Universidade do Minho
Escola de Engenharia

Sandro Ricardo da Costa Ferraz

Recomendações para a adoção de práticas ágeis no desenvolvimento de software: estudo de casos

Dissertação do

Mestrado em Engenharia Informática (MEI)

Trabalho realizado sob a orientação de

Professor Doutor João M. Fernandes

Professor Doutor Ricardo J. Machado

Julho de 2016

Agradecimentos

Antes de mais gostaria de agradecer aos meus orientadores, Professor Doutor João M. Fernandes e Professor Doutor Ricardo J. Machado, pela colaboração e disponibilidade, que permitiram a realização desta dissertação.

Agradeço ao Centro de Computação Gráfica e, em especial, à minha coordenadora, Mestre Ana Lima, pela flexibilidade e condições de trabalho que proporcionaram a realização deste trabalho. Quero agradecer também ao grupo EPMQ pela partilha de conhecimento e ideias, em particular à Doutora Paula Monteiro, uma vez que teve uma influência relevante através das suas críticas e sugestões ao longo deste trabalho.

Agradeço à minha namorada Cristiana Faria, por toda a paciência e compreensão que demonstrou estes meses.

Agradeço à minha família, amigos e colegas que sempre me ajudaram e motivaram nos momentos mais difíceis.

A todos estes e aqueles que direta ou indiretamente permitiram a conclusão desta dissertação, o meu sincero obrigado.

Resumo

Várias evidências na literatura demonstram que a adoção de práticas ágeis têm uma influência positiva no desenvolvimento de software, no entanto a sua adoção em organizações está condicionada por vários desafios.

Este trabalho apresenta os conceitos dos métodos ágeis, através da descrição das suas práticas e princípios. Também são apresentados os maiores desafios que as organizações enfrentam quando pretendem adotar práticas ágeis, assim como os impactos que a adoção destas práticas proporcionam para as organizações.

Neste trabalho realizou-se uma investigação qualitativa através do método de estudo de caso com a finalidade de encontrar evidências que suportam as recomendações para a adoção das práticas ágeis no desenvolvimento de software. Para esta investigação foram utilizados, como estudo de casos, dois projetos de desenvolvimento de software onde foram adotadas práticas ágeis.

Palavras-Chave: desenvolvimento de software, práticas ágeis, métodos ágeis, SCRUM, XP.

Abstract

Several evidences in literature demonstrates that the adoption of agile methods and practices have a positive influence on the development of software, but its implementation in organizations is constrained by several challenges.

This work presents the concept of agile methods through its practices and principles. The main challenges and impacts on organizations when seeking to adopt approaches and agile practices are also presented.

It is therefore proposed, conduct a qualitative research method through the case study method in order to find evidence that helped characterize recommendations for the adotaption of agile practices in software development. For this research was used as a case studies, two software development projects where agile practices were adopted.

Keywords: software development, agile practices, agile methods, SCRUM, XP.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Acrónimos

CCG Centro de Computação Gráfica

DTDA Desenvolvimento de *Timeline* e do *Dashboard* Analítico

HMIExcel *Human-Machine Interface Excellence*

I&DT Investigação e Desenvolvimento Tecnológico

ROI *Return of Investment* - Retorno do Investimento

SAMI *Sidky Agile Measurement Index*

TDD *Test-Driven Development*

TFS *Team Foundation Server*

TI Tecnologias da Informação

UML *Unified Modeling Language*

XP *Extreme Programming*

Capítulo 1

Introdução

Este capítulo pretende apresentar a descrição do contexto e o que motiva a realização deste trabalho, seguindo-se os objetivos e a abordagem metodológica de investigação utilizada. No final é apresentada a respetiva estrutura do documento.

1.1 Contextualização

Os métodos de desenvolvimento de software estão em constante evolução, devido à evolução das tecnologias e às novas necessidades dos utilizadores. Atualmente, o ambiente de negócios é dinâmico e deu origem a organizações em crescimento, a precisarem de adaptar continuamente as suas estruturas, estratégias e políticas para se adequarem ao novo ambiente. Essas organizações precisam de sistemas de informação que evoluem constantemente para atender às suas necessidades de mudança. Mas os métodos tradicionais para desenvolvimento de software não têm a flexibilidade necessária para ajustar dinamicamente o processo de desenvolvimento [5].

Os métodos tradicionais podem ser aplicados quando os requisitos do sistema são razoavelmente estáveis e o projeto deve ser realizado de forma linear [6]. Desta forma, é possível a criação de um cronograma com os prazos de cada fase de desenvolvimento do projeto [7]. Os sistemas de software são abstratos e intangíveis, não estando restringidos pelas propriedades dos materiais, nem governados pelas leis da física ou pelos processos de manufatura. Isso com-

plica a engenharia de software, porque não há limites naturais para o potencial do software. Por este motivo, o software pode tornar-se extremamente difícil de interpretar e demasiado caro para se modificar [8], sendo este um fator muito importante da escolha do método a utilizar no seu desenvolvimento.

Em projetos onde a mudança de requisitos é inevitável e as datas de entrega de software são curtas, é conveniente aplicar métodos ágeis [9]. No entanto, em organizações que estão habituadas aos métodos tradicionais, a adoção de métodos ágeis irá representar provavelmente vários desafios, uma vez que os dois tipos de métodos de desenvolvimento de software são baseados em conceitos muito distintos [5].

A motivação deste trabalho engloba-se sobretudo na adoção das práticas ágeis para desenvolvimento de software. Para esse efeito, foi realizada uma descrição pormenorizada sobre estas práticas (e métodos ágeis, pois incluem um conjunto de práticas pré-definidas) e uma análise no contexto de dois projetos de desenvolvimento de software onde foram adotadas práticas ágeis. Estes projetos foram desenvolvidos em diferentes organizações: um na WinTrust, com sede em Lisboa e um escritório em Braga, e o outro na Bosch Car Multimédia Portugal, com sede em Braga. Este trabalho pretende produzir um conjunto de recomendações que possam ser implementadas por aqueles que têm que introduzir práticas ágeis numa organização de desenvolvimento de software.

1.2 Objetivos

O objetivo geral desta dissertação é contribuir com recomendações para facilitar a adoção de práticas ágeis no âmbito de organizações que desenvolvem software. Assume-se que a organização pretende mudar a sua forma de desenvolver software (seja para todos os projetos futuros, seja apenas no contexto dum projeto específico) e que identificou vantagens na adoção de práticas ágeis (no limite, a adoção dum determinado método).

Os objetivos específicos deste trabalho são:

- Analisar casos onde as práticas ágeis foram adotadas e perceber como as práticas foram

introduzidas nas equipas.

- Caracterizar recomendações com base nas análises realizadas;
- Validar as recomendações caraterizadas através de um projeto real.

Com o trabalho desenvolvido foram encontradas evidências que reforçam a pertinência da adoção de práticas ágeis para desenvolver software.

Para atingir os objetivos foi realizada uma investigação qualitativa e aplicado o método de estudo de caso. A recolha de dados foi realizada através de entrevistas, análise de documentos e análise de artigos.

1.3 Abordagem Metodológica

Tendo em conta o objetivo deste trabalho, a abordagem de investigação foi qualitativa, uma vez que este estudo enquadra-se nas condições deste tipo de investigação, tais como [10]:

- Contacto intenso e prolongado com uma "situação real";
- Ter como objetivo uma visão holística do contexto em estudo;
- A recolha de dados a partir do conhecimento dos próprios atores;
- A análise dos dados não altera a originalidade dos mesmos;
- A recolha dos dados é efetuada com instrumentos não padronizados;
- Os dados são quase sempre em forma de "palavras" obtidas através de observações diretas ou indiretas, entrevistas e documentos;
- Os dados são analisados para comparar, contrastar e descobrir padrões.

Existem alguns métodos usados na investigação qualitativa, tais como, teoria fundamentada, etnografia, estudo de caso, entre outros [11]. Para este estudo, o método escolhido foi o de estudo de caso do tipo múltiplo.

Para Yin [12], um "caso" pode ser algo bem definido ou concreto (como um indivíduo, um grupo ou uma organização), mas também pode ser algo menos definido ou definido num plano mais abstrato (como decisões, programas, processos de implementação ou mudanças organizacionais).

Segundo Zelkowitz e Wallace [13], num estudo de caso os investigadores observam e monitorizam um projeto e recolhem dados ao longo do tempo. Segundo outro autor, Yin [12], *"um estudo de caso é uma estratégia de investigação mais adequada quando queremos saber o "como" e o "porquê" de acontecimentos atuais sobre os quais o investigador tem pouco ou nenhum controlo."*

Segundo Guba e Lincoln [14], o objetivo de um estudo de caso é relatar os factos como sucederam, descrever situações ou factos, proporcionar conhecimento acerca do fenómeno estudado e comprovar ou contrastar efeitos e relações presentes no caso e segundo Yin [12], o objetivo é explorar, descrever ou explicar.

Num estudo de caso é possível definir quatro fases sequenciais de estruturação [15]:

Definição da Unidade de Análise: esta fase consiste na escolha da unidade que constitui o caso, essa escolha é feita pelo próprio investigador, o mesmo tem de ter a habilidade de perceber que dados são suficientes para a compreensão do objeto como um todo;

Recolha de Dados: esta fase é realizada geralmente através do uso de múltiplos métodos de recolha de dados, tais como, entrevistas, observação, análise de documentos, questionários, entre outros. A recolha deve ser feita também através de múltiplas fontes, de forma a possibilitar diferentes perspetivas e permitir a triangulação dos dados;

Seleção, Análise e Interpretação dos Dados: esta fase consiste na avaliação e seleção dos dados que serão essenciais e tem ligação com os objetivos da investigação. Só os dados selecionados serão analisados. O investigador deve definir antecipadamente o plano de análise e considerar as limitações dos dados obtidos, caso contrário deve apresentar os resultados em termos de probabilidade;

Elaboração do Relatório: esta fase consiste na elaboração do relatório final.

1.4 Estrutura do Documento

O presente documento tem uma estrutura sob a forma de capítulos.

No capítulo 1 é apresentada uma breve contextualização sobre a motivação da realização deste estudo, identificação dos objetivos que se pretendem alcançar e a abordagem metodológica adotada como orientação desta investigação.

No capítulo 2 é apresentado o estado de arte no que se refere aos conceitos que estão relacionados com o objetivo principal deste estudo de caso, onde são expostos conceitos relacionados com práticas ágeis.

No capítulo 3 é apresentado o estudo de casos utilizados neste trabalho, enquadrados com a abordagem metodológica de investigação adotada, assim como a caracterização de recomendações para a adoção de práticas ágeis no desenvolvimento de software.

No capítulo 4 é apresentada a validação parcial do trabalho realizado, onde foram implementadas algumas das recomendações caraterizadas num projeto real.

No capítulo 5 e último é apresentada a conclusão deste trabalho e perspetivas de trabalho futuro.

Capítulo 2

Estado da Arte

Este capítulo pretende apresentar o estado de arte associado às temáticas abordadas no presente trabalho. Começa por apresentar um método tradicional, designado por modelo em cascata, seguindo-se com a apresentação dos métodos ágeis e uma apresentação mais aprofundada de dois desses métodos, o Scrum e o XP (*Extreme Programming*). Este capítulo termina com a apresentação dos maiores desafios que as organizações enfrentam quando pretendem adotar práticas ágeis, assim como os impactos que a adoção dessas práticas proporcionam para as organizações que desenvolvem software.

2.1 Modelo em Cascata

O modelo em cascata é um método tradicional, este modelo já é antigo, foi iniciado por Royce em 1970 e está dividido em várias fases sequenciais no ciclo de vida no desenvolvimento de software, ou seja, assim que uma fase estiver concluída, dará início à fase seguinte [16].

As fases estão definidas sequencialmente como é mostrado na Figura 2.1.

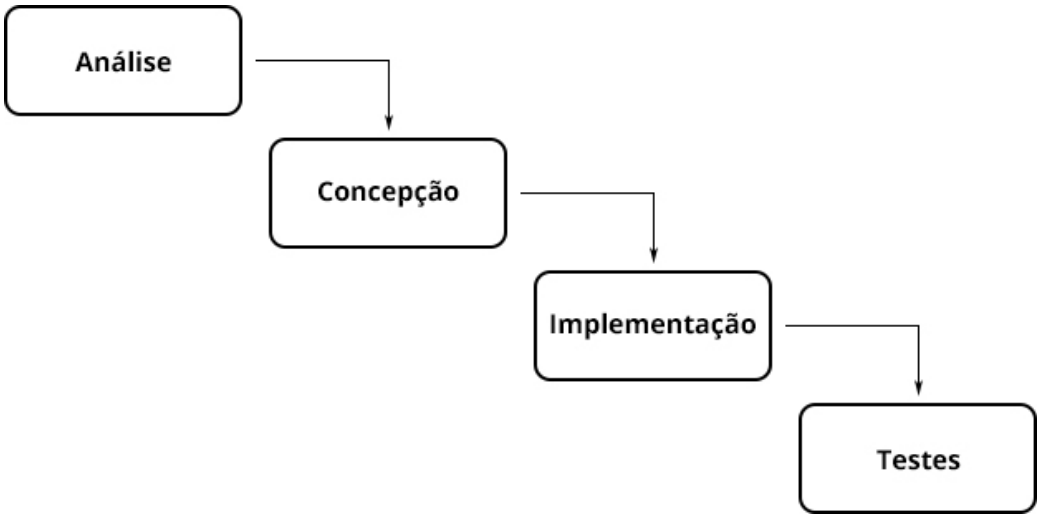

Figura 2.1: Fases do desenvolvimento de software baseado no modelo em cascata.

Cada fase tem um processo bem definido a ser seguido, e no final de cada uma delas devemos ter acesso à documentação respetiva. Segundo Sommerville as fases são definidas da seguinte forma [8]:

Análise. Nesta fase o objetivo é contactar as partes interessadas no sistema com a finalidade de levantar todos os requisitos ao pormenor desse mesmo sistema e preparar um documento com todos os requisitos especificados.

Conceção. Depois de concluir a fase de levantamento de requisitos, através do documento que foi preparado, é criada a arquitetura adequada para o sistema, envolvendo a identificação e descrição das principais abstrações do sistema e seus relacionamentos.

Implementação. É a fase onde é efetuada a construção do software e onde são efetuados os testes unitários.

Testes. Aqui é efetuada a integração do software de forma a testar se o sistema cumpre todos os requisitos que foram definidos, posteriormente, caso os requisitos sejam todos cumpridos o sistema é entregue ao cliente.

Normalmente só se deve avançar para a fase seguinte assim que a anterior estiver concluída. No entanto, existe ligação de uma fase para a outra e por norma só na fase seguinte se consegue detetar a necessidade de efetuar alterações na fase anterior, isto implica retrabalho na documentação de forma a refletir essas alterações. Normalmente esse retrabalho tem um custo elevado, levando mesmo a que após pequenas iterações deste procedimento, as mesmas sejam deixadas para trás. Desta forma o sistema pode acabar por não fazer o que o cliente pretendia. Este é um problema do uso do modelo em cascata, pois o cliente não tem muitas oportunidades de dar feedback sobre o sistema [17].

O modelo em cascata tem o foco sobre um plano, o problema é que desta forma o processo de desenvolvimento é realizado linearmente (Figura 2.1), onde não pode haver muitas mudanças. Mas as mudanças ocorrem muito frequentemente em quase todos os projetos de software. E logo na fase inicial, no levantamento de requisitos, é onde surgem as questões, que por norma quase nunca são suficientes, de seguida esses problemas são levados para a fase de conceção que acabam por resultar em conflitos e defeitos na fase de implementação [18].

O modelo em cascata só deve ser usado quando se sabe realmente o que é pretendido, e não deve sofrer mudanças radicais no desenvolvimento do sistema [8].

2.2 Métodos Ágeis

Em grande parte dos projetos que utilizam as abordagens tradicionais para o desenvolvimento de software, como o modelo em cascata, muito tempo é dispensado em reuniões para saber o que está a acontecer e quais são os problemas de forma a progredir no desenvolvimento do software. No entanto, quando surgem novos requisitos o impacto é enorme no desenvolvimento, quanto maior o progresso, maior será o trabalho para integrar os novos requisitos no software. Nas abordagens tradicionais é criada documentação em todas as fases do processo, alguma desta documentação por vezes é desnecessária [19].

Segundo Dennis Brandl [19], os gestores de projetos acreditam que se os clientes soubessem melhor os requisitos que pretendem, então os projetos não acabariam tarde e acima do orçamento previsto.

É importante referir também as razões para o insucesso da abordagem do modelo em cascata para o desenvolvimento de software segundo Peter DeGrace e Leslie Hulet [20]:

- Os requisitos não são totalmente compreendidos no início do projeto;
- Os utilizadores só sabem verdadeiramente o que querem depois de verem uma versão inicial do software;
- Os requisitos alteram muitas vezes durante o processo de desenvolvimento do software;
- Novas ferramentas e tecnologias fazem com que as estratégias de implementação sejam imprevisíveis.

Os métodos ágeis, como o Scrum por exemplo, podem ajudar a resolver o problema do insucesso da abordagem do modelo em cascata, uma vez que estes métodos baseiam-se no desenvolvimento iterativo. Os métodos ágeis têm como caraterísticas o desenvolvimento rápido e as mudanças contínuas, ou seja, tem uma caraterística adaptativa ao contrário dos métodos tradicionais que são preditivos. Estes métodos consistem em ciclos iterativos curtos, onde principalmente os desenvolvedores e cliente têm uma envolvência ativa nestes ciclos até ao fim do projeto [19].

Estes ciclos permitem realizar mudanças aos requisitos, através da inserção, eliminação ou alteração de requisitos previamente identificados [21]. Além disso a prioridade de cada requisito pode ser alterada. Desta forma, não há necessidade da criação de documentação exagerada como nas abordagens tradicionais, apenas é criada a necessária [19]. O Manifesto Ágil foi criado em Fevereiro de 2001, onde dezassete especialistas em desenvolvimento de software tornaram o termo "Métodos Ágeis" mais popular. Devido às diferentes experiências no desenvolvimento de software, estes especialistas uniram-se com o objetivo de descobrir as melhores formas de desenvolvimento, partilhando os princípios mais comuns. Através dessa partilha de conhecimento chegaram a um consenso em torno de quatro valores principais [22]:

- ***Indivíduos e interações*** *mais que processos e ferramentas;*
- ***Software em funcionamento*** *mais que documentação abrangente;*

- ***Colaboração com o cliente*** *mais que negociação de contratos;*
- ***Responder a mudanças*** *mais que seguir um plano.*

Além dos quatros principais valores, o Manifesto Ágil inclui doze princípios fundamentais [22], que torna na maioria dos casos os métodos de desenvolvimento ágil mais apropriados do que os métodos tradicionais para o desenvolvimento de software:

1. *A nossa maior prioridade é, desde as primeiras etapas do projeto, satisfazer o cliente através da entrega rápida e contínua de software com valor;*
2. *Aceitar alterações de requisitos, mesmo numa fase tardia do ciclo de desenvolvimento. Os processos ágeis potenciam a mudança em benefício da vantagem competitiva do cliente;*
3. *Fornecer frequentemente software funcional. Os períodos de entrega devem ser de poucas semanas a poucos meses, dando preferência a períodos mais curtos;*
4. *O cliente e a equipa de desenvolvimento devem trabalhar juntos, diariamente, durante o decorrer do projeto;*
5. *Desenvolver projetos com base em indivíduos motivados, dando-lhes o ambiente e o apoio de que necessitam, confiando que irão cumprir os objetivos;*
6. *O método mais eficiente e eficaz de passar informação para dentro de uma equipa de desenvolvimento é através de conversa pessoal e direta;*
7. *A principal medida de progresso é a entrega de software funcional;*
8. *Os processos ágeis promovem o desenvolvimento sustentável. Os promotores, a equipa e os utilizadores deverão ser capazes de manter, indefinidamente, um ritmo constante;*
9. *A atenção contínua à excelência técnica e ao bom design aumenta a agilidade;*
10. *Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial;*
11. *As melhores arquiteturas, requisitos e desenhos surgem de equipas auto-organizadas;*

12. *A equipa reflete regularmente sobre o modo de se tornar mais eficaz, fazendo os ajustes e adaptações necessárias.*

No final dos anos 90, surgiram vários métodos ágeis [23, 22], tais como: o Scrum, que já foi referido anteriormente, o XP, o *Lean Development Software* e o *Kanban*. Todos estes métodos ágeis recorrem aos valores fundamentais do Manifesto Ágil [24]:

1. Ênfase no desenvolvimento colaborativo onde as pessoas receberam privilégios sobre os processos que restringiam o seu trabalho;

2. Eliminação do trabalho desnecessário, especialmente no que diz respeito a criação de documentos desnecessários, restringindo-se a documentar o que é extremamente necessário e nada mais. Embora tenha sido mal interpretado por muitos como "sem documentação";

3. Os clientes e os *stakeholders* deixaram de ser apenas um elemento isolado às margens do desenvolvimento de software, mas passam a ter um papel fundamental no acompanhamento da evolução do produto a ser desenvolvido;

4. Aceitação do facto da incerteza fazer parte do processo de desenvolvimento de software e da necessidade de controlar essas tendências de mudanças ao longo do projeto.

2.3 Scrum

A ideia do Scrum vem do rugby em que as equipas andam com a bola para a frente em pequenos passos para atingir o objetivo, ganhar o jogo. No Scrum a ideia baseia-se na mesma lógica, em que a equipa de desenvolvimento em colaboração com os clientes e utilizadores vão evoluindo o software em pequenos passos, designados também de *sprints* no contexto do método Scrum. Cada *sprint* pode durar entre 1 a 4 semanas, fornecendo rapidamente valor de negócio para o cliente [19].

Mesmo que os requisitos alterem radicalmente, ou porque foram mal compreendidos, ou porque o cliente não sabe o que realmente quer, já não será um grande problema, pois este método foi pensado para responder a essas situações [19].

Esta abordagem transmite flexibilidade, adaptabilidade e produtividade, ajudando a melhorar as práticas de engenharia existentes numa organização, envolvendo atividades frequentes com o objetivo de identificar de forma consistente as deficiências ou impedimentos no desenvolvimento do software [19].

2.3.1 Papéis

O Scrum tem três papéis principais bem definidos [25]:

Scrum Master. Não é considerado um líder, mas sim o responsável por garantir que o projeto é realizado de acordo com as práticas, valores e regras do Scrum e que o projeto evolui como planeado.

Product Owner. Representa as partes interessadas (*stakeholders*) e o negócio. É responsável pela gestão e controlo do projeto, encarregando-se também de criar o *Product Backlog* e da tomada de decisão sobre o mesmo.

Equipa de Desenvolvimento. Tipicamente é composta por cinco a nove elementos [19]. A equipa é responsável pelo desenvolvimento e entrega do software. Deve ser auto-organizada com a finalidade de alcançar os objetivos de cada *sprint*.

Para entender melhor os papéis na gestão de projetos aplicando o método Scrum, muitas vezes é utilizada a fábula "A galinha e o porco". Nesta fábula a galinha e o porco cruzam-se na rua e a galinha sugere ao porco em abrir um restaurante. O porco aceita a proposta da galinha e pergunta-lhe qual é o nome que ela quer dar ao restaurante, onde a resposta dela é "presunto e ovos". Mas o porco diz à galinha que dessa forma ele está comprometido (necessidade de abater o porco para obter o presunto) enquanto a galinha está apenas envolvida (põe o ovo e vai-se embora).

Podemos assim afirmar que o *Srum Master*, o *Product Owner* e a equipa de desenvolvimento representam os porcos e os gestores e *stakeholders* do projeto representam as galinhas, na medida em que desempenham papéis auxiliares [19].

2.3.2 Processo

O Scrum inclui três fases no seu processo (ver Figura 2.2) [25] :

Pré-Jogo. Esta fase inclui duas subfases. A primeira é o planeamento, onde é criado o *Product Backlog* que contém todos os requisitos que são conhecidos até ao momento. Os mesmos são priorizados e é estimado o esforço necessário para o seu desenvolvimento. É também definida a equipa do projeto, as ferramentas a utilizar e é realizada a avaliação dos riscos e formação, caso seja necessário. Em cada iteração o *Product Backlog* é atualizado de forma a preparar a próxima iteração. A segunda subfase consiste na conceção da arquitetura de alto nível dos requisitos do *Product Backlog*.

Fase de Desenvolvimento ou Fase do Jogo. É considerada a parte ágil, onde o imprevisível é esperado. As diferentes variáveis ambientais e técnicas podem mudar durante o processo, e através dos *sprints* da fase de desenvolvimento é possível adaptar com flexibilidade a essas mudanças. Cada *sprint* inclui as fases tradicionais do desenvolvimento de software, como o modelo em cascata por exemplo, denotando-se a evolução do software a cada *sprint*.

Pós-Jogo. Nesta fase encontra-se o encerramento e lançamento do software, em que todos os requisitos estão implementados e não podem ser acrescentados novos requisitos.

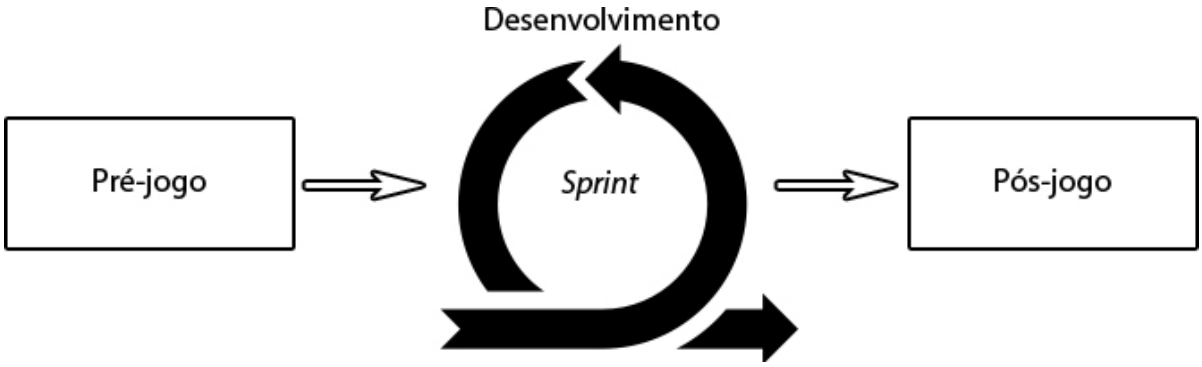

Figura 2.2: Processo Scrum.

2.3.3 Práticas

Existem algumas práticas de gestão que devem ser seguidas no Scrum [25]:

Product Backlog. Define tudo o que é necessário no produto final com base no conhecimento atual. É composto por uma lista de requisitos que é atualizada constantemente. Os requisitos do *Product Backlog* podem incluir, por exemplo, caraterísticas, funções, correções de erros, defeitos e melhorias. Esta prática inclui as tarefas para criar a lista de requisitos do *Product Backlog* e controlá-lo de forma consistente durante o processo de adicionar, remover, especificar, atualizar e priorizar requisitos. O *Product Owner* é responsável pela manutenção do *Product Backlog*.

Existe uma reunião associada a esta prática denominada de *Product Backlog Grooming*, que pode ser realizada com a finalidade de alterar, estimar e ordenar requisitos do *Product Backlog*. Esta reunião ocorre parcialmente durante um *Sprint*, não sendo obrigatória. Os participantes nesta reunião normalmente são o *Product Owner* e a equipa de desenvolvimento [26].

Estimativa de Esforço. Toda a equipa é responsável por esta prática, que consiste em estimar, para cada um dos requisitos do *Product Backlog*, o esforço necessário para o concluir.

Sprint. A equipa Scrum organiza-se de forma a desenvolver os requisitos selecionados para o

respetivo *sprint* (pode durar de 1 a 4 semanas). A partir do momento que for estipulada a duração de um *sprint*, essa duração deve ser mantida até ao fim do projeto. Dentro desta prática são realizadas outras práticas tais como o *Sprint Backlog*, as reuniões *Daily Scrum*, a reunião *Sprint Planning*, a reunião *Sprint Review* e a reunião *Sprint Restrospective*, que veremos de seguida. A Figura 2.3 mostra o esquema de desenvolvimento dos *sprints*.

Além dos *sprints* normais, caso haja necessidade, também se pode aplicar o *Sprint Zero*, que ocorre antes do início do desenvolvimento. O *Sprint Zero* permite à equipa compreender melhor as necessidades dos utilizadores, explorar o contexto e identificar os objetivos para o projeto como um todo. A consolidação do conhecimento é utilizada para negociar as prioridades do primeiro *sprint* e refinar o *Product Backlog* [27].

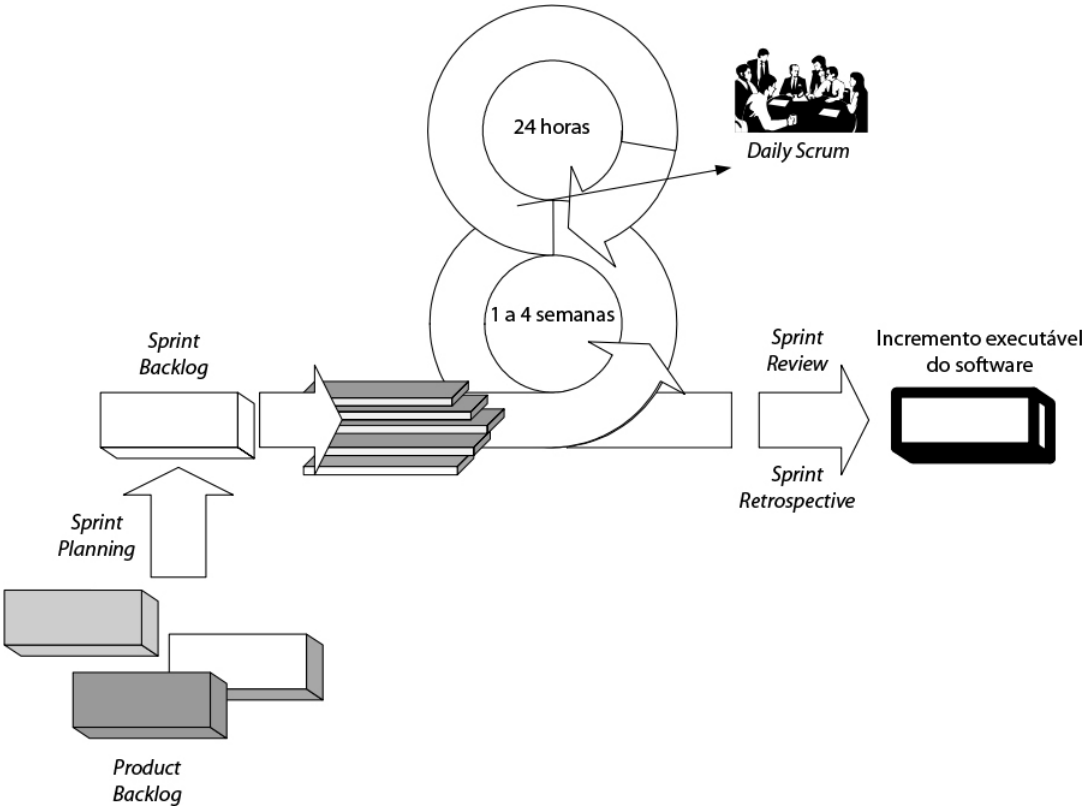

Figura 2.3: Esquema de desenvolvimento dos *sprints* (Adaptada de [3]).

Reunião *Sprint Planning*. Esta reunião está dividida em duas fases. A primeira é organizada

pelo *Scrum Master* e todos os intervenientes do processo estão presentes com o objetivo de decidir quais os requisitos a desenvolver no próximo *sprint*, ou seja, é efetuada uma priorização dos requisitos. A segunda é entre o *Scrum Master* e a equipa Scrum, concentrando-se na implementação a realizar durante o *sprint*.

***Sprint Backlog*.** É uma lista de requisitos selecionados do *Product Backlog* para um *sprint*, sabendo que esses requisitos são estáveis até o *sprint* estar concluído.

Reunião *Daily Scrum*. Durante o *sprint* é realizada uma reunião diária, sempre no mesmo horário, com a equipa de pé e com uma duração máxima de 15 minutos. Nestas reuniões não se vão resolver problemas nem fazer apresentações do estado do *sprint*, o objetivo é acompanhar o progresso da equipa e realizar compromissos perante a mesma [28]. O acompanhamento do progresso é realizado pelo *Scrum Master* que pede aos elementos da equipa Scrum para responder a três perguntas [29]:

1. O que foi feito desde a última reunião diária?
2. Que dificuldades surgiram, e como é que os outros membros podem ajudar?
3. O que será feito até à próxima reunião diária?

O *Scrum Master* vai orientando estas reuniões tentando sempre motivar a equipa.

Reunião *Sprint Review*. No último dia do *sprint*, é incrementado ao software todo o trabalho efetuado nesse *sprint* e é apresentado às partes interessadas para avaliação.

Reunião *Sprint Retrospective*. Esta reunião ocorre após a reunião *Sprint Review* e antes da próxima reunião *Sprint Planning*. O objetivo é identificar melhorias para implementar no próximo *sprint* [26].

2.4 XP

Um dos métodos mais populares e recomendados dos métodos ágeis é o XP. Este método foi desenvolvido para atender às necessidades de pequenas equipas de desenvolvimento de soft-

ware, que são confrontadas com a mudança de requisitos [30].

2.4.1 Valores

Ao avaliar certas atividades do processo de desenvolvimento de software deve haver referências para entender se há progressos. O XP tem quatro valores fundamentais que são usados para orientação das práticas aplicadas [30]:

Comunicação. A maioria das falhas dos projetos pode ser atribuída a problemas com a comunicação. A comunicação substitui quase todos os outros valores. A comunicação eficaz entre todos os envolvidos no projeto é a chave para o sucesso [31];

Simplicidade. "Qual é a coisa mais simples que poderia funcionar?" [32]. A mensagem é simples. Não tentar antecipar o futuro e sim deixá-lo acontecer. Muitas vezes quando se tenta fazer essa previsão fica demasiado caro. A satisfação dos requisitos funcionais atuais é o mais relevante [31];

Feedback. As práticas do XP são projetadas para o feedback constante. As práticas, tais como lançamentos de curta duração, integração contínua e teste, fornecem feedback muito claro. Esse feedback permite que o projeto possa avançar sobre uma base muito sólida;

Coragem. A coragem de reestruturar o código ou deitar todo o código fora e refazer novamente [32].

2.4.2 Práticas

O XP emprega 12 práticas ágeis [32, 31, 33].

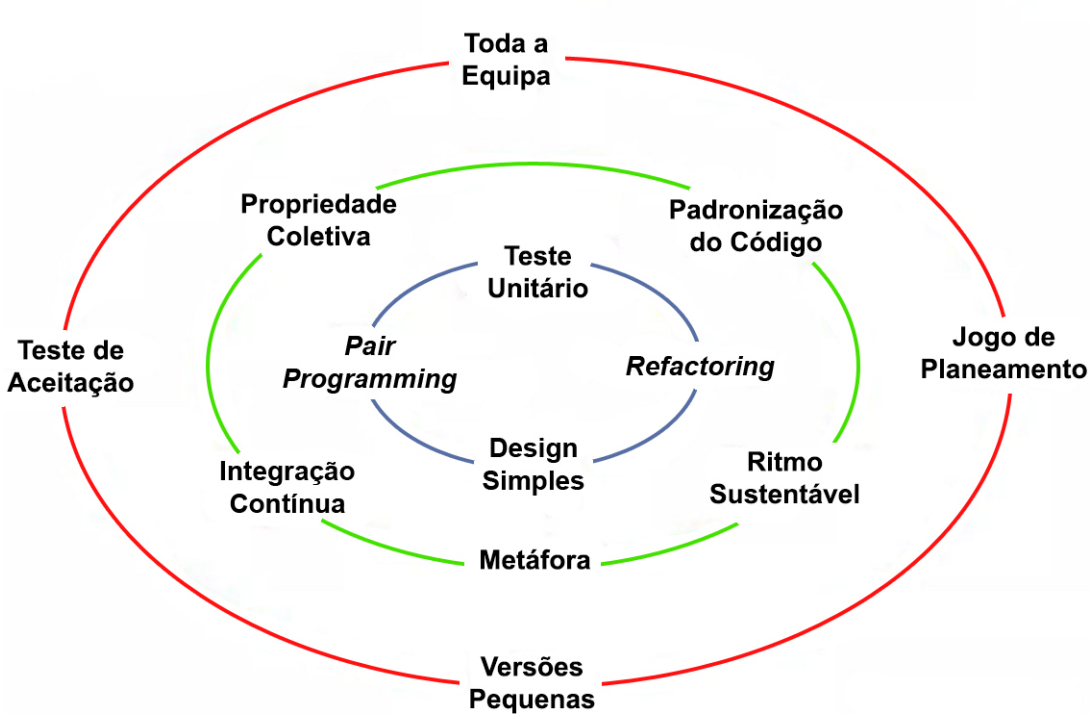

Figura 2.4: Práticas XP.

1. Jogo de Planeamento. Cliente e equipa de desenvolvimento trabalham juntos para o planeamento do produto. Isto pode acontecer em diferentes escalas. Por exemplo no início do desenvolvimento a sessão do planeamento será mais longa. Mas pequenas sessões podem ocorrer depois de cada iteração. Independentemente da escala, as regras são geralmente sempre as mesmas:

1. Cliente e equipa de desenvolvimento criam uma lista de novos requisitos para o produto.
2. Equipa de desenvolvimento cria estimativas para o esforço necessário da próxima iteração, e o esforço para concluir cada requisito. Aqui também podem ser realizadas *spikes*. A *spike* é uma técnica usada para atividades como pesquisa, inovação, conceção, investigação e prototipagem [2]. Com *spikes*, pode-se estimar adequadamente o esforço de desenvolvimento associado a um requisito ou mesmo entender melhor uma exigência, minimizando assim o risco.

3. O cliente e a equipa de desenvolvimento priorizam os requisitos e decidem quando será lançada a próxima versão do software funcional.

2. Versões pequenas. Colocar um sistema simples em produção rapidamente, em seguida, lançar novas versões em ciclos muito curtos. Sendo também um fator chave na obtenção de feedback sobre o software funcional.

3. Metáfora. O uso de metáforas possibilita a transmissão de ideias de uma forma mais clara e simples. Fazer com que o sistema seja fácil de explicar para os outros. Devemos estar aptos para explicar o sistema a terceiros que não têm nenhuma experiência técnica. Principalmente, é importante que o cliente e desenvolvedores possam compreender a metáfora. Um exemplo de metáfora pode ser visto em [34], onde uma descrição de um sistema de recuperação de informação baseada em agentes é descrita através da seguinte metáfora "este programa funciona como uma colmeia de abelhas, que saem para encontrar o pólen e trazem-no de volta para a colmeia".

4. Design simples. Os requisitos mudam constantemente. Não há interesse em detalhar o design para algo que não se sabe como irá ficar. Um dos lemas do XP é "Faça o mais simples que possa funcionar" [32].

5. Testes. Os testes incluem testes unitários, os quais são automatizados e escritos pelos desenvolvedores e testes de aceitação, onde os clientes escrevem os testes demonstrando que as caraterísticas foram alcançadas. Os testes são o indicador de conclusão.

6. *Refactoring*. Reestruturar o código sem alterar o comportamento do mesmo. Este melhoramento da estrutura interna do código permite remover a duplicação, melhorar a comunicação, simplificar e permitir que novas funcionalidades sejam adicionadas facilmente no futuro.

7. *Pair Programming*. Dois desenvolvedores trabalham lado a lado num computador para o desenvolvimento do código, permitindo a revisão constante do mesmo. Pode ser mais produtivo do que dois desenvolvedores trabalharem individualmente.

8. **Propriedade coletiva.** Qualquer desenvolvedor pode alterar qualquer código em qualquer parte do sistema a qualquer momento.

9. **Integração contínua.** Exige aos desenvolvedores integrar o seu código cada vez que uma tarefa é concluída. Os testes devem passar a 100% antes da integração.

10. **40 horas semanais.** Não ultrapassar 40 horas de trabalho por semana é uma regra.

11. **Cliente presente.** Convida o cliente a fazer parte da equipa de desenvolvimento em tempo integral, está presente para clarificar e responder a questões acima de tudo. O cliente também é responsável por escrever testes de aceitação. Ele também está envolvido em todas as fases de desenvolvimento.

12. **Padronização do código.** Comunicação é um valor fundamental. Adotar padrões de codificação melhora a comunicação, uma vez que o código é consistente de classe para classe. Todos os desenvolvedores devem ter o mesmo padrão de código.

2.5 Adoção de Métodos Ágeis

2.5.1 Desafios

Enquanto os métodos tradicionais continuam a dominar o desenvolvimento de sistemas, inúmeras opiniões e inquéritos demonstram claramente a crescente popularidade dos métodos ágeis. A chegada destes métodos dividiu a comunidade de desenvolvimento de software em campos opostos de tradicionalistas e agilistas [5].

Segundo Boehm [35], as organizações devem cuidadosamente evoluir para o melhor equilíbrio entre os métodos ágeis e adaptá-los para se enquadrarem corretamente na organização. Claramente, a maioria das organizações não podem ignorar a onda ágil, mas para as organizações que estão habituadas aos métodos tradicionais, a adoção de métodos ágeis provavelmente irá representar vários desafios, uma vez que os dois métodos de desenvolvimento de software são baseadas em conceitos opostos [5].

Pesquisas anteriores mostram que mudanças no processo de desenvolvimento de software representam fenómenos complexos de mudança organizacional e não pode ser realizado apenas pela substituição de ferramentas e tecnologias atuais por novas [36].

A estabilidade apresenta um dos maiores obstáculos à adoção de métodos no desenvolvimento ágil. As diferenças entre as métodos tradicionais e ágeis recomendam as organizações a repensar os seus objetivos [5].

A cultura organizacional tem um impacto significativo sobre a estrutura social das organizações, que por sua vez influência o comportamento e as ações das pessoas [36]. Uma vez que uma cultura é bem estabelecida, é difícil e demorado para mudar. Essa estabilidade cultural pode ser o desafio mais significativo para a adoção de abordagens ágeis [37].

Os métodos ágeis exigem uma mudança de gestão e controlo para a liderança e colaboração. As organizações que facilitam essa mudança precisam da combinação entre autonomia e cooperação, proporcionando flexibilidade e capacidade de resposta [37]. Com essa mudança o papel de um gestor de um projeto tradicional deve ser alterado para o papel de facilitador, que dirige e coordena os esforços de colaboração de todas as pessoas envolvidas no desenvolvimento [38].

A aplicação dos métodos tradicionais no desenvolvimento de software resulta numa grande quantidade de documentação, já as abordagens ágeis incentivam a redução de sobrecarga, principalmente na documentação. Grande parte do conhecimento dos membros da equipa de desenvolvimento ágil é adquirido pela experiência ao longo da vida. Por essa razão as organizações podem-se tornar dependentes das equipas de desenvolvimento e pode potencialmente alterar o equilíbrio de gestão para as mesmas [5].

Um processo social cooperativo caraterizado por colaboração e comunicação entre membros que valorizam e confiam uns nos outros, é fundamental para o sucesso dos métodos ágeis [39, 38]. No entanto para programadores acostumados a atividades solitárias ou que trabalham em grupos homogêneos de analistas e designers, a tomada de decisão colaborativa pode ser esmagadora [5].

Atualmente, há poucas evidências para sugerir que os métodos ágeis irão funcionar na ausência de pessoas competentes e acima da média [37, 35]. Isso pode causar problemas devido à dificuldade em encontrar pessoas para a equipa de desenvolvimento que usam métodos ágeis

e à desmotivação dos desenvolvedores não ágeis [5].

Num ambiente ágil, a equipa de desenvolvimento e o cliente tomam a maioria das decisões. Isso cria um ambiente de tomada de decisão diverso devido às várias atitudes, objetivos e disposições cognitivas dos elementos da equipa [40]. Para uma organização pode haver necessidade de um enorme esforço, tempo e paciência para construir uma cultura de confiança e respeito entre os seus funcionários para facilitar essa tomada de decisão colaborativa [5].

O problema da mudança de atitudes e práticas, principalmente em organizações que durante anos tentaram atingir maiores níveis de CMM (*Capability Maturity Model*), só pode ser solucionado através de um investimento significativo de tempo, esforço e capital [5].

Os processos tradicionais são baseados em medições orientadas para a conformidade e atividades. Os métodos ágeis dependem de especulação, ou de planear com a percepção de que tudo é incerto, para encaminhar o rápido desenvolvimento de sistemas flexíveis e adaptáveis de alto valor. Estes métodos salientam a importância de avaliação ao contrário da medição, e são altamente tolerantes à mudança. Uma das maiores barreiras para a migração é a mudança de um modelo de processo a partir de um modelo de ciclo de vida para um que suporta o desenvolvimento evolutivo e iterativo. Essa mudança implica grandes alterações nos procedimentos de trabalho, ferramentas e técnicas, canais de comunicação, estratégias de resolução de problemas, e os papéis de pessoas dentro das organizações [5].

O maior desafio para um gestor de projeto é a seleção do método mais apropriado dentro de um conjunto de métodos ágeis atualmente disponíveis. Eles diferem em termos de tamanho da equipa, propriedade do código, duração de cada ciclo iterativo e mecanismos de reação rápida e de mudança. Na ausência de uma abordagem ágil unificada, as organizações devem decidir o que é mais compatível com as suas práticas existentes [5].

As ferramentas desempenham um papel fundamental no sucesso da implementação de um método de desenvolvimento de software. As organizações que pretendem adotar métodos ágeis devem investir em ferramentas que suportem e facilitem o rápido desenvolvimento iterativo, gestão de versões e configurações, JUnits, refactoring e outras técnicas ágeis [5].

2.5.2 Impactos

A literatura mostra que com métodos ágeis muitas vezes o ROI (Retorno do Investimento) é mais rápido em relação aos métodos tradicionais [41]. Como confirmação, Leffingwell [42] afirma que o ROI torna-se maior do que zero imediatamente após o primeiro *sprint*. O ROI é uma maneira de medir o valor de negócio de métodos ágeis para novos produtos de desenvolvimento de software, onde o resultado pode ser obtido da seguinte forma [43]:

ROI = (Benefícios – Custos) / Custos

Onde Benefícios são os ganhos totais adquiridos a partir de métodos ágeis, incluindo benefício do uso do novo sistema e os Custos são o montante total gasto com os métodos ágeis, incluindo formação, *coaching*, ferramentas automatizadas, entre outros [43].

Em alguns métodos, como o Scrum, onde existe o *Product Backlog*, sabe-se que a priorização dos requisitos a serem desenvolvidos em todo o projeto é efetuada em função do valor de negócio esperado para cada requisito [44], maximizando o ROI logo nos primeiros *sprints* [41]. Quando se trata da adoção do Scrum, o ROI normalmente é obtido de forma diferente [45]:

ROI = (Valor de Negócio) / Esforço

Onde o Valor de Negócio e o Esforço são apresentados em pontos atribuídos aos requisitos do *Product Backlog* [41].

A adoção de métodos ágeis influência positivamente o custo, produtividade, qualidade e satisfação do cliente, o que resulta no aumento do ROI do projeto [46], a par disso existem cada vez mais organizações que querem adotar os métodos ágeis [4].

No que se refere às práticas ágeis, existem algumas que são usadas com mais frequência nas organizações de desenvolvimento de software [47]:

- Participação ativa dos *stakeholders*;
- Otimização do código fonte;
- Testes de regressão do código fonte;

- Guias de codificação comuns;
- Integração contínua;
- Otimização da base de dados;
- *Pair Programming*;
- TDD (*Test-Driven Development*).

No entanto, as equipas de desenvolvimento ágil de software não têm a obrigatoriedade de adotarem essas práticas ágeis, podem adaptá-las em função de diversos critérios e fatores [48]. A combinação de várias práticas ágeis que melhor se adequam ao projeto, revela um maior nível de maturidade ágil [49].

Ao longo dos últimos anos as organizações têm questionado: "Porque devemos adotar práticas ágeis?" [50]. São inúmeras as histórias de sucesso de organizações que adotaram práticas ágeis e que dão resposta a esta questão [51, 52, 53, 54], levando a muitas outras organizações quererem adotar essas práticas [4].

Uma das preocupações das organizações quando pretendem adotar práticas ágeis é determinar como o podem fazer, através da maturidade ágil, ou seja, o grau em que uma organização pode adotar práticas ágeis. Para permitir uma avaliação da maturidade ágil de uma organização é referida uma escala como o SAMI (*Sidky Agile Measurement Index*) [4].

SAMI foi desenvolvido na Virginia Tech por Dr. Ahmed Sidky na tentativa de orientar a adoção ágil numa organização [55]. O SAMI baseia-se na ideia de que quanto maior o números de práticas ágeis adotadas numa organização, maior a sua agilidade [55], determinando assim a sua maturidade ágil. Como a contagem de práticas adotadas é uma medida simples, Dr. Ahmed Sidky criou 5 níveis ágeis. Cada nível ágil tem um conjunto de práticas, as mesmas refletem os valores relevantes do manifesto ágil. Uma organização pode alcançar qualquer um dos níveis de agilidade, desde que adote todas as práticas desse nível e os níveis abaixo [4].

Existem estudos que demonstram que a utilização de vários métodos ágeis em simultâneo, em particular a combinação do XP e Scrum, trazem benefícios, tais como baixo custo, aumento

da produtividade individual dos elementos da equipa, qualidade do software e satisfação dos clientes [56, 57, 1].

Uma comparação do XP e Scrum é apresentada na Tabela 2.1. A comparação é baseada nos parâmetros de qualidade de práticas ágeis e nível de disponibilidade entre ambos os métodos [1].

Tabela 2.1: Comparação do XP e Scrum [1]

Parâmetro de qualidade	**XP**	**Scrum**
Práticas de engenharia	Sim	Não
Práticas de gestão de projeto	Não	Sim
Aceitação de mudança em cada iteração	Sim	Não
Prioritização de requisitos	Sim	Não
Refactoring	Sim	Não
Pair Programming	Sim	Não
Tamanho do projeto	Pequeno a médio	Médio a grande
TDD	Sim	Não
Auto-organização	Não	Sim
Testes unitários	Sim	Não
Conceção	Centrada no código	Centrada na conceção
Nível de documentação	Menos	Mais
Tamanho da equipa	<10	<10 e múltiplas equipas
Estilo de código	Limpo e simples	Não especificado
Ambiente tecnológico	Feedback rápido	Não especificado
Ambiente físico	Equipa no mesmo local e pouco distribuída	Não especificado
Cultura de negócio	Colaborativo e cooperante	Não especificado

Capítulo 3

Estudo de Casos

Neste capítulo é apresentado um estudo de dois casos. Os casos estudados foram definidos como unidades de análise. O primeiro é o projeto iFlow e o segundo o projeto CITT (*Cross Independent Test Tool*), ambos projetos de I&DT (Investigação e Desenvolvimento Tecnológico).

Para este estudo de casos houve a necessidade de recolha de dados. No caso do projeto iFlow foram utilizados documentos técnicos e principalmente um artigo [2] relacionado com o projeto. No caso do projeto CITT, além dos documentos relacionados com o projeto, também foram realizadas entrevistas a elementos envolvidos no processo de execução do projeto. Após a recolha de dados foi possível escrever este capítulo, que resultou da análise e interpretação dos dados.

Os aspetos importantes que estão contidos neste capítulo que se encontram relacionados com o tema deste trabalho são: (1) o processo de execução, e (2) as práticas ágeis. Estes aspetos foram definidos principalmente através de dois projetos de desenvolvimento de software, utilizados como unidades de análise. Em ambos os projetos analisou-se qual o processo de execução utilizados e quais as práticas ágeis envolvidas.

Este capítulo tem como principal objetivo contribuir com recomendações para a adoção de práticas ágeis no desenvolvimento de software, através da análise das evidências relacionadas com a adoção dessas práticas em cada um dos projetos.

3.1 Projeto iFlow

3.1.1 Contextualização

O projeto iFlow fez parte de uma das linhas de investigação de um projeto de I&DT chamado HMIExcel (*Human-Machine Interface Excellence*).

O projeto HMIExcel foi patrocinado pelo consórcio entre a Universidade do Minho (UMinho) e Bosch Car Multimedia Portugal (Bosch), foi projetado para enfrentar os desafios científicos e tecnológicos da Bosch Braga e para obter o reconhecimento desta unidade como um centro de competência internacional de interface homem-máquina (HMI). O projeto teve a duração de 28 meses, iniciou em março de 2013 e terminou em Junho de 2015. Embora HMIExcel para o organismo de financiamento fosse visto como um projeto, a sua complexidade e incerteza levou o consórcio (UMinho e a Bosch) a gerir como um programa, ou seja, um conjunto de projetos que estão de alguma forma relacionados a contribuir para o mesmo objetivo. O objetivo principal do HMIExcel foi promover o investimento em I&DT, para desenvolver e produzir conceitos de mobilidade futuras no domínio automóvel, onde a academia e a indústria trabalharam em conjunto tendo em vista produtos e processos inovadores. Estas soluções foram desenvolvidas para atender às necessidades de produção e preparação da Bosch para responder aos desafios da Quarta Revolução Industrial (Indústria 4.0).

O projeto HMIExcel dividiu-se em treze linhas de investigação, cada uma abordando um desafio específico. Cada linha de investigação seguiu um curso independente, todos os projetos foram coordenados por uma equipa focada nos objetivos e nas entregas esperadas em cada linha individual. Uma dessas linhas de investigação, o iFlow, será usado neste trabalho como um estudo de caso.

O iFlow é um projeto de I&DT, que teve como finalidade o desenvolvimento de um sistema de software de logística integrada para a rastreabilidade da cadeia de fornecimento interno. O iFlow é um sistema de software de monitorização em tempo real dos fretes em trânsito, desde os fornecedores até à fábrica Bosch, localizada em Braga. O principal objetivo do projeto foi desenvolver um sistema de software de rastreamento que através da integração da informação

de transitários e dos dispositivos de GPS (nos veículos) permitisse controlar o fluxo de matérias-primas de fornecedores remotos (asiáticos) e locais (europeus) para o armazém da Bosch, notificando os utilizadores em caso de qualquer desvio em relação ao tempo estimado de chegada (ETA) e antecipando desvios do tempo de entrega.

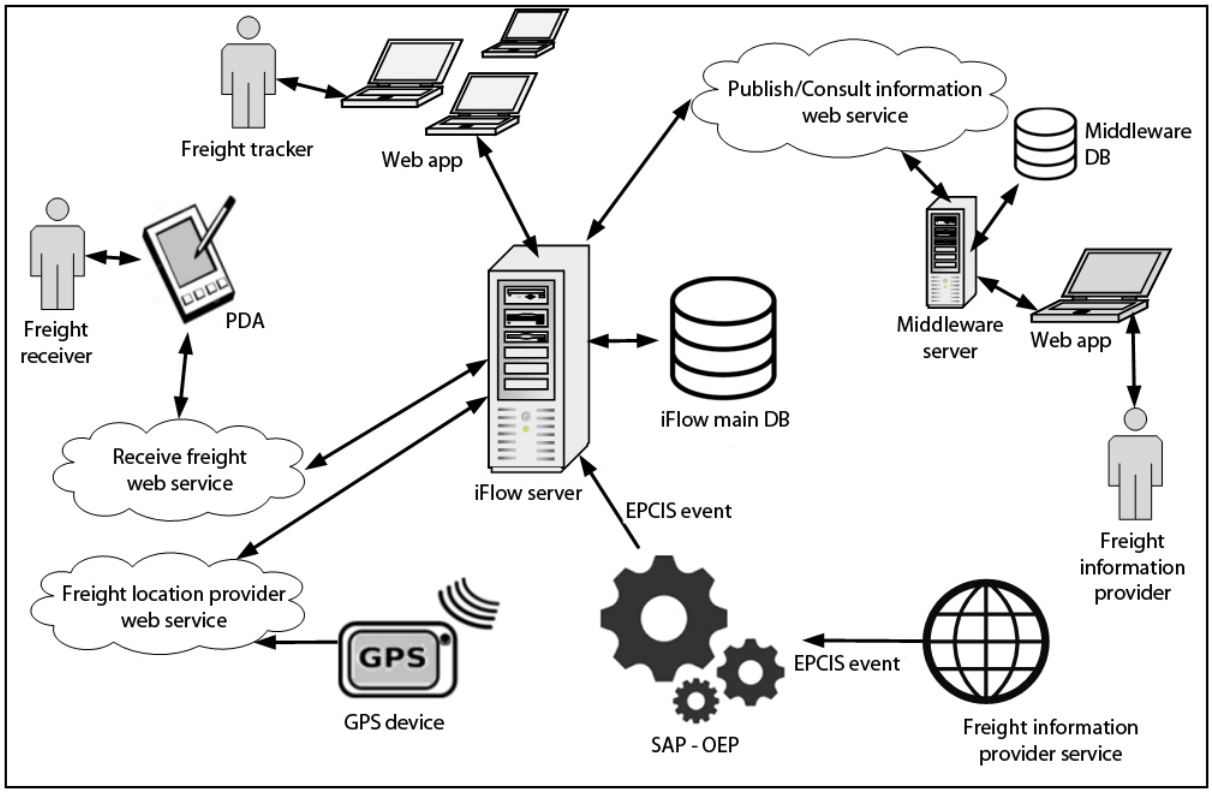

Figura 3.1: Arquitetura do sistema de software iFlow [2].

A Figura 3.1 ilustra a arquitetura do sistema de software iFlow, ou seja, a integração do servidor iFlow (iFlow *server*) com outros sistemas. A arquitetura permite representar os protocolos de interface significativos e os sistemas que foram envolvidos. O sistema utiliza informações que são fornecidas por diferentes fontes, como GPS, dispositivos de assistente pessoal digital (PDA) ou SAP-OER (objeto de repositório de eventos). O servidor *middleware* (*middleware server*) foi desenvolvido com o objetivo de padronizar a forma como a informação entre a Bosch e os fornecedores é trocada. Em seguida, o servidor iFlow executa toda a lógica de negócio, onde os funcionários da Bosch têm acesso a todos os recursos através de uma interface web (*app web*).

No caso do projeto iFlow, esta colaboração UMinho e Bosch foi baseada na premissa, uma vez que a Bosch era considerada principalmente como um cliente de software e a UMinho como uma entidade de desenvolvimento de software contratada. A equipa principal do iFlow foi composta por nove elementos com formações multidisciplinares:

- Bosch:
  - um *Product Owner*, que representava outros oito elementos do departamento de logística, e tinha a responsabilidade de estabelecer os requisitos;
  - um elemento do departamento de TI (Tecnologia da Informação), responsável por validar que cada incremento do produto desenvolvido poderia ser facilmente integrado com o sistema da Bosch. Também colaborou com a Equipa de Desenvolvimento em algumas funcionalidades do software;
- UMinho:
  - três coordenadores de I&DT, responsáveis por assegurar que o rigor científico (tanto do sistema como do processo de desenvolvimento de software) e os prazos do projeto fossem atingidos;
  - quatro desenvolvedores de software com competências metodológicas e tecnológicas (como análise de requisitos, conceção, modelação de base de dados, programação, testes, *deployment*, entre outras).

Todo o desenvolvimento de software foi realizado nas instalações da Bosch, onde os elementos da equipa iFlow (à exceção dos coordenadores de I&DT) estavam localizados diariamente. Os elementos da UMinho não tinham conhecimento prévio do domínio (neste caso, logística), pelo que a equipa decidiu que o arranque do projeto consistia na recolha exaustiva e cuidada dos requisitos do sistema a desenvolver.

Após a realização da engenharia de requisitos, e como o iFlow é um sistema de software para um contexto industrial, a equipa decidiu seguir o método Scrum como a abordagem iterativa para a fase de implementação. Esta fase foi realizada através de *sprints*. Com base nas

entregas de software incrementais, tanto a UMinho e a Bosch poderiam gerir as expectativas do seu projecto.

Como foi um projeto colaborativo de software Universidade-Indústria de I&DT, os papéis anteriormente apresentados são ligeiramente diferentes dos papéis definidos pelo Scrum (ou seja, *Product Owner*, *Scrum Master* e Equipa de Desenvolvimento), no entanto foram facilmente mapeados, conforme ilustrado na Tabela 3.1.

Tabela 3.1: Mapeamento entre os papéis iFlow e os papéis Scrum (Adaptada de [2]).

Papéis Scrum / Papéis iFlow	*Product Owner*	*Scrum Master*	Equipa de Desenvolvimento
Product Owner (Bosch)	●		
Elemento de TI (Bosch)	●		●
Coordenadores de I&DT (UMinho)		●	
Desenvolvedores de Software (UMinho)			●

O *Product Owner* foi o elemento responsável por estabelecer os requisitos, tal como referido anteriormente, mas também participou em reuniões com os coordenadores de I&DT da UMinho e desenvolvedores de software para o monitorização do projeto (o papel gestor de projeto não está incluído dentro de uma equipa Scrum). Assim, é mapeado diretamente para o papel de *Product Owner*. Além disso, o *Product Owner* era o ponto de contato entre o desenvolvimento e as entidades prestadoras de serviços que concebiam o ecossistema. O facto do sistema de informação da Bosch (não só das instalações de Braga, mas das instalações em todo o mundo da Bosch Car Multimedia) ser gerido pela Bosh Car Multimedia Group, na Alemanha, e do sistema em desenvolvimento estar incluído no sistema de informação da Bosch, inclui também o departamento de TI da Bosch da Alemanha no ecossistema do projeto. Assim, o *Product Owner* também se apresentou como um ponto de contato na negociação entre a equipa iFlow e o departamento de TI da Alemanha, sobre os requisitos estarem em conformidade com a segurança e a política geral da Bosch Car Multimedia Group.

O elemento do departamento de TI da Bosch foi responsável por validar o código desenvolvido durante os *sprints*, de modo a ser integrado nos sistemas de informação existentes. Ele interagia diretamente com os desenvolvedores de software da UMinho. Por esta razão, ele foi considerado como parte da Equipa de Desenvolvimento. No entanto, ele foi responsável por representar o departamento de TI a partir de Braga durante as negociações com o departamento de TI da Alemanha e com as entidades prestadoras de serviços. Então, dentro dessas tarefas, ele também desempenhou o papel de um *Product Owner*.

A coordenação de I&DT foi composta por três professores da UMinho, das áreas de engenharia de software e também de logística. Eles foram responsáveis por assegurar que as preocupações académicas desse tipo de projetos de I&DT fossem atingidas, onde as suas principais preocupações foram assegurar o desenvolvimento para além do estado da arte, e coordenar e acompanhar o desenvolvimento de entregas do projeto e artigos científicos. Como a Bosch nunca tinha adotado processos de desenvolvimento de software ágil, eles foram responsáveis pela formação e controlo da adoção de práticas do Scrum na equipa do projeto, quando o projeto entrou na fase de implementação. Para esta responsabilidade, estes elementos são mapeados para o papel de *Scrum Master*.

Os desenvolvedores de software realizaram tarefas relacionadas com a engenharia de software (como análise de requisitos, conceção, modelação de base de dados, programação, testes, *deployment*, entre outras), portanto, a responsabilidade destes elementos foi mapeada diretamente para o papel de Equipa de Desenvolvimento.

3.1.2 Processo de Execução

Esta secção apresenta o processo de execução do software do projeto iFlow e as adaptações necessárias devido à natureza de pesquisa do projeto e à integração com serviços de terceiros. Uma visão geral do processo está representada na Figura 3.2. O processo é composto por três fases: iniciação, implementação e *deployment*.

A fase de iniciação incluiu atividades de modelação de negócios, levantamento de requisitos, e análise e conceção. A fase de implementação foi realizada em pequenas versões iterativas e

incrementais através dos *sprints*. Deve notar-se que a Figura 3.2 representa sete ciclos realizados como *sprints* (círculos com borda cheia) e um ciclo que representa o *refactoring* (círculo com borda tracejada). Este ciclo diferente pode não ser necessário em outros projetos de I&DT se a arquitetura for estável durante todo o projeto.

E por fim foi realizada a fase de *deployment*.

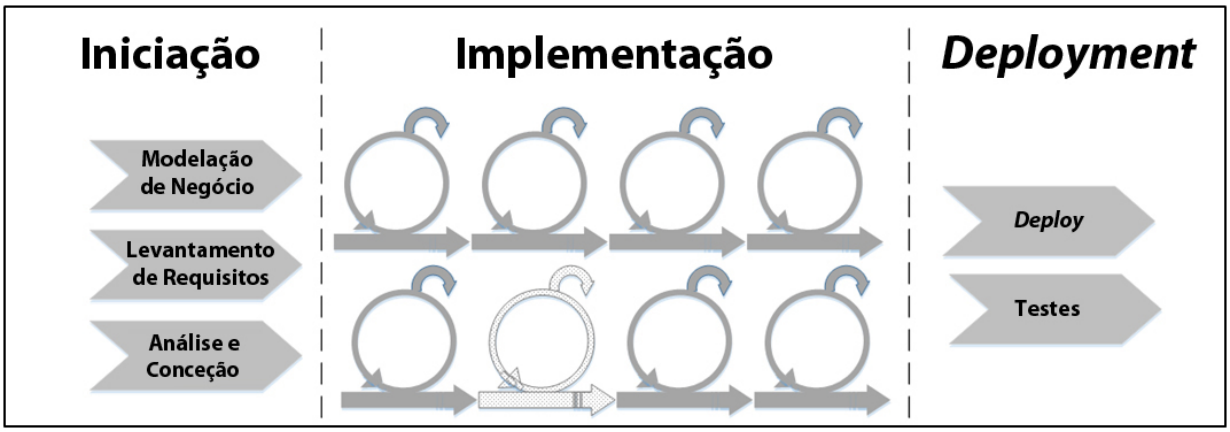

Figura 3.2: Visão geral do processo executado no projeto iFlow (Adaptada de [2]).

Fase de Iniciação

Dentro da fase de iniciação, o objetivo foi criar o *Product Backlog*, para dar início à fase de implementação na forma de *sprints*. Devido à complexidade do projeto, também foi entregue a documentação dos requisitos (o processo geral é representado na Figura 3.3).

Como o iFlow é um projeto de software de I&DT, algumas atividades de investigação foram realizadas ao longo do projeto e os resultados da investigação tiveram que ser documentados. O arranque inicial do projeto consistiu na recolha exaustiva e cuidada dos requisitos do sistema a desenvolver, e as tarefas iniciais foram conduzidas para especificar o âmbito do projeto, para caracterizar o domínio e as atividades (da logística) da organização, para definir os termos e para analisar fluxos e dados.

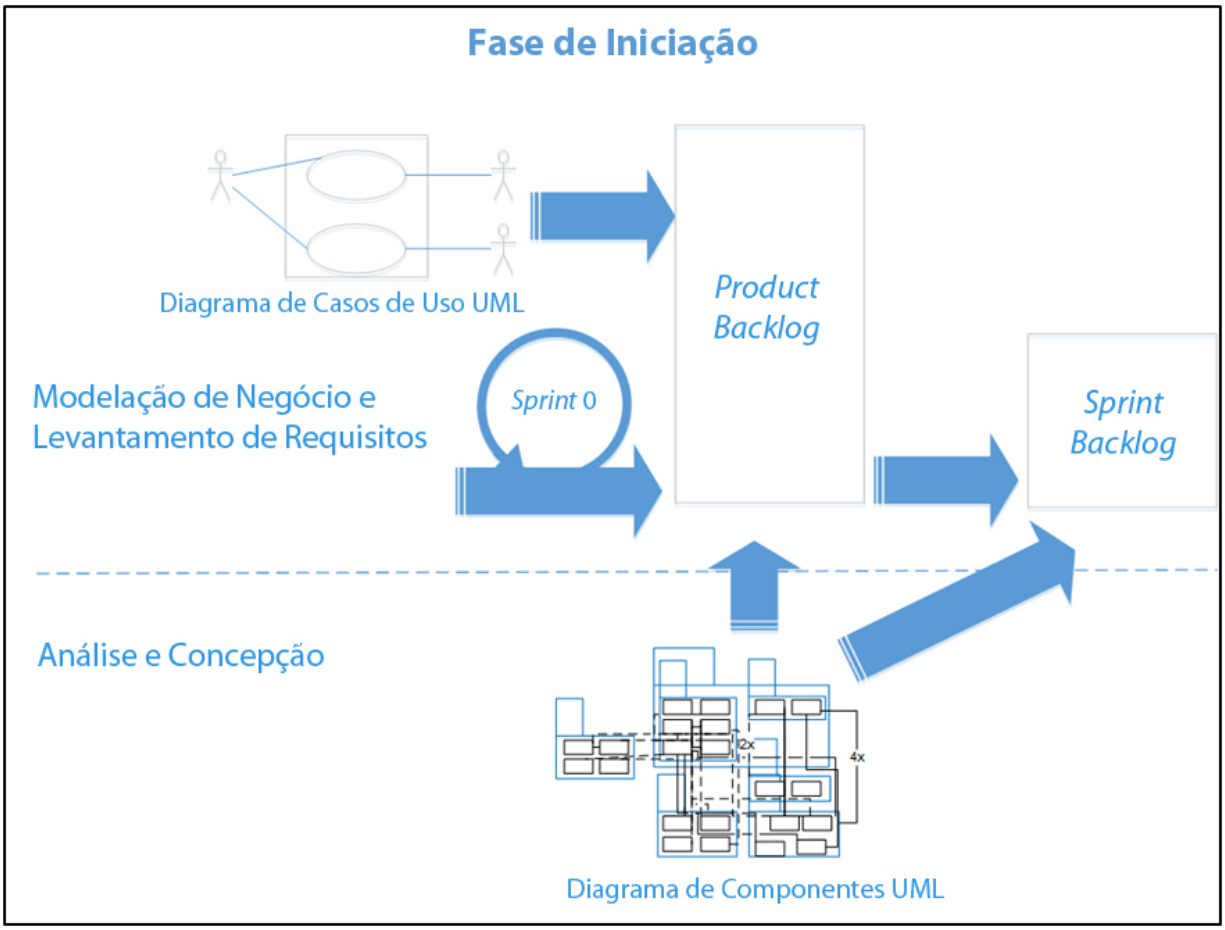

Figura 3.3: Fase de Iniciação (Adaptada de [2]).

Os processos logísticos relacionados com a organização e as lacunas do momento foram documentados num relatório designado de "*As-Is report*". Em seguida, os requisitos foram extraídos, formalmente especificados em casos de uso UML (*Unified Modeling Language*) , como mostra a Figura 3.4, dando origem a uma lista de requisitos de qualidade (não-funcionais) e uma primeira versão da arquitetura lógica (diagrama de componentes UML). Este conjunto de requisitos foi documentado em um relatório designado de "*To-Be report*" (que foi constantemente atualizado ao longo do desenvolvimento). Ambos os modelos de casos de uso (especialmente o "*To-Be*") foram usados como base para criar um *Product Backlog*. A criação do *Product Backlog* com casos de uso é diferente de outras estruturas ágeis, onde, por exemplo, em Scrum, o *Pro-*

duct Backlog é composto por *user stories*. Um *user story* é uma caracterização centrada de um requisito do cliente. Ele contém apenas as informações necessárias para os desenvolvedores do projeto perceberem claramente o que é necessário para desenvolver. No entanto, os casos de uso também são usados em estruturas ágeis.

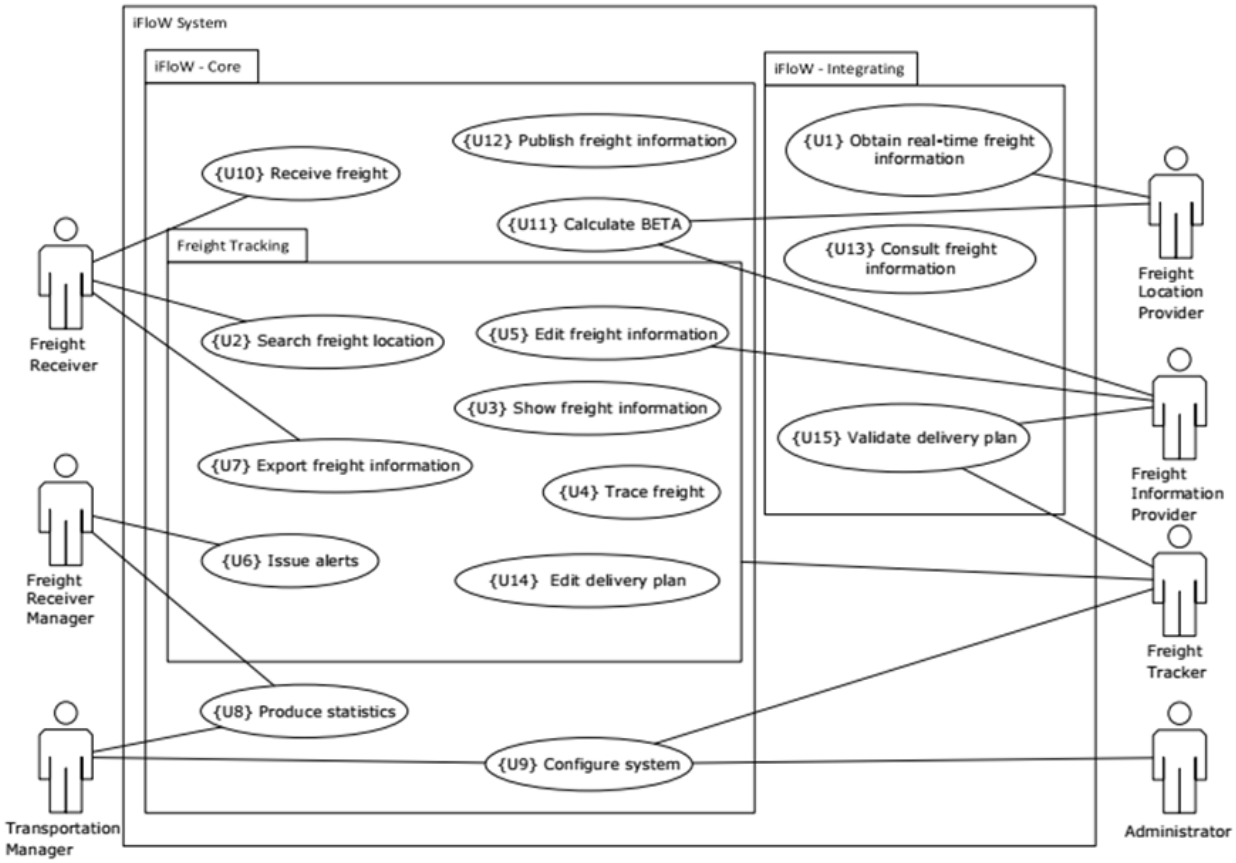

Figura 3.4: Diagrama de casos de uso [2].

Na Figura 3.4 é ilustrado o modelo de casos de uso global do fluxo do projeto. Cada um dos casos de uso foi funcionalmente decomposto, o que resultou num total de 90 casos de uso de mais baixo nível.

A fase de iniciação termina com o *Sprint Zero*. A maior parte da pesquisa tecnológica foi realizada durante este *sprint*, antes do desenvolvimento dos seguintes *sprints*. No *Sprint Zero*, cada requisito (caso de uso) foi priorizado pelo valor identificado pelas partes interessadas, neste

caso foi utilizada a técnica de priorização MoSCoW. Além disso, para cada requisito foi estimado o esforço quantitativo para a seu desenvolvimento. Uma técnica que costuma ser utilizada é a *use case points*, no entanto, neste projeto esta técnica não foi utilizada. Em vez disso, a equipa do projeto iFlow definiu que cada *sprint* corresponde a um esforço total de 20 pontos (o que resultou em cerca de cinco pontos por semana), como base para a distribuição desses pontos por requisito e seguir uma técnica comparativa semelhante à *planning poker*.

Fase de Implementação

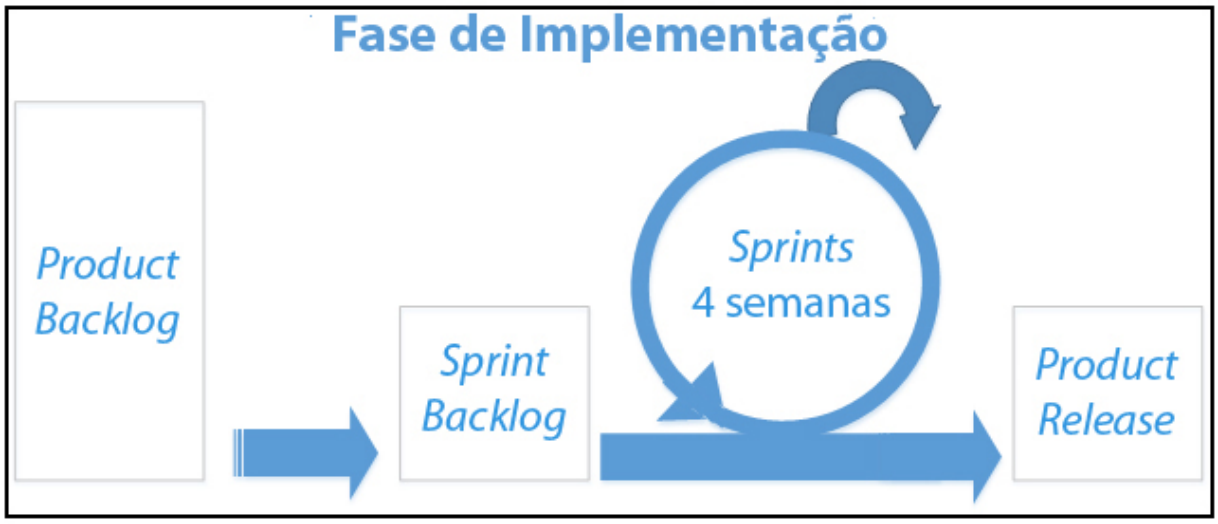

Figura 3.5: Fase de implementação (Adaptada de [2]).

Dentro da fase de implementação (ver Figura 3.5), os requisitos do *Product Backlog* foram construídos de forma iterativa e incremental durante sete *sprints* de quatro semanas. Nesta fase, foram realizadas as iterações Scrum. Na Figura 3.6 está representado um exemplo de um *Sprint Backlog* para uma das iterações realizadas.

Sprint #1Tracking Sheet

Project	*iFloW*
Sprint #	1
Start date	29/09/14

			Week			
			1	**2**	**3**	**4**
Task ID	**Description**	**Initial estimate (units)**	**Units Left**	**Units Left**	**Units Left**	**Units Left**
{U1.1.1}	Obtain freight information from Kuehne Nagel related to transit from Algeciras Consolidation Center	**7**	7	Use case moved to SP2		
{U1.1.2}	Obtain freight information from Kuehne Nagel related to transit from Hong-Kong Consolidation Center	**6**	6	4	3	2
{U3.1}	Show freight general information	**5**	4	2	1	0
{U2}	Search freight location	**2**	2	1,5	1	0,5
	Total estimate units	20				
	Remaining units (actual)		**19**	**7,5**	**5**	**2,5**
	Remaining units (ideal)		**15,0**	**10,0**	**5,0**	**0,0**

Figura 3.6: Exemplo de um *Sprint Backlog* baseado nos casos de uso.

Cada *sprint* teve um planeamento padrão e estrutura que consistia em várias etapas, previamente negociadas pelos elementos do projeto:

Desenvolvimento do *Sprint*: teve a duração de quatro semanas, e consistiu no desenvolvimento dos requisitos do *Sprint Backlog*;

Reunião da Monitorização do *Sprint*: ocorreu na segunda semana para mostrar o progresso do *sprint* e monitorizar as tarefas do *sprint*. Os participantes foram o *Product Owner*, os coordenadores de I&DT e a Equipa de Desenvolvimento;

Reunião de Verificação e Validação (V+V) do *Sprint*: ocorreu na quarta semana e o objetivo foi testar e validar os requisitos desenvolvidos pela Equipa de Desenvolvimento. Os participantes foram o *Product Owner*, a Equipa de Desenvolvimento, o elemento do departamento de TI da Bosch e um utilizador do produto da Bosch. Em cada reunião de V+V do *sprint*, o utilizador do produto dava a responsabilidade a diferentes utilizadores do departamento de logística para que os testes realizados pudessem abranger diferentes percepções da organização. Durante o *sprint*, se houvesse a necessidade de mover um requisito para um próximo *sprint* devido a uma determinada restrição, o mesmo não seria apresentado nesta reunião, e a equipa seria notificada;

Reunião de Conclusão do *Sprint*: ocorreu no máximo, dois dias após a reunião de V+V do *sprint*, e os participantes foram o *Product Owner*, os coordenadores de I&DT, o elemento do departamento de TI da Bosch e a Equipa de Desenvolvimento. É semelhante a uma reunião de *Sprint Retrospective* e uma reunião de *Sprint Planning*, realizadas dentro da mesma reunião. O objetivo principal foi analisar o progresso da fase de implementação, avaliar a percentagem e conclusão do desenvolvimento dos requisitos e atualizar o gráfico de *burndown*;

Reunião de Retrabalho do *Sprint*: ocorreu um dia depois da reunião da conclusão do *sprint*. Após a reunião do V+V do *sprint*, algumas ações de retrabalho surgiram devido a uma sugestão da equipa de V+V. Neste caso, a Equipa de Desenvolvimento teve que desenvolver essas ações de retrabalho até ao fim do *sprint*. As reuniões de retrabalho do *sprint* foram usadas para validar as ações de retrabalho realizadas. Os participantes foram os utilizadores do produto, o *Product Owner*, o elemento do departamento de TI da Bosch e a Equipa de Desenvolvimento.

Para o desenvolvimento de cada requisito, a equipa realizou tarefas que envolveram várias áreas de engenharia de software.

Ocasionalmente, a equipa realizou *spikes*. A utilização das *spikes* no projeto iFlow justifica a inclusão da área de levantamento de requisitos em cada *sprint*, como mostrado na Figura 3.7.

Em todos os *sprints*, foi necessário avaliar e atualizar a arquitetura lógica (área de análise e conceção). Em seguida, os *sprints* foram realizados nas seguintes áreas: implementação, testes e *deployment*. Estas *spikes* foram, em sua maioria, provenientes de requisitos baseados em *middleware* (por exemplo, relacionadas com a integração com serviços de terceiros, GPS, EPCIS ou SAP-REA). Dentro dos requisitos restantes, a área de levantamento de requisitos não foi necessária. Assim, em comparação com as áreas incluídas na Figura 3.8, o *sprint* foi realizado com as restantes áreas, com exceção de levantamento de requisitos. Na verdade, é o que acontece no processo Scrum (onde quase todos os esforços para o levantamento de requisitos são realizados antes dos ciclos *sprint*, como o *Sprint Zero* ou similar).

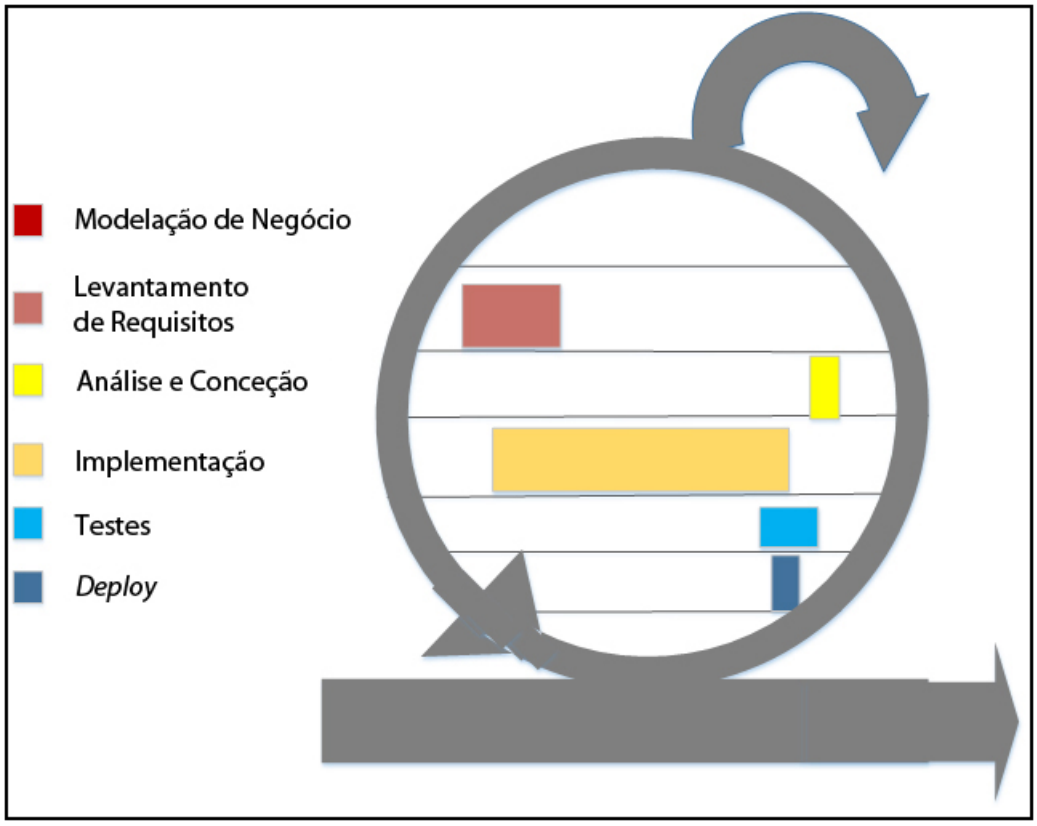

Figura 3.7: As áreas realizadas dentro dos *sprints* (Adaptada de [2]).

Além disso, a implementação de alguns requisitos de *middleware* envolveu a colaboração de terceiros. O desenvolvimento desses requisitos foi gerido a partir de uma perspetiva de equipa

distribuída. O esforço de desenvolvimento por terceiros esteve devidamente alinhado com o processo de desenvolvimento da equipa. No caso do projeto iFlow, a integração com as entidades de prestação de serviços necessitou que ambas as equipas trabalhassem juntas.

Em determinada altura, tanto a Bosch como a UMinho identificaram a necessidade de *refactoring*, para lidar com questões de segurança e de normalização. Esse *refactoring* levou a uma pausa nas tarefas de execução. A arquitetura lógica de software foi revista e os impactos foram analisados. Algumas *spikes* orientadas para a conceção (semelhantes às *spikes* de arquitectura do XP) foram realizadas, que seguiu depois para uma reestruturação da arquitetura. Neste caso, houve um foco na área de análise e conceção, em vez da execução (ver Figura 3.8, onde é detalhado o sexto *Sprint* incluído na Figura 3.2). Da mesma forma que os outros *sprints*, este esforço também durou quatro semanas. Este esforço foi necessário, neste caso, mas pode ocorrer, ou não, em qualquer projeto.

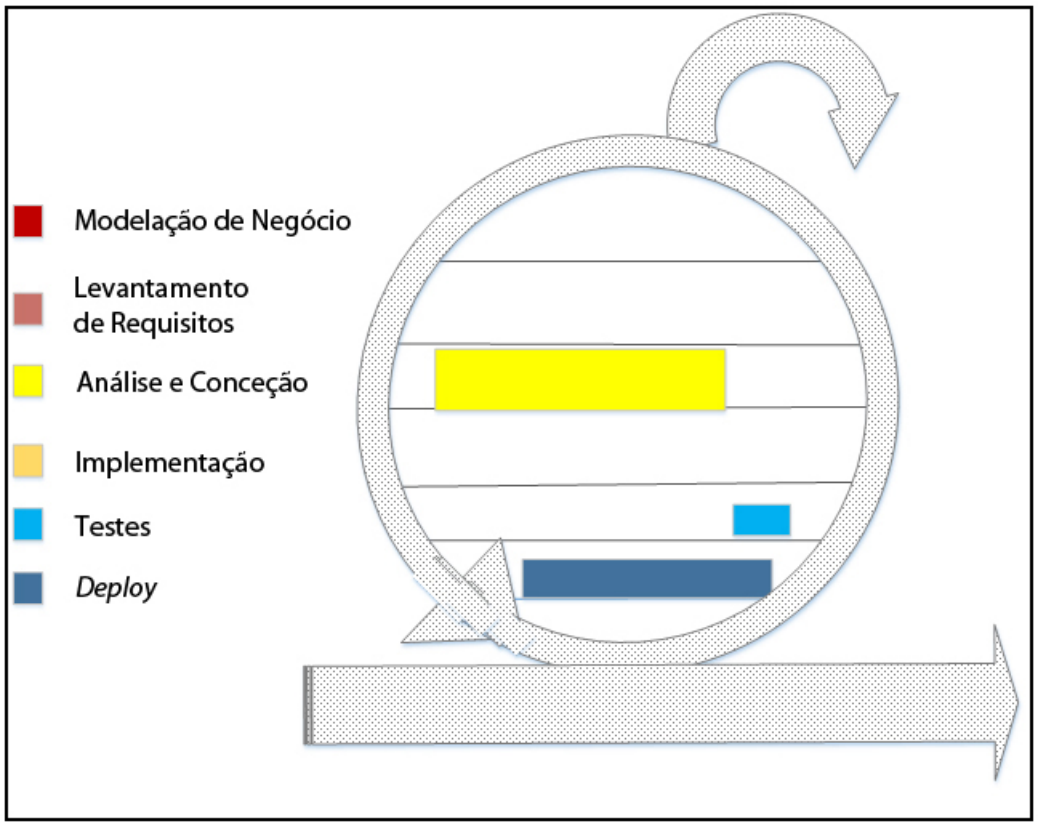

Figura 3.8: As áreas realizadas dentro do *sprint* através da técnica *spike* (Adaptada de [2]).

3.2 Projeto CITT

3.2.1 Contextualização

O projeto CITT foi desenvolvido através de um consórcio composto por três organizações, o CCG (Centro de Computação Gráfica), a WinTrust e a PRIMAVERA BSS.

O CITT foi um projeto de I&DT em co-promoção que teve a duração de 18 meses. O projeto teve como principal objetivo a definição de um processo de testes e a criação de uma ferramenta de testes automáticos, completamente diferenciadora dos produtos existentes no mercado, produtos esses conhecidos pela sua morosidade, complexidade e rigidez. Assim, este projeto visou desenvolver uma ferramenta inovadora que permite a reutilização de trabalho já realizado, obtendo um aumento significativo de produtividade no desenho de casos de teste. É uma ferramenta flexível, que possibilita o desenvolvimento de testes automáticos independentemente da tecnologia e da plataforma utilizada.

A ferramenta é capaz de acompanhar o ciclo de desenvolvimento do software, ou seja, testes que são definidos na fase inicial do desenvolvimento continuam a ser válidos ao longo do desenvolvimento, necessitando apenas serem configurados e parametrizados.

Este projeto foi capaz de ir ao encontro das necessidades anteriormente descritas, através de um processo e uma ferramenta de testes capazes de oferecer o suporte necessário às atividades de teste automático, de forma rápida e eficiente.

No início do projeto foram estudadas duas soluções para a arquitetura de alto-nível que se distinguiam apenas na camada de dados, sendo a primeira arquitetura composta por uma camada de dados partilhada, uma instância da aplicação (*Application*) e um ambiente com múltiplos utilizadores agrupados em organizações (*Multi-Tenant*). A Figura 3.9 ilustra a segunda arquitetura, sendo esta idêntica à primeira com a exceção da camada de dados, que neste caso foi composta por múltiplas instâncias (*Multi-Instance Database*), esta última solução acabou por ser escolhida porque foi mais simples de desenvolver e implementar.

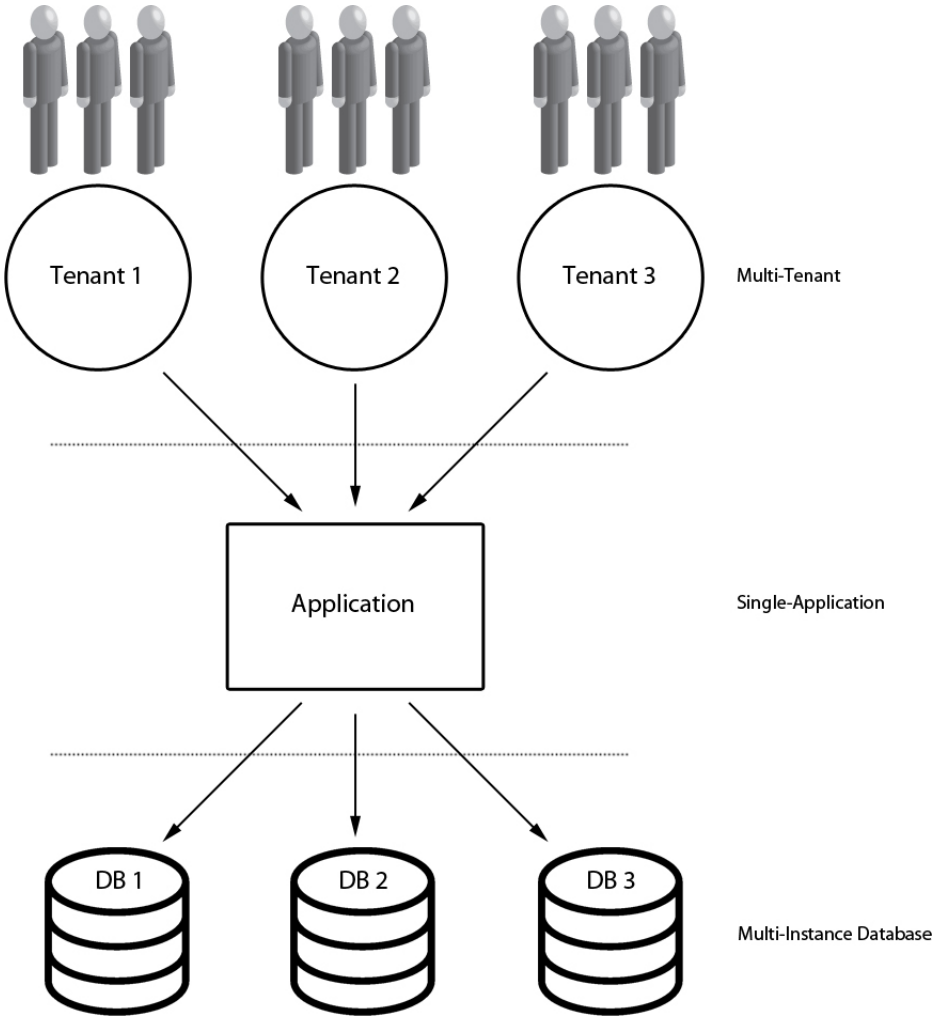

Figura 3.9: Arquitetura da ferramenta CITT.

A equipa principal do CITT foi composta por 6 elementos, a qual decidiu (por mútuo acordo entre as organizações) implementar o método Scrum (ou seja, as três organizações implementaram este método) na fase de implementação. A distribuição dos papéis dos elementos foi definida da seguinte forma:

***Scrum Master*:** um elemento da PRIMAVERA BSS, tinha o papel de assegurar que os princípios e os valores do método Scrum fossem respeitados e os prazos do projeto fossem atingidos;

***Product Owner*:** um elemento da WinTrust e um elemento da PRIMAVERA BSS, tinham a res-

ponsabilidade de estabelecer os requisitos e validar o desenvolvimento. O elemento com o papel de *Scrum Master* também assumia muitas vezes este papel, isto acontecia quando nenhum dos elementos que representava o papel de *Product Owner* estava presente nas reuniões Scrum;

Equipa de Desenvolvimento: um elemento do CCG (como analista/testador), dois elementos da PRIMAVERA BSS (um deles subcontratado ao CCG) e dois elementos da WinTrust (que estavam a trabalhar em Braga, lado a lado com as pessoas da PRIMAVERA BSS, dado possuírem os seus escritórios no mesmo edifício), que tinham a responsabilidade do desenvolvimento da ferramenta CITT.

3.2.2 Processo de Execução

O processo executado neste projeto é composto por três fases: iniciação, implementação e testes. Uma visão geral do processo está representada na Figura 3.10.

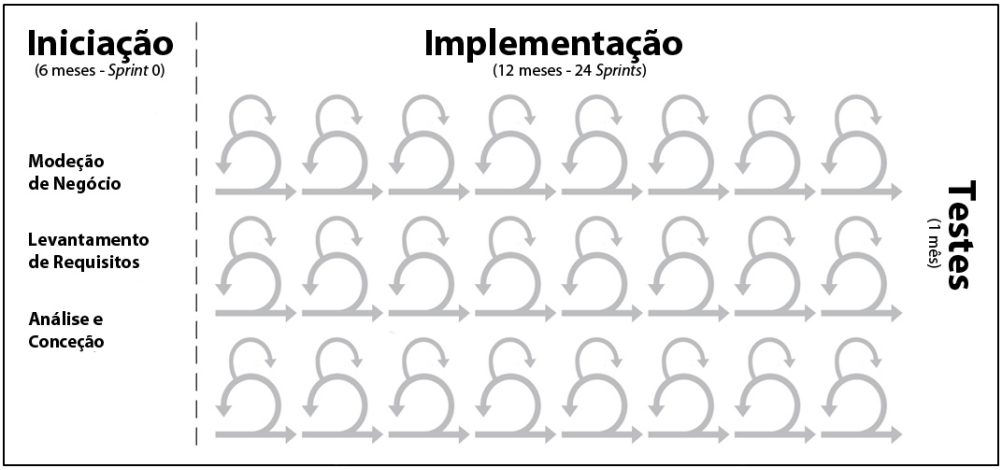

Figura 3.10: Visão geral do processo executado no projeto CITT.

Na fase de iniciação foram realizadas atividades de modelação de negócios, levantamento de requisitos e análise e conceção. A fase de implementação foi realizada através de *sprints*. E por último a fase de testes foi realizada em simultâneo com a fase de implementação no último mês (ou seja, 12º mês da fase de implementação).

Fase de Iniciação

Esta fase teve a duração de 6 meses e a sua finalidade consistiu no levantamento e definição dos requisitos dos principais elementos constituintes da ferramenta, desta forma foi possível a criação do *Product Backlog*. A partir do diagrama de casos de uso, que é ilustrado na Figura 3.11 foram extraídos os requisitos para adicionar ao *Product Backlog* em forma de *user stories* e registados na ferramenta TFS (*Team Foundation Server*). Na Figura 3.12 é mostrado como exemplo a decomposição do caso de uso *U7* em *user stories*.

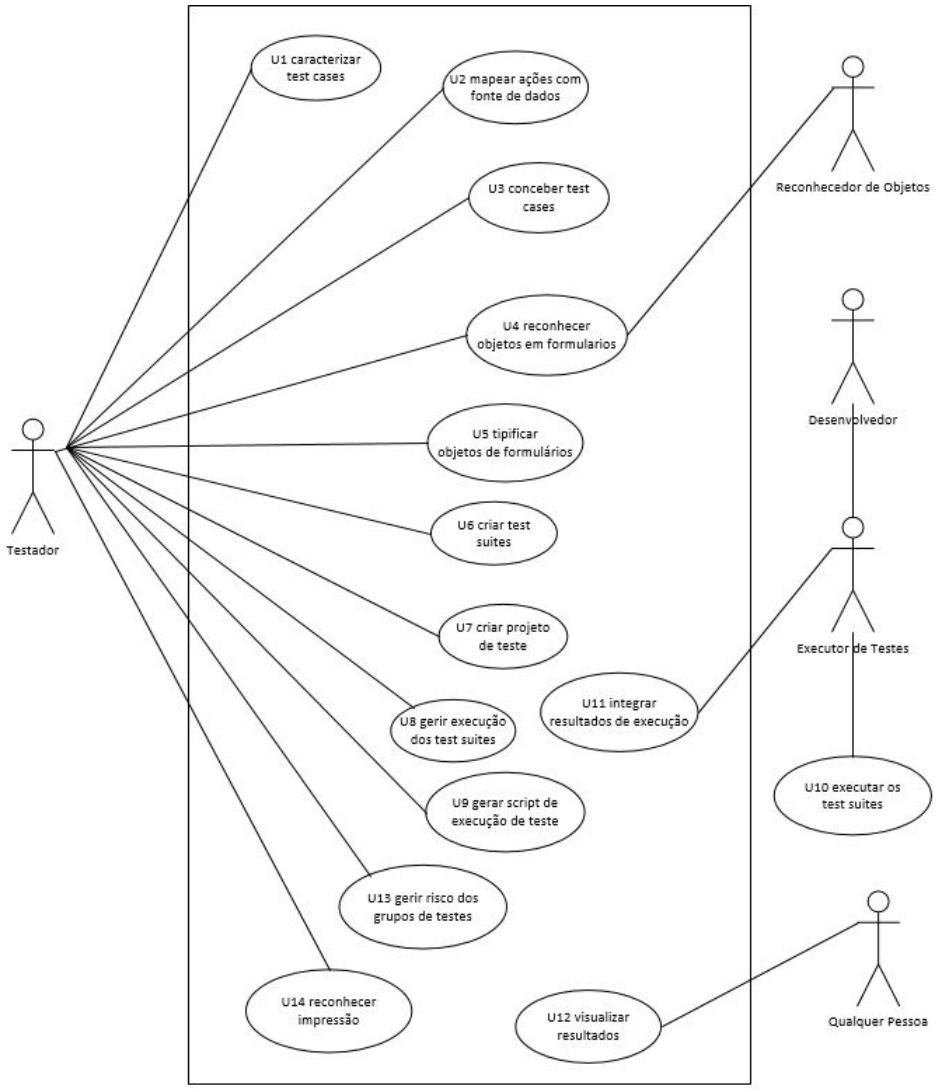

Figura 3.11: Diagrama de casos de uso do projeto CITT.

Ao longo desta fase foram realizadas 9 reuniões de levantamento de requisitos e definição da arquitetura, algumas nas instalações do CCG e outras nas instalações da PRIMAVERA BSS, que tiveram como participantes o *Scrum Master*, os dois *Product Owners* (um da WinTrust e outro da PRIMAVERA BSS) e o elemento subcontratado ao CCG.

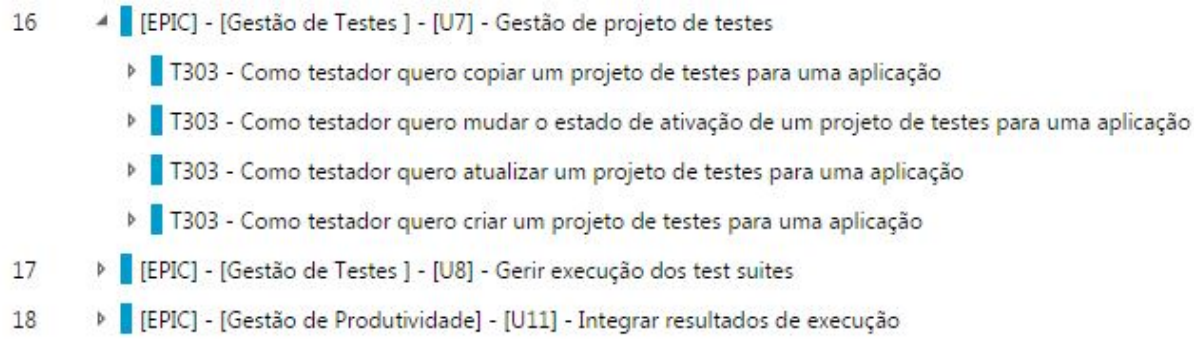

Figura 3.12: Exemplo de *user stories* do caso de uso *U7*.

Esta fase terminou com uma cerimónia *Sprint Zero*, em que foram atribuídas tarefas à Equipa de Desenvolvimento, tais como definir a estrutura da base de dados e desenvolver as funcionalidades para um MVP (Produto Minimamente Viável).

Fase de Implementação

Dentro desta fase, os *user stories* do *Product Backlog* foram implementados durante 24 *sprints*, em que cada *sprint* teve a duração de 2 semanas. Para cada *sprint* foi selecionado um subconjunto de *user stories* do *Product Backlog*, definindo assim o *Sprint Backlog*. Na Figura 3.13 está representado um exemplo do *Sprint Backlog* do *sprint* 5.

Cada *Sprint Backlog* continha um conjunto de *user stories* como referido anteriormente, assim como informação associada a cada *user story*, tais como o responsável, o estado, o esforço estimado através dos *Story Points*, entre outras. Os *Story Points* permitiram ao *Scrum Master* monitorizar o estado do desenvolvimento ao longo do projeto. Um exemplo de monitorização é o gráfico de *burndown* como ilustra a Figura 3.14.

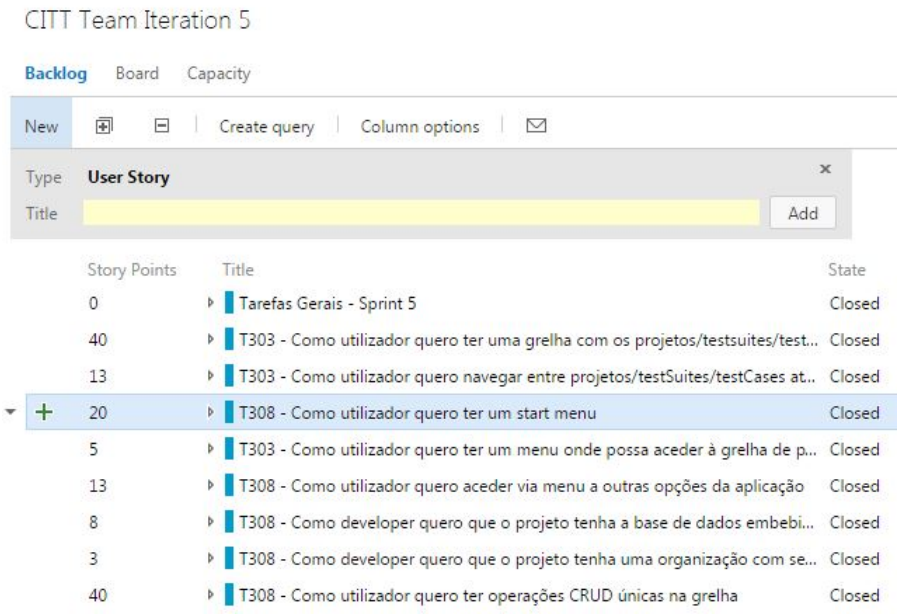

Figura 3.13: *Sprint Backlog* do *sprint* 5.

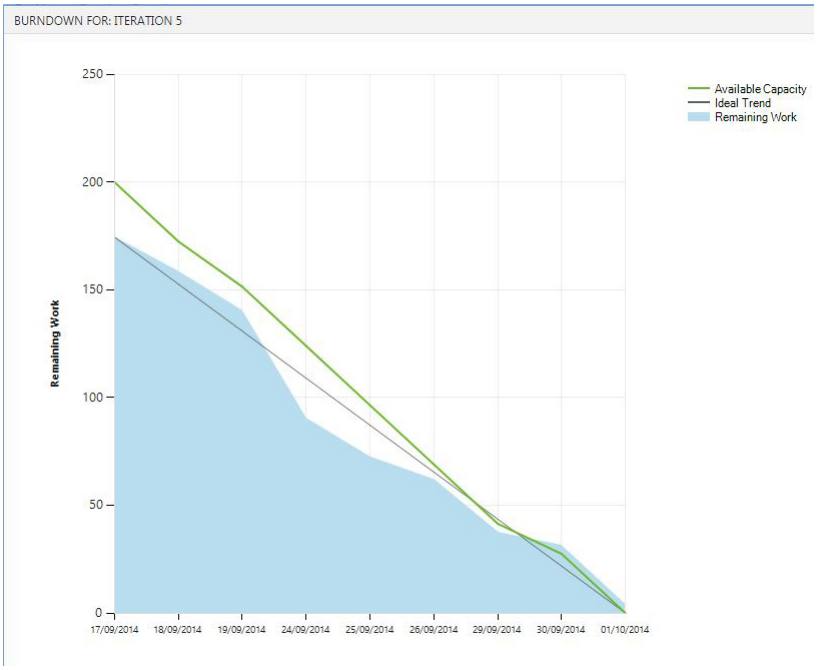

Figura 3.14: Gráfico de *burndown* do *sprint* 5.

Ao longo do projeto, foi decidido pelo *Scum Master* que de *sprint* a *sprint* cada uma das funcionalidades existentes no sistema deveria rodar por cada elemento da equipa, com o objetivo de adquirirem conhecimento tecnológico sobre todas as funcionalidades. Desta forma, conseguia-se algumas vantagens, tais como a possibilidade de um elemento ser capaz de substituir outro caso fosse necessário, os elementos não adquirirem maus hábitos, facilidade em detetar *bugs* e mais cuidados na codificação. Devido a esta decisão, foi necessário implementar em alguns *sprints* a prática *Pair Programming* do método ágil XP, conseguindo assim que elementos com competências mais especificas em relação a certas funcionalidades pudessem apoiar outros elementos.

No entanto, a partir dos *sprints* a meio da fase de implementação, o *Scrum Master* ajustou a Equipa de Desenvolvimento, optando por alocar cada elemento às funcionalidades que se enquadravam nas suas competências tecnológicas.

Foi proposto pela PRIMAVERA BSS a implementação do método Scrum, acabando por ser aceite pelas restantes organizações do consórcio:

Reunião *Product Backlog Groomings*: ocorria quando havia necessidade de refinar o *Product Backlog*. Um exemplo era quando o *Product Backlog* continha poucos *user stories* e havia a necessidade de adicionar mais. Os participantes eram o *Scrum master* (neste caso representava o papel de *Product Owner*) e a Equipa de Desenvolvimento.

Reunião *Sprint Review*: ocorria no final de cada *sprint*, sempre no mesmo dia e à mesma hora, segundas-feiras (2 em 2 semanas) às 16 horas. Os participantes eram o *Scrum Master*, pelo menos um dos *Product Owner* e a Equipa de Desenvolvimento. O objetivo desta reunião consistia em validar os *user stories* desenvolvidos durante o *sprint* e identificar possíveis novos *user stories*. Esta reunião era realizada via Skype, uma vez que as organizações se encontravam localizadas em cidades diferentes.

Reunião *Sprint Planning*: ocorria um dia após a *Sprint Review* com o objetivo principal de selecionar as *user stories* para o *Sprint Backlog* do novo *sprint*. De seguida, eram definidas tarefas para cada uma das *user stories*, assim como a atribuição da pessoa responsável por

essa tarefa e o tempo necessário (em horas) para a sua conclusão. Nos primeiros *sprints*, era difícil prever o tempo necessário para a conclusão de um requisito, porque a equipa trabalhou junta pela primeira vez e o seu ritmo de desenvolvimento era desconhecido. A partir dos primeiros *sprints*, foi possível prever o ritmo da equipa e estimar o tempo de conclusão de um requisito de forma mais eficiente. Os participantes eram o *Scrum Master* e a Equipa de Desenvolvimento.

Reunião *Daily Scrum*: ocorria todos os dias de manhã com a duração máxima de 10 minutos. Os participantes eram o *Scrum Master* e a Equipa de Desenvolvimento. O objetivo era ter um ponto de situação do trabalho realizado, na medida em que todos os participantes se comprometiam com o que seria realizado no próprio dia e tinham o conhecimento do que foi feito no dia anterior. Esta reunião por vezes era realizada via Skype.

Reunião *Sprint Retrospective*: ocorria imediatamente a seguir à *Sprint Review*. Os participantes eram o *Scrum Master* e a Equipa de Desenvolvimento. Todos os participantes podiam intervir de forma a dar a sua opinião (sobre o trabalho realizado e/ou sobre a equipa) do que correu bem ou mal no *sprint*, podendo também sugerir melhorias para os próximos *sprints*. Por exemplo, numa destas reuniões verificou-se que havia estimativas de esforço de *user stories* incorretas, pois não havia conhecimento suficiente para definir essas estimativas, então decidiram introduzir *spikes* nos próximos *sprints*. Esta reunião era realizada via Skype, devido à localização das organizações.

Ainda dentro desta fase, durante o último mês ocorreu a fase de testes, os quais foram realizados a todos os *sprints*. Após os testes aos *sprints* realizaram-se testes ao piloto.

3.3 Recomendações

Esta secção apresenta um conjunto de recomendações para a adoção de práticas ágeis no desenvolvimento de software, com base no estudo de casos utilizado neste trabalho.

As recomendações descritas nesta secção são dirigidas a organizações predispostas à adoção

de práticas ágeis, principalmente para organizações que se enquadrem num contexto idêntico a um dos casos analisados.

No entanto, aconselha-se a leitura do artigo [4], uma vez que apresenta uma estrutura (ver Figura 3.15) que orienta as organizações na adoção de práticas ágeis. Esta estrutura consiste em dois componentes: (1) o índice de medição ágil (mais especificamente a medição SAMI), e (2) um processo de quatro etapas. Como já foi referido anteriormente, o SAMI permite identificar a maturidade ágil de projetos e organizações. Por outro lado, o processo de quatro etapas ajuda a determinar se as organizações estão preparadas ou não para a adoção de práticas ágeis e identificar através da maturidade ágil das organizações, quais são as práticas ágeis que podem ser introduzidas.

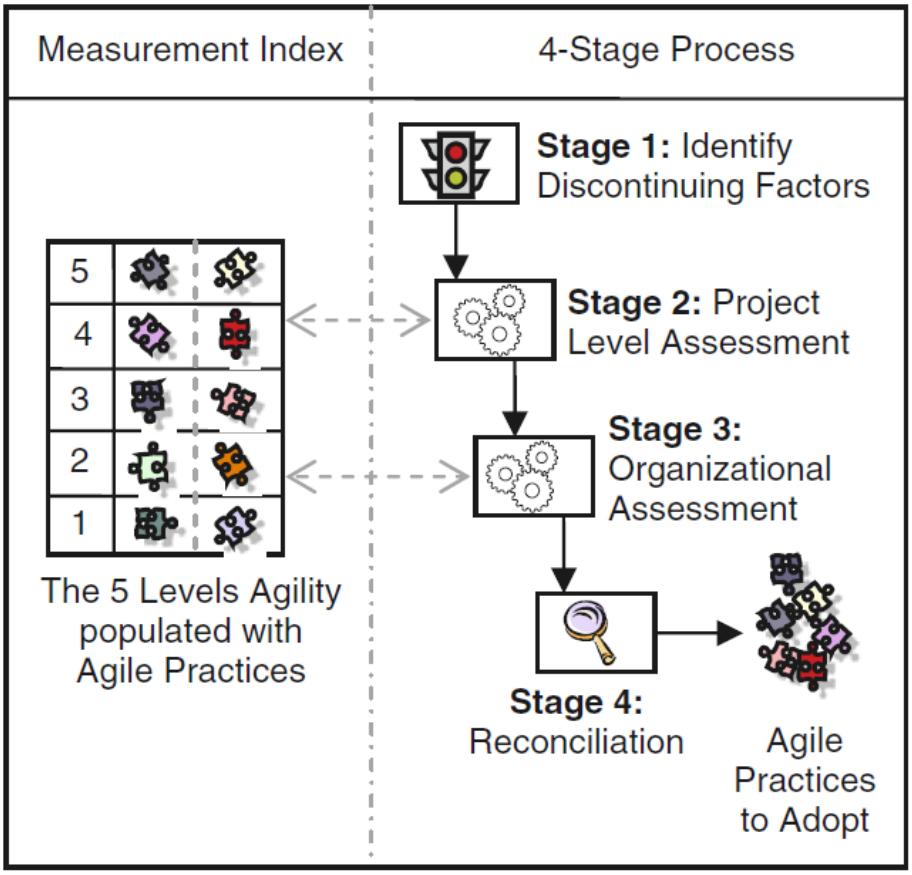

Figura 3.15: Estrutura de orientação para a adoção de práticas ágeis [4].

As recomendações serão caracterizadas pelo seguinte conjunto de propriedades:

Destinatário direto: Elemento do projeto a quem é dirigida a recomendação. Poderão existir um ou mais destinatários diretos.

Destinatário indireto: Elemento do projeto que participa na prática associada à recomendação implementada. O destinatário indireto não é obrigatório. No entanto poderão existir um ou mais destinatários indiretos.

Tipo de projeto: Esta propriedade apresenta qual o tipo de projeto a que se destina a recomendação. O projeto poderá ser do tipo simples ou complexo. No entanto, uma recomendação poderá ser direcionada para ambos os tipos de projeto. Considera-se um projeto simples quando tem um número reduzido de funcionalidades, todas elas com um grau de implementação baixo. Já um projeto complexo, poderá ter funcionalidades para as quais a equipa do projeto tem apenas conhecimento superficial do respetivo domínio, ou poderá ter um número elevado de funcionalidades independentemente do grau de conhecimento sobre o seu domínio. No tipo de projeto complexo, o risco de insucesso no desenvolvimento pode ser considerado elevado.

Condição: Esta propriedade restringe a implementação da recomendação nos casos em que a própria condição se verifica. Poderão existir uma ou mais condições. No caso de existir mais do que uma condição, a verificação de pelo menos uma delas é suficiente para implementar a recomendação.

Recomendação: Aqui é feita a descrição da recomendação.

Benefícios: Esta propriedade apresenta os benefícios dos valores e princípios ágeis que a implementação da recomendação irá fomentar no contexto do projeto.

Motivação: Esta propriedade permite perceber com base em que argumentos foi feita a caraterização da recomendação.

De seguida, serão apresentadas as recomendações caracterizadas através da análise do estudo de casos apresentado. Caso a maturidade ágil de uma organização não seja suficientemente alta para permitir a implementação de alguma das recomendações, a mesma deve decidir se pretende ou não efetuar as mudanças e melhorias necessárias para que seja possível adotar as práticas recomendadas, com a finalidade de obter benefícios na adoção dessas práticas.

3.3.1 Partilhar o conhecimento sobre os valores e princípios ágeis

Recomendação 1 - Partilhar o conhecimento sobre os valores e princípios ágeis	
Destinatário direto: • Equipa do projeto	**Tipo de projeto:** • Simples • Complexo
Condição: • Pelo menos um elemento da equipa do projeto não tem conhecimento sobre os valores e princípios ágeis no desenvolvimento de software.	
Recomendação	
Partilha de conhecimento sobre os valores e princípios ágeis entre os elementos da equipa do projeto, sempre que pelo menos um dos elementos dessa equipa não esteja com eles familiarizado no contexto de desenvolvimento de software. Esta partilha de conhecimento deverá ser realizada pelos elementos da equipa que defendam e que possuam conhecimentos sólidos sobre os valores e princípios ágeis. Aconselha-se a incorporação de pelo menos um elemento com esse conhecimento, caso não exista ninguém dentro da equipa do projeto com esse conhecimento. A partilha desse conhecimento deverá ser realizada antes do início do projeto.	
Benefício: • Partilha de conhecimento	
Motivação: Com base nos dados analisados do projeto iFlow, verificou-se que os elementos da Bosch não tinham conhecimento sobre os valores e princípios ágeis no desenvolvimento de software. Para colmatar este facto, os coordenadores de I&DT da UMinho que possuíam o conhecimento relevante sobre os valores e princípios ágeis ficaram responsáveis por partilhar esse conhecimento.	

3.3.2 Aceitar a adoção de práticas ágeis propostas por parceiros tecnológicos

<table>
<tr><th colspan="2">Recomendação 2 - Aceitar a adoção de práticas ágeis propostas por parceiros tecnológicos</th></tr>
<tr><td>Destinatários diretos:
Responsável pelo desenvolvimento
Equipa de Desenvolvimento</td><td>Tipo de projeto:
Simples
Complexo</td></tr>
<tr><td colspan="2">Condições:
Um elemento pertencente a uma organização parceira num projeto propõe a adoção de práticas ágeis;
Existirem evidências de benefícios de projetos passados.</td></tr>
<tr><td colspan="2">Recomendação</td></tr>
<tr><td colspan="2">Aceitar a adoção de práticas ágeis propostas por parceiros tecnológicos do projeto, sempre que essas práticas demonstrem benefícios, com base em indicadores de projetos anteriores realizados em contexto semelhante ao atual. Idealmente, este tipo de proposta deverá ser realizada antes do arranque do projeto.</td></tr>
<tr><td colspan="2">Benefícios:
Cliente mais envolvido no processo de desenvolvimento
Eventual aumento da satisfação do cliente</td></tr>
<tr><td colspan="2">Motivação: Com base nos dados analisados do projeto CITT, verificou-se que a PRIMAVERA BSS propôs a adoção do método Scrum (contém um conjunto de práticas ágeis) às restantes organizações, que acabaram por aceitar a proposta com base na experiência e benefícios apresentados.
As organizações que proponham a adoção de práticas ágeis no desenvolvimento de software devem garantir e reconhecer os benefícios obtidos com esta adoção.</td></tr>
</table>

3.3.3 Treinar práticas ágeis

<table>
<tr><th colspan="2">Recomendação 3 - Treinar práticas ágeis</th></tr>
<tr><td>Destinatário direto:
Responsável pelo desenvolvimento
Destinatário indireto:
Equipa de Desenvolvimento</td><td>Tipo de projeto:
Complexo</td></tr>
<tr><td colspan="2">Condição:
Equipa de Desenvolvimento não tem experiência com as práticas ágeis que fazem parte do método ágil que o responsável pelo desenvolvimento selecionou.</td></tr>
<tr><td colspan="2">Recomendação</td></tr>
<tr><td colspan="2">Treinar o método ágil que o responsável pelo desenvolvimento selecionou para o processo de desenvolvimento do projeto. Antes do projeto arrancar a Equipa de Desenvolvimento deverá ter um período de treino para se familiarizar com as práticas ágeis que serão adotadas. Para isso deverá participar no desenvolvimento de um protótipo de um projeto simples, durante 1 ou 2 iterações.</td></tr>
<tr><td colspan="2">Benefícios:
Partilha de conhecimento
Maior confiança nas práticas ágeis
Perceção das vantagens e desvantagens de cada prática ágil
Perceber o contexto em que as práticas ágeis podem ser utilizadas</td></tr>
<tr><td colspan="2">Motivação: Com base na literatura e nos dados analisados (principalmente numa entrevista realizada a um dos elementos da Equipa de Desenvolvimento) do projeto CITT, revelou-se importante a necessidade de Equipas de Desenvolvimento que não têm experiência com as práticas ágeis no desenvolvimento de software, realizem participações em iterações de outros projetos simples, antes do arranque do seu projeto.</td></tr>
</table>

3.3.4 Utilizar ferramentas de gestão ágil

<table>
<tr><th colspan="2">Recomendação 4 - Utilizar ferramentas de gestão ágil</th></tr>
<tr><td>Destinatário direto:
Responsável pelo projeto
Destinatário indireto:
Equipa do projeto</td><td>Tipo de projeto:
Simples
Complexo</td></tr>
<tr><td colspan="2">Condição:
Não é utilizada nenhuma ferramenta de gestão ágil ou é utilizada uma que não é considerada adequada.</td></tr>
<tr><td colspan="2">Recomendação</td></tr>
<tr><td colspan="2">Utilização de ferramentas direcionadas para a gestão ágil no desenvolvimento de software. Para isso deverá primeiramente verificar, se a organização já possui alguma ferramenta com essas caraterísticas. Caso contrário, deverá ser disponibilizada uma ferramenta para esses propósitos.</td></tr>
<tr><td colspan="2">Benefícios:
Organização do projeto
Monitorização do projeto</td></tr>
<tr><td colspan="2">Motivação: Com base nos dados analisados do projeto CITT, verificou-se que as ferramentas de gestão ágil (neste caso o TFS) disponibilizam um conjunto de caraterísticas que permitem adotar práticas ágeis com um grau de dificuldade baixo, no processo de desenvolvimento de software.</td></tr>
</table>

3.3.5 Utilizar ferramentas de comunicação

Recomendação 5 - Utilizar ferramentas de comunicação	
Destinatários direto: • Responsável pelo projeto **Destinatários indireto:** • Equipa do projeto	**Tipo de projeto:** • Simples • Complexo
Condição: • Pelo menos um elemento da equipa do projeto não tem o seu posto de trabalho nas mesmas instalações em que se encontra a restante equipa, quer seja de modo temporário ou mesmo definitivo.	
Recomendação	
Utilização de ferramentas de comunicação, tendo como principal caraterística a comunicação síncrona, como por exemplo, o Skype e o telefone. Os elementos da equipa do projeto que têm o seu posto de trabalho nas mesmas instalações deverão reunir-se presencialmente, comunicando através da ferramenta escolhida apenas com o(s) elemento(s) cujo posto de trabalho se encontra noutras instalações.	
Benefício: • Comunicação frequente	
Motivação: Com base nos dados analisados do projeto CITT, verificou-se que devido à localização em diferentes instalações do posto de trabalho de um dos elementos da Equipa de Desenvolvimento, houve a necessidade de utilizar estas ferramentas de modo a garantir a comunicação. A equipa seguia o método ágil Scrum e o uso das ferramentas de comunicação foi necessário apenas nas Reuniões *Daily Scrum*, Reuniões *Sprint Review* e Reuniões *Sprint Retrospective*. No entanto, estas ferramentas podem ser utilizadas em qualquer situação que seja necessário a comunicação entre quaisquer elemento que constitua a equipa.	

3.3.6 Avaliar o ritmo da Equipa de Desenvolvimento

Recomendação 6 - Avaliar o ritmo da Equipa de Desenvolvimento	
Destinatário direto: • Responsável pelo desenvolvimento **Destinatário indireto:** • Equipa de Desenvolvimento	**Tipo de projeto:** • Simples • Complexo
Condições: • Equipa de Desenvolvimento trabalha em conjunto pela primeira vez; • Necessidade de estimar o mais eficientemente possível o esforço para o desenvolvimento de requisitos.	
Recomendação	
Avaliação do ritmo da Equipa de Desenvolvimento nas primeiras iterações. Deverá ser registado o tempo real que a equipa demorou para concluir os requisitos que foram desenvolvidos ao longo das primeiras iterações. Consideram-se as primeiras iterações todas aquelas em que a estimativa e o desenvolvimento dos requisitos que foram realizados, tenham resultados muito díspares. Após essas iterações a Equipa de Desenvolvimento deverá ser eficiente na estimativa do esforço para o desenvolvimento de cada requisito.	
Benefício: • Minimização do risco	
Motivação: Com base nos dados analisados do projeto CITT, verificou-se que as estimativas de esforço nas primeiras iterações (*sprints*) eram difíceis de prever porque foi a primeira vez que a equipa trabalhou em conjunto e o seu ritmo de desenvolvimento era desconhecido. Através da informação recolhida sobre a equipa nas primeiras iterações, foi possível, a partir desse momento, estimar, o esforço para a conclusão de cada requisito.	

3.3.7 Realizar reuniões frequentes

<table>
<tr><th colspan="2">Recomendação 7 - Realizar reuniões frequentes</th></tr>
<tr><td>Destinatário direto:
Equipa do projeto</td><td>Tipo de projeto:
Simples
Complexo</td></tr>
<tr><td colspan="2">Condições:
O cliente não sabe bem quais são as suas necessidades e o cliente tem disponibilidade para reunir com a frequência prevista;
As necessidades do cliente precisam de ser esclarecidas e o cliente tem disponibilidade para reunir com a frequência prevista.</td></tr>
<tr><td colspan="2">Recomendação</td></tr>
<tr><td colspan="2">Realização de reuniões frequentes com o cliente e se possível com toda a equipa do respetivo projeto. Estas reuniões deverão ser realizadas em curtos espaços temporais, idealmente entre 1 a 4 semanas. O objetivo principal destas reuniões é obter feedback do cliente.</td></tr>
<tr><td colspan="2">Benefícios:
Feedback constante
Satisfação do cliente</td></tr>
<tr><td colspan="2">Motivação: Com base nos dois projetos e na leitura realizada para este trabalho surge a caracterização desta recomendação. Salienta-se a importância do contacto pessoal com o cliente, de modo a garantir o feedback constante, permitindo assim reajustar as suas necessidades.</td></tr>
</table>

3.3.8 Utilizar o *Sprint Zero*

<table>
<tr><th colspan="2">Recomendação 8 - Utilizar o Sprint Zero</th></tr>
<tr><td>Destinatário direto:
Responsável pelo projeto
Destinatário indireto:
Equipa do projeto</td><td>Tipo de projeto:
Complexo</td></tr>
<tr><td colspan="2">Condição:
Requisitos do projeto não estão definidos pelo dono do produto.</td></tr>
<tr><td colspan="2">Recomendação</td></tr>
<tr><td colspan="2">Utilização do Sprint Zero. O Sprint Zero consiste numa primeira iteração antes do arranque das iterações da fase de implementação, que permitirá à equipa do projeto identificar e refinar os requisitos iniciais do projeto. Para a realização desta iteração deverão estar presentes os elementos da equipa do projeto capazes de identificar os requisitos.</td></tr>
<tr><td colspan="2">Benefício:
Minimização do risco</td></tr>
<tr><td colspan="2">Motivação: Com base nos dados analisados dos dois projetos (CITT e iFlow), verificou-se a utilização do Sprint Zero, com o objetivo de identificar os principais requisitos do Product Backlog, definir a arquitetura do projeto e atribuir tarefas à Equipa de Desenvolvimento para as funcionalidades de um MVP.
Apesar do Sprint Zero não ser muito referenciado na literatura, tem vindo a ser adotado pelas organizações ao longo do tempo, de modo a colmatar a necessidade de identificar os principais requisitos do projeto.</td></tr>
</table>

3.3.9 Utilizar a técnica *spike*

<table>
<tr><th colspan="2">Recomendação 9 - Utilizar a técnica spike</th></tr>
<tr><td>Destinatário direto:
Responsável pelo desenvolvimento
Destinatário indireto:
Equipa de Desenvolvimento</td><td>Tipo de projeto:
Complexo</td></tr>
<tr><td colspan="2">Condições:
Equipa de Desenvolvimento tem dificuldades em estimar o esforço para o desenvolvimento de pelo menos um dos requisitos de um domínio, sobre o qual não possuem um elevado grau de conhecimento;
Necessidade de estimar o esforço para o desenvolvimento de requisitos o mais eficiente possível.</td></tr>
<tr><td colspan="2">Recomendação</td></tr>
<tr><td colspan="2">Utilização da spike, técnica pertencente ao método XP. Esta spike deverá ocorrer quando a Equipa de Desenvolvimento se deparar com pelo menos um requisito, de um domínio sobre o qual não têm conhecimento suficiente. A spike poderá ocorrer a qualquer altura da fase de implementação.</td></tr>
<tr><td colspan="2">Benefício:
Minimização do risco</td></tr>
<tr><td colspan="2">Motivação: Com base nos dados analisados dos dois projetos (CITT e iFlow), verificou-se que o conhecimento da Equipa de Desenvolvimento sobre o domínio de alguns dos requisitos era insuficiente, tendo sido necessário utilizar a técnica spike para realizar estimativas de esforço mais eficientes.</td></tr>
</table>

3.3.10 Utilizar a prática *Pair Programming*

Recomendação 10 - Utilizar a prática *Pair Programming*	
Destinatário direto: • Responsável pelo desenvolvimento • Equipa de Desenvolvimento	**Tipo de projeto:** • Complexo
Condições: • Pelo menos um elemento da Equipa de Desenvolvimento tem conhecimento técnico mais avançado em relação à restante equipa; • Pelo menos um elemento não está a conseguir acompanhar a restante equipa a nível de conhecimento técnico.	
Recomendação	
Utilização da prática *Pair Programming*. Esta prática deverá ser realizada por um elemento da Equipa de Desenvolvimento com um nível de conhecimento técnico avançado e outro elemento que não esteja a conseguir acompanhar a Equipa de Desenvolvimento. A prática *Pair Programming* poderá ser utilizada em qualquer altura na fase de implementação, de modo a garantir que a equipa tenha o mesmo nível de conhecimento técnico.	
Benefícios: • Partilha de conhecimento • Desenvolvimento colaborativo	
Motivação: Com base nos dados analisados do projeto CITT, verificou-se que houve a necessidade de toda a Equipa de Desenvolvimento estar no mesmo nível de conhecimento técnico. A prática *Pair Programming* permitiu que os elementos da equipa com mais experiência, partilhassem o seu conhecimento com os elementos com menor conhecimentos técnicos. Segundo a literatura, esta prática consiste em dois desenvolvedores trabalharem no mesmo computador, tendo como principal vantagem a revisão constante do código. Nesta situação esta prática foi utilizada para a partilha de conhecimento.	

3.3.11 Rodar os elementos da Equipa de Desenvolvimento

Recomendação 11 - Rodar os elementos da Equipa de Desenvolvimento	
Destinatário direto: • Responsável pelo desenvolvimento **Destinatário indireto:** • Equipa de Desenvolvimento	**Tipo de projeto:** • Simples • Complexo
Condição: • Os elementos da Equipa de Desenvolvimento precisam de obter conhecimento técnico sobre todas ou quase todas as funcionalidades existentes no sistema a desenvolver.	
Recomendação	
Rotatividade de todos os elementos da Equipa de Desenvolvimento a cada iteração. Durante a iteração os elementos da equipa deverão ser alocados a tarefas pertencentes a pelo menos uma funcionalidade, sendo que na próxima iteração deverão ser alocados a tarefas pertencentes a outra(s) funcionalidade(s), mesmo que as tarefas das funcionalidades anteriores não estejam concluídas.	
Benefícios: • Partilha de conhecimento • Desenvolvimento colaborativo	
Motivação: Com base nos dados analisados do projeto CITT, verificou-se que nos primeiros *sprints* (iterações) da fase de implementação houve rotação entre todos os elementos da Equipa de Desenvolvimento, com o objetivo principal de estes adquirirem conhecimento tecnológico sobre todas as funcionalidades do sistema. Esta rotação, para além do conhecimento tornou a Equipa de Desenvolvimento mais ágil e colaborativa.	

3.4 Conclusão

Este capítulo teve como objetivo contribuir para a adoção de práticas ágeis no desenvolvimento de software, através da caraterização de um conjunto de recomendações baseadas em evidências encontradas no estudo de dois casos, o projeto iFlow e o projeto CITT.

Através da análise dos dados recolhidos no estudo de casos, verificou-se que a adoção de práticas ágeis no processo de desenvolvimento de software trouxe benefícios, tais como a comunicação constante, a satisfação do cliente, a qualidade do software, entre outros. No caso do projeto iFlow houve a necessidade de realizar um mapeamento de papéis do projeto iFlow para os papéis do método ágil Scrum, conseguindo assim ter o aproveitamento dos princípios e valores das práticas ágeis no desenvolvimento de software.

Apesar de ambos os projetos seguirem o método ágil Scrum também foram adotadas práticas ágeis que não fazem parte desse método, foi o caso do *Pair Programming* (no projeto CITT) e do *Refactoring* (no projeto iFlow), as quais pertencem ao método XP. Nesse sentido as organizações adotaram práticas ágeis que mais se adequavam ao projeto a desenvolver, com a finalidade de obter benefícios com essa adoção. No entanto as organizações devem ter uma maturidade ágil que permita a adoção de práticas ágeis, caso contrário, deverão decidir se pretendem ou não implementar as melhorias necessárias para obter essa maturidade, de modo a serem capazes de adotar essas práticas com sucesso.

As recomendações que foram caraterizadas neste capítulo pretendem incentivar as organizações na adoção de determinadas práticas ágeis, perante as suas necessidades no desenvolvimento de software, de modo a que obtenham benefícios na sua incorporação no seu processo de desenvolvimento.

Capítulo 4

Validação do Trabalho

Neste capítulo é apresentada a validação deste trabalho. A validação foi realizada através da implementação num projeto real, de algumas das recomendações caraterizadas no capítulo anterior. As recomendações foram selecionadas tendo em conta determinadas condições encontradas no projeto e compatíveis com as condições necessárias para implementar essas recomendações.

4.1 Desenvolvimento de Timeline e do Dashboard Analítico

4.1.1 Contextualização

O projeto Desenvolvimento de *Timeline* e do *Dashboard* Analítico (DTDA) foi desenvolvido pela organização CCG. O projeto teve a duração de 6 meses, sendo o seu principal objetivo o desenvolvimento de dois componente, o *Dashboard* Analítico e a *Timeline*.

O componente do *Dashboard* Analítico consistia na implementação de uma plataforma de visualização de informação composta pelos seguintes módulos: (1) tabelas, (2) calendários, (3) horários e (4) gráficos simples.

O componente *Timeline* consistia no desenvolvimento de uma aplicação interativa, que apresenta um conjunto de eventos dispostos por ordem cronológica com os quais o utilizador poderá interagir.

A *Timeline* dispõe das seguintes funcionalidades:

- Funcionalidades de integração
  - Capacidade de definir intervalo temporal a apresentar;
  - Capacidade de carregar os itens temporais de várias fontes, em particular: (1) *json*, *Dataset* e (2) itens estáticos no HTML;
  - Capacidade de definir o *template* de estilos (css) a aplicar.
- Funcionalidades de interação com o utilizador
  - Capacidade de mostrar itens temporais com vários tipos de duração temporal (itens pontuais, itens que ocupam um intervalo de tempo);
  - Capacidades de *zooming*, e navegação na *Timeline*;
  - Capacidade de visualização detalhada dos itens da *Timeline*. A visualização é realizada através de uma janela *popup* com a informação detalhada do evento, *link* para o conteúdo;
  - Possibilidade de criar itens na *Timeline*.

Numa primeira abordagem ao projeto foi realizada a análise de requisitos. Aqui foram levantadas as necessidades do cliente, e posteriormente essas necessidades foram sistematizadas através de diagramas UML, o que incluiu a utilização de diagramas de várias naturezas, como casos de uso, atividades, sequências, estados ou classes. A Figura 4.1 ilustra um dos diagramas UML criados, neste caso, representando o diagrama geral dos casos de uso do componente *Timeline*.

Foram também definidos um conjunto de protótipos e mockups, que permitiram uma primeira apresentação visual do sistema.

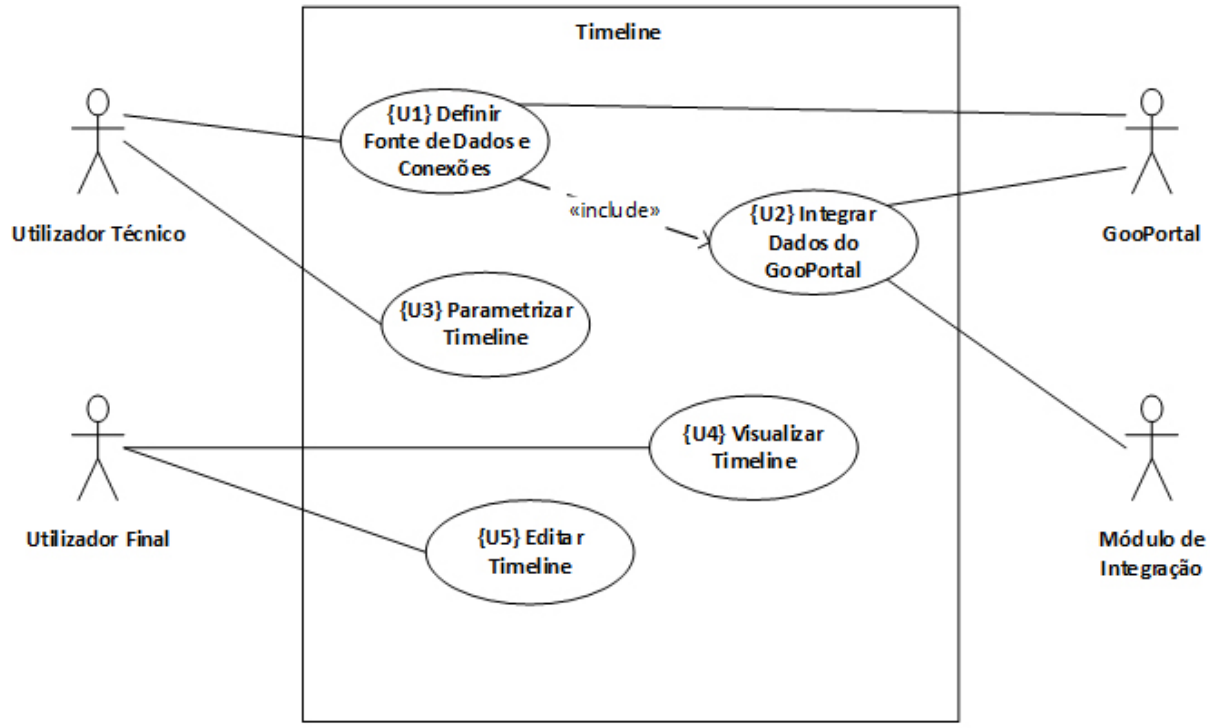

Figura 4.1: Diagrama geral dos casos de uso do componente *Timeline*.

A equipa principal do projeto DTDA foi composta por 4 elementos, pertencentes à organização CCG, para o desenvolvimento do componente *Dashboard* Analítico e posteriormente foi incorporado mais um elemento, também da organização CCG, para o desenvolvimento do componente *Timeline*. Os elementos da equipa adotaram os seguintes papéis:

Gestor de Projeto: um elemento responsável pela organização do projeto, bem como pela interação com o cliente, sempre que fosse necessário;

Equipa de Desenvolvimento: três elementos responsáveis pelo desenvolvimento do componente *Dashboard* Analítico. Posteriormente foi incorporado mais um elemento no desenvolvimento do componente *Timeline*. De salientar a minha participação neste projeto como o elemento incorporado para o desenvolvimento do componente *Timeline*.

4.1.2 Recomendações Implementadas

O CCG é uma organização que normalmente utiliza o modelo em cascata no processo de desenvolvimento. No entanto para o projeto DTDA sugeri a implementação das recomendações caraterizadas, caso se verificasse que as condições apresentadas pelas respetivas recomendações eram cumpridas. Esta sugestão teve como finalidade a validação de algumas das recomendações caraterizadas neste trabalho.

A sugestão foi aceite, e foram verificadas as condições para implementar cinco recomendações:

Partilhar o conhecimento sobre os valores e princípios ágeis

Recomendação 1 - Partilhar o conhecimento sobre os valores e princípios ágeis
Condição que se verificou: • Pelo menos um elemento da equipa do projeto não tem conhecimento sobre os valores e princípios ágeis no desenvolvimento de software.
Implementação da recomendação: No projeto DTDA, um dos elementos da equipa fez parte da Equipa de Desenvolvimento do projeto CITT, utilizado como estudo de caso neste trabalho. Através da envolvência no projeto CITT, esse elemento adquiriu conhecimentos sólidos sobre os valores e princípios ágeis no desenvolvimento de software, ficando assim responsável, antes do arranque do projeto DTDA, pela partilha desse conhecimento à restante equipa que não detinha esse conhecimento.
Resultado: Toda a equipa adquiriu conhecimentos sólidos sobre os valores e princípios ágeis no desenvolvimento de software.

Depois desta recomendação (Recomendação 1), o gestor de projeto decidiu adotar algumas das práticas do método ágil Scrum: (1) *Product Backlog*, (2) *Sprint* e (3) Reunião *Sprint Review*.

Utilizar ferramentas de gestão ágil

Recomendação 4 - Utilizar ferramentas de gestão ágil
Condição que se verificou: • Não é utilizada nenhuma ferramenta de gestão ágil ou é utilizada uma que não é considerada adequada.
Implementação da recomendação: A organização CCG já tinha instalada uma ferramenta, o Redmine, a qual foi utilizada para o projeto DTDA. Como foram adotadas algumas práticas do método ágil Scrum, houve a necessidade de instalar um *plugin* adicional na ferramenta Redmine orientado para o Scrum.
Resultado: Facilitou a adoção das práticas ágeis adotadas devido às suas caraterísticas direcionadas para a gestão ágil. No entanto, apesar de facilitar a adoção dessas práticas, esta ferramenta não foi uma experiência muito enriquecedora devido à sua interface bastante confusa e às limitações do *plugin*. Devido a este facto, a organização está em processo de migração para outra ferramenta de gestão ágil, que consideram mais adequada.

Utilizar ferramentas de comunicação

Recomendação 5 - Utilizar ferramentas de comunicação
Condição que se verificou: • Pelo menos um elemento da equipa do projeto não tem o seu posto de trabalho nas mesmas instalações em que se encontra a restante equipa, quer seja de modo temporário ou mesmo definitivo.
Implementação da recomendação: Um elemento da Equipa de Desenvolvimento não tinha o seu posto de trabalho nas mesmas instalações (CCG) da restante equipa, sendo assim disponibilizado o Skype como ferramenta de comunicação.
Resultado: Foram realizadas várias reuniões durante o processo de execução do projeto. Nestas reuniões participavam todos os elementos no projeto. Esta ferramenta foi muito importante, na medida em que foi possível a comunicação entre a equipa que se encontrava nas instalações do CCG e o elemento da Equipa de Desenvolvimento que se encontrava fora das instalações. Em algumas das reuniões o cliente também esteve presente, deslocando-se até às instalações do CCG.

Realizar reuniões frequentes

Recomendação 7 - Realizar reuniões frequentes
Condição que se verificou: • As necessidades do cliente precisam de ser esclarecidas e o cliente tem disponibilidade para reunir com a frequência prevista.
Implementação da recomendação: Como o cliente se mostrou disponível para a realização de reuniões frequentes, foram realizadas reuniões de 2 em 2 semanas, com a participação do cliente e de toda a equipa do projeto.
Resultado: Devido ao contacto frequente com o cliente, foi possível garantir o feedback constante, de modo a reajustar as suas necessidades.

Utilizar a prática *Pair Programming*

Recomendação 10 - Utilizar a prática *Pair Programming*
Condição que se verificou: • Pelo menos um elemento não está a conseguir acompanhar a restante equipa a nível de conhecimento técnico.
Implementação da recomendação: Na fase de implementação do componente *Timeline* (realizada após implementação do componente *Dashbord* Analítico), fui incorporado na Equipa de Desenvolvimento. Como um dos elementos da Equipa de Desenvolvimento já tinha um nível de conhecimento técnico avançado, ficou com a responsabilidade de partilhar esse conhecimento comigo. O conhecimento desse elemento foi adquirido devido à sua experiência no desenvolvimento do componente *Dashboard* Analítico.
Resultado: Devido à partilha de conhecimento, a Equipa de Desenvolvimento ficou com o mesmo nível de conhecimento técnico, de modo a progredir na implementação da *Timeline* de forma constante. Esta recomendação também foi enriquecedora na medida em que teve o beneficio do desenvolvimento colaborativo.

4.2 Conclusão

Com a implementação das cinco recomendações no projeto DTDA, verificou-se a existência de benefícios, tais como a satisfação do cliente, qualidade no projeto (em termos de resultados esperados) e o desenvolvimento colaborativo. Foi considerado um projeto com sucesso, na medida em que todas as necessidades do cliente foram implementadas dentro do prazo e orçamento.

Capítulo 5

Conclusões e Trabalho Futuro

Neste capítulo, são apresentadas as conclusões finais do trabalho, com foco nas contribuições e trabalho futuro.

5.1 Conclusões

O objetivo deste trabalho consistiu em contribuir com um conjunto de recomendações para facilitar a adoção de práticas ágeis no âmbito de organizações que desenvolvem software.

Através da análise de dois projetos utilizados como estudo de casos foi possível perceber como foram adotadas as práticas ágeis no desenvolvimento de software, e em que condições é que cada organização sentiu necessidade de adoção dessas práticas.

A análise do estudo de casos permitiu a caraterização de um conjunto de recomendações, assim como a caraterização dos benefícios da sua implementação, que poderão ser utilizadas em determinadas condições ou necessidades identificadas no desenvolvimento de software.

As organizações não têm a necessidade de adotar um método ágil por completo, podem adotar as práticas ágeis que mais se adequam ao projeto a desenvolver. No entanto as organizações devem ter uma maturidade ágil que permita a adoção de práticas ágeis, caso contrário deverão decidir se pretendem ou não implementar as melhorias necessárias para obter essa maturidade de modo a serem capazes de adotar essas práticas com sucesso.

5.2 Perspetivas de Trabalho Futuro

Na realização deste trabalho, foram identificadas algumas possibilidades de trabalho futuro:

- Identificar mais recomendações através do estudo de outros casos;
- Validar a implementação das recomendações em diferentes organizações.

Glossário

método Carateriza o processo com um conjunto de atividades utilizado no desenvolvimento de software.

método ágil Processo de gestão e desenvolvimento de software que usa uma abordagem de planeamento e execução iterativa e incremental.

processo Sequência de atividades que deverão ser realizadas com o objetivo de atingir um determinado resultado.

requisito Capacidade que um sistema deve possuir para satisfazer as necessidades dos utilizadores.

risco Circunstância potencialmente adversa que pode ter efeitos negativos ou perversos no processo de desenvolvimento e na qualidade final do sistema.

Referências Bibliográficas

[1] M. Qureshi, "Empirical evaluation of the proposed exscrum model: Results of a case study," *arXiv preprint arXiv:1202.2513*, 2012.

[2] N. Santos, J. M. Fernandes, M. S. Carvalho, P. V. Silva, A. Fernandes, M. P. Rebelo, D. Barbosa, P. Maia, M. Couto, and R. J. Machado, "Using scrum together with uml models: A collaborative university-industry r&d software project." 2011.

[3] J. Hunt, "Agile software construction," 2006.

[4] A. Sidky, J. Arthur, and S. Bohner, "A disciplined approach to adopting agile practices: the agile adoption framework," *Innovations in systems and software engineering*, vol. 3, no. 3, pp. 203–216, 2007.

[5] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," *Communications of the ACM*, vol. 48, no. 5, pp. 72–78, 2005.

[6] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[7] `http://www.oxagile.com/company/blog/the-waterfall-model/`, 2014. Acedido em 20 de Dezembro de 2015.

[8] I. Sommerville, *Software engineering - Ninth Edition*, pp. 30–32. Addison-Wesley, 2011.

[9] M. S. Soares, "Comparação entre metodologias ágeis e tradicionais para o desenvolvimento de software," *INFOCOMP Journal of Computer Science*, vol. 3, no. 2, pp. 8–13, 2004.

[10] C. P. Coutinho, *Metodologia de investigação em ciências sociais e humanas*. Leya, 2014.

[11] J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2013.

[12] R. K. Yin, "Case study research: Design and methods . thousands oaks," *International Educational and Professional Publisher*, 1994.

[13] M. V. Zelkowitz and D. R. Wallace, "Experimental models for validating technology," *Computer*, vol. 31, no. 5, pp. 23–31, 1998.

[14] E. G. Guba, Y. S. Lincoln, *et al.*, "Competing paradigms in qualitative research," *Handbook of qualitative research*, vol. 2, no. 163-194, 1994.

[15] A. C. Gil, "Como elaborar projetos de pesquisa. são paulo: Atlas, 1996," *Métodos e técnicas de pesquisa social*, vol. 5, 1995.

[16] W. W. Royce, "Managing the development of large software systems," in *proceedings of IEEE WESCON*, vol. 26, pp. 1–9, Los Angeles, 1970.

[17] F. Bomarius, M. Oivo, P. Jaring, and P. Abrahamsson, *Product-Focused Software Process Improvement, 10th International Conference, PROFES 2009, Oulu, Finland, June 15-17, 2009. Proceedings*, vol. 32 of *Lecture Notes in Business Information Processing*. Springer, 2009.

[18] T. J. Lehman and A. Sharma, "Software development as a service: agile experiences," in *SRII Global Conference (SRII), 2011 Annual*, pp. 749–758, IEEE, 2011.

[19] D. Brandl, "Agile software development. control engineering," vol. 56, p. 18, Julho 2009. `http://www.controleng.com/single-article/agile-software-development/d4fa37caa2001f03419164f469eb1908.html`.

[20] J. Sutherland, "Agile development: Lessons learned from the first scrum," *Cutter Agile Project Management Advisory Service: Executive Update*, vol. 5, no. 20, pp. 1–4, 2004.

[21] J. M. Fernandes and R. J. Machado, *Requirements in engineering projects*. Springer, 2016.

[22] A. Manifesto. `http://agilemanifesto.org`, 2001. Acedido em 12 de Novembro de 2015.

[23] B. Boehm, "A view of 20th and 21st century software engineering," in *Proceedings of the 28th international conference on Software engineering*, pp. 12–29, ACM, 2006.

[24] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213–1221, 2012.

[25] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods: Review and analysis*, pp. 18–34. VTT Electronics, 2002.

[26] K. Schwaber and J. Sutherland, "O guia do scrum." `http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese_European.pdf`, 2011. [Acedido em 21 de Março de 2016].

[27] N. Ganesh and S. Thangasamy, "Issues identified in the software process due to barriers found during eliciting requirements on agile software projects: Insights from india," *Issues*, vol. 16, no. 5, 2011.

[28] K. Schwaber and J. Sutherland, "The scrum guide. scrum. org. 25.09. 2013," 2013.

[29] H. Zhi-gen, Y. Quan, and Z. Xi, "Research on agile project management with scrum method," in *Services Science, Management and Engineering, 2009. SSME'09. IITA International Conference on*, pp. 26–29, IEEE, 2009.

[30] J. Newkirk, "Introduction to agile processes and extreme programming," in *Proceedings of the 24th international conference on Software engineering*, pp. 695–696, ACM, 2002.

[31] L. Liu and Y. Lu, "Application of agile method in the enterprise website backstage management system: Practices for extreme programming," in *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pp. 2412–2415, IEEE, 2012.

[32] K. Beck, *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[33] U. of Alberta, "Software processes and agile practices." `https://www.coursera.org/learn/software-processes-and-agile-practices`. Acessado em 13 de Dezembro de 2015.

[34] L. Lindstrom and R. Jeffries, "Extreme programming and agile software development methodologies," *Information systems management*, vol. 21, no. 3, pp. 41–52, 2004.

[35] B. Boehm, "Get ready for agile methods, with care," *Computer*, vol. 35, no. 1, pp. 64–69, 2002.

[36] S. Sircar, S. P. Nerur, and R. Mahapatra, "Revolution or evolution? a comparison of object-oriented and structured systems development methods," *MIS Quarterly*, pp. 457–471, 2001.

[37] B. Boehm and R. Turner, *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional, 2003.

[38] J. Highsmith, "Agile project management: Principles and tools," *Cutter consortium*, vol. 4, pp. 1–37, 2003.

[39] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *Computer*, vol. 34, no. 9, pp. 120–127, 2001.

[40] S. Cavaleri and K. Obloj, *Management systems: A global perspective*. Wadsworth, 1993.

[41] G. Milanov and A. NJEGUŠ, "Analysis of return on investment in different types of agile software development project teams," *Informatica Economica*, vol. 16, no. 4, pp. 7–18, 2012.

[42] D. Leffingwell, *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.

[43] D. F. Rico, H. H. Sayani, J. V. Sutherland, and S. Sone, *The business value of agile software methods: maximizing ROI with just-in-time processes and documentation*. J. Ross Publishing, 2009.

[44] A. Miguel, *Gestão de projectos de software*. FCA - Editora de Informática, Lda, 2010.

[45] P. Deemer, G. Benefield, C. Larman, and B. Vodde, "The scrum primer," *Scrum Primer is an in-depth introduction to the theory and practice of Scrum, albeit primarily from a software development perspective, available at: http://assets. scrumtraininginstitute. com/downloads/1/scrumprimer121. pdf*, vol. 1285931497, p. 15, 2010.

[46] D. F. Rico, "What is the return on investment (roi) of agile methods," 2008.

[47] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, and S. Z. Sarwar, "Agile software development: Impact on productivity and quality," in *Management of Innovation and Technology (ICMIT), 2010 IEEE International Conference on*, pp. 287–291, IEEE, 2010.

[48] D. West, T. Grant, M. Gerush, and D. D'silva, "Agile development: Mainstream adoption has changed agility," *Forrester Research*, vol. 2, p. 41, 2010.

[49] M. A. Santos, P. H. S. Bermejo, M. S. Oliveira, A. O. Tonelli, *et al.*, "Agile practices: An assessment of perception of value of professionals on the quality criteria in performance of projects," *Journal of Software Engineering and Applications*, vol. 4, no. 12, p. 700, 2011.

[50] J. Highsmith, "Agile: from rogue team to enterprise acceptance," *The Cutter Business Technology Council*, 2006.

[51] B. Schatz and I. Abdelshafi, "Primavera gets agile: a successful transition to agile development," *IEEE software*, no. 3, pp. 36–42, 2005.

[52] L. Barnett, "Agile survey results: Solid experience and real results," *Agile Journal*, pp. 70–77, 2006.

[53] S. Kuppuswami, K. Vivekanandan, P. Ramaswamy, and P. Rodrigues, "The effects of individual xp practices on software development effort," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 6, pp. 6–6, 2003.

[54] A. Law and R. Charron, "Effects of agile practices on social factors," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, pp. 1–5, ACM, 2005.

[55] A. Sidky, *A structured approach to adopting agile practices: The agile adoption framework*. PhD thesis, Virginia Polytechnic Institute and State University, 2007.

[56] D. Parsons, H. Ryu, and R. Lal, "The impact of methods and techniques on outcomes from agile software development projects," in *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, pp. 235–249, Springer, 2007.

[57] D. Mishra and A. Mishra, "Complex software project development: agile methods adoption," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 8, pp. 549–564, 2011.

Apêndices

Guião de Entrevista de Análise do Projeto CITT

Com a criação deste questionário, definiu-se os seguintes objetivos:

1. Entender qual foi a finalidade do projeto
2. Descrever o processo de execução do projeto
3. Perceber porque foram adotadas práticas ágeis na execução do projeto

Através da definição do objetivo foram formuladas as seguintes questões:

1. O que foi desenvolvido no projeto?
2. Qual foi a duração do projeto?
3. Quais foram os métodos (incorpora um conjunto de práticas ágeis) ou práticas ágeis utilizados?
    (a) Porquê?
    (b) Caso tenha sido adotado o método Scrum
        i. Qual foi a duração de cada *sprint*?
        ii. Como descreve a importância das práticas definidas neste método?
4. Como foi o processo de implementação dos métodos ou práticas ágeis no projeto?

5. Quantos elementos constituíam a equipa?
    (a) Quais foram os seus papéis?
    (b) Quais foram as suas responsabilidades?
6. Qual era a experiência dos elementos da equipa?
7. A equipa tinha conhecimento sobre as práticas ágeis?
8. Qual foi a reação da equipa na implementação das práticas ágeis?
9. Como foi realizada a comunicação entre os elementos da equipa?
10. O processo de execução foi composto por fases?
    (a) Quais?
11. Como foi realizada a gestão de requisitos?
12. Foram definidas ferramentas para apoiar a gestão de requisitos?
    (a) Quais?
    (b) Porquê?
13. Durante o desenvolvimento do projeto foi realizada monitorização?
    (a) Como?
    (b) Porquê?
14. Quais os desafios (dificuldades) encontrados na adoção das práticas ágeis?
15. Quais os impactos (benefícios) encontrados na adoção das práticas ágeis?