesso de 64% enquanto os projetos Waterfall têm apenas uma taxa de sucesso de 49% e as empresas que adotaram Agile tiveram um crescimento médio de 60% em receita e lucro (Flynn, 2022).

Nesta dissertação, ao recorrer à metodologia de casos de estudo, pretende-se realizar um estudo sobre a utilização do Agile em grandes organizações, avaliando as suas metodologias, impactos e benefícios. Neste contexto, pretende-se responder à diversas questões desta área, como, Quais os benefícios e desafios do Agile em larga escala? e Qual o impacto do Agile na cultura, na organização, na forma de trabalhar e nos resultados das grandes empresas?

Na revisão de literatura deste trabalho recorreu-se a várias fontes de informação como livros, artigos e documentos académicos. Esta dissertação desenvolve-se em seis capítulos: no segundo capítulo é abordado o domínio, o enquadramento histórico da metodologia Agile, as suas vantagens, as suas desvantagens, a comparação entre as metodologias Waterfall e Agile, e o Agile em larga escala; no terceiro capítulo explicita-se o método de pesquisa e o tipo de investigação adotados para a realização deste trabalho; no quarto capítulo apresentam-se os quatro casos de estudo, em cada caso apresenta-se o enquadramento da empresa, a transformação Agile e as considerações finais; e no quinto capítulo apresenta-se uma análise dos resultados obtidos de cada caso. Por fim, apresenta-se a conclusão onde se reflete sobre os casos de estudo e se indica limitações experienciadas e sugestões de trabalhos futuros na área.

Capítulo 2 - Estado da Arte

2.1 Agile

O início da mudança no desenvolvimento de *software* aconteceu no início dos anos 90 quando na maioria dos projetos o prazo de entrega e o orçamento eram ultrapassados, como consequência disso quando o projeto se encontrava finalizado já estava desatualizado e fora de contexto do mercado (Alam et al., 2017). Durante estes anos os programadores começaram a desenvolver aplicações mais complexas, muito desse trabalho foi realizado em *start-ups* onde as equipas eram pequenas e muitas das vezes não tinham uma educação tradicional em informática. O surgimento de tantos problemas levou muitos a questionar as metodologias clássicas e a procurar formas de ser mais eficiente, era inaceitável ter de escrever documentação detalhada antecipadamente e era necessário um processo mais iterativo e colaborativo.

A filosofia das metodologias clássicas defende que os requisitos de um projeto de *software* são definidos no início do projeto, antes das fases de design e desenvolvimento, e não podem ser mais alterados. Esta filosofia não é sempre aplicável, por vezes, é necessária uma filosofia mais flexível onde os programadores podem realizar alterações tardias nos requisitos (Abrahamsson et al., 2017). As metodologias clássicas são demasiado mecânicas para serem utilizadas em detalhe (Nandhakumar & Avison, 1999). Ao longo da década de 90, muitas equipas de desenvolvimento de *software* começaram a mudar a sua abordagem de planeamento e entrega de novos produtos e surgiram novos métodos de desenvolvimento de *software* como Scrum, XP, Crystal, Pragmatic Programming, RAD, DSDM e FDD (Beck, 1999) (Cockburn, 2000) (Alam et al., 2017).

Em 2000, um grupo de 17 programadores com *backgrounds* diferentes, reuniu-se nas montanhas Wasatch em Oregon, EUA, para discutir e refletir como poderiam diminuir os tempos de desenvolvimento para entregar novos *softwares* no mercado. Pretendiam dessa forma acelerar as vantagens dos utilizadores de maneira a resolver os problemas de adequação do produto ao mercado e obter rápido feedback para confirmar a utilidade do novo *software* e continuar a melhorá-lo (Sommerville, 2011).

Em fevereiro de 2001, o grupo voltou a reunir-se, num resort de ski em Snowbird, Utah, EUA, com o objetivo de expandir ainda mais o seu progresso e encontrar uma solução para os principais problemas de desenvolvimento da época. Em três dias o grupo criou o Manifesto para Desenvolvimento Agile de *Software*, mais conhecido como Manifesto Agile, composto por 4 valores, que serão descritos na secção 2.1.1, e 12 princípios, que serão descritos na secção 2.1.2 (Beck et al., 2001). Metodologias como Scrum, Kanban e XP já existiam antes da criação do Manifesto, mas só depois das reuniões é que o grupo se apercebeu que apesar de serem metodologias diferentes tinham os mesmos princípios fundamentais (Abbas et al., 2008).

O grupo também criou uma organização chamada Agile Alliance que tem como objetivos partilhar informações sobre Agile, continuar a evoluir a abordagem para responder às necessidades que se encontram em constante mudança e fornecer recursos para equipas que querem adotar a abordagem. Ao longo dos anos, as equipas que adotaram as metodologias contribuíram para o desenvolvimento do Agile ao introduzir novas práticas como reuniões diárias e retrospetivas.

Agile é um conjunto de metodologias e práticas que têm como objetivo ajudar na gestão de projetos e no desenvolvimento de produtos e têm como principais prioridades a capacidade de adaptação às mudanças, a satisfação do cliente, a boa comunicação entre elementos da mesma equipa e a entrega rápida de *software* (Abbas et al., 2008) (Abrahamsson et al., 2017).

Os processos Agile são leves e suficientes, ou seja, são facilmente manobráveis e satisfazem os mínimos requeridos (Cockburn, 2001). Cada processo Agile é dividido por 3 etapas: preparação, planeamento do *sprint* e *sprint*. Na etapa da preparação o *product owner* cria o *backlog* do produto, o *backlog* de funcionalidades que têm de ser incluídas no produto final, e a equipa de desenvolvimento faz uma estimativa de quanto tempo irá demorar a realizar cada funcionalidade. Na etapa do planeamento do *sprint* a equipa decide quais as funcionalidades do *backlog* do produto que irá trabalhar durante o *sprint*, um *sprint* é um período normalmente de duas semanas, onde a equipa de desenvolvimento deve atingir um objetivo específico. Na etapa do *sprint* a equipa trabalha nas tarefas do *backlog* do produto e no fim do *sprint* a equipa tem de ter concluído todas as funcionalidades do *backlog* do produto, caso isso não aconteça as funcionalidades são transferidas para o próximo *sprint*. No fim a equipa realiza duas reuniões, uma reunião de revisão do *sprint* onde é apresentada uma

demo das funcionalidades concluídas ao *product owner* e aos *stakeholders*, e uma reunião retrospetiva onde se discute os aspetos positivos e negativos da realização do *sprint* e como podem melhorar no próximo *sprint*.

Existem imensos benefícios para as equipas que decidem adotar Agile (Masood & Farooq, 2017):

- Aumento da satisfação do cliente - o cliente é envolvido no processo de desenvolvimento e fornece feedback em cada fase do projeto
- Melhor comunicação entre o *product owner* e a equipa de desenvolvimento
- Aumento da flexibilidade - Agile é mais flexível do que outras metodologias, as equipas têm mais facilidade de fazer alterações no projeto
- Reduz riscos - os gestores de projeto dividem os projetos em *sprints* para conseguirem responder melhor às necessidades
- Reduz os custos de retrabalho

Apesar dos vários benefícios existem também alguns desafios (Masood & Farooq, 2017):

- Controlo limitado - o *project owner* pode ter dificuldades em ter controlo sobre o projeto todo, isto pode ser um grave problema principalmente para projetos que têm de cumprir orçamentos e prazos rigorosos
- Colaboração entre equipas - pode ser desafiante quando existem equipas remotas que não estão habituadas a trabalhar juntas, isto pode causar atrasos no projeto
- Pouca documentação - alguns projetos necessitam de muita documentação, na metodologia Agile normalmente não se escreve muita documentação
- Duração de projeto - a duração de um projeto costuma ser mais longa ao usar a metodologia Agile do que outras metodologias

O ciclo de vida de desenvolvimento de *software* de um projeto Agile está dividido em pequenas partes chamadas de iterações que afetam cada uma das fases de desenvolvimento (Leau et al., 2012), é normalmente composto por sete fases: conceito, começo, iteração, lançamento, entrega, manutenção e reforma. Cada metodologia Agile tem o seu próprio ciclo de vida que leva a mudanças tecnológicas e na gestão da organização (Bhalerao et al., 2009). Na figura 1 está demonstrado o ciclo de vida de um projeto Agile.

Na fase do conceito o *product owner* discute com o cliente quais são os principais requisitos e prepara a documentação para descrevê-los, nisto inclui-se as funcionalidades que serão suportadas e os resultados propostos. Para além disso, o *product owner* faz uma estimativa do tempo e do custo de potenciais projetos para decidir se um projeto é viável (Bhalerao et al., 2009) (Gheorghe et al., 2020).

Na fase do começo o *product owner* escolhe a melhor equipa para o projeto e fornece-lhes as ferramentas e recursos necessários. Nesta fase a equipa constrói a arquitetura do projeto e cria o modelo da interface do utilizador e são realizadas reuniões regulares com os *stakeholders* para assegurar que todos os requisitos são incorporados no processo de design (Bhalerao et al., 2009).

Na fase da iteração, a fase mais longa do ciclo, o objetivo é construir as funcionalidades mais básicas do produto até o final do primeiro *sprint*. As equipas trabalham em conjunto tendo em conta os requisitos estabelecidos e o feedback do cliente para transformar o design em código (Bhalerao et al., 2009).

Na fase do lançamento a equipa de garantia de qualidade executa vários testes para garantir que o *software* está funcional. Se forem encontrados *bugs* no código o programador resolve-os o mais rapidamente possível. Também, nesta fase é realizado a formação dos utilizadores o que significa que é necessário escrever mais documentação (Gheorghe et al., 2020).

Na fase da entrega existem duas subfases a pré-entrega e a produção. Na subfase pré-entrega são executados testes extra e verifica-se se os requisitos funcionais e não funcionais estão prontos para serem lançados em produção. Na subfase produção o produto é entregue ao cliente (Bhalerao et al., 2009) (Gheorghe et al., 2020).

Na fase da manutenção a equipa de desenvolvimento fornece apoio para manter o sistema a funcionar, resolver novos *bugs*, dar formação adicional aos utilizadores de forma a garantir que eles utilizam o produto corretamente, e atualizar o produto com atualizações e recursos adicionais (Bhalerao et al., 2009) (Gheorghe et al., 2020).

Existem duas razões para um produto ir para a fase da reforma, o produto é substituído ou o produto tornou-se incompatível com a organização. Caso alguma destas situações ocorra a equipa de desenvolvimento realiza as atividades de fim de vida do produto,

remove o apoio para o produto, notifica os utilizadores de que o produto vai ser descontinuado e, se houver substituição, os utilizadores serão migrados para o novo *software*.

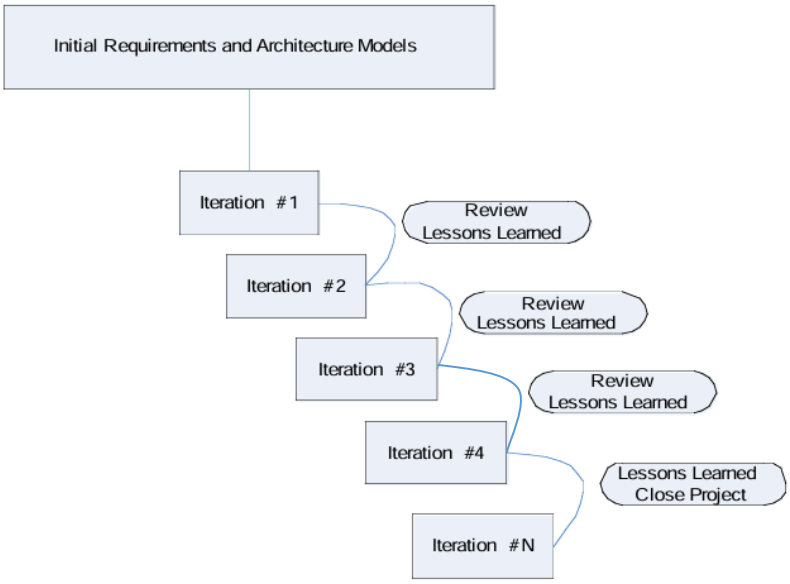


Figura 1 - Ciclo de Vida de Projeto Agile
Fonte: (Adaptado de Hass, 2007)

2.1.1 Valores Agile

Do manifesto Agile surgiram quatro valores de gestão de projetos (Beck et al., 2001):

Indivíduos e interações acima de processos e ferramentas - são as pessoas que respondem às necessidades de negócio e conduzem o processo de desenvolvimento, se os processos ou as ferramentas conduzirem o desenvolvimento a equipa não irá responder tão bem às mudanças e às necessidades do cliente. A comunicação é um bom exemplo da diferença que existe entre valorizar os indivíduos em vez de valorizar os processos, nos indivíduos a comunicação é fluida e acontece quando surge uma necessidade enquanto nos processos a comunicação é agendada e requer conteúdo específico.

***Software* funcional sobre documentação completa** - nas metodologias tradicionais gasta-se demasiado tempo na documentação dos produtos. A maioria dessa documentação é realizada para as fases de desenvolvimento e entrega e distribui-se em vários tópicos como as especificações técnicas, requisitos técnicos, planos de teste e documentos de design. Agile não elimina toda a documentação, mas reduz bastante ao ponto de permitir que os programadores trabalhem sem se preocuparem com pormenores desnecessários.

Colaboração do cliente em vez de negociação de contratos - a negociação de contratos é o período em que o cliente e o gestor de produto definem os detalhes de uma entrega de produto. Na metodologia Waterfall o cliente apenas está envolvido no processo de desenvolvimento antes do início do projeto. Na metodologia Agile o cliente colabora em todo o processo de desenvolvimento o que facilita os programadores a resolverem os seus pedidos. Este envolvimento do cliente acontece de algumas formas diferentes como o cliente estar presente em algumas *demos* do produto e o cliente participar nas reuniões das equipas, garantindo diariamente que o produto corresponde às suas expetativas.

Responder à mudança ao invés de seguir um plano - o desenvolvimento de *software* tradicional considera a mudança uma despesa, portanto sempre que possível deve ser evitada. Agile considera que as mudanças melhoram sempre um projeto ao fornecerem valor adicional ao projeto, as prioridades podem ser alteradas em cada iteração e podem ser adicionadas novas funcionalidades nas próximas iterações.

2.1.2 Princípios Agile

O manifesto Agile enumera doze princípios (Beck et al., 2001):

Satisfazer os clientes por meio de melhorias e entregas antecipadas e contínuas - se os clientes receberem novas atualizações regularmente os programadores irão receber mais

feedback e dessa forma poderão aplicar as alterações necessárias à medida que o projeto avança e isso irá originar clientes mais satisfeitos e receitas mais recorrentes.

Aceitar a mudança de requisitos - a adaptabilidade é muito importante na metodologia Agile, os processos iterativos têm de ser flexíveis. Num mundo de negócios cada vez mais competitivo é essencial desenvolver projetos em que seja possível, em qualquer momento, realizar alterações.

Entregar valor com frequência - garantir a entrega regular de *software* funcional aos clientes.

Colaboração entre os *stakeholders* do negócio e os programadores ao longo do projeto - a colaboração é fundamental no Agile, o objetivo é que as equipas colaborem com mais frequência. Quando as equipas e o negócio estão alinhados são tomadas melhores decisões.

Construir projetos em torno de indivíduos motivados - os projetos apresentam melhores resultados quando as equipas estão comprometidas, motivadas e trabalham juntas para atingir os objetivos. O gestor de projeto tem a função de dar apoio à equipa no que for necessário e de manter a equipa motivada ao longo de todo o projeto.

Habilitar interações cara a cara - a comunicação é melhor quando as equipas trabalham no mesmo local. Um exemplo disso são as reuniões de planeamento de iterações. Estas reuniões são eficazes porque os membros da equipa falam cara a cara o que torna a compreensão de ideias mais rápida e clara. Esta prática é melhor do que a prática aplicada na metodologia tradicional onde se utiliza uma folha de registos de tarefas.

***Software* funcional é a principal medida do progresso** - a entrega de *software* funcional ao cliente é o fator que mede o progresso, logo o desenvolvimento de *software* deve ser sempre a principal prioridade.

Manter um ritmo de trabalho sustentável - alguns processos dos projetos podem e devem ser rápidos, mas não devem ser demasiado rápidos ao ponto das equipas e dos *stakeholders* ficarem exaustos. O objetivo é manter a sustentabilidade ao longo de todo o projeto.

Excelência contínua aumenta a agilidade - O desenvolvimento contínuo de um bom trabalho permite que as equipas avancem mais rapidamente no projeto. Para garantir que o produto final é o que o cliente deseja é fundamental, durante todo o projeto, realizar revisões dos requisitos técnicos e de design.

Simplicidade é essencial - por vezes, a solução mais simples é a melhor solução. Agile procura não complicar demasiado e encontrar respostas simples para os problemas mais complexos. Agile simplifica o trabalho ao exigir pouca documentação, isso poupa tempo e faz com que o foco da equipa esteja apenas no desenvolvimento o que leva a entregas antecipadas.

Equipas auto-organizadas geram mais valor - equipas proativas são ativos valiosos para as organizações porque se esforçam diariamente para criar mais valor.

Refletir e ajustar regularmente a maneira de trabalhar para aumentar a eficácia - é essencial que haja um tempo dedicado para que as equipas reflitam sobre o seu desempenho e ajustem os seus comportamentos para os próximos projetos, um exemplo disto são as reuniões retrospetivas. O facto do Agile dividir os projetos em pequenas iterações é uma grande vantagem porque permite que no final de cada iteração se possa refletir sobre o

trabalho realizado em vez de só se poder refletir no final do projeto como é o caso na metodologia Waterfall.

2.1.3 Metodologias Agile

Existem várias metodologias Agile, abaixo aborda-se algumas das mais usadas, as suas abordagens, vantagens e desvantagens.

Scrum

Scrum é a metodologia Agile mais utilizada e é caracterizada por ciclos, conhecidos como *sprints* que duram de 2 a 4 semanas. A palavra Scrum é originária do rugby onde o objetivo de toda a equipa é rumar com a bola numa direção. Nesta metodologia a equipa é liderada pelo *scrum master* que tem como função gerir o processo e remover todos os obstáculos para que o resto da equipa possa trabalhar normalmente.

As equipas Scrum têm todos os dias uma pequena reunião de 15 minutos, chamada de *daily scrum*, em que debatem as tarefas que estão a realizar, os obstáculos que enfrentaram e outros temas que possam afetar o desenvolvimento do projeto. Scrum é mais usado na gestão do desenvolvimento de produtos de *software*, mas também é usado com sucesso em outros contextos relacionados a negócios (Al-Saqqa et al., 2020).

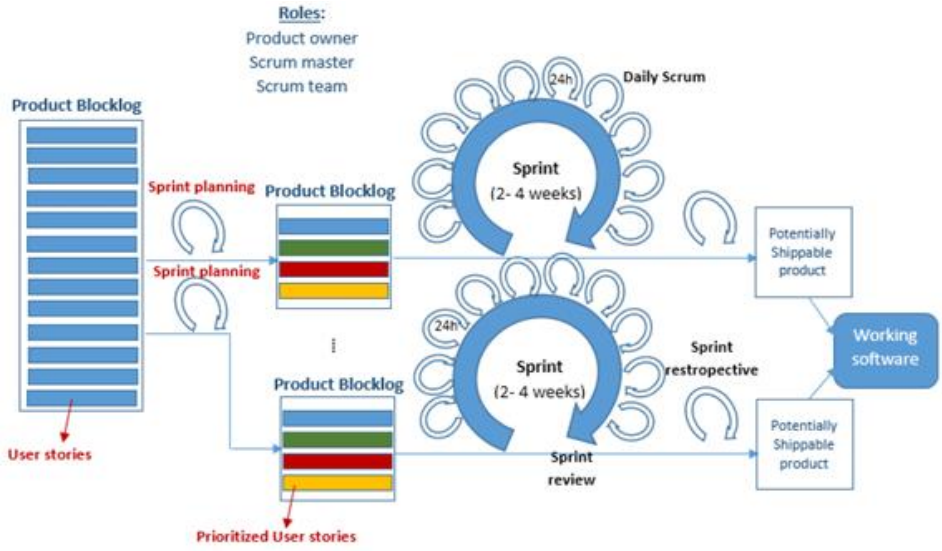


Figura 2 - Ciclo de Vida do Scrum
Fonte: (Al-Saqqa et al., 2020)

Vantagens (Al-Saqqa et al., 2020)

- Transparência
- Motivação da equipa
- Foco na qualidade
- Reorganização das prioridades
- Processos simples
- Comunicação

Desvantagens (Al-Saqqa et al., 2020)

- Perda do foco no projeto como um todo
- Papel de cada programador não está bem definido

Extreme Programming

A metodologia XP é baseada na simplicidade (Beck, 1999). O XP é composto por cinco valores: comunicação, simplicidade, feedback, coragem e respeito, e a sua principal prioridade é a satisfação do cliente (Alam et al., 2017). No XP o *software* é testado desde o início do projeto, desta forma é recolhido feedback para melhorar o desenvolvimento. O XP também promove várias práticas Agile como por exemplo programação em pares.

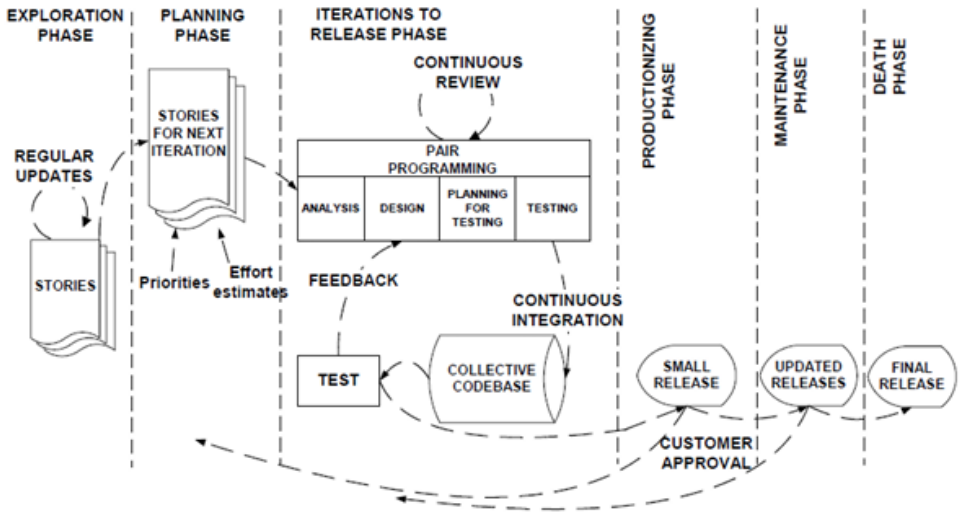


Figura 3 - Ciclo de Vida do Extreme Programming

Fonte: (Al-Saqqa et al., 2020)

Vantagens (Alam et al., 2017) (Al-Saqqa et al., 2020)

- Simplicidade do código
- Eleva o talento da equipa
- Promove um bom ambiente de trabalho

Desvantagens (Alam et al., 2017) (Al-Saqqa et al., 2020)

- Dificuldade de adoção para equipas que trabalham em diferentes locais
- Falta de atenção no design

SAFe

A metodologia SAFe foi desenvolvida por Dean Leffingwell em 2012 com o principal objetivo de escalar práticas Agile em grandes organizações. SAFe procura ajudar na gestão, organização e controlo do processo de desenvolvimento em projetos com muitas equipas. Algumas características importantes do SAFe são o foco no cliente, na entrega contínua e no design *thinking* (Marinho et al., 2021). Esta metodologia está dividida em três partes: equipa, programa e portfólio. Cada parte tem as suas atividades e processos de integração (Almeida & Espinheira, 2021).

Vantagens (Almeida & Espinheira, 2021) (Marinho et al., 2021)

- Escalabilidade
- Melhor tempo para obtenção de valor
- Alinhamento com os objetivos de negócio
- Agilidade
- Planeamento de PI e gestão de dependências

Desvantagens (Almeida & Espinheira, 2021) (Marinho et al., 2021)

- Demasiadas terminologias
- Abordagem *top-down*
- Não é recomendável para *start-ups*

Kanban

Kanban é uma abordagem visual do Agile, a palavra é de origem japonesa e está associada ao termo *just in time*. As equipas usam um quadro ou uma tabela, chamado de *kanban board*, como está ilustrado na figura 4, que está dividido em colunas para representar onde cada tarefa se encontra no processo de desenvolvimento.

Cada tarefa é representada por um cartão, à medida que as equipas trabalham nas tarefas as equipas vão movendo os cartões de uma coluna para outra que represente a fase em que a tarefa se encontra. Os quadros são compostos por pelo menos três colunas: a coluna das tarefas que se irão realizar, a coluna das tarefas que estão a ser realizadas e a coluna das tarefas que estão concluídas. A figura abaixo tem uma coluna extra, a coluna dos testes.

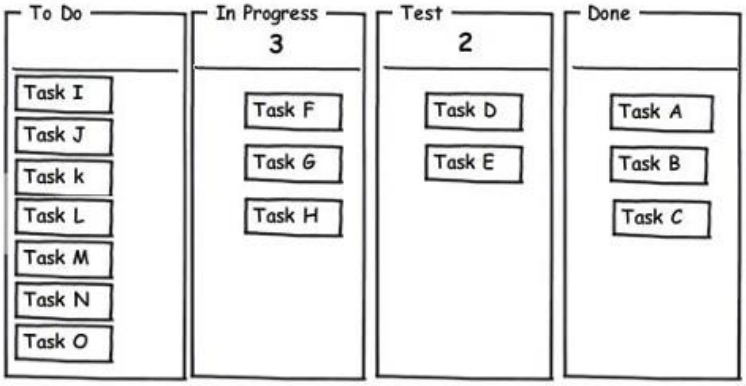


Figura 4 - Kanban Board
Fonte: (Kirovska & Koceski, 2015)

Esta metodologia é muito útil e exige transparência e comunicação para que todas as equipas identifiquem obstáculos e visualizem o trabalho que está a ser realizado. Além disso, este método é mais adequado para processos que sofrem pequenas alterações. O principal objetivo do Kanban é maximizar a produtividade através da redução do tempo dos processos. O Kanban implementado da maneira correta é um processo com poucos custos (Wakode et al., 2015) (Anderson & Carmichael, 2016).

Vantagens (Anderson & Carmichael, 2016)

- Visualização de todas as tarefas
- Foco na duração da tarefa desde o início até ao fim, através do *kanban board*
- Entregas contínuas
- Maior flexibilidade

- Limitação do número de tarefas que estão a ser realizadas
- Colaboração e Cooperação
- Simples de implementar

Desvantagens (Anderson & Carmichael, 2016)

- Má interpretação do *kanban board*
- Atrasos das tarefas

Feature Driven Development

FDD surgiu em 1997 e foi desenvolvido por Jeff De Luca quando ele se apercebeu que esta metodologia era a melhor para desenvolver grandes projetos num determinado período. O principal objetivo do FDD é gerir o desenvolvimento de *software* baseado na lista de requisitos das necessidades de negócio (Al-Saqqa et al., 2020).

O ciclo de vida do FDD, ilustrado na figura 5, é composto por cinco processos: desenvolvimento do modelo geral, construção da lista de funcionalidades, criar um plano para cada funcionalidade, design por funcionalidade e construir por funcionalidade. Existem seis papéis fundamentais no FDD: gestor de projeto, programador chefe, gestor de desenvolvimento, arquiteto chefe, especialista do domínio e proprietário da classe. Capa papel pode ser feito por mais de uma pessoa e cada membro da equipa pode ter vários papéis (Anwer et al., 2017) (Al-Saqqa et al., 2020).

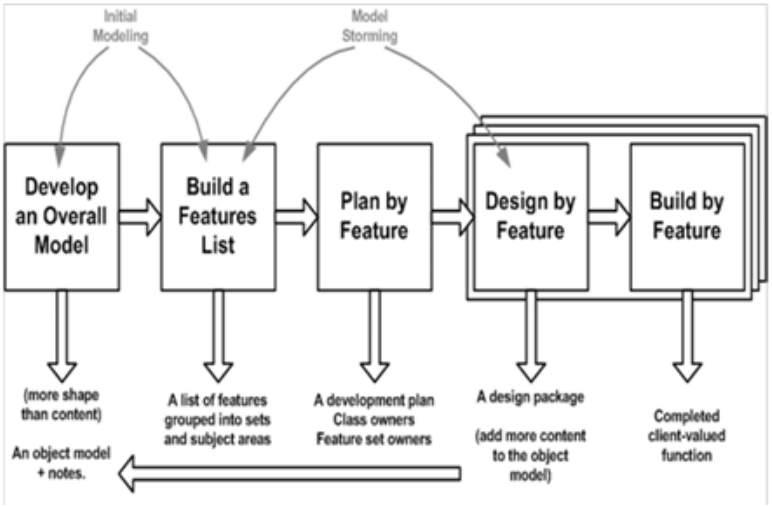


Figura 5 - Ciclo de Vida do Feature Driven Development

Fonte: (Anwer et al., 2017)

Vantagens (Anwer et al., 2017) (Al-Saqqa et al., 2020)

- Maior flexibilidade
- Entregas rápidas e frequentes
- Melhor comunicação
- Maior Foco

Desvantagens (Anwer et al., 2017) (Al-Saqqa et al., 2020)

- Controlo limitado
- Complexidade
- Requer uma equipa especializada, especialmente em design e modelagem

Crystal

Crystal é uma metodologia Agile que foi desenvolvida por Alistair Cockburn no início da década de 90. Crystal é um conjunto de metodologias de desenvolvimento de *software* que podem ser utilizadas para todo o tipo de projetos tendo em conta a complexidade, tamanho do projeto, tamanho da equipa, tecnologia disponível, criticidade e nível de habilidade. Esta metodologia concentra-se na comunicação e colaboração dos indivíduos em vez de apenas nos processos e a duração de cada iteração é no máximo 4 meses (Alam et al., 2017) (Anwer et al., 2017).

A família de metodologias Crystal está representada na figura 6 tendo em conta dois parâmetros, o tamanho do projeto e a criticidade do sistema. Em relação ao tamanho, os projetos estão divididos por cores, os projetos brancos são de pequena dimensão, os amarelos são de média dimensão, os laranjas são de grande dimensão e os vermelhos são de muito grande dimensão. Em relação à criticidade, os projetos estão divididos por letras, a letra C significa o conforto, a letra D significa o dinheiro discricionário, a letra E significa o dinheiro essencial e a letra L significa a vida. Cada metodologia representada na figura tem as suas funções, práticas, técnicas e produtos de trabalho, contudo têm alguns valores e regras em comum (Anwer et al., 2017).

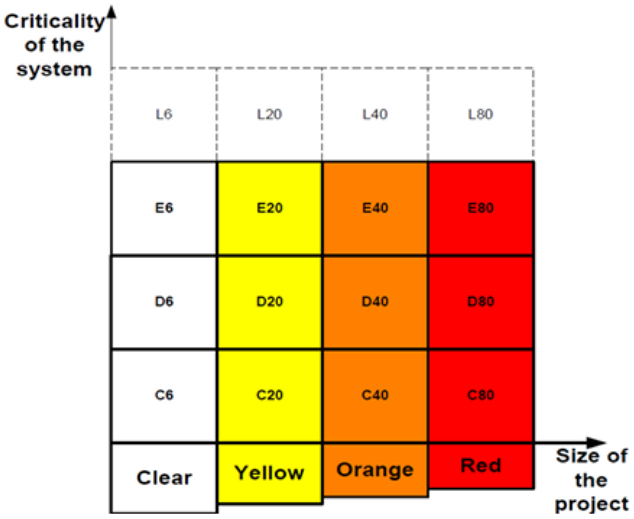


Figura 6 - Família das Metodologias Crystal

Fonte: (Anwer et al., 2017)

Vantagens (Anwer et al., 2017)

- Bom controlo de risco
- Satisfação do cliente
- Boa comunicação entre a equipa
- Aplicabilidade para diferentes projetos
- Melhor eficiência

Desvantagens (Anwer et al., 2017)

- Mudança cultural
- Necessário a adesão dos líderes

Lean

A metodologia Lean vem do Lean Manufacturing, criado pela Toyota, e é aplicada principalmente ao desenvolvimento de *software*. Lean obriga a equipa a remover as atividades que não fornecem valor ao produto final, trabalha na construção de soluções simples, apresenta-as aos clientes e usa os comentários deles sobre o produto para melhorá-lo durante o projeto (Poppendieck & Poppendieck, 2003) (Cawley et al., 2010).

Lean tem sete princípios essenciais: eliminar as coisas que não interessam, desenvolvimento de qualidade, criar conhecimento, comprometimentos diferenciados, entrega rápida, respeitar a equipa e otimizar o todo.

Vantagens (Poppendieck & Poppendieck, 2003)

- Reduz os custos
- Diminui o tempo das entregas
- Facilmente adaptável
- Não exagera nas soluções e requisitos de negócio

Desvantagens (Poppendieck & Poppendieck, 2003)

- Dependente da habilidade das equipas de seguirem os princípios
- Fácil de perder o foco
- Muita documentação

Scrumban

Scrumban resulta da combinação de duas metodologias, Scrum e Kanban. Combina a previsibilidade e a estrutura do Scrum com o fluxo de trabalho contínuo e a flexibilidade do Kanban. Existem cinco passos cruciais para implementar o Scrumban: criar um *scrumban board*, definir os limites de trabalho, ordenar as prioridades da equipa no *scrumban board*, priorizar os projetos mais importantes e definir as reuniões diárias. As organizações devem usar o Scrumban em diversas situações como quando pretendem dar mais independência às suas equipas e na manutenção de projetos (Stoica et al., 2016) (Banijamali et al., 2017).

Vantagens (Stoica et al., 2016) (Banijamali et al., 2017)

- Melhoria contínua
- Maior flexibilidade
- Reduz o desperdício
- Melhor colaboração
- Reduz o tempo das tarefas

Desvantagens (Stoica et al., 2016) (Banijamali et al., 2017)

- Complexidade
- Dificuldade de implementação
- Falta de clareza

Test Driven Development

TDD foi criado por Kent Beck em 2003 e baseia-se em desenvolver *software* que obriga a escrever testes automatizados e código para passar nesses testes, o código poderá ser melhorado no fim deste processo. O desenvolvimento tradicional de *software* é o oposto do TDD, primeiro constrói-se o design, depois escreve-se o código e só no fim se realizam os testes. TDD tem duas regras, se um teste falhar escrever código para resolvê-lo e não haver duplicações no código (Anwer et al., 2017) (Al-Saqqa et al., 2020).

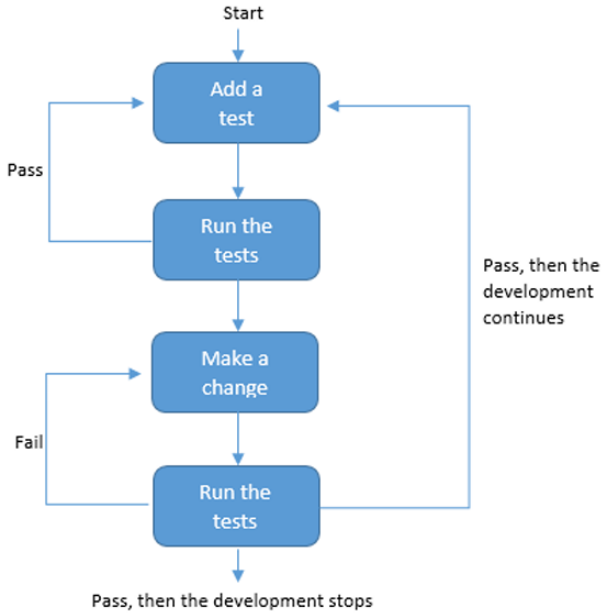


Figura 7 - Processo de Test Driven Development

Fonte: (Al-Saqqa et al., 2020)

Vantagens (Anwer et al., 2017) (Al-Saqqa et al., 2020)

- Melhor qualidade de código
- Descobre *bugs* antecipadamente
- Fácil integração com funcionalidades

- Processos de desenvolvimento simples

Desvantagens (Anwer et al., 2017) (Al-Saqqa et al., 2020)

- Falta de princípios de gestão
- Pouca documentação

2.2 Comparação Waterfall e Agile

As metodologias Waterfall e Agile são duas formas muito populares de desenvolvimento de *software*. Waterfall é melhor para projetos com prazos concretos e requisitos bem definidos enquanto Agile é melhor para projetos flexíveis em que se desenvolvem novos produtos. Agile permite planear o projeto a cada *sprint* e desta forma o projeto evolui à medida que o desenvolvimento também evolui.

No início dos anos 70 surgiu a metodologia Waterfall com o objetivo de ajudar os projetos de desenvolvimento de *software*. Cada ciclo de vida do projeto tem algumas etapas sequenciais: definição de requisitos, design, implementação, testes, entrega e manutenção. Os ciclos de vida Waterfall e Agile são bastante diferentes, um é incremental e rígido e o outro é iterativo e flexível. Os maiores benefícios da metodologia Waterfall são a importância dos requisitos e o planeamento detalhado de todo o processo de desenvolvimento. As principais limitações são que os projetos, grande parte das vezes, não seguem o caminho sequencial pretendido e a dificuldade em definir os requisitos no início do projeto (Hass, 2007).

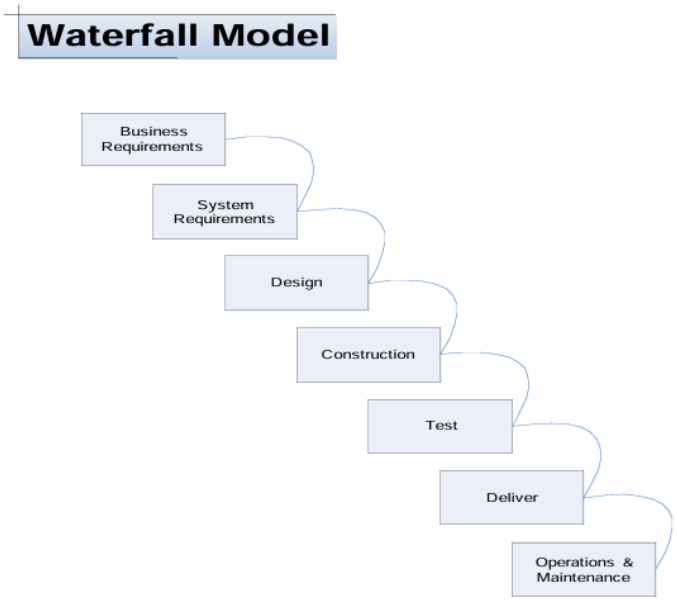


Figura 8 - Ciclo de Vida de Projeto Waterfall
Fonte: (Hass, 2007)

Na metodologia Waterfall prioriza-se os processos acima de tudo enquanto no Agile dá-se mais valor aos indivíduos e às interações. A escolha da equipa é uma etapa crucial para qualquer projeto, uma equipa disfuncional pode originar muitos problemas como atrasos nas

entregas, falhas de comunicação e duplicação de trabalho. No Agile as equipas têm de ser capazes de estabelecer uma boa relação com o cliente e com as outras equipas.

Apesar das suas diferenças, ambas as metodologias têm objetivos em comum, como o foco no cliente, diminuir o tempo de entrega de produtos no mercado e o investimento na qualidade. Para escolher qual metodologia utilizar para determinado projeto deve-se primeiro analisar todas as características do projeto, de modo a tomar a melhor decisão. Deve se ter em conta vários fatores como o tamanho do projeto, o envolvimento do cliente, os requisitos indispensáveis, a experiência da equipa, o prazo do projeto e o orçamento disponível. Por vezes, a melhor solução é uma abordagem híbrida onde se combina as duas metodologias, identificam-se e aplicam-se as práticas mais adequadas de cada metodologia para que o projeto seja bem-sucedido.

As principais diferenças entre as metodologias Waterfall e Agile estão descritas na Tabela 1 (Mitchell & Seaman, 2009) (Leau et al., 2012) (Casteren, 2017).

Tabela 1 - Principais Diferenças entre as Metodologias Waterfall e Agile
Fonte: (Autor)

Waterfall	Agile
Processo de design sequencial	Processo de design incremental, desenvolvido através de iterações
A medida do sucesso é feita em termos de produtos concluídos	A medida do sucesso é feita em termos de funcionalidades entregues
Modelo rígido - não permite modificar os requisitos depois do processo de desenvolvimento ser iniciado	Modelo flexível - permite modificar os requisitos depois do processo de desenvolvimento ser iniciado
Pouca interação com o cliente, o produto só é entregue após o desenvolvimento estar concluído	Muita interação com o cliente, a cada iteração do projeto existe feedback por parte do cliente
Equipas grandes	Várias equipas pequenas
Plano de teste é revisto após o desenvolvimento estar concluído	Plano de teste é revisto após cada *sprint*
A equipa de teste, na maioria das vezes, não participa na fase de mudança de requisitos	A equipa de teste participa na fase de mudança de requisitos

Mais utilizada por governos, bancos, companhias de seguros e grandes empresas	Mais utilizada por *start-ups*, produtos SaaS e pequenas empresas
Muita documentação	Pouca documentação

2.3 Agile nas Organizações

As metodologias Agile são frequentemente utilizadas no desenvolvimento de *software*, mas podem ser utilizadas em qualquer tipo de projeto. Antes de qualquer organização implementar uma metodologia deve primeiro ponderar se essa é a melhor abordagem, o Agile não é adequado para todos os projetos e organizações. Agile tem sido mais usado em equipas de desenvolvimento pequenas e ágeis, mas ultimamente as grandes organizações têm usado cada vez mais esta metodologia nos seus grandes e complexos projetos onde estão envolvidas várias equipas e *stakeholders* e por essa razão é necessário que haja uma excelente comunicação e coordenação entre todos. É fundamental que nos projetos de grande dimensão haja uma boa relação com o cliente de forma a receber regularmente o seu feedback para assegurar que o processo de desenvolvimento está no caminho certo e para garantir que o produto final está de acordo com as necessidades dos utilizadores (Lindvall et al., 2004). Outro aspeto muito importante do Agile nas organizações são os *sprints*, nesta etapa são definidas as metas que se pretende alcançar, isto divide o projeto em partes mais pequenas e torna o progresso mais fácil de acompanhar.

A implementação do Agile nas organizações é um processo gradual. Para mudar a forma como o trabalho é realizado é indispensável que toda a organização esteja de acordo e isso pode demorar o seu tempo. As equipas têm de assimilar as práticas e têm de se habituar à nova forma de trabalhar (Lindvall et al., 2004). Adotar Agile nas organizações envolve alguns passos:

- Escolher a metodologia mais adequada - as metodologias Agile são investigadas e analisadas e é escolhida a mais adequada para a organização tendo em conta os seus objetivos e necessidades
- Construir uma equipa Agile - construir uma equipa com as habilidades especificas para o projeto que se adapta bem às mudanças e trabalhe de forma colaborativa
- Implementar práticas Agile - como *sprints*, retrospetivas e reuniões *stand-up*
- Avaliação dos processos de gestão de projetos - análise da abordagem atual de gestão de projetos e identificação de áreas que precisam de ser alteradas para integrar os princípios Agile com sucesso

- Formação - formação sobre a metodologia escolhida ajuda a equipa a compreender melhor os valores, princípios, práticas e vantagens dessa metodologia
- Monitorização - monitorização do progresso do projeto
- Revisão - revisão regular do estado do projeto
- Relatórios - entrega de relatórios aos *stakeholders* sobre os resultados e problemas encontrados

É vital que em projetos de grande escala sejam usadas metodologias colaborativas e adaptáveis, como é o caso das metodologias Agile. De seguida descreve-se os benefícios destas metodologias para as organizações (Dikert et al., 2016).

A **flexibilidade** é uma característica essencial de todas as metodologias Agile porque permite que as equipas respondam com rapidez às mudanças e aos novos requisitos.

A comunicação transparente e frequente permite que os *stakeholders* tenham **visibilidade** sobre o progresso do projeto e isso permite-lhes tomar decisões informadas, consequentemente, isto vai conduzir a um aumento de confiança dos *stakeholders*.

Para diminuir o risco de atrasos e prazos ultrapassados o projeto é dividido em *sprints*, **pequenas entregas de *software* funcional**, com o objetivo de obter feedback e realizar as alterações rapidamente.

Nas metodologias Agile promove-se uma boa **colaboração e comunicação** onde as equipas trabalham juntas para acrescentar valor e os *stakeholders* fazem parte do processo de tomada de decisão. Isso leva a um melhor entendimento dos requisitos e alinhamento dos objetivos.

Algumas metodologias como o Scrum realizam retrospetivas regularmente, a equipa reflete sobre o que correu bem e o que pode ser melhorado. Isso promove uma cultura de **melhoria contínua** o que ajuda as equipas a entregar melhores resultados finais.

Agile é bastante eficaz em projetos empresariais com a abordagem, metodologias ferramentas, mentalidade e estruturas corretas. Além disso, é importante compreender os objetivos, os *stakeholders* e as expetativas do projeto e ter um plano para enfrentar as adversidades. Apesar dos diversos benefícios que existem ao implementar metodologias Agile

as organizações continuam a enfrentar alguns desafios, como estão descritos abaixo (Dikert et al., 2016).

A **resistência à mudança** é um desses desafios, Agile obriga a uma mudança significativa na mentalidade e nas práticas. Pode haver membros da equipa resistentes às mudanças, na maioria dos casos são indivíduos que estão habituados a utilizar métodos tradicionais e mais estruturados.

Escalar metodologias Agile para grandes projetos pode se tornar num grande desafio, as metodologias Agile são mais adequadas para equipas pequenas e multifuncionais do que para projetos grandes e complexos com várias equipas e *stakeholders* e por essa razão as metodologias podem ser menos eficazes.

A **falta de requisitos claros** é um problema que muitas organizações enfrentam porque o foco do Agile é direcionado à flexibilidade e à adaptabilidade o que dificulta a definição antecipada de requisitos.

Em projetos de grande escala pode haver **dificuldades em coordenar equipas** devido a muitas equipas trabalharem no mesmo projeto, algumas de forma remota, o que dificulta a comunicação e coordenação e pode resultar em atrasos na conclusão do projeto.

As metodologias Agile fornecem muita autonomia às equipas, mas isso pode levar à **falta de governança e supervisão** o que pode ser um problema grave nas organizações, pode levar a faltas de responsabilidade.

Ao utilizar metodologias Agile torna-se **mais difícil medir o sucesso** porque as métricas tradicionais de medir o sucesso como o custo e o cronograma são mais complicadas de utilizar.

Capítulo 3 - Abordagem de Investigação

O método de pesquisa adotado foi o método de caso de estudo porque é o mais apropriado para este trabalho, neste caso são analisados todos os parâmetros do Agile em grandes organizações.

Começou-se por realizar uma revisão da literatura, onde se inclui vários casos reais publicados em artigos de revistas da especialidade sobre a utilização da metodologia Agile em projetos de grande dimensão, de modo a responder a todas as questões da investigação e a cumprir os objetivos propostos. Desta maneira, enquadrou-se e analisou-se de forma imparcial os casos de estudo que este trabalho aborda.

O objetivo principal da investigação foi descobrir e compreender os benefícios que o Agile fornece nas grandes organizações. Para além disso, foram analisadas as diferentes metodologias, as metodologias mais utilizadas nos projetos atualmente e as diferenças entre as metodologias Waterfall e Agile.

No caso A, Spotify, analisam-se os princípios, benefícios e desafios derivados do modelo de engenharia criado pela organização. Este é um caso especial visto que a organização criou o seu próprio modelo Agile de modo a ter um modelo personalizado que corresponde melhor a sua forma de trabalhar, estrutura, recursos e objetivos. A maior inovação deste modelo foi a maneira como organiza as equipas por várias estruturas, como *squads* e *chapters*. Isto traz uma enorme vantagem ao tornar a comunicação e colaboração mais eficientes o que consequentemente torna as equipas mais produtivas.

No caso B, Google, estuda-se a implementação do Agile numa das mais bem-sucedidas organizações do mundo. Isto foi realizado através da adoção de várias técnicas e práticas Agile como o *sprint* de design, o TBD, a integração contínua e a automação de testes. Descreve-se estas técnicas, como são utilizadas e as suas particularidades. Esta organização utiliza várias metodologias Agile como XP e Scrum.

No caso C, Cisco, analisa-se a transição de metodologias da organização, a troca de uma metodologia tradicional para as metodologias Agile. São analisados dois grandes projetos da organização onde foram aplicadas novas metodologias, nos quais obteve excelentes resultados. Descreve-se os problemas encontrados e as soluções aplicadas para superá-los.

No caso D, Amazon, investiga-se a implementação do Agile numa das maiores organizações do mundo. Similarmente com o caso B, também neste caso a implementação é realizada através de várias práticas e técnicas originárias de várias metodologias, nomeadamente Scrum, como a cultura de dia 1, o modelo de equipa *two-pizza* e a obsessão pelo cliente. Esta organização deve o seu crescimento à aplicação do Agile e ao seu foco na satisfação dos clientes com o objetivo final de entregar produtos com valor.

Com a análise destes casos pretende-se explorar a implementação de diferentes metodologias Agile em grandes organizações, sendo o foco principal os resultados, em relação a diversos aspetos, como, estrutura organizacional, cultura e processos de desenvolvimento. Pretende-se também demonstrar que a abordagem Agile é eficaz e fornece imensas vantagens a organizações de grande escala e de áreas de mercado e de origem distintas. Esta investigação é realizada através do estudo de princípios e práticas Agile que as organizações aplicam diariamente e da adoção de metodologias Agile para projetos específicos.

Capítulo 4 - Casos de Estudo

Neste capítulo pretende-se apresentar casos de estudo de empresas que adotaram a metodologia Agile nos seus processos e assim melhoraram a sua forma de trabalhar e obtiveram um crescimento significativo no mercado.

4.1 Análise do Caso A - Spotify

4.1.1 Enquadramento da Empresa

No caso A analisa-se a implementação de práticas Agile na empresa Spotify Ltd. O Spotify é um serviço de *streaming* de música, podcast e vídeo, o seu conteúdo está protegido por direitos digitais a partir de um vasto número de editoras discográficas e editoras independentes como a Sony, a Universal, a EMI e a Warner Music Group. O Spotify foi fundado por Daniel Ek e Martin Lorentzon em Estocolmo, Suécia, desenvolvido em 2006 pela Spotify AB, lançado oficialmente em 7 de outubro de 2008 e tem cerca de 9.000 funcionários.

O Spotify é um serviço freemium, ou seja, fornece serviços básicos grátis que podem conter publicidade ou limitações, mas quem optar por subscrever à assinatura paga tem acesso à recursos adicionais como *downloads* de música e melhor qualidade de transmissão. Tem um total de 587 milhões de utilizadores sendo que 361 milhões usam o serviço grátis e 226 milhões são assinantes pagantes. O Spotify está disponível em 184 mercados, os seus utilizadores estão localizados principalmente nos EUA e na Europa, que representam cerca de 53% de todos os utilizadores e 67% de todas as receitas. O seu principal rival é Apple Music, mas tem também concorrência feroz de outras empresas como YouTube Music, Deezer, Amazon Music e Sound Cloud.

4.1.2 Transformação Agile

Um dos principais fatores de sucesso de qualquer organização é a sua cultura. A cultura Agile no Spotify é muito importante no dia a dia da empresa. Todas as equipas percebem e implementam a cultura da empresa no seu trabalho, graças a isto a cultura Agile continua a crescer e a ficar cada vez mais forte. Quando a empresa foi criada em 2008 a metodologia inicialmente utilizada foi o Scrum, mas à medida que a empresa cresceu também as equipas aumentaram de tamanho e chegou-se à conclusão de que as práticas do Scrum, como as reuniões de planeamento de *sprint*, que até esse ponto estavam a ser implementadas estavam a impedir que a empresa atingisse todo o seu potencial. Por essa razão a empresa escolheu outro caminho e começou a criar e implementar a sua própria metodologia Agile (Manninen, 2018) (Almeida & Espinheira, 2021).

Neste âmbito surgiu o modelo de engenharia Agile do Spotify, a Spotify Tribe, que foi criado em 2011. Em 2012, dois funcionários, Henrik Kniberg e Anders Ivarsson, a pedido dos seus colegas, publicaram um artigo com o objetivo de demonstrar a visão geral de como o desenvolvimento de produtos é organizado no Spotify. No artigo está incluído a hierarquia das equipas, como as equipas são organizadas e o tipo de cultura empresarial implementado. O Spotify ao usar este modelo defende as necessidades de transparência, colaboração e simplicidade como parte essencial na sua cultura (Kniberg & Ivarsson, 2012).

4.1.2.1 Spotify Tribe

O modelo de engenharia Agile do Spotify é centrado na simplicidade. O Spotify começou por se organizar em torno do seu trabalho identificando os aspetos mais importantes sobre como cada membro e as suas equipas deveriam ser organizadas, desta forma foi criado o modelo de engenharia do Spotify, chamado de Spotify Tribe. que se encontra ilustrado na figura 9 onde está demonstrada a organização dos diferentes elementos.

Muitas organizações ainda têm uma estrutura organizacional tradicional onde cada departamento é dirigido por um chefe que supervisiona várias equipas e cada equipa tem o seu líder. O chefe do departamento realiza reuniões com os lideras da organização e comunica

informação aos líderes das equipas que, por sua vez, comunicam aos elementos da equipa. A transição de um modelo tradicional para o modelo Spotify não deve ser totalmente copiada, mas sim adaptada às características de cada organização. Cada departamento deve ser dividido em *squads*, o Spotify dividiu as *squads* tendo em conta as funções do website. No caso de uma empresa que desenvolva produtos pode se dividir as *squads* pelos diferentes produtos. Para cada *squad* deve ser escolhido um líder. Esse líder representa a *squad* nas reuniões com os outros líderes.

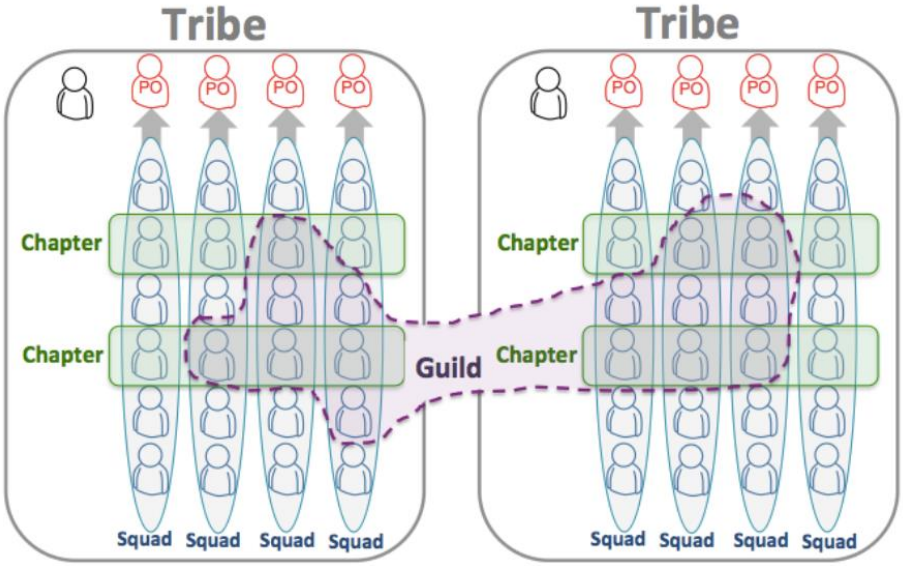


Figura 9 - Modelo Original de Engenharia Spotify
Fonte: (Kniberg & Ivarsson, 2012)

Squad

Numa organização pode haver várias *squads*, cada uma composta por 6 a 12 pessoas incluindo um *agile coach* e um *product owner*. Cada pessoa trabalha numa área específica e todas as *squads* têm contacto direto com os *stakeholders*.

As *squads* podem ser vistas como *mini-startups*, são auto-organizadas, multifuncionais, autónomas e autogeridas. Cada *squad* tem a sua missão de longo prazo focada no produto, como, por exemplo, criar a experiência de rádio do Spotify, e tem a sua missão de curto prazo que é negociada a cada trimestre (Salameh & Bass, 2018). Cada *squad* escolhe que metodologia implementa, pode ser Scrum, TDD, Kanban, XP ou uma

combinação de duas ou mais metodologias (Kniberg & Ivarsson, 2012) (Šmite et al., 2018) (Manninen, 2018).

As *squads* costumam aplicar várias técnicas derivadas dos princípios do Lean Startup, como a técnica MVP, que consiste em entregar um produto na sua versão mais básica para verificar se as necessidades do cliente são cumpridas, com o objetivo de entregar trabalho antecipadamente e com frequência. Outra técnica aplicada é a aprendizagem validada que utiliza métricas e A/B *testing* para descobrir o que funciona (Kniberg & Ivarsson, 2012) (Salameh & Bass, 2019).

Os espaços de trabalho das *squads* estão otimizados, com as pessoas a trabalharem próximas umas das outras de modo a encorajar a colaboração. Desses espaços incluem áreas de trabalho pessoal, áreas de lazer e salas de reuniões. Uma característica interessante destes espaços é que a maioria das paredes são *whiteboards*, como se consegue ver na parte vermelha da parede na figura 10 (Manninen, 2018).

Figura 10 - Sala de Reuniões de uma Squad
Fonte: (Kniberg & Ivarsson, 2012)

Figura 11 - Espaço de Trabalho e de Lazer de uma Squad
Fonte: (Kniberg & Ivarsson, 2012)

Tribe

Uma *tribe* é formada por várias *squads* que trabalham na mesma área, como por exemplo no reprodutor de música, composta por 40 a 150 pessoas, mas o ideal é no máximo 100 pessoas incluindo o líder da *tribe* que pode também fazer parte de uma *squad*. O líder da *tribe* é responsável por criar um ambiente produtivo e inovador para as *squads*. As *tribes* realizam regularmente reuniões informais onde se apresenta o que se está a trabalhar e o que foi desenvolvido de maneira a haver uma ajuda mútua com a partilha de diferentes experiências, isto inclui *demos* ao vivo de *software*, projetos de *hack-day* e apresentações de novas ferramentas e técnicas (Kniberg & Ivarsson, 2012) (Šmite et al., 2018).

Figura 12 - Reunião de uma Tribe
Fonte: (Kniberg & Ivarsson, 2012)

Chapter

Um *chapter* é constituído por indivíduos de diferentes *squads*, com competências semelhantes, que são agrupados em uma única *squad* e são formados dentro de uma *tribe*. O líder do *chapter* tem como principais funções apoiar os membros do *chapter* em determinados desafios e no crescimento pessoal de cada membro. Os membros de um *chapter* partilham conhecimento e discutem os desafios dos projetos (Šmite et al., 2018). Alguns exemplos de *chapters* no Spotify são o *chapter* de desenvolvimento web, o *chapter* de testes e o *chapter* de back-end (Kniberg & Ivarsson, 2012).

Guild

Guild é um grupo informal composto por indivíduos de diferentes *tribes* que têm em comum um interesse ou problema. Qualquer indivíduo de qualquer *tribe* pode pertencer a uma *guild*. Os objetivos da *guild* são garantir a transparência ao resolver problemas e manter as equipas alinhadas e focadas. As *guilds* têm reuniões semanais onde os membros da *guild* apresentam questões e discutem entre si uma solução. *Workshops* de *guilds* como o *hackathon* têm sido atividades bem aceites pelos membros das *guilds*. Na figura abaixo vê-se um exemplo de trabalho das *guilds*, foi realizada em Estocolmo, Suécia uma *Web Guild Unconference*, evento onde todos os programadores web do Spotify se reuniram para discutir desafios e soluções das suas áreas de especialidade (Šmite et al., 2018). Alguns exemplos de *guilds* no Spotify são a *guild* de engenheiros de testes, a *guild* de *agile coaches* e a *guild* de programadores web (Kniberg & Ivarsson, 2012).

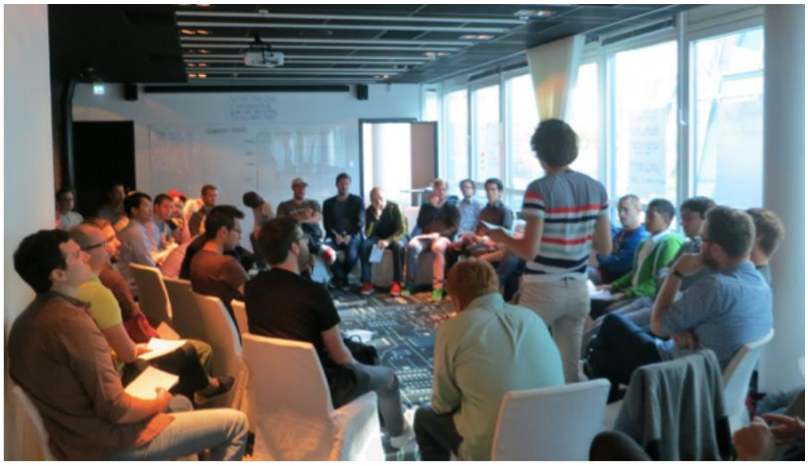


Figura 13 - Web Guild Unconference

Fonte: (Kniberg & Ivarsson, 2012)

Ao longo dos anos, o Spotify adaptou o seu modelo para corresponder às mudanças constantes da organização. Neste contexto surgiram dois novos elementos, trio e *alliance*, que se encontram ilustrados na figura abaixo, de modo a resolver os novos desafios que a organização encontrou no seu processo de crescimento.

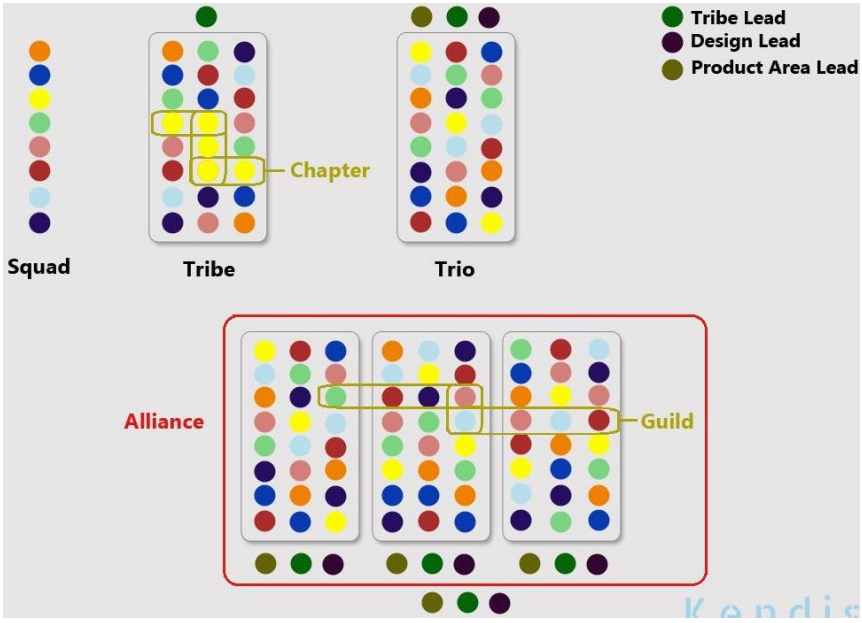


Figura 14 - Evolução do Modelo de Engenharia Spotify

Fonte: (Kendis, 2018)

Trio

Um trio é formado por um líder da área de produto, um líder de design e um líder da *tribe* (Almeida & Espinheira, 2021). Cada *tribe* possui um trio para assegurar que existe alinhamento contínuo entre as três perspetivas dos líderes e que todos estão focados nas suas tarefas.

Alliance

Uma *alliance* é geralmente formada por três ou mais trios e é liderada por um líder da área de produto, um líder de design e um líder da *tribe* (Almeida & Espinheira, 2021). As organizações ao crescerem implica que, por vezes, várias equipas necessitem de trabalhar juntas para ajudarem-se umas às outras de modo a cumprirem os objetivos.

No Spotify os papéis individuais têm um grande impacto no bom funcionamento do seu modelo. Estas pessoas são fundamentais para a colaboração das equipas e para o crescimento profissional dos seus elementos. De seguida, descreve-se os papéis mais importantes.

Chief Architect

O *chief architect* orienta todos os projetos, define a visão arquitetónica e se necessário lida com os problemas de dependência da arquitetura do sistema. Todos os sistemas têm um proprietário do sistema que lida com as dependências da arquitetura e verifica as entregas frequentes. De forma a mitigar eventuais riscos com a arquitetura do sistema, as responsabilidades do proprietário do sistema são divididas pelos outros membros, essas responsabilidades podem ser de qualquer membro de qualquer *squad* (Almeida & Espinheira, 2021).

Agile Coach

O *agile coach*, equivalente ao *scrum master*, ajuda a melhorar a forma da *squad* trabalhar, conduz as retrospetivas e as reuniões de planeamento do *sprint* (Manninen, 2018)

(Salameh & Bass, 2019). Também é responsável por resolver conflitos, organizar *workshops*, promover a comunicação e modelar valores Agile. É essencial que o *agile coach* esteja sempre presente e tome decisões rápidas no momento. Na maioria dos casos, os *agile coaches* são pessoas que têm formação em gestão de projetos (Bäcklander, 2019).

Product Owner

Cada *squad* tem um *product owner* que define a visão e a estratégia do produto, organiza o trabalho dos membros da *squad* de modo a priorizar o *backlog* do produto e mantem um *roadmap* de alto nível. O *product owner* deve estar diariamente envolvido com a *squad* de forma a ajudar e fornecer os recursos necessários, deve atender às reuniões de gestão, reuniões diárias e retrospetivas (Kristinsdottir et al., 2016) (Salameh & Bass, 2019).

4.1.2.2 Princípios do Modelo Spotify

O modelo Spotify aplica princípios determinantes para uma cultura de confiança, aprendizagem rápida e autonomia em toda a organização. Seguidamente, apresenta-se os melhores princípios implementadas pelo Spotify que originam impactos positivos na forma como a organização colabora.

Transparência e Confiança

Grande parte do sucesso do Spotify é devido ao seu foco na construção de uma comunidade e na transparência em torno do seu trabalho. Isto foi alcançado graças à confiança dos membros na organização que foi conseguida através de formas transparentes e inclusivas de obter feedback e de obter alinhamento sobre como a organização deve trabalhar no futuro (Bäcklander, 2019).

Autonomia

O Spotify dá o máximo de autonomia possível aos seus funcionários de modo a ajudá-los a trabalhar mais eficientemente, exemplos disso são as *squads* que podem alterar o código de outras *squads* a qualquer momento e podem escolher as suas ferramentas de desenvolvimento. Para além disso, o Spotify delegou algumas decisões para as *squads, tribes* e *chapters* em vez de serem impostas pelos líderes da organização que, muitas vezes, estão desconectados do trabalho diário (Manninen, 2018) (Salameh & Bass, 2018) (Salameh & Bass, 2019).

Alinhamento

O alinhamento das *squads* refere-se aos objetivos e à estratégia da organização serem compreendidos por todos e à priorização de interações em vez de processos. Nas *squads* os programadores focam-se apenas numa área específica o que os torna especialistas dessa área (Salameh & Bass, 2018) (Salameh & Bass, 2020).

Incentivo ao erro

O Spotify passou por muitas mudanças até chegar ao seu estado atual e continua a realizar experiências para melhorar o seu funcionamento. O Spotify incentiva o erro porque só desta forma é possível existir evolução, a melhoria envolve experimentação e aprender com os sucessos e com os fracassos. Foi criado um ambiente favorável às falhas, o objetivo não é descobrir de quem é a culpa, mas sim identificar os erros rapidamente para aprender e garantir que não voltam a acontecer.

4.1.2.3 Benefícios do Modelo Spotify

O modelo Spotify é baseado na autonomia e na confiança onde o fracasso é visto como uma oportunidade para aprender e evoluir, isto aumenta o moral e o crescimento individual. À medida que o Spotify desenvolveu e implementou o seu modelo foi

compreendendo melhor que benefícios estavam a ganhar. Os principais benefícios deste modelo estão descritos abaixo.

Criatividade

O modelo incentiva a autonomia e a criatividade entre as equipas ao confiar nos indivíduos para tomarem decisões de forma independente. O Spotify concentra-se na descentralização da tomada de decisões e na transferência dessa responsabilidade para as equipas, isto promove um sentimento de responsabilidade e propriedade que se traduz em processos de resolução de problemas e de tomada de decisão mais rápidos (Manninen, 2018) (Salameh & Bass, 2020).

Adaptabilidade e Flexibilidade

As equipas facilitam a adaptabilidade e a flexibilidade na cultura organizacional. As prioridades mudam à medida que a organização cresce. As equipas são facilmente reconfiguradas e isso permite ajustes contínuos para atender às necessidades que se encontram em constante evolução.

Colaboração multifuncional

As equipas organizam-se em torno dos projetos ou recursos disponíveis, neste aspeto o modelo diferencia-se dos modelos tradicionais. Isto incentiva a colaboração entre equipas com competências funcionais distintas o que resulta na melhoria da qualidade e da inovação dos produtos e serviços. As equipas são formadas por pessoas de diversas áreas de negócio, isto contrasta com outros modelos em que existe uma separação clara entre os departamentos de negócio e de TI (Salameh & Bass, 2019) (Salameh & Bass, 2022).

Poucos processos formais

O modelo foca-se na organização em torno do trabalho em vez de nos processos. Desta forma a organização tem maior flexibilidade em relação ao funcionamento das *squads*,

a organização concentra-se em alinhá-las e impulsionar os resultados individuais de cada *squad* em vez de exigir que mudem a forma como trabalham.

Menos documentação

Quando as equipas tomam as suas próprias decisões conseguem trabalhar mais rapidamente no código dos projetos. O modelo incentiva a partilha de informações entre equipas sem que haja demasiados processos formais o que consequentemente diminui a quantidade de documentação necessária.

Maior rapidez

As equipas ao terem controlo sobre o seu trabalho vai ter como efeito um aumento de rapidez no trabalho e nas entregas de *software* visto que não necessitam de esperar por decisões tomadas pelos líderes (Salameh & Bass, 2020) (Salameh & Bass, 2022).

Desafios de curto prazo

Este modelo permite a resolução de desafios de curto prazo de uma maneira mais rápida e eficiente, caso alguma *squad* esteja a ter problemas pode sempre solicitar a ajuda de outra *squad*. Esta rede de entreajuda de *squads* cria um ambiente de confiança e autonomia.

Alinhamento com a visão do Spotify

A estrutura do modelo está alinhada com a abordagem de desenvolvimento rápida e inovadora do Spotify, isto permite que a empresa continue na vanguarda da indústria de *streaming* de música e responda rapidamente às mudanças do mercado.

Motivação

Os funcionários terem controlo sobre o seu trabalho resulta numa maior satisfação e motivação. Na cultura do Spotify todas as ideias são ouvidas e valorizadas e a criatividade é

sempre bem recebida. O Spotify dá muita importância aos questionários, num questionário obteve 91% de satisfação dos funcionários, mas a sua preocupação foi com os 4% que não estavam satisfeitos (Manninen, 2018).

Tempo de entrega

O Spotify devido a ter equipas pequenas e focadas faz com que os ciclos de desenvolvimento sejam mais rápidos. Isto permite que o Spotify desenvolva novos produtos mais cedo do que a sua concorrência, mantendo assim uma vantagem competitiva no mercado.

4.1.2.4 Desafios do Modelo Spotify

O modelo Spotify tem como base a forma de trabalhar da organização. Algumas organizações implementaram este modelo com a esperança de obter as mesmas vantagens, mas na maioria das vezes não foram bem-sucedidas. Isto acontece porque é necessário ter em consideração a cultura e a estrutura da organização em cada modelo. A estrutura do modelo Spotify pode parecer uma estrutura organizacional simples, onde os funcionários reportam a uma equipa funcional, denominada de *chapter*, mas trabalham com uma equipa multifuncional, denominada de *squad*, mas a estrutura tem as suas complexidades. O modelo apresenta alguns desafios que estão descritos abaixo.

Risco para a arquitetura do sistema

O Spotify possui mais de 100 sistemas diferentes que combinados formam o produto geral (Kniberg & Ivarsson, 2012). Normalmente, quando as *squads* precisam de realizar uma atualização elas acabam por ter de realizar várias de modo a não comprometer o sistema. O risco existente é de a arquitetura não funcionar da maneira desejada se todos se concentrarem apenas nas suas partes. Uma forma de mitigar este risco é ter um proprietário do sistema que possua a integridade da arquitetura (Manninen, 2018) (Salameh & Bass, 2022).

Custos

Por ser um modelo que é implementado num ambiente complexo, as despesas têm de estar de acordo com o modelo e muitas das vezes as despesas são avultadas. Por essa razão este modelo é mais adequado a organizações de média e grande escala. As despesas podem estar relacionadas com os equipamentos de *hardware* e *software* e com os salários altos de funcionários qualificados e experientes.

Implementação

Para uma organização realizar uma transição do seu modelo atual, principalmente modelos que não são Agile, para o modelo Spotify, tem que primeiro investigar e analisar minuciosamente se faz sentido para o seu negócio fazer essa transição (Manninen, 2018). O modelo Spotify, como o próprio nome indica, é um modelo que foi criado para responder às necessidades específicas do Spotify, por esta razão este modelo não funcionou em muitas organizações. O modelo Spotify está em constante evolução para se ajustar à sua organização que também está em constante mudança com novos produtos e atualizações a serem desenvolvidos diariamente. Neste âmbito surgiram novos elementos no modelo, como os trios e as *alliances*, para resolver os novos problemas que a organização enfrentou à medida que crescia. Cada organização deve adaptar o modelo Spotify às suas características, deve ter em conta vários aspetos como os seus produtos, a sua cultura e os seus processos de desenvolvimento.

Coordenação entre equipas

Com o crescimento da empresa o volume de trabalho aumentou e surgiu a necessidade de criar mais equipas. A coordenação entre múltiplas equipas autónomas pode-se tornar complicada, a falta ou a má coordenação das equipas pode levar à redundância, duplicação de esforços e conflitos (Salameh & Bass, 2022).

Inconsistências

O facto de haver diferentes equipas a trabalharem de forma independente usando cada uma os seus próprios métodos e ferramentas nos mesmos projetos pode originar inconsistências no desenvolvimento e manutenção de produtos e serviços. Isto pode ser especialmente prejudicial para a experiência do utilizador.

Equipas multifuncionais

As equipas multifuncionais trazem imensas vantagens a forma de trabalhar do Spotify, como as diferentes perspetivas que os vários elementos das equipas têm sobre o mesmo projeto, mas este tipo de equipas pode não ser sempre a maneira mais eficiente de abordar determinadas tarefas. Uma equipa ao ter vários elementos de áreas distintas e com hábitos de trabalho diferentes pode, por vezes, gerar conflitos e consequentemente provocar atrasos nos projetos (Salameh & Bass, 2022).

Conhecimentos e práticas restritos

O modo como a estrutura está desenhada pode originar uma fragmentação do conhecimento, logo isso significa que conhecimentos valiosos e boas práticas podem ficar restritos apenas às equipas que as praticam. Isto pode dificultar a transferência de conhecimento e impedir a capacidade da organização de aprender com as experiências das equipas.

Difícil para pequenas empresas

O modelo foi projetado para organizações de média e grande escala e de rápido crescimento, como é o caso do Spotify. As empresas de pequena escala terão dificuldades em implementar esta estrutura de forma eficaz devido ao número limitado de funcionários e recursos.

Risco de perder o foco

Com as equipas focadas nos seus objetivos individuais, existe o risco de negligenciar os objetivos organizacionais. A falta de alinhamento pode levar a conflitos de interesses e prejudicar o progresso geral da empresa.

4.1.3 Considerações Finais sobre o Caso A

A criação do modelo Spotify contribui decisivamente para o sucesso e crescimento da organização ao longo dos anos, permitindo estar sempre atualizada com as novas tendências de modo a corresponder às necessidades dos clientes e continuar no topo do mercado de *streaming* de música. Os princípios aplicados, como, a elevada autonomia e o alinhamento, combinados com a colaboração que as equipas têm dentro da organização levou ao rápido desenvolvimento e à entrega de produtos de qualidade. Este modelo apresenta diversas vantagens se implementado corretamente, como, aumento da criatividade e motivação dos funcionários e tempos de entrega mais rápidos. Contudo, o modelo tem os seus desafios, principalmente quando se aborda temas como a coordenação, a partilha de conhecimento e a escalabilidade para organizações de pequenas dimensões. Este modelo não deve ser totalmente copiado por nenhuma organização. Cada organização que queira implementar este modelo deve manter os princípios, mas adaptar as práticas. Para além disso, deve adaptá-lo consoante a sua estrutura e necessidades.

4.2 Análise do Caso B - Google

4.2.1 Enquadramento da Empresa

No caso B analisa-se a implementação de práticas Agile na empresa Google LLC. O Google é uma empresa de *software* e de serviços online que foi fundada em 4 de setembro de 1998 na Califórnia, EUA por Larry Page e Sergey Brin, também conhecidos como os «Google Guys». O Google tem a sua sede localizada na Califórnia, EUA, apelidada de «GooglePlex» e tem cerca de 140.000 funcionários.

O Google começou por ser apenas um motor de pesquisa, mas ao longo dos anos evoluiu e tornou-se numa das maiores empresas tecnológicas do mundo e no motor de pesquisa mais popular, tendo mais de 85% do mercado global de pesquisas. Atualmente, através da Alphabet Inc., possui um vasto leque de produtos de sucesso como Youtube, Android, Waze, Pixel e fornecem vários serviços como Gmail, Google Ads, Google Talk. A principal fonte de receitas é a publicidade. Outras fontes são as vendas de aplicações, as compras dentro de aplicações, as taxas de serviço, as taxas de licenciamento e a venda de *hardware*. Os seus principais concorrentes no que diz respeito ao motor de pesquisa são o Bing da Microsoft, a Yahoo! e o AOL.

4.2.2 Transformação Agile

O Google desenvolve *software* em larga escala usando práticas e princípios Agile. O Google apesar de não admitir publicamente, utiliza várias metodologias Agile como o XP, Lean e Kanban, mas a mais comum é o Scrum. Devido à sua enorme escala o Google necessita de desenvolver as suas próprias ferramentas e infraestruturas. Um princípio Agile essencial que é aplicado pelo Google é agir e aprender com as experiências do passado de modo a não cometer os mesmos erros no futuro. Uma das práticas Agile utilizada é a reserva de um dia por semana a cada funcionário para a inovação. A partir desta medida podem surgir projetos bastante interessantes, para que tenham sucesso é necessário o envolvimento de mais

pessoas e isso cria um mercado no qual os programadores do Google escolhem em que ideias investem (Harracá, 2017).

O Google foi pioneiro no «permanent beta» que significa entregar *software* inacabado (Whittaker et al., 2012). Essa abordagem reduz as expectativas de qualidade e permite que os utilizadores experimentem o *software* sem ficarem irritados com os defeitos que possam encontrar, desta maneira a empresa recebe feedback dos utilizadores e a partir daí resolve os *bugs*. Todos os produtos do Google têm o seu próprio caminho, a cada trimestre se necessário são realizados pequenos ajustes. Os produtos são projetados para serem automatizados e documentados o máximo possível através de informações incorporadas na interface do utilizador.

O Google tem muitos casos de sucesso no desenvolvimento de produtos através da adoção da abordagem Agile, como o Google Chrome, o Gmail e o Google Cloud Platform. Em 2008, uma pequena e multifuncional equipa começou a desenvolver o Chrome, neste processo priorizaram-se três aspetos: desenvolvimento focado no cliente, iterações rápidas, e colaboração e transparência (International Agile Federation, 2024). Estes aspetos foram implementados através de práticas como reuniões diárias e realização de questionários. O resultado foi excelente com a criação de um rápido, inovador e seguro *browser*, atualmente é o *browser* mais usado no mundo. O Google utilizou Scrum para desenvolver o Gmail, várias equipas trabalharam em *sprints* ao mesmo tempo, mas em diferentes tarefas. Após cada *sprint*, a equipa de integração implementou as funcionalidades desenvolvidas pelas equipas no sistema.

O Google utiliza várias técnicas que têm as suas raízes no desenvolvimento Agile, como o *sprint* de design criado pelo próprio Google, o *trunk-based development*, a integração contínua e a automação de testes. Todas estas técnicas de desenvolvimento Agile, o *sprint* de design inspirado pelo *sprint* do Scrum e as outras três originárias do XP, são relevantes para este trabalho porque são técnicas Agile que são aplicadas frequentemente e com enorme sucesso em organizações de larga escala, como se demonstra ao longo dos seguintes capítulos com exemplos de outras empresas de renome e de grande dimensão, como, Lego e Airbnb, e de produtos de êxito do Google, como, o Gmail e o Chrome. De seguida, descrevem-se e analisam-se estas técnicas ao detalhe.

4.2.2.1 Sprint de Design do Google

O *sprint* de design do Google foi criado por Jake Knapp em 2010, a sua inspiração foi a cultura de desenvolvimento de produtos do Google, a sua experiência na criação e desenvolvimento de vários produtos como Gmail e Hangouts, e *workshops* de design *thinking*. Knapp apercebeu-se que apesar do entusiasmo à volta dos *workshops* de *brainstorming* as melhores ideias surgiam dos funcionários que enfrentavam grandes desafios e tinham pouco tempo para resolvê-los (Larusdottir et al., 2019) (Wangsa et al., 2022).

De 2010 a 2012, o *sprint* de design do Google continuou a ser desenvolvido pelo Google em várias equipas como Chrome e Google X. Em 2012, o *sprint* foi levado para Google Ventures onde o seu desenvolvimento prosseguiu. O *sprint* do Google é diferente do *sprint* do Scrum, mas ambos são Agile. O *sprint* de design do Google é um conjunto de atividades estruturadas desenvolvidas para ajudar as organizações a responder a questões críticas de negócio, projetar, fazer protótipos e testar soluções. Após a publicação do livro Sprint em 2016, o *sprint* do Google foi implementado com sucesso em várias organizações em todo o mundo, em empresas como Lego, Airbnb e Slack, em governos como no Reino Unido e na cidade de Chicago e em universidades como Columbia e Stanford (Knapp, Kowitz & Zeratsky, 2016) (Edmonds, 2018) (Courtney, 2024).

Existem múltiplas razões para usar o *sprint* do Google, o *sprint* permite realizar várias ações de elevada importância:

- Trabalho individual para criar soluções
- Votação estruturada
- Construir protótipos
- Testar os protótipos com clientes

O *sprint* do Google é ótimo para projetos com grandes desafios, como, projetos de alto risco, com prazos curtos e que precisam de um novo começo. O *sprint* é ideal para *start-ups* de TI, mas funciona em organizações de qualquer tamanho e de qualquer setor. Com o *sprint* pode-se experimentar abordagens estruturadas de alto risco e alta recompensa que permite que se concentrem todos os esforços num curto período e pode impulsionar a criatividade e a tomada de decisões dos funcionários sem ter de investir muito dinheiro e tempo (Knapp et al., 2016) (Wangsa et al., 2022).

O *sprint* do Google é constituído por no máximo sete indivíduos, estes indivíduos têm de querer estar envolvidos no *sprint* e é importante que tenham diferentes perspetivas (Larusdottir et al., 2019). Os papéis principais deles são os seguintes:

Decisor - No mínimo tem de haver 1 na equipa, conhece extremamente bem as questões do negócio e tem autoridade para tomar decisões.

Facilitador - São essenciais para o *sprint* de design do Google devido ao *sprint* ser bem estruturado, rápido e muito colaborativo. *Agile coaches* e *scrum masters* são ótimos candidatos para este papel.

Perturbador - Indivíduos inteligentes e teimosos que tendem a ter uma visão diferente do resto da equipa.

Especialista - É necessário ter especialistas de várias áreas, como especialistas do projeto, da tecnologia, do marketing, da logística e dos clientes da organização. Também é importante ter alguns especialistas de fora da organização.

Seguidamente, são descritas as ações realizadas em cada um dos 5 dias do *sprint* e são usadas algumas imagens de casos de estudo para ilustrar cada etapa. Um desses casos é uma aplicação para trabalhadores remotos encontrarem espaços públicos para trabalhar.

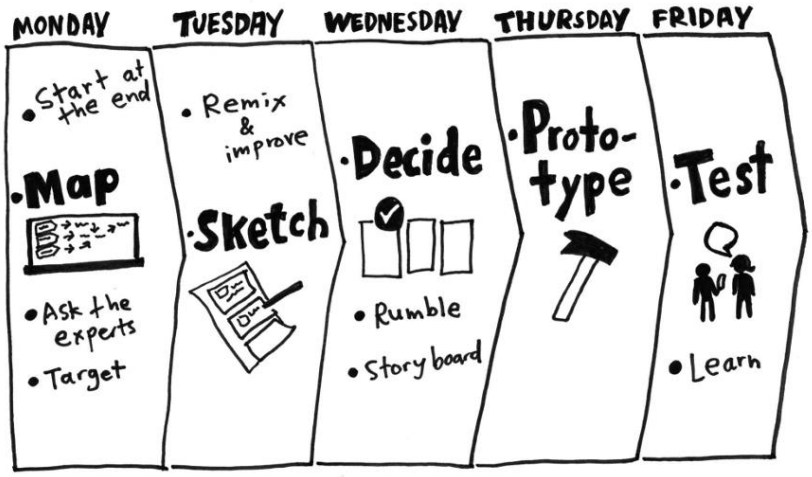


Figura 15 - Fases do Sprint de Design do Google
Fonte: (Knapp et al., 2016)

Segunda-feira - Entender e mapear o problema

No primeiro dia é explicado todo o processo do *sprint* de design do Google e as funções de cada elemento. Neste dia é definido o objetivo do *sprint*, a equipa extrai o máximo de informações possíveis dos especialistas através de entrevistas e são realizadas as 2 perguntas obrigatórias no início de cada *sprint*: O que precisa de ser verdade para o projeto ter sucesso? e O que pode causar o fracasso do projeto? (Knapp et al., 2016). No início de cada *sprint* mapeia-se o problema ao desenhar um fluxograma de 5 a 15 etapas, o fluxograma vai sendo melhorado ao longo do dia conforme é necessário (Ferreira & Canedo, 2018).

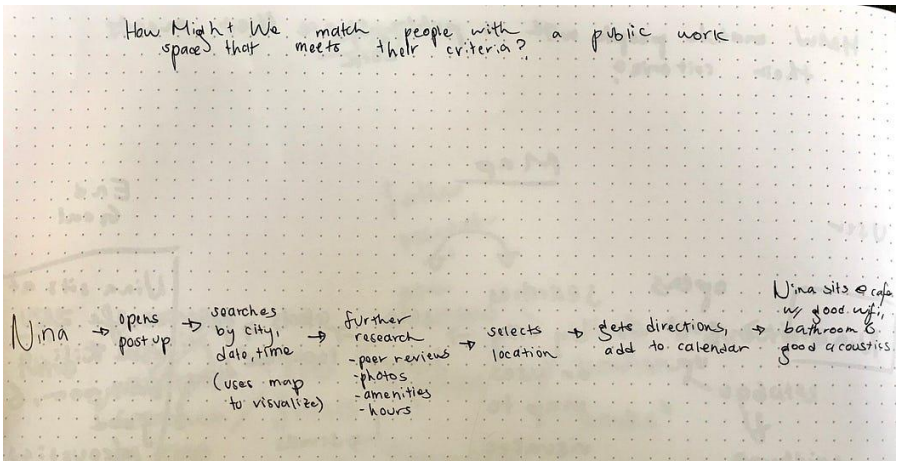


Figura 16 - Mapa de um Caso de Estudo

Fonte: (Tajima, 2019)

No fluxograma tem-se uma visão do caminho completo até à meta e é nele que se identifica as melhorias a realizar através da técnica *How Might We*. Esta técnica consiste em cada elemento da equipa escrever perguntas em cartões sobre os problemas que descobriram (Ferreira & Canedo, 2018).

Figura 17 - Cartões How Might We

Fonte: (Knapp et al., 2016)

Os cartões de todos os elementos são colocados num quadro e organizados por categorias. Depois é realizada uma votação, os cartões com mais votos vão para o lugar adequado no fluxograma (Ferreira & Canedo, 2018).

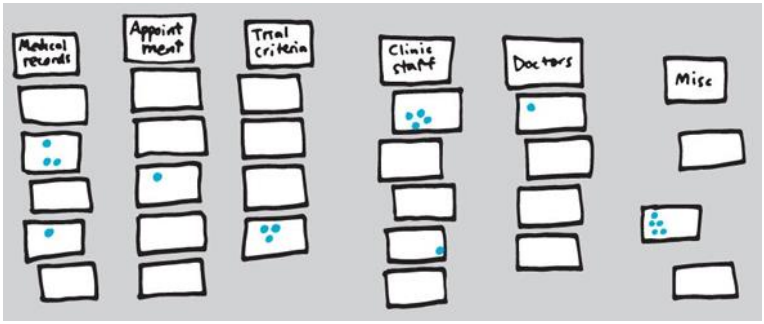


Figura 18 - Quadro de Votação dos Cartões How Might We
Fonte: (Knapp et al., 2016)

As últimas tarefas do dia são o decisor escolher um cliente alvo e o recrutamento para os testes de utilizador que irão ser realizados no último dia do *sprint*, sexta-feira (Araújo et al., 2019).

Terça-feira - Desenhar soluções

O segundo dia é exclusivamente dedicado às soluções, são realizadas *Lightning Demos* em que cada pessoa tem três minutos para apresentar informalmente uma ou duas soluções que criou. O facilitador dá um nome a cada solução e ilustra-a com um esboço simples. As melhores soluções são colocadas num quadro para que todos possam segui-las (Knapp et al., 2016).

Os criadores do *sprint* de design do Google concluíram que desenhar esboços é a maneira mais rápida e fácil de transformar ideias abstratas em soluções concretas. Após as *Lightning Demos*, cada indivíduo executa uma técnica de desenhar esboços dividida em quatro etapas (Ferreira & Canedo, 2018) (Larusdottir et al., 2019). O objetivo desta técnica é ajudar as pessoas a desenvolver as suas ideias.

- Notas - tirar notas relacionadas com o objetivo e com o mapa
- Ideias - escrever todas as ideias de como resolver o problema
- Crazy 8s - um exercício acelerado onde se desenha 8 variações de uma ideia em 8 minutos

- Esboço da solução - desenhar um esboço da solução o mais completo possível e que seja fácil de entender

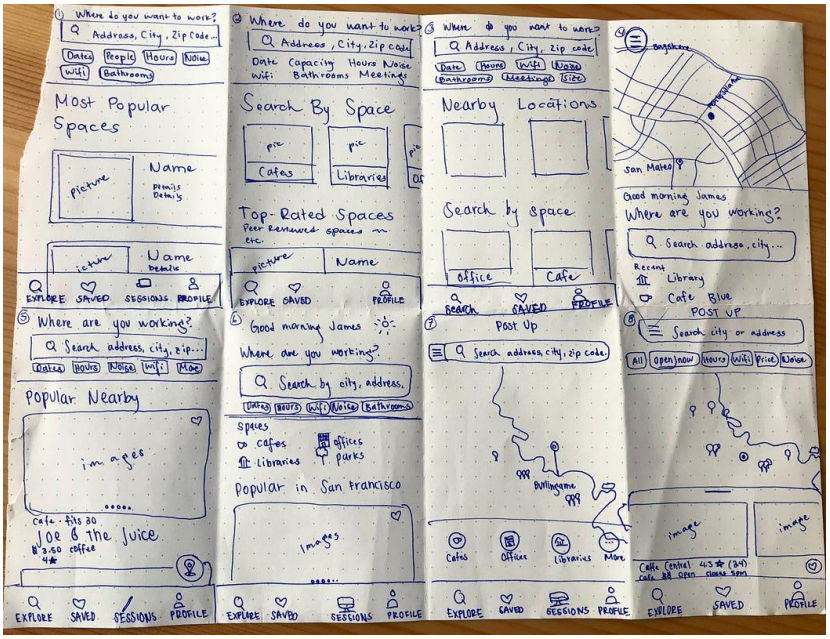


Figura 19 - Crazy 8s de um Caso de Estudo

Fonte: (Tajima, 2019)

O esboço da solução final deve ser autoexplicativo, ter um título criativo, usar as palavras certas, ser anónimo e ser natural, ou seja, o esboço não tem de ser perfeito (Araújo et al., 2019).

Quarta-feira - Escolher a melhor solução

No terceiro dia todos os esboços são colocados na parede, esta técnica é denominada de *art museum*. De seguida todos os elementos da equipa colocam *post-its* ao lado das partes que consideram interessantes dos esboços. Cada elemento pode colocar *post-its* em mais do que um esboço incluindo o seu próprio esboço. Estes *post-its* criam um *heatmap* com aglomerados reunidos em torno dos elementos mais interessantes. Se houver alguma questão sobre algum esboço escreve-se essa questão num *post-it* e coloca-se abaixo do esboço indicado. Depois a equipa discute os destaques de cada solução e regista as principais ideias (Knapp et al., 2016).

A seguir, cada pessoa vota numa solução e tem um minuto para explicar a sua escolha, cada voto vale um ponto. Os decisores têm 3 super votos, isto significa que as

soluções votadas por eles vão ser obrigatoriamente implementadas. Outra decisão que é tomada é se vão ser testados dois protótipos um contra o outro ou se se vai combinar o melhor das soluções que receberam super votos numa única solução.

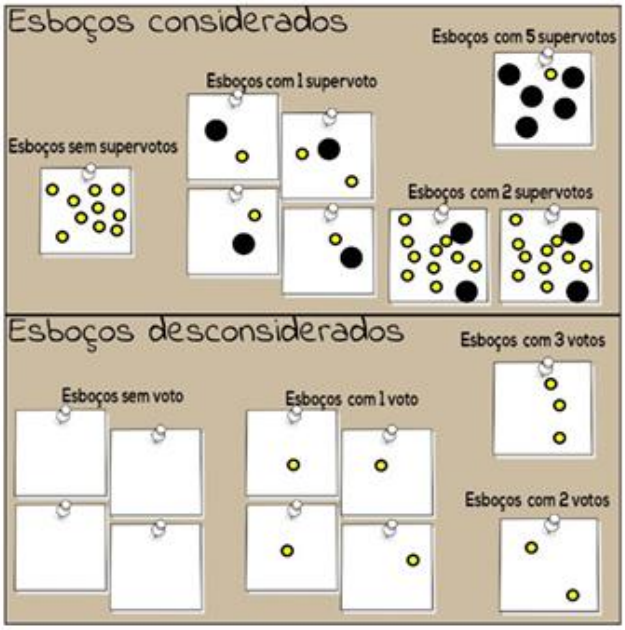


Figura 20 - Votação de Esboços

Fonte: (Ferreira & Canedo, 2018)

Por fim, alguém é escolhido para desenhar o *storyboard* com cerca de 15 painéis que vai ser utilizado para planear o protótipo.

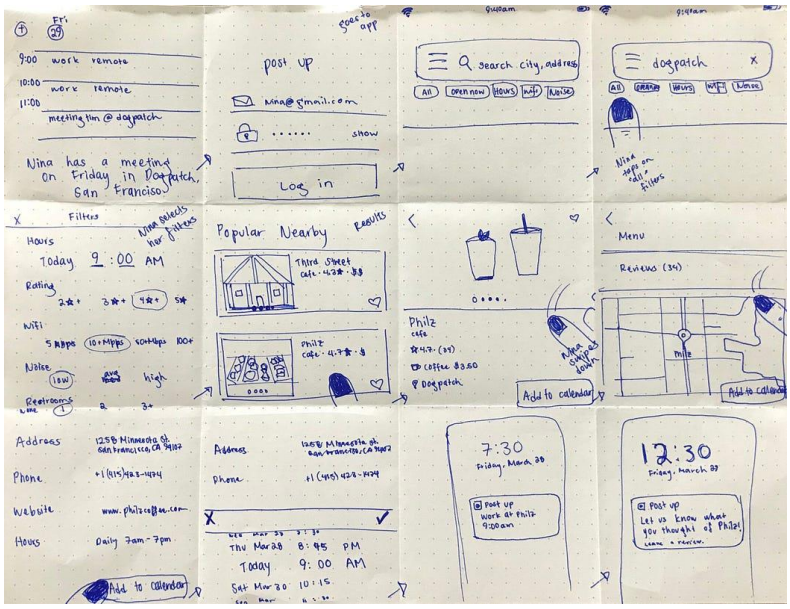


Figura 21 - Storyboard de um Caso de Estudo

Fonte: (Tajima, 2019)

Quinta-feira - Construir o protótipo

Construir um protótipo é uma excelente ideia porque permite testar um produto antes de começar a construí-lo e assim poupa-se tempo e recursos valiosos. Ao construir um protótipo está-se a criar uma superfície realista voltada para o cliente com base no *storyboard* previamente desenhado (Knapp et al., 2016). Para protótipos de aplicações e de websites é recomendável usar ferramentas como Keynote e PowerPoint, enquanto para protótipos de *hardware* é recomendável usar uma impressora 3D (Larusdottir et al., 2019).

Nesta fase, com a ajuda do facilitador são atribuídos diferentes papéis à equipa (Araújo et al., 2019):

- Criador - cria componentes individuais como ecrãs e páginas
- Costureiro - combina tudo, garante que datas, nomes e fontes estão consistentes e coerentes
- Escritor - torna o texto realista, usa frases simples e diretas
- Coletor de recursos - reúne itens como fotos, ícones e símbolos
- Entrevistador - escreve o guião das entrevistas dos testes de utilizador

No fim, a equipa verifica novamente o protótipo para confirmar que está de acordo com o *storyboard* e com as questões do *sprint*.

Sexta-feira - Testes de utilizador

A melhor maneira de descobrir como os clientes alvo reagirão ao produto é observá-los em ação a testar. Em relação à configuração dos testes, idealmente existem duas salas, uma para o entrevistador e clientes e outra para o resto da equipa observar os clientes a testarem o produto. Os clientes são filmados de modo a obter as suas reações ao produto, se for o caso de o produto ser num telemóvel, tablet ou computador grava-se também o ecrã do dispositivo (Knapp et al., 2016).

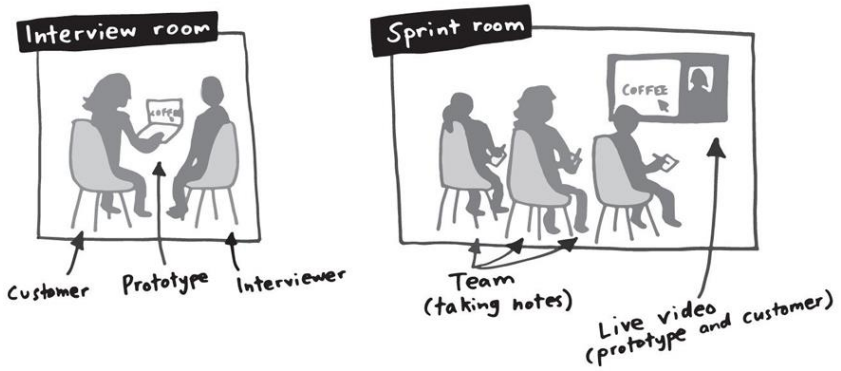


Figura 22 - Salas das Entrevistas
Fonte: (Knapp et al., 2016)

Em relação à entrevista, vários estudos defendem que cinco entrevistas é o número ideal de forma a obter o máximo de feedback com um investimento mínimo. O Google divide a entrevista em cinco partes (Knapp et al., 2016):

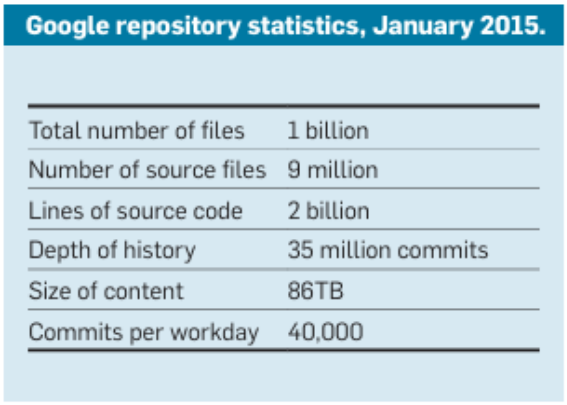
- Boas-vindas amigável - fazer com que os utilizadores se sintam bem-vindos, cumprimentá-los com um sorriso
- Perguntas de contexto - perguntas abertas sobre a experiência do utilizador ajudam a entender as suas reações
- Introdução ao protótipo - informar no início da entrevista que não é o produto completo e que por isso pode haver partes que não funcionem, assegurar aos clientes que não tenham receio de fazer algo errado, pois o produto está na fase de testes, e pedir que pensem em voz alta enquanto experimentam
- Tarefas e sugestões - pedir que imaginem cenários em que estão a usar o produto nas suas vidas, dar tarefas para completar e fazer perguntas para obter mais reações
- Resumo rápido - fazer algumas questões para avaliar a reação geral

No fim dos testes a equipa volta a ver os vídeos das entrevistas com especial atenção às reações dos clientes (Araújo et al., 2019). Cada elemento da equipa revê as suas notas à procura de padrões. Depois tentam perceber o que os padrões identificados dizem sobre os protótipos e como eles podem atingir o objetivo e responder às perguntas do *sprint*. No fim do *sprint* faz-se uma escolha, se há um protótipo vencedor, se se combinam elementos de alguns protótipos, se simplesmente se faz alterações no protótipo ou se se constrói um protótipo mais completo antes de iniciar outra ronda de testes (Knapp et al., 2016).

4.2.2.2 Trunk-Based Development

TBD é uma técnica Agile, proveniente do XP, adotada pela primeira vez no Google em 1999 quando os engenheiros de *software* decidiram que era melhor ter um único repositório com todo o código do que dividir o código em vários repositórios (Potvin & Levenberg, 2016). Esta técnica é usada por 95% dos programadores do Google nos seus projetos, sobretudo pelas equipas de DevOps. TBD é uma técnica de gestão de controlo de versões de código em que os programadores submetem *commits* num repositório. Atualmente, a maioria do código do Google está armazenado num repositório que pode ser acedido por todos os engenheiros, a maioria dos projetos partilha esse repositório. As únicas exceções são os projetos *open-source* Chrome e Android e alguns pedaços de código, de outros projetos, valiosos ou críticos para a segurança dos projetos (Henderson, 2017).

Ao longo dos anos, o repositório tem crescido imenso, como se pode comprovar nas estatísticas da figura 23, uma grande parte deste crescimento é devido, em 2012, ao Google ter adicionado suporte para utilizadores Windows e Mac, antes só havia suporte para utilizadores Linux. Antes de qualquer código ser submetido para o *trunk* central, a infraestrutura do Google realiza uma análise das alterações e uma série de testes automatizados. Para medir o desempenho do código e o envolvimento do utilizador quando são realizadas mudanças o Google usa A/B *testing* (Potvin & Levenberg, 2016).



Google repository statistics, January 2015.

Total number of files	1 billion
Number of source files	9 million
Lines of source code	2 billion
Depth of history	35 million commits
Size of content	86TB
Commits per workday	40,000

Figura 23 - Estatísticas do Repositório do Google

Fonte: (Potvin & Levenberg, 2016)

Os programadores costumavam desenvolver duas versões de *software* simultaneamente de modo a acompanhar as alterações implementadas e se necessário voltar

atrás e modificar o código. Com a evolução das técnicas e práticas, este processo tornou-se ineficiente, trabalhoso e dispendioso. Atualmente, a maioria dos programadores utiliza um de dois modelos de desenvolvimento para entregar *software* de qualidade, Gitflow ou TBD. Existe um conjunto de boas práticas de desenvolvimento que é importante ter em conta ao usar o TBD (Potvin & Levenberg, 2016):

- Executar revisões de código assíncronas
- Desenvolver o código em pequenos segmentos
- Implementar testes automatizados detalhados
- Criar e executar rapidamente
- Usar sinalizadores de recursos
- Poucas fases de integração
- Ter no máximo 3 *branches* ativos no repositório
- Juntar os *branches* ao *trunk* pelo menos uma vez por dia
- Poucos congelamentos de código

Num estudo realizado com participantes que já trabalharam com o repositório monolítico do Google foram feitas duas questões: qual foi o grau de satisfação ao usar o repositório do Google e qual foi o grau de satisfação ao usar vários repositórios (Jaspan et al., 2018).

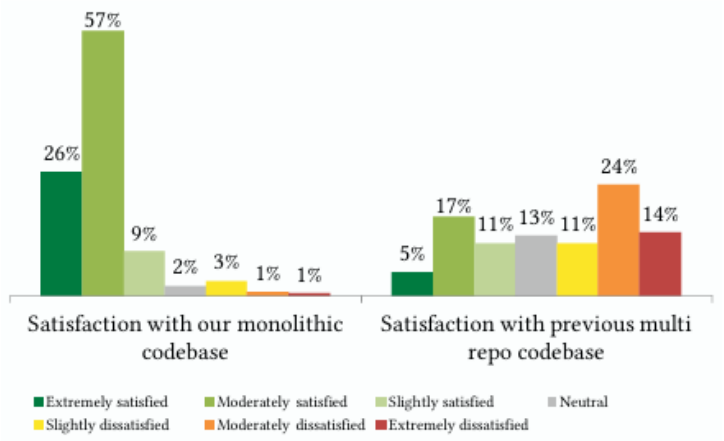


Figura 24 - Satisfação dos Engenheiros em relação à Utilização de Repositórios
Fonte: (Jaspan et al., 2018)

Como provado na figura 24, TBD facilita imenso a vida dos programadores e traz inúmeras vantagens aos projetos das organizações (Potvin & Levenberg, 2016) (Jaspan et al., 2018):

Visibilidade

A visibilidade do repositório permite que os programadores tenham acesso a todo o código do repositório, isto tem um impacto positivo na qualidade e na velocidade do código. Além disso, os programadores podem e são encorajados a editar código, isto é especialmente importante quando se realiza a migração de clientes de uma API.

Entregas regulares

Um dos objetivos do TBD é ter o *trunk* sempre verde, ou seja, estar pronto para ser implementado em qualquer altura. Para assegurar isto existem medidas a tomar como testes automatizados, revisões de código e convergência de código. Estas medidas garantem que a equipa tem agilidade para fazer entregas regulares e definir metas de lançamentos diárias de produção.

Poucos problemas

TBD é um modelo que gera poucos problemas comparativamente com modelos como Gitflow, porque no Gitflow quando se está a trabalhar num pedaço de código, ele já se encontra desatualizado com o *trunk*. Isto acontece devido a este modelo utilizar diferentes *branches* primárias para desenvolvimento, entregas, correções e recursos, enquanto no TBD utiliza-se só o *branch* principal para essas tarefas.

Trabalho cognitivo reduzido

Os programadores ao terem acesso a exemplos de código e às APIs têm como benefício que o seu próprio código seja mais simples de entender.

CI/CD

Com o crescimento do CI/CD os modelos de *branching* foram otimizados o que levou a uma grande evolução do TBD. Atualmente, o TBD é um requisito da integração

contínua, os programadores depois de realizarem os *commits* executam logo de seguida os testes automatizados para garantir que o projeto se encontra funcional em qualquer momento.

Ferramentas

Os programadores preferem trabalhar com as suas ferramentas de desenvolvimento preferidas. O Google desenvolveu especificamente para o seu repositório monolítico duas ferramentas: navegação de código e revisão de código. No estudo acima referido 88% dos inquiridos afirmaram que preferem as ferramentas do repositório do Google em vez das ferramentas que usam quando trabalham com vários repositórios.

Integração contínua

No TBD existe um único repositório onde são realizados todos os *commits* para o *branch* principal. Os testes automatizados de integração são executados sempre que é feito um novo *commit*, isto é feito para validar a qualidade do código. Esta prática é explorada em mais detalhe na secção 4.2.2.3.

Revisão do código

Commits pequenos tornam a revisão do código num processo mais eficiente. É mais fácil rever pequenos pedaços de código do que grandes. Nos grandes pedaços é necessário analisar muitas linhas de código para verificar se todas as alterações foram implementadas com sucesso.

Contudo, existem algumas desvantagens ao adotar esta técnica (Jaspan et al., 2018):

- Maior pressão - o código tem de estar sempre funcional
- Complexidade do código
- Risco de perder código - projetos grandes em que há várias equipas a trabalhar simultaneamente e existem milhares linhas de código

- Investimento em ferramentas de desenvolvimento, execução e integridade do código
- Equipa experiente - por vezes, é difícil recrutar uma equipa com a experiência desejada, preferencialmente que já tenha usado este modelo em projetos anteriores
- Difícil implementação - projetos complexos ou de grandes dimensões é aconselhável a utilização de modelos mais flexíveis como Feature Branching ou Gitflow

Em conclusão, TBD evoluiu bastante nos últimos anos e tornou-se numa prática comum das equipas de engenharia de alto desempenho. As principais razões disto ter acontecido é devido ao TBD fornecer mais flexibilidade e controlo sobre as entregas aos utilizadores. Este modelo de gestão de código é mais adequado para organizações com uma cultura colaborativa do que para organizações com uma cultura menos aberta onde partes do código são privadas.

4.2.2.3 Integração Contínua

A IC é uma prática Agile que automatiza a integração de alterações de código realizadas por vários programadores num projeto de *software*. Esta prática permite que se submeta frequentemente alterações de código no repositório. A IC ajuda o Google a agilizar as suas operações, esta prática automatiza a *build*, os testes e a implementação de *software*. O sistema de controlo de versão do código é o ponto-chave do processo que é complementado com ferramentas de revisão de estilo de sintaxe e com testes automatizados de qualidade de código.

A maioria das equipas do Google usa o Test Automation Platform, ferramenta responsável pela IC da maioria do código do Google e pela execução da maioria dos testes automatizados. O Google lida diariamente com mais de 50 mil alterações de código e executa mais de 4 biliões de testes individuais. Cada equipa cria um conjunto de testes unitários para o seu projeto para que sejam executados antes de uma alteração ser submetida. Quando um teste falha no TAP por causa de uma alteração implementada é crucial que essa alteração seja corrigida o mais rapidamente possível para evitar bloquear outros programadores. Para lidar

com estas falhas cada equipa tem um «Build Cop» que é o responsável por manter todos os testes aprovados no projeto. Quando ocorre uma falha num teste, ele abandona a sua tarefa e vai corrigir essa falha. As falhas geralmente são resolvidas de duas formas, reverte-se a alteração que provocou a falha, esta é a forma mais usada, ou corrige-se essa falha mais tarde, esta é a forma mais arriscada (Memon et al., 2017).

Ao implementar alterações antes de serem verificadas por todos os testes, o tempo médio de espera para submeter uma alteração diminui, para cerca de 11 minutos. O Google com esta prática e com o «Build Cop» é capaz de identificar e resolver com sucesso as falhas detetadas por testes mais longos com o mínimo de disrupção.

Cada *commit* tem um identificador único denominado de *changelist* ID. *Affected tests* são *test targets* que diretamente ou indiretamente dependem do código modificado, estes testes podem ser um script de teste *end-to-end*, um teste Java ou um conjunto de testes JUnit. O TAP tem de executar *affected tests* para assegurar que as últimas alterações não causaram quebras no código. Na figura abaixo tem-se um exemplo de como o TAP adia a execução de *affected tests*, representados pelos quadrados brancos, de modo a poupar tempo e recursos. Os *test targets*, representados pelos quadrados pretos, são apenas executados na última *changelist* ID, representadas como cl, por exemplo apesar do t11 ser afetado pelo cl4, cl7, cl15 e cl16, o t11 só é executado uma vez no cl16 porque o cl16 é o último cl que o afeta. No final cada *test target* tem o seu resultado, se positivo todos os testes foram concluídos com sucesso, se negativo pelo menos um teste falhou (Memon et al., 2017).

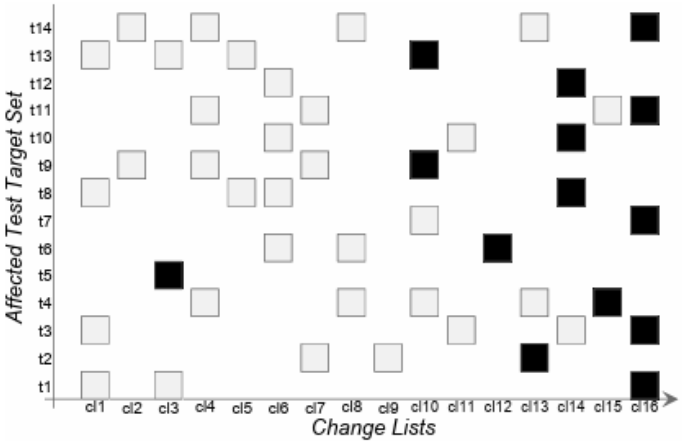


Figura 25 - Changelists e Test Targets
Fonte: (Memon et al., 2017)

Os grandes conjuntos de testes no Google tornam difícil encontrar a alteração específica que causa com que um teste falhe. Isto acontece devido a alguns problemas com a infraestrutura de testes, à prevalência de falhas e ao TAP ter de avaliar muitas alterações por dia, mais de uma por segundo, o que faz com que não consiga executar todos os testes a cada alteração implementada (Ziftci & Reardon, 2017). De modo a acelerar a identificação de falhas são usadas duas abordagens diferentes. A primeira abordagem é usar o TAP para dividir automaticamente um conjunto de falhas em alterações individuais e executar de novo os testes em relação a cada alteração isoladamente, este processo pode ser moroso (Memon et al., 2017). A segunda abordagem é usar ferramentas de localização de falhas onde o programador pesquisa binariamente um conjunto de alterações e identifica qual é a falha provável, estas ferramentas foram criadas pelo Google.

Para além do TAP, os programadores do Google usam um sistema distribuído de *builds* e testes, denominado Forge, para executar a maioria dos testes. Apesar da elevada quantidade de recursos computacionais que o Google possui, o TAP e o Forge têm recursos limitados (Memon et al., 2017). De modo a ultrapassar isto, foram criadas maneiras de identificar quais testes devem ser executados e quando para assegurar que se gasta a quantidade mínima de recursos para validar uma determinada alteração. O mecanismo principal é a análise do gráfico de dependências *downstream* para cada alteração. As ferramentas Forge e Bazel que o Google utiliza mantêm uma versão quase em tempo real do gráfico de dependências global e disponibilizam essa versão para o TAP. O TAP determina rapidamente que testes estão *downstream* de qualquer alteração e executa o mínimo de testes possível para garantir que se pode realizar a alteração. Outro mecanismo é a velocidade de execução dos testes, o TAP executa alterações com menos testes mais rápido do que alterações com mais testes. A diferença do tempo de espera entre uma alteração que realiza 100 testes e outra que realiza 1.000 pode ser de vários minutos. Os engenheiros que pretendem gastar menos tempo acabam por realizar alterações mais pequenas e focadas.

Um estudo realizado com foco no TAP tinha dois objetivos. O primeiro objetivo era fornecer aos programadores melhores dados dos resultados dos testes do TAP de modo a evitar quebras no código. Isto pode ser alcançado através de algumas medidas como usar ferramentas de análise estática e adicionar mais testes. O segundo objetivo era diminuir a carga de trabalho do TAP ao evitar reexecutar testes com altas probabilidades de sucesso, desta forma iria-se reduzir o tempo dos testes e poupar recursos preciosos. Dos 5,5 milhões de

testes analisados por este estudo apenas 63.000 falharam pelo menos uma vez (Memon et al., 2017).

Em 2016, John Micco, gestor de produtividade de engenharia no Google, abordou a IC tendo em conta a escala do Google. A IC fornece informação em tempo real ao lidar com testes instáveis e identificar rapidamente falhas e mudanças prejudiciais. A IC também fornece *green builds* regularmente que permite lidar com testes instáveis e mostrar os resultados de todos os testes. Testes instáveis são testes que podem ser impactados por fatores incontroláveis ou desconhecidos, como por exemplo: um recurso estar indisponível, um recurso não responder no momento da execução e o tempo de resposta de um servidor. Outra tarefa que a IC auxilia é ajudar os programadores a desenvolverem *software* com segurança, isto é feito através de um conjunto de medidas: identificar sempre se as mudanças no código irão interromper a *build* antes de submeter, sincronizar com a última lista de *green builds*, submeter código com confiança e lidar com testes instáveis (Micco, 2016).

A utilização da IC no Google trouxe vários benefícios (Micco, 2016):

- Identificar falhas
- Manter *green builds*
- Tempos de iteração rápidos, em alguns projetos os tempos de envio para produção são inferiores a 36 horas
- Identificar mudanças prejudiciais com precisão
- Equipas motivadas
- Reduzir os custos de computação

Apesar de todos os benefícios, o Google tem alguns custos na implementação. Requer atualização de dependências em cada alteração, isso demora tempo e faz com que os testes se atrasem, e requer um grande investimento em recursos computacionais. Os recursos crescem proporcionalmente a diversos fatores como o tempo dos testes e o aumento de dependências das bibliotecas (Micco, 2016).

4.2.2.4 Automação de Testes

A estratégia do Google foi criar uma abordagem de testes bem definida e focada nas equipas de forma a ajudar a IC. No início as equipas dependiam muito de operações manuais e apenas alguns engenheiros se concentravam nos testes e nas entregas de *software* (Whittaker et al., 2012). Com o passar dos anos o Google cresceu e percebeu que os longos ciclos de testes manuais atrasavam as entregas e a identificação e correção de *bugs* demorava cada vez mais tempo. O Google resolveu estes problemas ao automatizar os testes (Copeland, 2010). No Google existem três tipos de engenheiros na área dos testes (Whittaker et al., 2012):

- Engenheiro de *Software* - escreve o código, os testes unitários e a documentação e desenha a arquitetura
- Engenheiro de *Software* em Teste - foco na testabilidade, escreve estruturas de testes unitários automatizadas e melhora o código em relação ao desempenho e à segurança
- Engenheiro de Teste - principal foco nos testes de utilizador e automatiza características de testes de API e UI

O Google divide os testes em relação ao seu objetivo em três tipos: pequenos, médios e grandes. Outras empresas dividem em quatro tipos, têm mais um tipo de teste, os testes enormes. Todos os engenheiros podem executar qualquer tipo de testes. Os testes podem ser executados manualmente ou automatizados, a maioria dos testes automatizados são testes pequenos. Todos os testes que podem ser automatizados e não requerem intervenção humana devem ser automatizados (Whittaker et al., 2012). Apesar de haver cada vez mais testes automatizados nos projetos, os testes manuais continuam a ser necessários. Num estudo 80% dos inquiridos discordaram da visão de que os testes automatizados irão substituir todos os testes manuais (Rafie et al., 2012). Testes relacionados com temas como a privacidade e a criatividade devem continuar a ser testes manuais.

O foco dos **testes pequenos** são a corrupção de dados, erros funcionais, condições de erro e erros pontuais. Este tipo de teste é escrito pelos engenheiros de *software* e normalmente é executado em poucos segundos num ambiente fictício. A pergunta que cada teste pequeno deve conseguir responder é: Este código faz o que é suposto? (Whittaker et al., 2012).

A maioria dos **testes médios** é automatizada, escritos pelos engenheiros de *software* e envolvem duas ou mais funcionalidades, o seu foco é na interação das funcionalidades, chamada de funções *nearest neighbor*. Estes testes são executados em ambientes reais e fictícios. A pergunta que os testes médios tentam responder é: Um conjunto de funções *nearest neighbor* opera entre si da maneira que deve? (Whittaker et al., 2012).

Os **testes grandes** envolvem mais de três funcionalidades e representam situações reais. Todos os tipos de engenheiros estão envolvidos no processo de escrita dos testes. Este tipo de teste demora horas a ser executado. A pergunta feita nos testes grandes é: O produto funciona da maneira que um utilizador esperaria e produz os resultados desejados? (Whittaker et al., 2012).

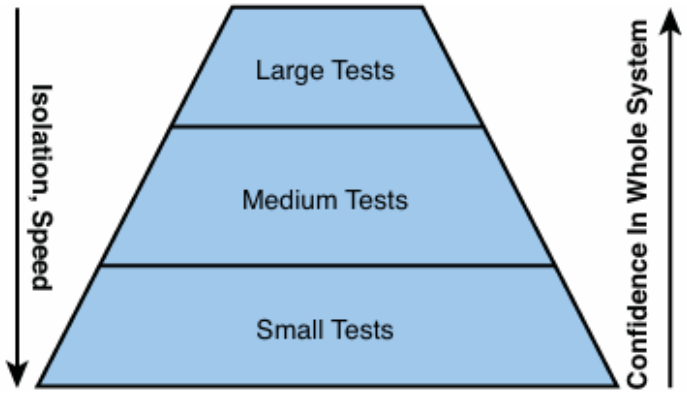


Figura 26 - Benefícios e Limitações dos Tipos de Teste
Fonte: (Whittaker et al., 2012)

Na área dos testes no Google existem alguns aspetos fundamentais como o foco nos testes de *software*, boa comunicação e a abordagem ágil e rápida das equipas. A comunicação fluida e eficiente nas equipas foi alcançada através de vários fatores como o facto do Google dar grande ênfase à comunicação interna da qualidade do produto. O Google publica internamente dados sobre cobertura de código o que facilita o aumento da adoção de práticas recomendadas. Os engenheiros que trabalham na área dos testes, ao contrário de outras organizações, estão integrados com o resto da equipa o que facilita a comunicação (Whittaker et al., 2012).

A abordagem das equipas permite pequenas e frequentes entregas e que os testes sejam pequenos e específicos. Os programadores são incentivados a minimizar as falhas nos testes e a procurar maneiras de acelerar os ciclos de testes. As equipas mudaram a sua forma de entregar produtos aos utilizadores. Atualmente, um produto ou uma atualização de um

produto é entregue ao utilizador logo que exista algum benefício, deste modo as equipas obtêm feedback imediato e começam logo a trabalhar na próxima etapa, em vez de só ser entregue nas grandes entregas (Whittaker et al., 2012). Para além disso, as equipas executam vários tipos de testes como testes unitários, testes de integração e testes de regressão (Copeland, 2010) (Henderson, 2017).

Os maiores impedimentos do sucesso e da melhoria contínua do Google são os *bugs* dos testes incompletos, as principais razões disto são:

- Os programadores não escrevem código de teste suficiente
- O código de teste tem menos manutenção do que o código da aplicação, por esta razão normalmente o código de teste tem mais *bugs* do que o código da aplicação
- A maioria dos programadores não realiza testes completos

A primeira base de dados de *bugs* do Google foi o BugsDB que foi utilizado até 2005. Esta simples base de dados tinha como objetivos guardar informação e calcular estatísticas dos *bugs* (Whittaker et al., 2012). Após o BugsDB, surgiu uma nova ferramenta para ajudar a combater os *bugs* no Google, o Buganizer, ferramenta de rastreio de *bugs*. Esta ferramenta rastreia diversos problemas como *bugs*, processos de cliente e pedidos de funcionalidades (Henderson, 2017). As equipas do Google verificam regularmente os *bugs* e gerem-nos de acordo com as suas prioridades, como se pode ver na figura 27, os *bugs* estão organizados do P0 ao P4, ou seja, do mais prioritário ao menos prioritário. A triagem dos *bugs* é realizada nas reuniões regulares da equipa ou por um elemento escolhido pela equipa (Henderson, 2017).

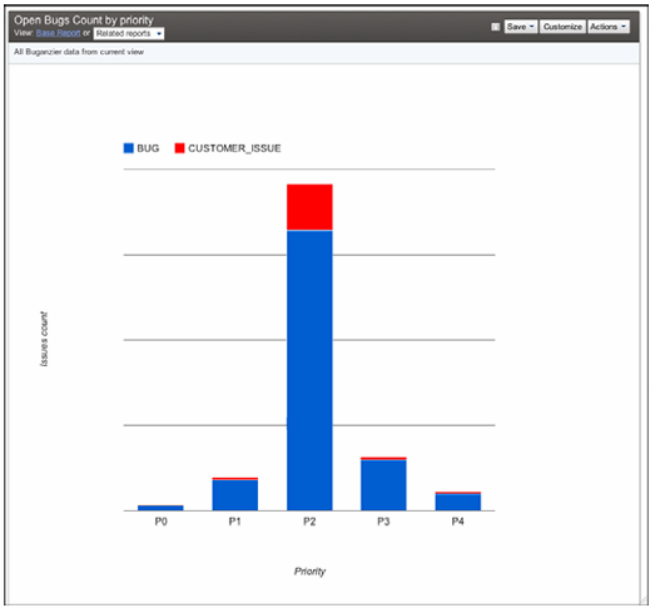


Figura 27 - Número de Bugs no Buganizer Organizados por Prioridade
Fonte: (Whittaker et al., 2012)

O Google usa múltiplas ferramentas nos seus processos de testes como o FindBugs, ferramenta *open-source* de análise estática para Java. Um estudo realizado sobre um FindBugs *fixit*, realizado em 2009, teve como finalidades avaliar o valor da ferramenta, identificar *bugs* e descobrir quais eram os *bugs* mais importantes para os engenheiros (Ayewah & Pugh, 2010). A principal conclusão deste estudo foi que os engenheiros consideram que os avisos do FindBugs devem ser resolvidos, mais de 77% das avaliações identificaram os avisos como defeitos reais. A maioria dos *bugs* encontrados não causavam problemas nos ambientes de produção, mas se tivessem sido detetados mais cedo os custos de resolução teriam sido menores.

Segundo um estudo, os principais benefícios da automação de testes são a cobertura dos testes e a reutilização de testes, e as principais limitações são as poucas ferramentas de testes, os custos envolvidos com os processos iniciais de automação e a formação dos funcionários. Nesse estudo, 45% dos inquiridos afirmaram que as ferramentas disponíveis no mercado não correspondiam às suas necessidades (Rafie et al., 2012).

Um exemplo onde a automação de testes foi implementada com sucesso no Google é o Gmail. O Gmail é um produto que esteve em versão beta durante quatro anos e durante esse tempo teve propositadamente a etiqueta beta para que os utilizadores soubessem que o

produto ainda estava a ser melhorado. A etiqueta só foi removida quando a meta de 99,99% de tempo de atividade para dados de e-mail de um utilizador real foi alcançada.

4.2.3 Considerações Finais sobre o Caso B

A adoção de metodologias Agile por parte do Google revelou desde muito cedo ser uma boa opção, o uso de práticas Agile foi fundamental porque permitiu que a organização continuasse inovadora e rápida enquanto crescia e tornava-se numa empresa líder e de referência no setor da tecnologia. A utilização de técnicas Agile e das suas ferramentas tem um grande impacto em diversos aspetos da organização. A utilização de ferramentas, como, TAP, Forge e Bazel na IC e Buganizer e FindBugs na automação de testes, desempenhou um papel crucial no aumento da rapidez e da qualidade do desenvolvimento de produtos. As metodologias de desenvolvimento, TBD, IC e automação de testes ajudam os programadores a desenvolver, organizar, integrar e testar o código mais eficientemente e com menos custos de retrabalho. O *sprint* de design permite que as equipas criem e testem protótipos em pouco tempo. O Google começou por dominar a área de pesquisas, mais tarde alargou o seu domínio para outras áreas como o e-mail e os mapas através do Gmail e Google Maps, respetivamente, neste momento está a tentar dominar a área da inteligência artificial.

4.3 Análise do Caso C - Cisco

4.3.1 Enquadramento da Empresa

No caso C analisa-se a implementação de práticas Agile na empresa Cisco Systems, Inc. A Cisco é uma empresa que desenvolve, fabrica e vende *hardware*, *software*, equipamentos de telecomunicação, produtos de alta tecnologia e outros serviços. A empresa foi fundada em 10 de dezembro de 1984 na Califórnia, EUA por Leonard Bosack e Sandy Lerner e tem cerca de 85.000 empregados.

A Cisco é uma empresa multinacional americana que se especializou em várias áreas como tecnologia de rede baseada em IP, sistemas de comutação, rede ótica, segurança e tecnologia sem fios. Os lucros da empresa vêm principalmente de duas formas. A primeira é a venda de produtos de *hardware* de alta qualidade, a empresa procura sempre melhorar os seus produtos para que acompanhem a evolução tecnológica. Os artigos mais vendidos são equipamentos de telecomunicações e infraestrutura de rede. A segunda são as taxas das subscrições. A Cisco fornece às empresas as ferramentas que necessitam para gerir os seus negócios online e offline. A Cisco possui vários produtos de sucesso como o WebEx, Jabber, Duo Security, OpenDNS e Jasper. Os seus principais competidores são Microsoft, Lenovo, Dell, NVIDIA e Alcatel-Lucent.

4.3.2 Transformação Agile

A Cisco começou por adotar a metodologia Waterfall nos seus projetos, mas isto trouxe alguns desafios complexos como processos de tomada de decisão lentos e complicados, dificuldade de adaptação às mudanças, longos e grandes ciclos de desenvolvimento e de testes e falhas de comunicação entre departamentos (Chen et al., 2016) (Power, 2016). Em 2006, de modo a acelerar a inovação, a Cisco começou por experimentar algumas práticas Agile (Pradhan & Nanniyur, 2021). Após várias experiências bem-sucedidas, em novembro de 2009, o Collaboration and Communications Group (CCG) da Cisco anunciou Agile como principal metodologia da empresa (Cisco, 2011). Nesta fase

foram contratados novos líderes seniores com um vasto conhecimento em Agile para orientar, treinar e promover o processo. Também houve um grande investimento na formação dos funcionários e em novas ferramentas. As equipas são livres de escolher as suas metodologias, as mais populares e usadas eram o Scrum, Kanban e XP (Power, 2016).

Em 2011, 85% das equipas de produto da CCG e 90% das equipas de sistema utilizavam metodologias Agile. O *software* era entregue em ciclos de 6 a 8 meses em vez de ciclos de 12 a 18 meses como era o caso na metodologia Waterfall, e os clientes recebiam *software* mais personalizado para os seus negócios (Chen et al., 2016). Ainda assim, grande parte dos projetos funcionava num ciclo de desenvolvimento híbrido Agile e Waterfall de modo a não haver uma completa disrupção, assim enquanto decorria a mudança para o Agile mantinham-se algumas práticas Waterfall, na sua maioria práticas relacionadas com a estabilidade e integridade ao nível do sistema (Pradhan & Nanniyur, 2021). Também em 2011, a Cisco desenvolveu a Cisco Standard Agile, metodologia Agile personalizada às suas necessidades, disponível a todas as equipas na Cisco, que fornece formação e acesso à ferramenta de gestão (Cisco, 2011).

Mais tarde, o departamento de Cloud and Software IT da Cisco ao seguir a estratégia de TI digital adotou um desenvolvimento mais flexível. O CSIT apercebeu-se que as práticas Agile apenas eram adotadas por algumas equipas pequenas e que Waterfall ainda era a norma para equipas grandes e em projetos complexos (Cisco, 2011). Por causa disso, o CSIT decidiu que o Agile seria a metodologia usada pela maioria das equipas. O uso do Agile ajuda a Cisco nos seus três princípios essenciais: foco no cliente, liderança no *software* e inovação.

As metodologias SAFe e Scrum foram aplicadas em dois grandes projetos na Cisco, na sua plataforma de cobrança de subscrições e na sua aplicação WebEx para tablets Samsung, respetivamente. Nas secções 4.3.2.2 e 4.3.2.3, estes dois casos são descritos ao detalhe, os seus problemas, as suas soluções e os seus resultados.

4.3.2.1 Práticas da Transição Agile na Cisco

A Cisco ao realizar a transição para a abordagem Agile enfrentou dois grandes desafios, o desenvolvimento de novas práticas de gestão que sejam compatíveis com as

práticas Agile e ajudar as equipas de engenharia e as unidades de negócio a adotar a metodologia Agile. É difícil para as equipas mudarem de metodologias, isso envolve muitas mudanças profundas, as equipas têm de mudar os seus hábitos, ferramentas e práticas. Para combater esta situação a Cisco criou a Agile Tiger Team (ATT) que ajuda as equipas nas tarefas de transição para a metodologia Agile. Abaixo abordam-se as práticas necessárias para a transição Agile implementadas na Cisco (Chen et al., 2016).

Liderança

Nos projetos Agile os líderes fornecem mais liberdade às equipas, nesta metodologia os líderes têm um papel de suporte, as suas funções são: ajudar as equipas a melhorar os processos, ajudar cada membro no desenvolvimento profissional e motivar os funcionários. Os líderes atendem a *demos* dos produtos de modo a acompanharem o progresso do projeto e fazer sugestões de melhorias.

Suporte

Para auxiliar a transição a ATT toma várias medidas. Primeiro, a ATT examina as equipas para identificar que aspetos precisam de ser alterados, como por exemplo é verificado se as equipas têm os papéis certos como *product owner* e *scrum master*. Depois a ATT fornece formação sobre os aspetos essenciais do Agile. Após a conclusão da formação, a ATT coloca um *agile coach* na equipa durante alguns meses para facilitar o início do processo de transição. Inicialmente, o *agile coach* e o responsável pela formação eram pessoas diferentes, mas isso causava alguns problemas, para evitar isso a ATT decidiu que seriam a mesma pessoa.

Coordenação

A coordenação na Cisco está dividida em três partes: entre equipas Agile, entre diferentes equipas e departamentos, e entre a organização e os seus parceiros externos. As equipas Agile estão sob grande pressão para entregar novas funcionalidades no fim de cada *sprint*, isto requer colaboração intensa entre todos. Para alcançar bons resultados em cada *sprint*, a Cisco aplicou algumas medidas como otimizar o tamanho das equipas para cerca de

7 a 9 pessoas, reduzir o número de tarefas de cada membro e recolocar todos os membros de uma equipa no mesmo local, foram criados espaços abertos para facilitar a colaboração das equipas.

A coordenação de equipas de diferentes áreas e departamentos pode-se tornar extremamente complexa, principalmente em projetos onde as equipas usam diferentes metodologias. Para minimizar estes desafios a Cisco desenvolveu algumas soluções como modularizar as interfaces de diferentes componentes e melhorar o design dos produtos.

Os processos Agile dependem muito da interação com os parceiros externos. As equipas de desenvolvimento precisam de receber o feedback dos clientes para corrigir os erros e melhorar no próximo *sprint*. As falhas na comunicação podem levar a resultados catastróficos, num projeto a Cisco teve de voltar a trabalhar em 80% do código de um produto.

Avaliação de benefícios

A ATT trabalha com as unidades de negócio para identificar os possíveis benefícios da implementação das práticas Agile. A ATT concluiu que os principais benefícios vêm de quatro áreas: satisfação dos clientes, tempo de entrega ao mercado, produtos adequados e motivação dos funcionários. A ATT demonstra os benefícios a todos os envolvidos no processo de transição de modo a ter a aprovação dos líderes e a colaboração de todos os membros das equipas. Apenas equipas que possam usufruir da totalidade dos benefícios devem adotar esta abordagem.

Planeamento

No desenvolvimento Agile o planeamento a longo prazo pode ser uma tarefa complicada. Alguns estudos afirmam que não é possível criar um planeamento detalhado para vários meses. Uma equipa da Cisco desenvolveu um processo que melhora substancialmente a previsibilidade do planeamento. A cada *sprint* esta equipa recebe o feedback dos clientes sobre as novas funcionalidades e a cada trimestre entrega novas funcionalidades no mercado. Esta equipa usa uma abordagem *bottom-up* onde duas equipas trabalham em conjunto, a equipa de execução e a equipa de planeamento.

Quando a equipa de execução começa a trabalhar num trimestre a equipa de planeamento organiza os dois trimestres seguintes com atenção especial no próximo trimestre, este processo repete-se até ao fim do projeto. A equipa de planeamento para organizar as tarefas detalhadas dos trimestres tem em conta o feedback fornecido pelos clientes, as informações do trimestre anterior e as reuniões bissemanais com a equipa de execução.

Recrutamento de clientes

O processo de recrutamento de clientes deve ser realizado tendo em conta as características, qualificações, e posição no mercado dos clientes e o novo produto que vai ser desenvolvido. Deve-se recrutar vários clientes de modo a minimizar os riscos de ser influenciado pelos maus clientes. Os clientes recrutados são diferentes dos clientes normais, os clientes recrutados têm de saber usar produtos inacabados e trabalhar com equipas Agile. Um mau recrutamento pode trazer imensos problemas e aumentar os custos do projeto. Num projeto na Cisco a equipa demorou mais de um ano a corrigir os erros cometidos devido ao mau recrutamento.

Avaliação de disponibilidade

A ATT identificou três condições para avaliar se uma organização está preparada para a transição: desenvolvimento do produto em fase inicial, adesão da liderança e interdependência das tarefas. As equipas ao usar Agile podem realizar mudanças na fase inicial dos projetos com o mínimo de prejuízo. Os líderes das equipas têm de estar comprometidos com o processo de transição, eles ajudam a construir uma nova cultura na organização e fornecem apoio às equipas. As equipas de engenharia estão em constante comunicação e colaboração com outras equipas o que, por vezes, causa que as tarefas das equipas estejam interligadas o que consequentemente aumenta a complexidade do projeto. A ATT avalia se a tarefa de uma equipa está interligada com outras equipas e favorece as equipas que trabalham em projetos com tarefas autónomas.

4.3.2.2 Plataforma de Cobrança de Subscrições da Cisco

Desde 2012 que a plataforma de cobrança de subscrições da Cisco cobra uma anuidade a vários serviços, tais como Cisco WebEx e Service Exchange Platform. A plataforma é um SaaS onde inclui portais internos e integração com Oracle Business and Revenue Management. No início foi adotada a metodologia Waterfall, foram formadas equipas diferentes para projetar, construir, testar e implementar. Cada equipa começava a trabalhar assim que a equipa anterior concluíra a sua parte. Isto levou a várias situações problemáticas, tais como ciclos de entrega demasiado longos, superiores a três meses em alguns casos, equipas não conseguirem entregar nas datas previstas, problemas de qualidade, encerramento tardio em documentos de requisitos, processos mais complexos e equipas trabalharem muitas horas e fins de semana para compensar os atrasos (Cisco, 2017).

Em 2015, de maneira a solucionar estes problemas, a Cisco adotou a metodologia SAFe. A Cisco lançou três Agile Release Trains (ARTs): capacidades, defeitos e correções, e projetos. ARTs são equipas multifuncionais focadas a trabalhar para um único objetivo. Inicialmente, havia alguma desconfiança na implementação dos ARTs porque as equipas pretendidas não estavam disponíveis num só local. Por causa disso, as equipas foram formadas com membros em diferentes países, a maioria dedicava todo o seu tempo a esse projeto, enquanto alguns tinham de dividir o seu tempo entre vários projetos. Idealmente, as equipas são formadas com todos os membros no mesmo local porque membros em diferentes países tendem a trabalhar como equipas separadas. As razões disto acontecer são as diferenças de fuso horário, as diferenças culturais e os diferentes líderes (Cisco, 2017).

As equipas de capacidade trabalham para o desenvolvimento da plataforma e as equipas de projeto trabalham no desenvolvimento e em entregas estáveis, ambas estão focadas na entrega de produto ao cliente. Os três ARTs trabalham juntos para criar, testar e entregar pequenas funcionalidades para a equipa de integração e teste do sistema. As equipas realizam *daily scrum* para determinar as tarefas de trabalho, alinhar o progresso e inserir itens no *kanban board*. Os funcionários em trabalho remoto visualizam o *kanban board* através da aplicação WebEx. Na figura abaixo está representada a diferença entre os ciclos de entrega antes e depois da implementação dos três ARTs.

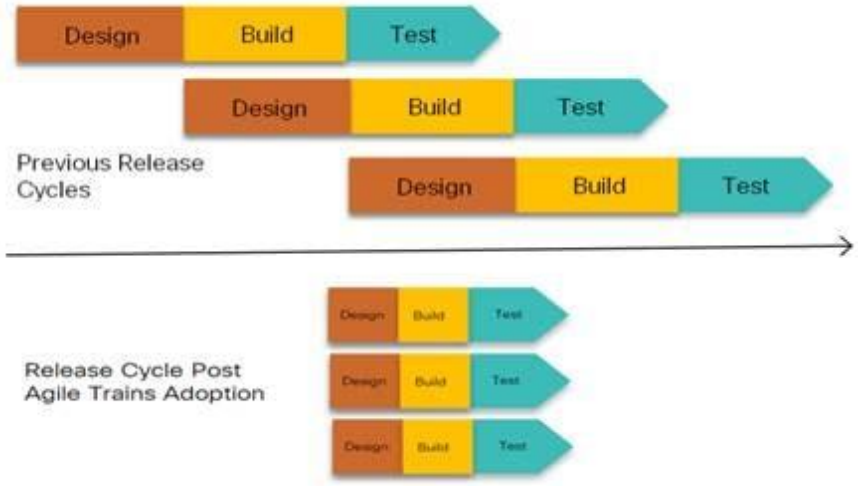


Figura 28 - Ciclos de Entrega Antes e Depois dos ARTs

Fonte: (Cisco, 2017)

Os resultados da implementação dos ARTs foram excelentes, a nova versão da plataforma de cobrança de subscrições foi entregue no prazo e com todos as capacidades planeadas. Com esta nova versão da plataforma a taxa de rejeição de defeitos diminui em 16%, os defeitos críticos diminuíram em 40% e os ciclos de entrega diminuíram em 50% (Cisco, 2017). Os ciclos de entrega de novas funcionalidades eram realizados com mais frequência o que ajudou a conseguir uma vantagem considerável no mercado em relação aos concorrentes. O CSIT atribui a melhoria da qualidade a vários fatores:

- Colaboração das equipas
- Foco no produto e no cliente
- Equipas independentes
- Estado do projeto disponível para toda a equipa consultar

Também se registou um aumento da eficiência de remoção de defeitos em 14%. Isto foi alcançado através da melhoria da colaboração entre equipas de diferentes países e a identificação de oportunidades de melhoria durante as *daily scrum* (Cisco, 2017). Os novos ciclos de entrega eliminaram a necessidade de as equipas terem de trabalhar horas extra, reduziu o número de reuniões por dia e ajudou os funcionários a perceber como se enquadravam no panorama geral da organização.

4.3.2.3 Aplicação WebEx para Tablets Samsung

Desde janeiro de 2014, a aplicação WebEx Meetings veio pré-instalada em tablets Android, cada pessoa que comprasse um tablet recebia seis meses de acesso gratuito ao WebEx Meeting Center. De modo a ter a aplicação concluída na data de lançamento os programadores tiveram de trabalhar rapidamente para cumprir os requisitos que mudavam frequentemente e para acelerar as alterações pedidas pela unidade de negócios (Cisco, 2017).

A Cisco adotou a metodologia Scrum para resolver esta situação. Foram aplicados três *sprints* de acordo com a localização. O primeiro *sprint* teve a duração de 3 semanas e o seu foco foi nos EUA e no Canadá. O segundo *sprint* também teve a duração de 3 semanas e o seu foco foi em 4 países europeus. O terceiro *sprint* teve a duração de 5 semanas e o seu foco foi em 7 locais na Ásia. Na fase de planeamento, alguns departamentos da Cisco incluindo o de TI, reuniram requisitos e avaliaram a preparação dos parceiros, ambientes e equipas de engenharia e marketing (Cisco, 2017). A equipa de desenvolvimento usou práticas do XP como o TDD. Primeiro escreveram casos de teste automatizados para as novas funcionalidades, depois escreveram o mínimo de código necessário para passar nos testes e por fim melhoraram o código para torná-lo mais simples e fácil de manter.

Como resultado da implementação da abordagem Agile neste projeto, os defeitos de garantia de qualidade diminuíram em 25% (Cisco, 2017). Para além disso, a unidade de negócios analisou novas funcionalidades mais cedo no ciclo comparativamente com os ciclos anteriores devido aos programadores verificarem o código várias vezes por dia e por causa disso cada *sprint* foi executado com mais eficiência que o anterior. Os *sprints* estão demonstrados nos gráficos *burndown* da figura 29 em que o eixo vertical representa o trabalho pendente em horas e o eixo horizontal representa o tempo em dias. Foram vendidos mais de 35 milhões de tablets Samsung pré-carregados com a aplicação WebEx o que significou um aumento significativo do reconhecimento da marca.

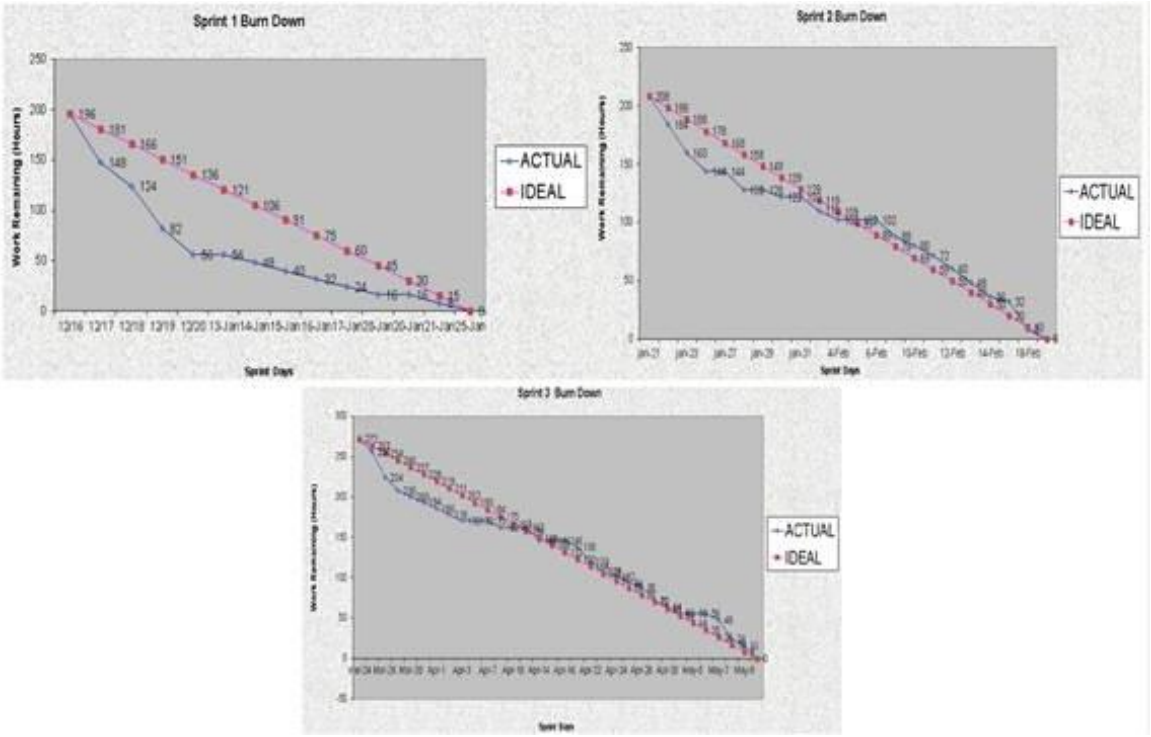


Figura 29 - Gráficos Burndown dos 3 Sprints
Fonte: (Cisco, 2017)

4.3.3 Considerações Finais sobre o Caso C

A Cisco ao mudar de abordagem transformou o seu negócio e tornou-se numa empresa mais competitiva e bem-sucedida. Neste processo foi criada a ATT para auxiliar as equipas nas tarefas de transição. Neste âmbito trabalhou-se em áreas como a liderança, o planeamento, o suporte, a coordenação e o recrutamento de clientes. Neste caso abordou-se dois projetos da Cisco em que ocorreu uma mudança para metodologias Agile. No primeiro projeto adotou-se SAFe e lançou-se três ARTs com o objetivo de solucionar os problemas relacionados com a plataforma, como, os ciclos de entrega longos e os atrasos dos documentos de requisitos. No segundo projeto adotou-se Scrum e implementou-se três *sprints* com o objetivo de ter a aplicação concluída dentro do prazo estabelecido.

4.4 Análise do Caso D - Amazon

4.4.1 Enquadramento da Empresa

No caso D analisa-se a implementação de práticas Agile na empresa Amazon.com Inc. A Amazon é uma empresa multinacional americana que se foca no *e-commerce*, *streaming*, *cloud computing* e inteligência artificial. A empresa foi fundada em 5 de julho de 1994 em Washington, EUA por Jeff Bezos e tem cerca de 1.125.000 funcionários.

A Amazon começou por vender livros online e mais tarde expandiu-se para vender videojogos, brinquedos, roupa, móveis entre outros, atualmente vende quase de tudo. A Amazon possui outros produtos de sucesso como Amazon Prime Video, Amazon Studios, Amazon Music e Amazon Web Services. A principal fonte de receitas da empresa é o *e-commerce*. Outras fontes significativas são publicidade, *cloud services* e serviços de subscrição. Os seus principais rivais em relação ao *e-commerce* são JD, LightInTheBox, Overstock, Alibaba, Vipshop e Wayfair.

4.4.2 Transformação Agile

A Amazon tem aplicado práticas Agile desde a sua criação, a transformação que a empresa sofreu ao longo dos anos é descrita por muitos como descentralizada e não planeada (Denning, 2019b). Desde 1999 que a empresa utiliza práticas Agile na gestão dos seus projetos e das suas equipas. Entre 2004 e 2009, o Scrum foi sendo cada vez mais adotado pelas equipas, nomeadamente pelas equipas de desenvolvimento de *software* (Atlas, 2009). Esta adoção foi facilitada através das equipas pequenas e autónomas e da cultura da empresa. Em 2010, a empresa começou a realizar entregas contínuas em grande escala. A Amazon ao dar às equipas o poder de decisão em relação à forma como trabalham e ao criar equipas estáveis e de longo prazo contribuiu para que a abordagem Agile tivesse sucesso (Harracá, 2017). A Amazon concentra-se em partilhar conhecimento e promover um melhor entendimento através da formação, dos *agile coaches* e das comunidades.

Ao longo dos anos a Amazon tem demonstrado uma enorme agilidade estratégica ao desenvolver novos serviços de sucesso como AWS. Isto tem sido alcançado graças à mentalidade ágil do seu antigo CEO, Jeff Bezos, que tem como principal prioridade a visão a longo prazo (Denning, 2018). Os principais elementos da abordagem da Amazon são tomar decisões rápidas, a obsessão de entregar mais valor aos clientes e a criação de uma cultura *start-up* que valoriza a inovação, criatividade e crescimento e que arrisca e aprende com os seus erros. Seguidamente, descreve-se práticas Agile implementadas pela Amazon que a ajudaram a transformar a forma de entregar valor.

4.4.2.1 Dia 1 da Amazon

O dia 1 da Amazon é uma cultura onde o cliente é o centro de tudo o que é realizado na empresa. O dia 1 significa ter coragem para experimentar, não ter medo de errar, ser flexível, tomar rápidas e boas decisões, adotar uma cultura *start-up* e encorajar os funcionários a serem curiosos. Em 1997, Jeff Bezos numa carta escrita aos *stakeholders* destaca três aspetos fulcrais para o sucesso da Amazon: obsessão pelo cliente, visão a longo prazo e incentivo à inovação (Denning, 2019b). Nessa carta, Bezos escreveu “Este é o Dia 1 para a Internet… e, se fizermos tudo certo, para a Amazon.com” (Slater, 2023).

À medida que uma empresa cresce é crucial que ela continue a corresponder às diferentes necessidades dos clientes e adapte-se aos ambientes externos. O objetivo da Amazon é que seja sempre dia 1 para que nunca se passe para o dia 2. A cultura do dia 2 é uma cultura menos flexível onde os principais aspetos são o foco nos processos e as decisões são mais lentas. Na carta escrita aos *stakeholders*, em 2017, Bezos afirma “O Dia 2 é uma estagnação. Seguida por irrelevância. Seguida por um declínio doloroso e excruciante. Seguida pela morte. E é por isso que é sempre Dia 1" (Slater, 2023). De forma a evitar o dia 2 as empresas devem seguir um conjunto de práticas: continuar com o foco no cliente, atenção contínua aos processos diários e continuar a inovação em novos produtos e serviços. Apenas implementar a cultura de dia 1 não é suficiente para as empresas continuarem a ser relevantes, é preciso adotar as práticas corretas e ter a estrutura organizacional mais adequada. Na tabela abaixo demonstra-se as diferenças entre o dia 1 e o dia 2 (Galetti et al., 2019) (Slater, 2023).

Tabela 2 - Diferenças entre o Dia 1 e o Dia 2 da Amazon

Fonte: (Autor)

Dia 1	Dia 2
Obsessão pelo cliente	Obsessão pelos processos de negócio
Tomada de decisões rápida	Tomada de decisões lenta e burocrática
Promove a criatividade e a tomada de riscos	Promove o controlo
Estruturas organizacionais ágeis	Estruturas organizacionais em camadas
Visão a longo prazo	Visão a curto prazo
Equipas pequenas e autónomas	Equipas grandes e com muitas dependências
Adota tendências externas	Inabilidade de adotar tendências externas
Evita usar intermediários	Usa intermediários para medir o sucesso do negócio
Não tem medo de errar	Medo de errar

4.4.2.2 Obsessão com a Experiência do Cliente

Um princípio primordial da gestão Agile é os clientes serem o centro do negócio. O cliente na Amazon é visto como um influenciador na forma como a empresa deve trabalhar para entregar produtos e serviços (Denning, 2019a). Para qualquer empresa ter sucesso é fundamental conseguir ganhar a confiança dos clientes de modo a construir e manter uma boa relação. A obsessão pelo cliente é o primeiro princípio dos 16 princípios de liderança na Amazon (Harracá, 2017). Na AWS 90% dos recursos desenvolvidos são o resultado das necessidades dos clientes (Slater, 2023).

A obsessão em aumentar o impacto da experiência do cliente é implementada por todos na organização, desde os líderes das equipas de marketing e vendas até aos funcionários dos armazéns (Denning, 2019a). Nas reuniões da Amazon existe sempre uma cadeira onde ninguém se senta, isto acontece porque, segundo Bezos, essa cadeira está reservada para a pessoa mais importante da reunião, o cliente. O feedback dado pelos clientes é importantíssimo para a Amazon, Bezos costumava ler as reclamações dos clientes e enviava-as para os gestores dos departamentos com um ponto de interrogação, isto servia de lembrete para pensar nos clientes e no feedback.

Esta obsessão é possível através das métricas orientadas ao cliente (Denning, 2019a). Na maioria das empresas os dados financeiros são a principal influência no processo de tomada de decisões, mas na Amazon são os dados dos clientes. As métricas do cliente são incluídas em todas as decisões, as equipas só iniciam uma atividade após perceberem como irão medir a resposta do cliente (Denning, 2019b).

4.4.2.3 Modelo de Equipa Two-Pizza

As equipas *two-pizza*, criadas por Bezos em 2002, são equipas pequenas, independentes e multidisciplinares organizadas em torno dos serviços e capacidades (Denning, 2019b). Uma equipa *two-pizza* significa que o tamanho da equipa é o número de pessoas que conseguem comer duas pizas, normalmente 6 a 10 pessoas. Os principais objetivos destas equipas são tomar decisões rápidas, obsessão pelo cliente e evitar distrações. A comunicação com as outras equipas é limitada e bem documentada com a intenção de diminuir a sobrecarga de comunicação. Estas equipas têm os seus próprios dados que as outras equipas não podem aceder, exceto através de APIs. Os principais benefícios deste modelo de equipa são:

- Manter a produtividade
- Acelerar a inovação
- Promover a experimentação
- Reduzir os custos dos erros

O Single-Threaded Owner (STO) é o líder responsável por gerir equipas *two-pizza* e fornecer-lhes suporte (Galetti et al., 2019). A Amazon usa um processo anual de planeamento, conhecido como OP1, em que as equipas descrevam os recursos, investimentos e metas pretendidos para o próximo ano. Para além disso, são realizadas revisões de programas, revisões de *roadmaps* e análises de negócios semanalmente. Estes processos ajudam os STOs a tomarem melhores decisões.

O indicador de performance usado pelas equipas *two-pizza* é uma função de aptidão. Esta função é uma avaliação quantitativa acordada entre a equipa e o seu líder que funciona como um indicador de desempenho (Harracá, 2017).

A Amazon através das equipas *two-pizza* aplica o princípio Agile de ter equipas pequenas e focadas que projetam, desenvolvem e entregam produtos rapidamente e frequentemente. No Agile o uso de equipas pequenas permite o desenvolvimento rápido com menor custo e risco para as empresas, na Amazon a maioria dos ciclos de produtos têm a duração entre 10 a 15 dias. Equipas pequenas permitem um maior foco nas necessidades de cada cliente, aumentam a satisfação dos funcionários e reduzem a burocracia que é normalmente associada a equipas de grande dimensão (Atlas, 2009).

4.4.2.4 Método Kaizen

A Amazon adotou o método Kaizen para melhorar a qualidade dos seus processos. O Kaizen é uma abordagem de negócios que se foca em aumentar a produtividade e melhorar o ambiente de trabalho. Esta abordagem *bottom-up* ajuda a descobrir problemas operacionais relacionados com as tarefas diárias nos locais de trabalho, neste caso nos armazéns da Amazon. Este método faz com que haja maior consciência sobre os desafios que os funcionários enfrentam diariamente. O método Kaizen ajuda a Amazon a melhorar as tarefas nas linhas de produção e expedição e permite redefinir os seus processos para atender às mudanças nos requisitos.

Um exemplo deste método foi quando o antigo CEO da Amazon, Jeff Bezos, teve um acidente quando trabalhou na área de embalagem num dos armazéns, ele deixou cair uma encomenda de produtos de higiene e as garrafas de vidro da encomenda partiram-se no chão o que causou um risco, alguém podia escorregar e magoar-se. Ele percebeu que este pequeno acidente podia ter originado um grave acidente de trabalho para um dos seus funcionários e podia ter atrasado os prazos de entrega da Amazon. De modo a resolver este tipo de situações, ele decidiu que as regulamentações relacionadas com os materiais, conteúdo e posição das embalagens no armazém necessitavam de ser ajustados para evitar outros acidentes deste tipo (Onetto, 2014).

4.4.2.5 Liderança

A liderança na Amazon é descentralizada, ou seja, é uma liderança onde cada pessoa é responsável pelo seu próprio trabalho. Isto permite que haja espaço para inovar na forma como as pessoas resolvem os desafios diários sem terem de se preocupar demasiado com a burocracia. Na Amazon os líderes pensam nos projetos a longo prazo, incentivam o trabalho de equipa e promovem uma cultura de constante procura pela perfeição (Denning, 2018) (Denning, 2019b).

As equipas descentralizadas podem tomar dois tipos de decisões, *one-way-door* ou *two-way-door*. As decisões *one-way-door* são decisões que depois de tomadas é muito difícil de revertê-las, enquanto as decisões *two-way-door* são decisões que depois de tomadas são facilmente revertidas. As equipas devem demorar o tempo que for necessário a tomar decisões *one-way-door* e devem ser rápidas a tomar decisões *two-way-door*. Na maioria dos projetos, as equipas tomam mais decisões *two-way-door* do que *one-way-door*.

Segundo vários estudos realizados pela Organizational Network Analysis (ONA), as principais razões dos processos de tomada de decisões das equipas serem lentos são: as muitas reuniões, a dificuldade de alinhar prioridades, a dificuldade para obter aprovações, demasiados erros, dados insuficientes e demasiada documentação (Galetti et al., 2019). Para combater isso, a Amazon decidiu implementar uma forte e bem definida liderança.

4.4.2.6 Processo Working Backwards

O processo *working backwards* é um processo onde se começa pelo cliente e só no fim se desenvolve o produto. Ao longo dos anos, vários produtos de sucesso como Amazon Prime Video e Alexa utilizaram este processo. Este processo é realizado antes de se iniciar um projeto para testar uma ideia, através de um comunicado de imprensa, FAQs e um manual do produto (Denning, 2019b). O comunicado de imprensa é um documento de uma página, escrito após se decidir o que se vai desenvolver. O objetivo do comunicado é descrever de maneira simples o produto ou o serviço e as vantagens que fornecem ao cliente. As FAQs são

usadas para descrever o funcionamento do produto, a sua forma de execução, identificar riscos e ajudar na decisão de investir num projeto.

4.4.2.7 Aprender com os Erros

A Amazon adotou uma cultura *start-up* onde uma das principais características é o não ter medo de falhar, só deste modo poderá haver inovação. As equipas na Amazon correm riscos ao desenvolver novos produtos, elas experimentam, inovam e criam com o objetivo final de obter produtos focados no cliente e tornar os ciclos de desenvolvimento mais rápidos. As equipas não têm medo de falhar, elas aprendem com os seus erros de forma a melhorar sempre o produto. A Amazon falhou em diversos projetos como Fire Phone, Amazon Wallet e Amazon WebPay, mas teve muito sucesso em outros como AWS que atualmente representa 10% das suas receitas e 74% dos seus lucros (Quast, 2024).

4.4.2.8 Flexibilidade

A flexibilidade nas equipas e na tecnologia são dois aspetos muito importantes na Amazon. Manter as equipas flexíveis é uma grande passo para aumentar a agilidade da empresa e gerir melhor os conflitos. Um dos principais benefícios da arquitetura da Amazon é a facilidade de dimensionar e mover a infraestrutura. De forma a gerir os dados provenientes do feedback dos clientes foram adotados *cloud services* filtrados por várias APIs. A utilização de ESBs e de APIs para extrair os dados resultou numa infraestrutura ágil e eficiente. Ao alinhar com sucesso a TI com os seus negócios a Amazon promove a inovação e o desenvolvimento de produtos com o foco no cliente.

De forma a ajudar os gestores a gerirem as suas equipas o departamento de recursos humanos da Amazon utiliza uma ferramenta chamada Connections que fornece feedback da equipa. Esta ferramenta faz uma questão a cada funcionário todos os dias, as questões podem ser sobre vários temas como os desafios do projeto, a liderança do gestor, e a comunicação da equipa. A utilização desta ferramenta permite que a Amazon compreenda, detete e resolva os

desafios que os seus funcionários enfrentam diariamente o que resulta num aumento da produtividade e num melhor ambiente de trabalho (Kim, 2018).

4.4.3 Considerações Finais sobre o Caso D

Apesar da Amazon não afirmar ser Agile, o sucesso que tem tido ao longo dos anos só é possível por causa da sua cultura e filosofia Agile. As práticas mencionadas acima inspiram as equipas a evoluir e a resolver problemas com rapidez e agilidade. A evolução da Amazon é feita através de uma liderança descentralizada com equipas pequenas com muita autonomia. O aspeto organizacional Agile da Amazon é evidente com a criação das equipas *two-pizza* que são fundamentais na estrutura da empresa, elas foram criadas para que os funcionários se mantenham próximos dos clientes e tomem decisões rápidas. Práticas como o método Kaizen e o processo *working backwards* têm um grande impacto nos processos de desenvolvimento. Para além disso, a Amazon implementou algumas práticas adaptativas como conferências e *hackathons* com o objetivo de ajudar as equipas a inovar ao desenvolverem novos produtos e serviços focados no cliente.

Capítulo 5 - Análise dos Casos de Estudo

Neste capítulo pretende-se analisar os quatro casos de estudo apresentados no capítulo anterior com ênfase nos resultados obtidos através da implementação de práticas Agile. A análise realizada neste capítulo é uma análise qualitativa. Não é possível comparar quantitativamente os casos de estudo porque as áreas de mercado de cada caso são todas diferentes, apesar de estarem todas relacionadas com a tecnologia. No caso A é o *streaming* de música, no caso B são várias áreas como o motor de pesquisa e o Gmail, no caso C é a venda de *hardware* e *software* relacionado com as telecomunicações e no caso D é o *e-commerce*.

No caso A a empresa criou e desenvolveu o seu próprio modelo Agile de modo a conseguir acompanhar o seu crescimento. A empresa reestruturou a organização das suas equipas em estruturas como *squads*, *tribes* e *chapters* de forma que elas tivessem mais independência e flexibilidade e houvesse mais colaboração entre as equipas. O sucesso do modelo é devido ao uso de uma abordagem baseada na experimentação para a resolução de problemas num ambiente relaxado, de confiança e de transparência.

A estrutura do modelo Spotify é um excelente caso de estudo para o design organizacional moderno, fornecendo bastante conhecimento sobre como a autonomia, o alinhamento e a colaboração multifuncional impulsionam o sucesso de uma empresa num cenário de negócios em constante mudança. A implementação do modelo Spotify trouxe imenso valor para toda a organização o que permitiu a sua evolução de uma pequena *start-up* para uma das maiores empresas do mundo na sua área.

As principais vantagens do uso do Agile neste caso foram a adoção de uma cultura ágil e flexível e a reestruturação da organização com a criação de equipas autónomas e multifuncionais. A principal desvantagem é o modelo desta organização não ser recomendável para pequenas empresas.

No caso B a organização aplica várias práticas de diferentes metodologias Agile, nomeadamente Scrum e XP, conforme os projetos em causa. Inspirado no *sprint* do Scrum, o Google criou o seu próprio *sprint*, o *sprint* de design, com o principal objetivo de ajudar as equipas a desenhar protótipos e a testar novas funcionalidades. TBD é uma técnica utilizada pelo Google que ajuda os programadores a gerirem mais eficientemente todo o código dos

vários serviços do Google. Algumas práticas Agile adotadas foram a IC e a automação de testes, estas duas práticas muitas das vezes são aplicadas em conjunto, tendo a última sido aplicada num produto de muito sucesso do Google, o Gmail.

A implementação destas práticas Agile ajudou de forma decisiva na evolução da empresa. Estas práticas tiveram um grande impacto na forma de desenvolver e testar código, os principais resultados disso foram a redução de custos e a rápida entrega de *software*. As figuras abaixo demonstram como apenas em quatro anos estas práticas ajudaram no crescimento do Google em relação à sua velocidade e escala.

Figura 30 - Velocidade e Escala do Google em 2012

Fonte: (Micco, 2016)

Figura 31 - Velocidade e Escala do Google em 2016

Fonte: (Micco, 2016)

As principais vantagens do uso do Agile neste caso são as técnicas utilizadas nos processos de desenvolvimento que contribuem no planeamento, desenvolvimento, organização, integração, testes e entrega de *software*. A principal desvantagem é ser necessário ferramentas mais especializadas o que pode implicar investimentos avultados.

No caso C a organização também aplica várias práticas Agile nos seus projetos, a metodologia mais utilizada é SAFe. Neste caso são abordadas as práticas de transição para as metodologias Agile e são descritos dois projetos onde foram implementadas abordagens Agile com excelentes resultados. No primeiro projeto foram resolvidos vários problemas relacionados com a plataforma de cobrança de subscrições como entregas tardias, ciclos de lançamento demasiado longos e problemas de qualidade. Os principais resultados obtidos foram que os defeitos críticos diminuíram em 40% e os ciclos de entrega diminuíram em 50%. No segundo projeto as equipas conseguiram cumprir o objetivo principal que era ter a aplicação WebEx concluída na data prevista, conseguiram também que os defeitos de garantia de qualidade diminuíssem em 25%.

O enorme sucesso da Cisco deve-se a vários fatores que influenciam como toda a organização funciona: a mudança da cultura da empresa para uma cultura onde se dá valor à colaboração, à melhoria contínua e ao feedback; o investimento em formação para os funcionários aprenderem e ficarem familiarizados com as práticas Agile; a utilização de métricas para acompanhar a transformação Agile e fazer ajustes se necessário; e, por fim, uma forte adesão e apoio da liderança de toda a empresa à transformação Agile.

As principais vantagens do uso do Agile neste caso foram a cultura implementada e os excelentes resultados obtidos nos projetos em que houve mudança para uma metodologia Agile. A principal desvantagem é o investimento que é fundamental para realizar a transição de metodologias, este investimento tem de ser efetuado em várias áreas como na formação dos funcionários e em novas ferramentas.

No caso D a organização implementou várias práticas de diferentes metodologias Agile nos seus projetos, sendo a principal Scrum. A organização é conhecida por ser um local exigente de trabalhar onde são esperados resultados perfeitos. Os excelentes resultados obtidos pela organização só têm sido possíveis por causa da adoção de uma cultura de *start-up* e da implementação de práticas Agile como o método Kaizen, a liderança descentralizada e o modelo de equipa *two-pizza*. Estas práticas têm sido aplicadas desde os primeiros projetos e têm tido o papel principal na evolução da organização.

As principais vantagens do uso do Agile neste caso foram a cultura inovadora, a criação de equipas independentes e multidisciplinares e os métodos e processos implementados. A principal desvantagem é que as práticas deste caso não devem ser replicadas em pequenas e médias empresas, cada empresa deve adaptar as práticas consoante as suas características.

Em todos os casos deste trabalho é possível identificar áreas de impacto da abordagem Agile. As principais áreas são a cultura, a estrutura organizacional e os processos de desenvolvimento. A implementação desta abordagem em larga escala implica mais do que simplesmente aplicar metodologias Agile nos projetos, implica alterações significativas em toda a organização, principalmente nas áreas acima referidas.

Cultura

A cultura afeta o trabalho de todos os funcionários de uma organização, este é um aspeto bem evidenciado em todos os casos, com a adoção de metodologias Agile a cultura das organizações tornou-se mais ágil e dinâmica e o ambiente de trabalho melhorou significativamente. A cultura Agile incentiva a experimentação e o erro porque só assim é possível evoluir e criar produtos e serviços inovadores, em alguns casos esses produtos poderão vir a ser ainda mais rentáveis que o produto principal, isto aconteceu no caso D com a criação da AWS que atualmente representa uma excelente fonte de lucro. Também no caso D evidencia-se a cultura de dia 1, implementada pelo seu antigo CEO, Jeff Bezos, em que o ponto principal é a obsessão em satisfazer o cliente. A cultura Agile dá mais poder aos níveis mais baixos da organização, tornando-os mais autónomos e com maior responsabilidade.

Estrutura organizacional

A estrutura organizacional é um dos aspetos mais importantes de cada empresa, a maneira como as equipas e os seus elementos são organizados e a forma como a comunicação das equipas, principalmente equipas de diferentes áreas e de diferentes países, é realizada é determinante no dia a dia da empresa. Este aspeto é abordado no caso A com a criação do modelo Spotify, em que foram criadas estruturas como *squads* e *chapters*, e no caso D com a criação do modelo de equipa *two-pizza*. A estrutura organizacional ágil e flexível destes dois casos é diferente da clássica estrutura organizacional, com departamentos e hierarquias rígidas e bem definidas, que a maioria das empresas utilizam, como os casos B e C. A estrutura organizacional Agile cria equipas mais autónomas pois contêm as competências técnicas e de

negócio necessárias para a sua atividade, onde anteriormente havia departamentos separados, muitas vezes em clima de tensão entre o cliente interno e o fornecedor interno, passou a haver funcionários que vieram desses departamentos e agora pertencem á mesma equipa.

Processos de desenvolvimento

As metodologias escolhidas pelas organizações têm um grande impacto nos seus processos. Cada metodologia Agile tem as suas práticas que são incorporadas nos processos de desenvolvimento dos produtos e serviços das organizações. Isto está demonstrado nos casos B, C e D, onde se aborda algumas das práticas Agile utilizadas, no caso B com a criação do *sprint* de design e com a adoção de técnicas como TBD, IC e automação de testes, no caso C com a adoção de três ARTs na plataforma de cobrança de subscrições e com a implementação de três *sprints* na aplicação WebEx Meetings, e no caso D com técnicas como o processo *working backwards* e o método Kaizen. Em cada caso a organização criou o seu próprio processo de desenvolvimento com a escolha da metodologia mais adequada, por vezes mais que uma no mesmo projeto, e com a personalização das práticas Agile às suas necessidades e características. As metodologias e técnicas Agile foram consequências de ter equipas mais autónomas e de se conseguir realizar alterações mais frequentemente do que se realizava no modelo tradicional, várias dezenas por dia em vez de algumas por semana ou mês.

A resposta à primeira questão apresentada na introdução deste trabalho é que os principais benefícios são a melhoria dos processos de trabalho e a criação de equipas multifuncionais e autónomas e os principais desafios são que as metodologias Agile usadas em larga escala na maioria das vezes não são recomendadas para pequenas e médias empresas e a necessidade de estruturas e ferramentas mais complexas.

A resposta à segunda questão apresentada na introdução deste trabalho é que o Agile tem um impacto muito positivo em todas as áreas, a cultura melhora tornando-se numa cultura flexível e ágil que incentiva a experimentação e que se foca no cliente, a estrutura das organizações muda ao reorganizarem as suas equipas tornando-as assim mais independentes e adaptáveis o que resultou numa melhoria na comunicação e colaboração das equipas, a adoção de metodologias Agile melhora significativamente a forma de trabalhar de toda a organização, e todas estas áreas contribuem para a obtenção de resultados excelentes sendo os principais o aumento da satisfação do cliente, entregas mais rápidas e frequentes, aumento da produtividade, aumento dos lucros, e recursos e tempo poupados.

Conclui-se que em todos os casos de estudo as práticas e técnicas Agile tiveram um grande impacto no crescimento e inovação das empresas e na criação e desenvolvimento dos seus produtos e serviços. Nos casos A, C e D as empresas têm uma metodologia Agile bem definida que é usada na maioria dos seus projetos, modelo Spotify, SAFe e Scrum, respetivamente. Enquanto no caso B a empresa usa várias metodologias, como Scrum e XP, dependendo dos projetos e por vezes usa várias metodologias no mesmo projeto. Em todos os casos houve benefícios em comum como o aumento da autonomia e agilidade das equipas e a redução dos custos e do tempo das entregas de *software*.

Capítulo 6 - Conclusão

Este trabalho teve como principal objetivo a compreensão dos benefícios da utilização do Agile em larga escala. Desta forma foram investigados casos reais da implementação da abordagem Agile, foram apresentados quatro casos de estudo de grandes empresas, e foram analisados os seus impactos em três áreas, a cultura, a estrutura organizacional e os processos de desenvolvimento. Em cada caso demonstra-se a transformação Agile sofrida pela empresa desde a sua criação até hoje, como as metodologias são utilizadas e os resultados obtidos. Com a evolução das TI as metodologias que as empresas escolhem tornaram-se de extrema importância para os seus projetos e para o seu futuro, as empresas têm de ter em consideração diversos fatores nesta escolha, como as suas características e os seus objetivos. De modo a acompanhar a evolução do mercado e da sociedade as empresas necessitam de inovar e estar atentas às oportunidades de negócio que possam surgir.

Ao investigar empresas de diferentes setores do mercado e origens foi possível observar que apesar de utilizarem diferentes metodologias todas tiveram excelentes resultados ao optar pela abordagem Agile. Algumas optaram desde o início por esta abordagem, casos A, B e D, enquanto outras começaram por utilizar a abordagem Waterfall, mas com o seu crescimento foi inevitável alterarem para uma abordagem mais ágil e adaptável, caso C.

Apesar dos diferentes caminhos que cada empresa teve pode-se afirmar que a cultura Agile está enraizada nas suas equipas. Os principais resultados obtidos foram o aumento dos lucros e a melhoria dos seus processos de trabalho. Os recursos e tempo poupados tornaram as empresas mais eficientes e produtivas nos seus processos. Um aspeto fundamental nesses processos é a interação constante com o cliente, que permitiu que houvesse um regular fluxo de feedback durante o ciclo de vida dos projetos. A utilização das metodologias Agile fornece mais flexibilidade e autonomia às equipas, o que consequentemente resulta em mais criatividade, motivação e na criação de um ambiente mais relaxado e positivo entre equipas e departamentos.

Na realização deste trabalho experienciou-se algumas limitações, tais como não ter acesso ao funcionamento diário das empresas de modo a observar e entender melhor as práticas Agile em ação, e a falta de informação clara e transparente por parte das empresas sobre os seus métodos de trabalho e resultados.

Este trabalho deixa o tema ainda em aberto para novas investigações, neste âmbito propõe-se como trabalhos futuros a realização de um caso de estudo dos benefícios da utilização do Agile em empresas de pequena e média escala, e uma comparação dos resultados obtidos em empresas de larga escala com empresas de pequena e média escala, sugere-se que isto seja realizado através de contacto constante e próximo com os funcionários das empresas e observação das suas práticas diárias no local de trabalho.

Bibliografia

Abbas, N., Gravell, A. M., & Wills, G. B. (2008). *Historical Roots of Agile Methods: Where did "Agile Thinking" Come from?*. Agile Processes in Software Engineering and Extreme Programming: 9th International Conference, Proceedings 9, pp. 94-103.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). *Agile Software Development Methods: Review and Analysis*. VTT publication 478, Espoo, Finland.

Alam, S., Bhatti, S. N., Shah, S. A. A., & Jadi, A. M. (2017). *Impact and Challenges of Requirement Engineering in Agile Methodologies: A Systematic Review*. International Journal of Advanced Computer Science and Applications, Vol. 8, No. 4.

Almeida, F., & Espinheira, E. (2021). *Large-Scale Agile Frameworks: A Comparative Review*. Journal of Applied Sciences, Management and Engineering Technology, Vol. 2, Issue 1, pp. 16-29.

Al-Saqqa, S., Sawalha, S., & AbdelNabi, H. (2020). *Agile Software Development: Methodologies and Trends*. International Journal of Interactive Mobile Technologies, Vol. 14, No. 11, pp. 246-270.

Anderson, D. J., & Carmichael, A. (2016). *Essential Kanban Condensed*. Blue Hole Press.

Anwer, F., Aftab, S., Waheed, U., & Muhammad, S. S. (2017). *Agile Software Development Models TDD, FDD, DSDM, and Crystal Methods: A Survey*. International Journal of Multidisciplinary Sciences and Engineering, Vol. 8, No. 2, pp. 1-10.

Araújo, C. M., Santos, I., Canedo, E., & Araújo, A. P. (2019). *Design Thinking versus Design Sprint: A Comparative Study*. Design, User Experience, and Usability, Design Philosophy and Theory: 8th International Conference, Proceedings, Part I 21, pp. 291-306.

Atlas, A. (2009). *Accidental Adoption: The Story of Scrum at Amazon.com*. 2009 Agile Conference, pp. 135-140.

Ayewah, N., & Pugh, W. (2010). *The Google FindBugs Fixit*. Proceedings of the 19th International Symposium on Software Testing and Analysis, pp. 241-252.

Bäcklander, G. (2019). *Doing complexity leadership theory: How agile coaches at Spotify practise enabling leadership*. Creativity and Innovation Management, 28(1), pp. 42-60.

Banijamali, A., Dawadi, R., Ahmad, M. O., Similä, J., Oivo, M., & Liukkunen, K. (2017). *Empirical Investigation of Scrumban in Global Software Development*. Model-Driven Engineering and Software Development: 4th International Conference, Revised Selected Papers 4, pp. 229-248.

Beck, K. (1999). *Embracing Change With Extreme Programming*. Computer, 32(10), pp. 70-77.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Manifesto for Agile Software Development*. Acedido em 20 de junho de 2023 em https://agilemanifesto.org.

Bhalerao, S., Puntambekar, D., & Ingle, M. (2009). *Generalizing Agile Software Development Life Cycle*. International Journal on Computer Science and Engineering, Vol. 1(3), pp. 222-226.

Casteren, W. (2017). *The Waterfall Model and the Agile Methodologies: A comparison by project characteristics*. Research Gate, 2, pp. 1-6.

Cawley, O., Wang, X., & Richardson, I. (2010). *Lean/Agile Software Development Methodologies in Regulated Environments – State of the art*. Lean Enterprise Software and Systems: First International Conference, pp. 31-36.

Chen, R. R., Ravichandar, R., & Proctor, D. (2016). *Managing the transition to the new agile business and product development model: Lessons from Cisco Systems*. Business Horizons, 59(6), pp. 635-644.

Cisco. (2011). *Agile Product Development at Cisco: Collaborative, Customer-Centered Software Development*. White Paper.

Cisco. (2017). *How Cisco IT Uses Agile Development with Distributed Teams and Complex Projects*. Acedido em 4 de março de 2024 em https://www.cisco.com/c/en/us/solutions/collateral/enterprise/cisco-on-cisco/cs-boit-06172016-agile-development.html.

Cockburn, A. (2000). *Writing Effective Use Cases*. Boston: Addison-Wesley.

Cockburn, A. (2001). *Agile Software Development*. Boston: Addison-Wesley.

Copeland, P. (2010). *Google's Innovation Factory: Testing, Culture, And Infrastructure*. Third International Conference on Software Testing, Verification and Validation, pp. 11-14.

Courtney, J. (2024). *LEGO Design Sprints at Scale*. Acedido em 28 de abril de 2024 em https://www.thesprintbook.com/stories/lego-design-sprints-at-scale.

Denning, S. (2018). *The role of the C-suite in agile transformation: The case of Amazon*. Strategy & Leadership, Vol. 46, No. 6, pp. 14-21.

Denning, S. (2019a). *How Amazon practices the three laws of Agile management*. Strategy & Leadership, Vol. 47, No. 5, pp. 36-41.

Denning, S. (2019b). *How Amazon Became Agile*. Acedido em 18 de fevereiro de 2024 em https://www.forbes.com/sites/stevedenning/2019/06/02/how-amazon-became-agile.

Dikert, K., Paasivaaraa, M., & Lassenius, C. (2016). *Challenges and success factors for large-scale agile transformations: A systematic literature review*. Journal of Systems and Software, Vol. 119, pp. 87-108.

Edison, H., Wang, X., & Conboy, K. (2022). *Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review*. IEEE Transactions on Software Engineering, Vol. 48, No. 8, pp. 2709-2731.

Edmonds, M. (2018). *Making City Government Easier Than a Starbucks*. Acedido em 28 de abril de 2024 em https://sprintstories.com/making-city-government-easier-than-a-starbucks-67d2fec939b.

Ferreira, V. G. & Canedo, E. D. (2018). *Google Design Sprint como um Recurso Educacional: uma Pesquisa Exploratória*. Nuevas Ideas en Informática Educativa, Vol. 14, pp. 49-59.

Flynn, J. (2022). *16 Amazing Agile Statistics [2023]: What Companies Use Agile Methdology*. Acedido em 9 de dezembro de 2023 em https://www.zippia.com/advice/agile-statistics.

Galetti, B., Golden III, J., & Brozovich, S. (2019). *Inside Day 1: How Amazon Uses Agile Team Structures and Adaptive Practices to Innovate on Behalf of Customers*. People & Strategy, 42(2), pp. 36-41.

Gheorghe, A. M., Gheorghe, I. D., & Iatan, I. L. (2020). *Agile Software Development*. Informatica Economică, Vol. 24, No. 2, pp. 90-100.

Harracá, M. (2017). *Business models and organizational forms: searching the edge of innovation in Google and Amazon*. Tese apresentada ao Université Paris 13 para obtenção do grau de mestre, orientada por Benjamin Coriat, Paris.

Hass, K. B. (2007). *The Blending of Traditional and Agile Project Management*. PM World Today, Vol. 9, No. 5, pp. 1-8.

Henderson, F. (2017). *Software Engineering at Google*.

Hoda, R., Salleh, N., & Grundy, J. (2018). *The Rise and Evolution of Agile Software Development*. IEEE Software, 35(5), pp. 58-63.

International Agile Federation. (2024). *Chrome: A Case Study in Agile Triumph*. Acedido em 2 de maio de 2024 em https://medium.com/@internationalagilefederation/chrome-a-case-study-in-agile-triumph-b4863c02261f.

Jaspan, C., Jorde, M., Knight, A., Sadowski, C., Smith, E. K., Winter, C., & Murphy-Hill, E. (2018). *Advantages and Disadvantages of a Monolithic Repository: a case study at Google*. Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, pp. 225-234.

Kendis. (2018). *Exploring Key Elements of Spotify's Agile Scaling Model*. Acedido em 29 de maio de 2024 em https://medium.com/scaled-agile-framework/exploring-key-elements-of-spotifys-agile-scaling-model-471d2a23d7ea.

Kim, E. (2018). *Amazon employees start their day by answering a simple question about work*. Acedido em 2 de junho de 2024 em https://www.cnbc.com/2018/03/30/amazon-employee-reaction-to-hr-programs-connections-forte.html.

Kirovska, N., & Koceski, S. (2015). *Usage of Kanban methodology at software development teams*. Journal of Applied Economics and Business, Vol. 3, Issue 3, pp. 25-34.

Knapp, J., Zeratsky, J., & Kowitz, B. (2016). *Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days*. New York: Simon & Schuster.

Knaster, R. (2021). *15th State of Agile Report: Agile leads the way through the pandemic and digital transformation*. Acedido em 16 de março de 2024 em https://digital.ai/catalyst-blog/15th-state-of-agile-report-agile-leads-the-way-through-the-pandemic-and-digital.

Kniberg, H., & Ivarsson, A. (2012). *Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds*.

Kristinsdottir, S., Larusdottir, M., & Cajander, Å. (2016). *Responsibilities and Challenges of Product Owners at Spotify - An Exploratory Case Study*. Human-Centered and Error-Resilient Systems Development: Joint Working Conference, 6th International Conference on Human-Centered Software Engineering, and 8th International Conference on Human Error, Safety, and System Development, Proceedings 8, pp. 3-16.

Larusdottir, M., Roto, V., Stage, J., Lucero, A., & Šmorgun, I. (2019). *Balance Talking and Doing! Using Google Design Sprint to Enhance an Intensive UCD Course*. 17th IFIP Conference on Human-Computer Interaction, pp. 95-113.

Leau, Y. B., Loo, W. K., Tham, W. Y., & Tan, S. F. (2012). *Software Development Life Cycle AGILE vs Traditional Approaches*. International Conference on Information and Network Technology, Vol. 37, No. 1, pp. 162-167.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., et al. (2004). *Agile Software Development in Large Organizations*. Computer, 37(12), pp. 26-34.

Manninen, V. (2018). *The Agile Transformation*. Tese apresentada ao Business Information Technology da Oulu University of Applied Sciences para obtenção do grau de bacharelado, orientada por Ilkka Mikkonen, Oulu.

Marinho, M., Camara, R., & Sampaio, S. (2021). *Toward Unveiling How SAFe Framework Supports Agile in Global Software Development*. IEEE Acess, Vol. 9, pp. 109671-109692.

Masood, Z., & Farooq, S. (2017). *The Benefits and Key Challenges of Agile Project Management under Recent Research Opportunities*. International Research Journal of Management Sciences, Vol. 5(1), pp. 20-28.

Memon, A., Gao, Z., Nguyen, B., Dhanda, S., Nickell, E., Siemborski, R., et al. (2017). *Taming Google-Scale Continuous Testing*. IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, pp. 233-242.

Micco, J. (2016). *Continuous Integration at Google Scale*.

Mitchell, S. M., & Seaman, C. B. (2009). *A Comparison of Software Cost, Duration, and Quality for Waterfall vs. Iterative and Incremental Development: A Systematic Review*. Third International Symposiumm on Empirical Software Engineering and Measurement, pp. 511-515.

Nandhakumar, J., & Avison, D. E. (1999). *The fiction of methodological development: a field study of information systems development*. Information Technology & People, Vol. 12, No. 2, pp. 176-191.

Onetto, M. (2014). *When Toyota met e-commerce: Lean at Amazon*. McKinsey Quarterly, 41(1), pp. 1-7.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Boston: Addison-Wesley.

Potvin, R., & Levenberg, J. (2016). *Why Google Stores Billions of Lines of Code in a Single Repository*. Communications of the ACM, Vol. 59, No. 7, pp. 78-87.

Power, K. (2016). *Sensemaking and Complexity in Large-Scale Lean-Agile Transformation: A Case Study from Cisco*. 49th Hawaii International Conference on System Sciences, pp. 5417-5426.

Pradhan, S., & Nanniyur, V. (2021). *Large scale quality transformation in hybrid development organizations – A case study*. Journal of Systems and Software, Vol. 171, 110836.

Quast, J. (2024). *If You Think That Amazon Makes Its Money By Selling Products Online, Then You'll Be Shocked By What Its Real Moneymaker Is*. Acedido em 30 maio de 2024 em https://finance.yahoo.com/news/think-amazon-makes-money-selling-100800785.html?guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sg=AQAAAKGQ1Q-8vUrWKrffbZhx6clt3X_aZoMzs7uVV3YnwnQaXnR9S4ronroBTMmr8x-HgpVA0MEIZ91VcR_EmdJr2mSX_4xLbb&guccounter=2.

Rafi, D. M., Moses, K. R. K., Petersen, K., & Mäntylä, M. V. (2012). *Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey*. 7th International Workshop on Automation of Software Test, pp. 36-42.

Salameh, A., & Bass, J. (2018). *Influential Factors of Aligning Spotify Squads in Mission-Critical and Offshore Projects – A longitudinal embedded case study*. Product-Focused Software Process Improvement: 19th International Conference, Proceedings 19, pp. 199-215.

Salameh, A., & Bass, J. M. (2019). *Spotify Tailoring for Promoting Effectiveness in Cross-Functional Autonomous Squads*. Agile Processes in Software Engineering and Extreme Programming – Workshops, pp. 20-28.

Salameh, A., & Bass, J. M. (2020). *Heterogeneous Tailoring Approach Using the Spotify Model*. Proceedings of the 24th international conference on evaluation and assessment in software engineering, pp. 293-298.

Salameh, A., & Bass, J. M. (2022). *An architecture governance approach for Agile development by tailoring the Spotify model*. AI & Society, Vol. 37, pp. 761-780.

Slater, D. (2023). *Elementos da cultura de Dia 1 da Amazon*. Acedido em 21 de fevereiro de 2024 em https://aws.amazon.com/pt/executive-insights/content/how-amazon-defines-and-operationalizes-a-day-1-culture.

Šmite, D., Moe, N. B., Levinta, G., & Floryan, M. (2018). *Spotify Guilds – Cultivating Knowledge Sharing in Large-scale Agile Organizations*.

Sommerville, I. (2011). *Software Engineering* (9ª ed.). London: Pearson.

Stoica, M., Ghilic-Micu, B., Mircea, M., & Uscatu, C. (2016). *Analyzing Agile Development - from Waterfall Style to Scrumban*. Informatica Economică, Vol. 20, No. 4, pp. 5-14.

Tajima, S. (2019). *Google Ventures Design Sprint Case Study*. Acedido em 15 de fevereiro de 2024 em https://medium.com/@seratajima/google-ventures-design-sprint-de9f5fa8e79f.

Taylor, H. (2023). *20+ Agile Statistics: All About Agile Adoption*. Acedido em 9 de dezembro de 2023 em https://www.runn.io/blog/agile-statistics.

Verwijs, C., & Russo, D. (2023). *Do Agile Scaling Approaches Make A Difference? An Empirical Comparison of Team Effectiveness Across Popular Scaling Approaches*.

Wakode, R. B., Raut, L. P., & Talmale, P. (2015). *Overview on Kanban Methodology and its Implementation*. International Journal for Scientific Research & Development, Vol. 3, Issue 2, pp. 2321-0613.

Wangsa, K., Chugh, R., Karim, S., & Sandu, R. (2022). *A comparative study between design thinking, agile, and design sprint methodologies*. International Journal of Agile Systems and Management. Vol. 15, Issue 2, pp. 225-242.

Whittaker, J. A., Arbon, J., & Carollo, J. (2012). *How Google Tests Software*. Boston: Addison-Wesley.

Ziftci, C., & Reardon, J. (2017). *Who Broke the Build? Automatically Identifying Changes That Induce Test Failures In Continuous Integration at Google Scale*. IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, pp. 113-122.

Glossário

Bazel - Ferramenta do Google de automatização de *builds* e testes de *software*

Branch - Duplicação de um objeto gerido num sistema de controlo de versão

Commit - Operação de submeter alterações, maioritariamente de código, para um repositório

Debug - Processo de encontrar e solucionar *bugs* no código

Feature Branching - Modelo de *branching* em que o programador cria um *branch* para desenvolver uma nova funcionalidade

Gitflow - Modelo de *branching* do Git

Green Build - Todos os testes executados com sucesso

Trunk - *Branch* principal que contém o código executável do projeto