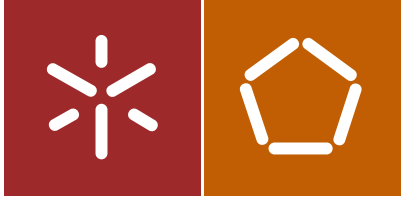
Universidade do Minho
Escola de Engenharia

Fidel Inácio Kussunga

# Caracterização de um Ambiente Visual para Apoiar as Cerimónias do SCRUM

outubro de 2019

Caracterização de um Ambiente Visual para Apoiar as Cerimónias do SCRUM
Fidel Inácio Kussunga
UMinho | 2019



Universidade do Minho
Escola de Engenharia

Fidel Inácio Kussunga

# Caracterização de um Ambiente Visual para Apoiar as Cerimónias do SCRUM

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia Eletrónica Industrial e Computadores

Trabalho efectuado sob a orientação do
Professor Doutor Pedro Miguel Gonzalez Abreu Ribeiro
Professor Doutor Paulo Francisco Cardoso

outubro de 2019

# AGRADECIMENTOS

A realização de uma dissertação é uma tarefa árdua que sem apoio de muitos, não seria possível a sua concretização. Apesar de que exige passar grande parte do tempo solitário, é reconfortante saber que existem pessoas ao nosso lado que nunca nos abandonam e que sempre estão lá para nos ajudar e apoiar. É muito importante lembrarmo-nos desses, que apesar de todas as dificuldades sempre estiveram presentes de uma forma direta ou indireta.

Agradeço em particular:

Ao meu orientador, Prof. Pedro Ribeiro, pela disponibilidade demonstrado e por todo apoio prestado e toda a orientação dada.

Ao meu prof. Paulo Cardoso, pela vontade e disponibilidade demonstrado em me orientar e por todos os seus conselhos.

Aos meus colegas, de Engenharia Eletrónica Industrial e Computadores, pelo companheirismo demonstrado ao longo desses anos. Um especial agradecimento para aqueles com quem diretamente trabalhei pelo todo apoio prestado.

Á minha família, porque sem elas nada disso seria possível, que apesar da distância nunca deixaram de me apoiar, principalmente a minha mãe (Madalena Nfiaussi).

Ainda, a minha querida namorada Jaquelina David, pela toda paciência e apoio prestado.

Por último, não poderia esquecer de agradecer ao Nuno Santos, pelo todo suporte dado e que contribuiu e muito para que a validação do trabalho fosse possível. Obrigado por toda a sua disponibilidade e vontade de ajudar.

## DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

# RESUMO

Metodologias ágeis de desenvolvimento de *software*, como o *Scrum*, ganharam enorme popularidade e foram bem-sucedidos em oferecer grandes benefícios para os seus utilizadores, como a aceleração de processos e recursos para lidar com a instabilidade de ambientes tecnológicos. O *feedback* rápido do cliente e o suporte para requisitos voláteis resultam num valor de produto mais alto, no entanto, a modelação de requisitos iniciais usando técnicas de modelação, não são comumente usados em processos ágeis como o *Scrum* para preparar melhor a fase de implementação do projeto de *software.*

Esta dissertação, irá propor um ambiente visual adequado para apoiar o *sprint* e *product backlog*. Para tal, será modelada uma solução/abordagem técnica utilizando uma linguagem de modelação, como por exemplo a *Unified Modeling Language* (UML) para apoiar a priorização de requisitos, melhorar a qualidade da solução e facilitar a manutenção de *software*.

Este procedimento seguirá a organização e gestão padrão de *Scrum*, e fornecerá uma implementação detalhada baseada no processo de modelação UML ou noutra linguagem de modelação de *software*.

O ambiente visual a ser proposto será validado com estudo de caso e painel de especialistas tornando assim a solução uma alternativa para resolver ou reduzir os problemas relacionados com a complexidade e qualidade de *software* produzidos pelas organizações. Serão estudadas as técnicas informais e formais de modelação de *software* no sentido de se encontrar o método adequado ao problema, menos complexa e que envolva menor custo para o desenvolvimento de um produto.

A metodologia que será utilizada para o desenvolvimento desta dissertação, é a *Design Science Research*. Esta metodologia consiste na revisão de literatura ajustada nos conceitos mais importantes para o problema em estudo, seguido de uma proposta de um artefacto que contribua com um novo conhecimento para a ciência, sendo avaliado posteriormente num contexto real.

**Palavras – chave**: Ambiente visual, desenvolvimento de *software*, metodologias ágeis, modelação de *software*, *SCRUM*

# ABSTRACT

Agile software development methodologies, such as Scrum, have gained tremendous popularity and have been successful in delivering great benefits to their users, such as accelerating processes and resources to deal with the instability of technological environments. Rapid customer feedback and support for volatile requirements result in a higher product value; however, modeling of initial requirements using modeling techniques is not commonly used in agile processes such as Scrum to better prepare the implementation phase of the project. software project.

This dissertation will propose a suitable visual environment to support the sprint and product backlog. For this, a technical solution / approach will be modeled using a modeling language such as the Unified Modeling Language (UML) to support the prioritization of requirements, improve the quality of the solution and facilitate the maintenance of software.

This will follow the standard Scrum organization and management and will provide a detailed implementation based on the UML modeling process or other software modeling language.

The visual environment to be proposed will be validated with a case study and expert panel thus making the solution an alternative to solve or reduce the problems related to the complexity and quality of software produced by the organizations. The informal and formal techniques of software modeling will be studied in order to find the appropriate method to the problem, less complex and involving less cost for the development of a product.

The methodology that will be used to develop this dissertation is Design Science Research. This methodology consists of a literature review adjusted to the most important concepts for the problem under study, followed by a proposal for an artifact that contributes to a new knowledge for science and is evaluated later in a real context.

**Keywords:** *Visual environment, Software development, SCRUM, Software modeling, Agile methodologies*.

# ÍNDICE

# LISTA DE FIGURAS

# LISTA DE TABELAS

# ABREVIATURAS

| | |
|---|---|
| ASD | *Adaptive Software Development* |
| CRC | Classe, Responsabilidade e Colaboração |
| DFD | Diagrama de Fluxo de Dados |
| DSDM | *Dynamic Systems Development Method* |
| ERM | *Entity – Relationship Modeling* |
| FDD | *Feature Driven Development* |
| FSM | *Finite States Machines* |
| 4SRS | *Four Step Rule Set* |
| *MDA* | *Model Driven Architecture* |
| MIEEIC | Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores |
| NLP | *Natural Language Processing* |
| OMG | *Object Management Group* |
| PO | *Product Owner* |
| *SWEBoK* | *Software Engineering Body of Knowledge* |
| TSI | Tecnologias de Sistemas de Informação |
| UMGAR | *UML Model Generator Analysis of Requirements* |
| UML | *Unified Modeling Language* |
| UC | *Use Case* |
| US | *User Story* |
| XP | *eXtreme Programming* |

# 1. INTRODUÇÃO

Neste capítulo introdutório, é apresentado enquadramento do tema do trabalho em desenvolvimento. É abordado ainda nesta parte da dissertação, a motivação da escolha do estudo deste problema assim como também é discutida a abordagem metodológica de investigação utilizada para a execução desta dissertação. Por fim, é apresentada a estrutura do documento.

## 1.1 Motivação

A preocupação existente no meio das organizações em implementar um bom *software* leva-nos a modelarmos os sistemas para se obter um *software* com a qualidade desejável. Os modelos servem para comunicar a estrutura e o comportamento desejados do sistema, visualizar e controlar a arquitetura do mesmo e compreender melhor o sistema que está a ser desenvolvido.

Outra grande motivação tem a ver com o facto de a modelação de requisitos iniciais usando técnicas de modelação não serem comumente usados em processos ágeis como o *Scrum* para preparar melhor a fase de implementação do projeto de *software.* Através de modelos, consegue-se obter visões diferentes do sistema, minimizando a complexidade do sistema para facilitar o seu entendimento, e atuando como meio de comunicação entre intervenientes do projeto.

## 1.2 Enquadramento

O tema em estudo está enquadrado na área científica de Tecnologias de Sistemas de Informação (TSI), concretamente na área de Gestão de Projetos, com o foco para a caracterização de um ambiente visual (baseado em técnicas de modelação) para apoiar o *Scrum* (*product backlog* e *sprint backlog*).

Para contextualizar o tema, começou-se por se abordar as metodologias ágeis de desenvolvimento de *software*.

Os sistemas de *software* são cada vez mais complexos devido aos requisitos complicados e diversificados dos clientes [1]. Ao mesmo tempo, o mercado é muito competitivo e exigente, amplificadas pela fraca qualidade de *software* produzido e a satisfação do cliente são questões cada vez mais importantes [2]. Portanto, as equipas de *software* devem desenvolver produtos de *software* para atender a todos os requisitos em tempo pré-determinado e sob custo definido. Os métodos tradicionais de desenvolvimento de *software* são cada vez menos adequados. Esses métodos são processos pesados, que dão mais atenção à industrialização e à padronização. Isso geralmente leva a um ciclo de desenvolvimento mais longo e um custo adicional [1]. O surgimento de métodos ágeis nos últimos anos atraiu cada vez mais atenção, uma vez que são utilizadas com sucesso para melhorar os processos de *software*.

As metodologias ágeis provenientes do Manifesto Ágil em [3] são métodos mais leves para projetos de *software* comparados com a maioria dos métodos tradicionais. As metodologias ágeis apresentam equipas auto-organizadas que são capacitadas para atingir metas de negócios específicas. As metodologias ágeis focam a sua atenção em entregas rápidas e frequentes de soluções parciais de maneira iterativa e incremental [1]. Metodologias ágeis demonstraram fornecer produtos de alta qualidade em menos tempo, resultando em maior satisfação do cliente.

O *Scrum* é uma abordagem para o desenvolvimento de projetos que não requer o preenchimento de uma especificação técnica de várias páginas, como no modelo tradicional [4]. O *Scrum* é uma metodologia para gerir o desenvolvimento de sistemas de informação, que coloca uma forte ênfase no controlo de qualidade do processo de desenvolvimento. Além de gerir projetos de desenvolvimento de *software*, a metodologia é usada por equipas de suporte de *software*, bem como uma abordagem para gerir o desenvolvimento e a manutenção de *software* [4]. A metodologia ágil *Scrum*, é baseada na divisão de projetos em iterações (*sprints*). Cada *sprint* inclui cinco fases, que são [5]: (1) inicializar, (2) analisar, (3) projetar, (4) realizar e (5) testar um conjunto de *user stories*. Uma *user story* descreve a funcionalidade que será valiosa para o utilizador / cliente de um sistema ou *software* [6]. No contexto da Arquitetura Orientada a

Modelos (MDA) [7], de facto, várias empresas de desenvolvimento de *software* adotaram essa notação, as *user stories* são descritas em nível de Modelo Independente Computacional (CIM). Este modelo representa o nível mais alto de abstração e de processo de desenvolvimento ágil [5].

Os mecanismos de planeamento, estimativa e controlo dos modelos de processos ágeis dependem significativamente de um conjunto fixo de tarefas estabelecido para cada *sprint*. Essas tarefas são criadas como refinamentos de itens do *Product Backlog* no início de cada *sprint*. No entanto, o entendimento de uma equipa de projeto sobre as implicações e dependências de negócios dos itens do *Product Backlog* pode não ser profundo o suficiente para identificar todas as tarefas necessárias tão cedo, portanto, além das tarefas definidas no início do *sprint*, mais tarefas necessárias podem ser descobertas [8].

A linguagem de modelação UML contém diversos diagramas que cobrem a análise e *design* de sistemas orientadas a objetos e é uma linguagem padrão no mercado de desenvolvimento de *software*. Esta ferramenta de modelação apoia as práticas em nível técnico em cada fase do desenvolvimento de *software*. Além disso, a modelação UML é independente de modelos de processo. Verifica-se, que em [1] foi proposto uma abordagem que combina a metodologia *Scrum* com a ferramenta da linguagem de modelação UML.

Pretende-se com esta dissertação, na primeira fase, estudar detalhadamente e categorizar todas as técnicas de modelação existentes com o intuito de justificar a escolha de um procedimento para aplicação na metodologia *Scrum.*

## 1.3 Objetivos

A questão de investigação desta dissertação é: “Quais serão as características adequadas de um ambiente visual para apoiar o *sprint* e *Product Backlog*?”. Será, portanto, a partir desta questão que todo trabalho será desenvolvido.

Pretende-se com esta dissertação propor um ambiente visual que facilite a interação entre os intervenientes no projeto, selecionando uma abordagem que torne este processo acessível a todos.

Sendo assim, é necessário um estudo das abordagens existentes sobre caracterização de um ambiente visual para apoiar as cerimónias do *Scrum* ou de uma outra metodologia ágil.

Outros objetivos são:

- Estudo detalhado das metodologias ágeis, com foco principal no *Scrum*;
- Análise de propostas existentes na literatura sobre técnicas para apoiar cerimónias *Scrum*;
- Proposta do ambiente visual adequado para apoiar o *sprint* e *product backlog*;
- Validação do ambiente proposto com estudo de caso;
- Formalização de uma nova versão do ambiente com base no *feedback* da validação.

## 1.4 Metodologia de Investigação

Para um projeto de investigação é fundamental a definição de uma abordagem metodológica adequada, uma vez que, a metodologia adotada irá contribuir para o alcance dos objetivos definidos para projeto.

As abordagens metodológicas geralmente abordam um problema existente a partir do qual uma hipótese é formada e analisada. A análise pode envolver o desenvolvimento de sistema (s) protótipo (s) para fornecer prova de conceito. Para a investigação fundamental, essa evidência ou artefacto é importante, pois torna-se o foco para expandir ou continuar a investigação [9].

De acordo com Nunamaker, Chen e Purdin [9] uma metodologia de investigação consiste na utilização de processos, métodos e ferramentas necessárias para a realização de trabalho de investigação sobre um determinado assunto.

O sucesso e a conclusão de um projeto de investigação depende da escolha de métodos apropriados e sistemáticos [10]. De acordo com Berndtsson, Hansson, Olsson e Lundell [10] um método é uma abordagem organizada para resolução de um determinado problema que inclui: (1) recolha de dados, (2) formulação de uma hipótese ou proposição, (3) teste da hipótese, (4) interpretação dos resultados e (5) indicação das conclusões que podem posteriormente ser avaliadas de forma independente por outros [10].

Existem diferentes abordagens de investigação, a sua escolha prende-se com a natureza do problema e com o que se pretende fazer [10].

A escolha de uma abordagem metodológica adequada vai possibilitar executar de forma mais bem-sucedida o trabalho de investigação, servindo de guião para responder à questão de investigação a que o trabalho pretende responder.

Tendo em conta a natureza do problema deste trabalho e os resultados esperados, a abordagem de investigação para esta dissertação (trabalho de investigação) é a *Design Science Research*, visto que, com o artefacto criado pretende-se resolver um problema real identificado.

Um aspeto importante da abordagem metodológica *Design Science Research* é a avaliação dos artefactos; em outras palavras, a utilidade dos artefactos propostos deve ser demonstrada [11]. Para a validação da abordagem proposta concretamente para esta dissertação, é apresentado primeiro um caso de estudo para estabelecer a validade dos constructos e métodos propostos e demonstrar o seu uso na caracterização de um ambiente visual. Com esta validação pretendeu-se obter (1) *feedback*, (2) maior compreensão e motivação para o problema em discussão, (3) uma avaliação da utilidade prática da abordagem proposta em contexto real e (4) sugestões de melhorias e trabalho futuro.

Peffers et al. [12] apresentam um modelo de processos para o desenvolvimento de investigação com recurso ao método *Design Science Research.* Esta metodologia divide-se em seis fases segundo [13]: Identificação do problema, Sugestão, Desenvolvimento, Avaliação, Comunicação e Conclusão.

1. **Identificação do problema e motivação** – nesta etapa define-se o problema específico de investigação. A definição do problema vai ser usada para o desenvolvimento de artefactos que podem ajudar a chegar a solução. Nesta fase é necessário justificar o valor da solução [12] e a sua relevância para a área em questão. Para isso é necessário um bom conhecimento sobre o estado da arte na área do problema e da importância ou relevância da solução. O Objetivo principal desta fase é a identificação da questão de investigação.
2. **Definição dos objetivos para a solução** – depois de conhecido o problema e depois do levantamento do estado da arte estar concluído, devem-se definir os objetivos para a solução. Os objetivos podem ser quantitativos ou qualitativos. Se forem quantitativos, a

solução proposta deve ser melhor que as já existentes. Se forem qualitativos, a solução deve descrever a forma como o novo artefacto suporta a solução do problema. Nesta fase, caso existam outras soluções, deve ser definida a sua eficiência para servir de termo de comparação [12].

3. ***Design* e desenvolvimento** – Nesta etapa, como o nome indica, desenham-se e constroem-se o(s) artefacto(s). Conceptualmente um artefacto pode ser qualquer objeto concebido pelo qual a investigação contribui para o seu desenvolvimento. Podem ser modelos, métodos ou novas propriedades técnicas [12]. Esta atividade inclui a construção do artefacto desde a definição das suas funcionalidades e arquitetura, até ao seu *design* e desenvolvimento.
4. **Demonstração** – nesta fase, é feita a demonstração da utilidade do artefacto para a resolução de uma ou mais instâncias do problema. A demonstração pode envolver um problema especificado. Para que a demonstração tenha valor é necessária a criação de um ambiente apropriado e bem definido. A demonstração pode servir, se correr bem, como prova de que a ideia funciona [12].
5. **Avaliação** – a fase de avaliação consiste na verificação, observação ou medição, se realmente o artefacto suporta a solução para o problema. Esta atividade envolve a comparação dos objetivos definidos com os resultados reais produzidos pelo artefacto na demonstração. Nesta fase é necessário conhecer as técnicas de análise e métricas relevantes. Dependendo da natureza do problema, a avaliação pode assumir várias formas: pode envolver a comparação das funcionalidades do artefacto com os objetivos da solução enumerados na atividade 2; pode medir quantitativamente a performance do artefacto através de simulações ou medidas de tempo de resposta, por exemplo [14].

   No final desta atividade, tendo em conta os resultados desta avaliação, o investigador pode decidir retroceder à atividade 3 (*design* e desenvolvimento) e tentar melhor o artefacto, retroceder para a atividade 2 e redefinir os objetivos da solução ou continuar para a atividade seguinte e deixar as possíveis melhorias para o trabalho futuro.

6. **Comunicação e divulgação do resultado** – no final, é necessário comunicar e divulgar os resultados obtidos, a sua importância, o artefacto, a sua utilidade e o que traz de novo, rigor da conceção, e a sua eficácia.

Na Figura 1 estão representadas as atividades desenvolvidas durante o processo de investigação usando a metodologia *Design Science Research* [12].

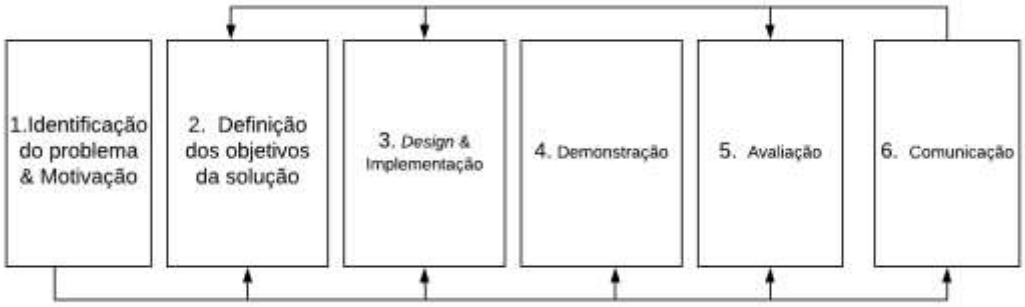



*Figura 1 - Atividades principais do método Design Science Research, adaptado de Perffers et al.* [12]

## 1.5 Estrutura da dissertação

Para que seja possível alcançar os objetivos definidos no item anterior, esta dissertação será dividida em 5 capítulos.

No primeiro, é feito o enquadramento do problema em análise, a metodologia de investigação, assim como os objetivos que se pretendem atingir com a realização deste trabalho e a motivação.

O segundo capítulo faz a revisão da literatura existente, é feita uma introdução dos tópicos a serem abordados. Serão ainda estudadas nesta seção, os conceitos sobre a engenharia de *software* e ainda o desenvolvimento ágil de *software* com foco sobre o *Scrum.* Ainda neste segundo capítulo, são estudadas técnicas de modelação de *software*.

No terceiro capítulo, faz-se a análise dos trabalhos já desenvolvido que abordam o tema deste projeto.

No quarto capítulo desta dissertação, será apresentado o procedimento proposto, e ainda, far-se-á uma análise crítica e propostas de melhorias.

Por fim, o capítulo 5 é a conclusão da dissertação, apresentando os resultados esperados e o trabalho futuro, que poderá enriquecer a solução apresentada.

# 2. METODOLOGIAS ÁGEIS E TÉCNICAS DE MODELAÇÃO

O objetivo deste capítulo é apresentar uma revisão bibliográfica sobre o tema em discussão. Este capítulo é composto por cinco secções, a primeira é referente a uma breve introdução sobre a engenharia de *software,* a segunda secção aborda as metodologias ágeis de desenvolvimento de *software*, a terceira secção apresenta uma comparação das metodologias ágeis estudadas, a quarta secção discute as técnicas de modelação de *software* com destaque para a modelação baseada em UML e a quinta secção aborda a *técnica Four Step Rule Set.*

## 2.1 A Engenharia de *Software*

Sommervill [15],[16],[17] define engenharia de *software* como uma disciplina de engenharia que se preocupa com todos os aspetos da produção de *software* desde etapas iniciais da especificação do sistema até a manutenção do sistema durante a sua utilização.

Hoje em dia assistimos a uma grande transformação do *software* assim como a sua expansão em função das crescentes necessidades, sendo preciso acompanhar o seu ritmo de crescimento. Isso significa que a trajetória do *software* está a impulsionar a engenharia de *software*, e não o contrário [18].

*Software* é um sistema criado pelo homem e, cada vez mais, a sua complexidade é maior. À medida que o *software* se torna imprescindível, as expetativas dos clientes aumentam também. Mas a necessidade e a disponibilidade de um engenheiro de *software* competente não se fez sentir com o crescimento da indústria de *software* [19].

*Tabela 1 - Atributos essenciais para um software de qualidade* [15]

| **Características do produto** | Descrição |
|---|---|
| Manutenção | O *software* deve ser escrito de tal forma que possa evoluir para atender às necessidades de mudança dos clientes. Este é um atributo crítico porque a mudança de *software* é um requisito inevitável de um ambiente de negócios em mudança. |
| Confiabilidade e segurança | A fiabilidade do *software* inclui uma variedade de características, incluindo confiabilidade e segurança. O *software* confiável não deve causar danos físicos ou económicos em caso de falha do sistema. Utilizadores mal-intencionados não devem poder aceder ou danificar o sistema. |
| Eficiência | O *software* não deve fazer uso desnecessário de recursos do sistema, como ciclos de memória e de processador. A eficiência inclui, portanto, a capacidade de resposta, o tempo de processamento, a utilização da memória, etc. |
| Aceitabilidade | O *software* deve ser aceitável para o tipo de utilizador para o qual foi projetado. Isso significa que deve ser compreensível, utilizável e compatível com outros sistemas que eles usam. |

Segundo Sommerville [15], a engenharia de *software* é importante por dois motivos:

1. Cada vez mais a sociedade conta com sistemas de *software* avançados. É necessária uma capacidade de produzir sistemas fiáveis de forma económica e rápida;
2. Geralmente, é mais barato, a longo prazo, utilizar métodos e técnicas de engenharia de *software* para sistemas de *software*, em vez de apenas desenvolver programas como se fosse um projeto de programação pessoal. Para a maioria dos sistemas, o custo tem a ver com a manutenção depois da utilização do sistema de *software*.

Para Sommerville [15], a abordagem sistemática usada na engenharia de *software* é normalmente chamada processo de *software*. Um processo de *software* é uma sequência de

atividades que leva à produção de um produto de *software*. Existem quatro atividades fundamentais que são comuns a todos os processos de *software*. Essas atividades são [15]:

1. Especificação de *software*, na qual clientes e engenheiros definem o *software* que deve ser produzido e as restrições do seu funcionamento;
2. Desenvolvimento de *software*, onde o *software* é implementado;
3. Validação de *software*, onde o *software* é verificado para garantir que é o que o cliente requer;
4. Evolução do *software*, onde o *software* é modificado para refletir mudanças nos requisitos do cliente e do mercado.

Diferentes tipos de sistemas precisam de diferentes processos de desenvolvimento [1]. Por exemplo, o *software* em tempo real de uma aeronave precisa de ser completamente especificado antes do início do desenvolvimento.

Na secção 2.2 são descritos os processos de desenvolvimento de *software* utilizando as metodologias ágeis.

### 2.1.1 Sistemas de *Software*

Existem muitos tipos diferentes de sistemas de *software*, desde simples sistemas embutidos até sistemas complexos de informação em todo o mundo. Não faz sentido procurar notações, métodos ou técnicas universais para engenharia de *software*, porque diferentes tipos de *software* exigem abordagens diferentes. Desenvolver um sistema de informação de uma organização é completamente diferente do desenvolvimento de um controlador para um instrumento científico. Nenhum desses sistemas tem muito em comum com um jogo de computador com uso intensivo de gráficos. Todas essas aplicações precisam de engenharia de *software,* mas nem todas precisam das mesmas técnicas de engenharia de *software*.

A engenharia de *software* é criticada como inadequada para o desenvolvimento de *software* moderno [15]. No entanto, segundo Sommerville [15] muitas dessas falhas de *software* são consequência de dois fatores:

**1.Necessidades crescentes**. À medida que novas técnicas de engenharia de *software* nos ajudam a construir sistemas maiores e mais complexos, as demandas mudam. Os sistemas precisam ser construídos e entregues mais rapidamente; sistemas maiores e mais complexos são necessários;

os sistemas precisam ter novas capacidades que antes eram consideradas impossíveis. Os métodos de engenharia de *software* existentes não conseguem lidar e novas técnicas de engenharia de *software* precisam ser desenvolvidas para atender a essas novas demandas.

**2.Baixas expectativas**. É relativamente fácil escrever programas de computador sem usar métodos e técnicas de engenharia de *software*. Muitas empresas evoluíram para o desenvolvimento de *software* à medida que os seus produtos e serviços evoluíram. Elas não usam métodos de engenharia de *software* no seu trabalho diário. Consequentemente, o seu software é geralmente mais caro e menos fiável do que deveria ser. Precisamos de uma melhor formação em engenharia de *software* e preparação para resolver esse problema.

A Engenharia de *Software* ocupa-se desta vertente e aborda as teorias, métodos e ferramentas que são necessárias para a construção dos sistemas da computação de *software* [15].

## 2.2 Metodologias Ágeis de Desenvolvimento de *Software*

O campo de desenvolvimento de *software* continua a introduzir novas metodologias. De facto, nos últimos anos, um grande número de abordagens diferentes para o desenvolvimento de *software* foi introduzido. Um estudo de NandKumar e Avison [20] argumenta que as metodologias tradicionais de desenvolvimento de sistemas de informação "são tratadas principalmente como uma ficção necessária para apresentar uma imagem de controlo ou fornecer um status simbólico". O mesmo estudo afirma ainda que essas metodologias são mecanicista demais para ser usado em detalhes. Parnas e Clements [21] apresentaram argumentos semelhantes desde o início. Como resultado, os desenvolvedores de *software* industriais tornaram-se céticos em relação às "novas" soluções difíceis de entender e, portanto, não são usadas [22]. Esse é o pano de fundo para o surgimento de métodos ágeis de desenvolvimento de *software*.

O desenvolvimento ágil de *software* foi definido como uma forma de produzir *software* que obedece às seguintes regras [3]:

i. Pessoas e interações sobre processos e ferramentas;
ii. *Software* que trabalha sobre uma documentação completa;

iii. Colaboração com o cliente sobre negociação e contrato;

iv. Responder a mudanças sobre o seguimento do plano.

Ou seja, embora haja valor nos itens à direita, valoriza-se mais os itens à esquerda.

Assim, pode-se ver claramente que os métodos ágeis são uma família de processos de desenvolvimento e não abordagens únicas para o desenvolvimento de *software* [22]. A maioria das metodologias ágeis tenta minimizar o risco desenvolvendo entregas em iterações curtas, cada uma é como um projeto de *software* em miniatura (análise de requisitos, *design*, codificação, teste). As metodologias ágeis enfatizam a comunicação em tempo real, preferencialmente face a face. Isto é, as metodologias ágeis produzem muito pouca documentação escrita. No entanto, isso pode transformar-se numa enorme desvantagem, quando os membros da equipa abandonam a empresa, e não deixam lá o conhecimento adquirido [22].

O desenvolvimento de *software* usando metodologias ágeis é cada vez mais adotado pelos profissionais de *software*, pois promove o desenvolvimento inicial de *software* e produtos de *software* de alta qualidade. Além disso, oferece capacidade de resposta às mudanças nos requisitos do utilizador, proporcionando a sua rápida absorção durante o desenvolvimento do *software* [23], [24].

Uma das metodologias ágeis mais utilizada é o *Scrum* [25],[27], com uma descrição mais completa nesta secção. Outras abordagens ágeis incluem *eXtreme Programming* (XP) [6], [28], *Adaptive Software Development* (ASD) [29], *Dynamic Systems development method* (DSDM) [30],[31] e *Feature Driven Development (FDD)* [32]. Estas são apenas algumas metodologias ágeis apresentadas das várias existentes.

Embora as metodologias ágeis sejam todas baseadas na noção de desenvolvimento e entrega incremental, eles propõem diferentes processos para conseguir isso [15]. No entanto, elas compartilham um conjunto de princípios, baseados no manifesto ágil e, portanto, têm muito em comum [33]. Esses princípios são mostrados na Tabela 2. Diferentes metodologias ágeis instanciam esses princípios de maneiras diferentes [15]. São apresentados alguns métodos ágeis de forma resumida devido a limitação do espaço, fornecendo as respetivas citações para mais detalhes, concentrando-se neste caso em metodologia *Scrum*.

*Tabela 2 - Os princípios de métodos ágeis* [15]

| Princípio | Descrição |
|---|---|
| Envolvimento do cliente | Os clientes devem estar intimamente envolvidos durante todo o processo de desenvolvimento. A sua função é fornecer e priorizar novos requisitos do sistema e avaliar as iterações do sistema |
| Entrega incremental | O *software* é desenvolvido em incrementos com o cliente a especificar os requisitos a serem incluídos em cada incremento. |
| Pessoas não processos | As competências da equipa de desenvolvimento devem ser reconhecidas e exploradas. Os membros da equipa devem ter liberdade para desenvolver suas próprias maneiras de trabalhar sem processos prescritivos |
| Abraçar a mudança | Esperar que os requisitos do sistema sejam alterados e assim, projetar o sistema para acomodar essas mudanças. |
| Manter a simplicidade | Concentrar-se na simplicidade do *software* que está a ser desenvolvido e do processo de desenvolvimento. Sempre que possível, trabalhar ativamente para eliminar a complexidade do sistema. |

Há interesse crescente no uso das metodologias ágeis para outros tipos de desenvolvimento de *software*. Contudo, devido ao seu foco em equipas pequenas e fortemente integradas, há problemas em escalá-las para grandes sistemas. No entanto, devido à necessidade de análise de segurança, privacidade e fiabilidade nos sistemas críticos, as metodologias ágeis

exigem mudanças significativas antes que possam ser frequentemente utilizadas para desenvolvimento de sistemas críticos [15].

### *2.2.1 eXtreme Programming*

*eXtreme Programming* (XP) é uma coleção de práticas de engenharia de *software* bem conhecidas. O XP visa permitir um desenvolvimento de *software* bem-sucedido, apesar dos requisitos de *software* livres ou em constante mudança [3]. Beck [28] descreve o *eXtreme Programming* como uma disciplina de desenvolvimento de *software* que organiza as pessoas para produzir *software* de maior qualidade de forma mais produtiva [34].

O XP Foi inicialmente introduzido por Beck [28] (um dos autores do manifesto ágil) e provou ser muito bem-sucedido em diferentes empresas nas mais variadas dimensões e em vários setores. Esta abordagem tem como foco a satisfação do cliente, fazendo com que os desenvolvedores respondam às mudanças nos requisitos do cliente e forneçam *software* de alta qualidade de forma rápida e contínua. O *software* é entregue ao cliente geralmente num intervalo de tempo de 1 a 3 semanas [35]. XP melhora os projetos de *software*, através de comunicação frequente com o cliente, simplicidade, *feedback*, respeito e coragem. O XP propõe 12 regras [34]: *Planning Game*, Fases Pequenas, Testes de Aceitação do Cliente, *Design* Simples, Programação por Pares, Desenvolvimento Orientado a Testes, *Refactoring*, Integração Contínua, Propriedade Coletiva do Código, Padrões de Codificação, Metáfora e Ritmo Sustentável. Como acontece com todos os processos ágeis estas regras não são escritas no sentido de serem permanentes e com o passar do tempo algumas regras foram modificadas, novas regras surgiram por exemplo Shore e Werden [36] citado por D. Stankovic, et al. [35]. Os autores fazem diferenciação entres duas versões do XP descritas por Beck [28] e Beck e Andres [37] introduzindo mesmo a sua própria abordagem ao XP que despontou da sua experiência empresarial.

Na prática, muitas empresas que adotaram o XP não usam todas as práticas extremas de programação. Elas configuram o processo de acordo com as suas formas específicas de trabalho. Para acomodar diferentes níveis de competências, alguns programadores não fazem *refactoring* em partes do sistema que não desenvolveram, e os requisitos convencionais podem ser usados em vez de *user stories* (notação textual usada para captura de requisitos de *software*). No

entanto, a maioria das empresas que adotaram uma variante XP usam pequenas versões, desenvolvimento de teste inicial e integração contínua [38].

**Papéis e Responsabilidades**

Existem diferentes funções no XP para diferentes tarefas e propósitos durante o processo e suas práticas [39] . A seguir, essas funções são apresentadas de acordo com Beck [28].

**Programador.** Os programadores escrevem testes e mantêm o código do programa tão simples e definido quanto possível. A primeira questão que torna o XP bem-sucedido é uma boa comunicação e coordenação com outros programadores e membros da equipa.

**Cliente.** O cliente escreve as *User* e os testes funcionais e decide quando cada exigência é satisfeita. O cliente define a prioridade de implementação para cada requisito.

**Testador.** Os testadores ajudam o cliente a escrever testes funcionais. Eles executam testes funcionais regularmente, transmitem resultados de testes e mantêm ferramentas de teste.

***Tracker.*** O *Tracker* dá *feedback* no XP. Ele acompanha as estimativas feitas pela equipa (por exemplo, estimativas de esforço) e dá *feedback* sobre o quão precisas elas são para melhorar as estimativas futuras. Ele também rastreia o progresso de cada iteração e avalia se a meta é alcançável dentro dos recursos e do tempo especificados.

**Treinador (Coach).** *Coach* é a pessoa responsável pelo processo como um todo. Uma boa compreensão do XP é importante nesta função, permitindo que o *Coach* oriente os outros membros da equipa para a interiorização do processo.

**Consultor.** Consultor é um membro externo que possui o conhecimento técnico específico necessário. O consultor orienta a equipa na solução dos seus problemas específicos.

**Gestor (*Big Boss*).** Gestor toma as decisões. Para poder fazer isso, ele comunica com a equipa do projeto para determinar a situação atual e para resolver qualquer dificuldade ou impedimento.

**Práticas XP**

Nesta secção estão enumeradas e descritas as 12 práticas XP (ver Tabela 3) de acordo com Beck [28].

*Tabela 3 - Práticas propostas no XP* [15]

| Prática | Descrição |
|---|---|
| ***Planning Game***. | Essa prática sugere um relacionamento próximo entre o cliente e a equipa técnica do projeto. Cada parte é responsável por definir e identificar um conjunto de atributos específicos do projeto, como propósito, prazos, estimativa de esforço e desvantagens tecnológica. |
| **Fases Pequenas** | O objetivo dessa prática é colocar rapidamente em produção (ou seja, implementar e testar) um sistema simples. Cada lançamento deve ser o menor possível e conter os requisitos mais valiosos para o cliente. |
| **Metáfora** | As metáforas permitem descrever um recurso a ser implementado, criando uma visão comum do cliente e da equipa técnica sobre como o produto deve funcionar. Desta forma, pode-se reduzir o uso de expressões técnicas, muitas vezes difíceis de serem compreendidas pelo cliente. |
| ***Design* Simples** | A arquitetura e o código (incluindo os testes de unidade) devem ser o mais simples possível. |
| **Desenvolvimento Orientado a Testes** | Todos os recursos implementados devem ser cobertos por testes de unidade, que devem ser sempre satisfeitos, em um esforço para eliminar erros de nível de unidade e de regressão durante o desenvolvimento. No XP, um recurso só está pronto para ser integrado numa versão quando atende a esses requisitos. |
| **Teste** | Testes de unidades são implementados antes do código e são executados continuamente. Os clientes escrevem os testes funcionais. |
| ***Refactoring*** | O *refactoring* visa simplificar o código implementado removendo a ambiguidade do código e a redundância |
| ***Pair Programming*** | Essa prática consiste em ter dois programadores trabalhando simultaneamente no mesmo computador. Cada programador tem um papel específico. Enquanto um elemento é responsável por escrever o |

| | |
|---|---|
| | código, o outro é responsável por verificar e validá-lo, com atenção especial para recursos não testados ou bloqueio de código. |
| **Propriedade coletiva** | Cada membro da equipa é incentivado a realizar todas as alterações necessárias no código. Assim, todos os membros da equipa são donos do código. Essa prática evita esperas desnecessárias por alterações de terceiros no código. |
| **Integração contínua** | Depois de um novo recurso ser implementado ou haver ajustamentos no código e após a execução de todos os testes com sucesso, uma nova *release* deve ser criada refletindo todas as alterações. |
| **40 horas semanais** | Não trabalhar mais de 40 horas por semana é uma regra |
| **Cliente no local** | O XP propõe não apenas um relacionamento próximo com o cliente, mas também que um cliente (ou um representante) deve estar sempre presente durante o ciclo de vida do projeto, sendo, portanto, parte da equipa do projeto. |
| **Padronização do código** | Os padrões de codificação permitem uma interpretação mais fácil do código implementado por todos os programadores. |

**Processo XP**

O ciclo de vida do XP consiste em seis fases: Exploração, Planeamento, Iterações para libertação, Produção, Manutenção e Término (Figura 2).

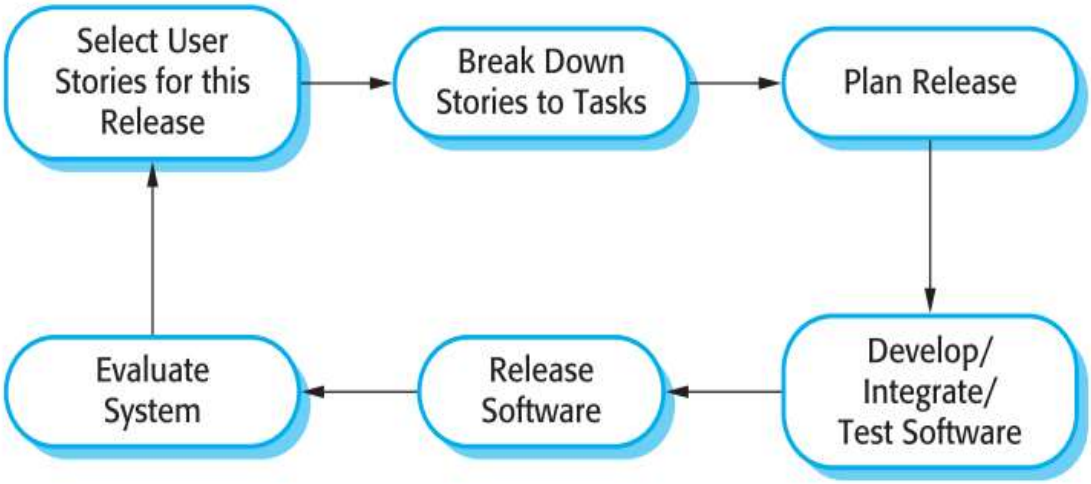



*Figura 2 - Ciclo de vida do processo XP* [15]

Na *eXtreme Programming*, os requisitos são expressos como *user stories* (US), que são implementados diretamente como uma série de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes de escrever o código. Todos os testes devem ser executados com sucesso quando um novo código é integrado ao sistema. Há um curto intervalo de tempo entre os lançamentos do sistema [15].

A *eXtreme Programming* envolve diversas práticas, que refletem os princípios dos métodos ágeis [15]:

a) O desenvolvimento incremental é suportado por pequenas e frequentes versões do sistema. Os requisitos são baseados em *US* simples que são usadas como base para decidir qual funcionalidade deve ser incluída num incremento do sistema.
b) O envolvimento do cliente é promovido através da integração do cliente na equipa de desenvolvimento. O representante do cliente participa no desenvolvimento e é responsável pela definição dos testes de aceitação do sistema.
c) As pessoas, e não o processo, são apoiadas por meio da programação por pares, da propriedade coletiva do código do sistema e de um processo de desenvolvimento sustentável que não envolve horas de trabalho excessivamente longas.

d) A mudança é adotada por meio de *releases* regulares do sistema para os clientes, desenvolvimento de teste inicial, *refactoring* para evitar a degeneração de código e integração contínua de novas funcionalidades.
e) Manter a simplicidade é suportado pelo *refactoring* constante que melhora a qualidade do código e usando o *design* simples não antecipando desnecessariamente futuras mudanças no sistema.

Num processo de XP, os clientes estão intimamente envolvidos na especificação e priorização dos requisitos do sistema. Os requisitos não são especificados como listas de funções do sistema necessárias. Em vez disso, o cliente faz parte da equipa de desenvolvimento e discute os cenários com outros membros da equipa [40].

Na prática, muitas empresas que adotaram o XP não usam todas as práticas extremas de programação. Elas escolhem de acordo com as suas formas locais de trabalho. Para acomodar diferentes níveis de habilidade, alguns programadores não fazem *refactoring* em partes do sistema que não desenvolveram e os requisitos convencionais podem ser usados em vez *user stories*. No entanto, a maioria das empresas que adotaram uma variante XP usa pequenas versões, desenvolvimento de teste inicial e integração contínua [15], [40], [41].

### 2.2.2 *Adaptive Software Development* (ASD)

O ASD foi proposto no ano 2000 para resolver questões sobre o desenvolvimento de sistemas grandes e complexos. O método tenta fornecer orientação para evitar falhas em projetos, mas ao mesmo tempo não é uma orientação excessiva que pode impedir a criatividade ou fazer com que o processo se torne lento e não flexível. Conceitos fortes relacionados com o ASD são desenvolvimento iterativo, abordagem incremental para recursos e prototipagem [42].

O Desenvolvimento de *Software* Adaptativo substitui o ciclo de cascata tradicional por uma série repetida de ciclos de especulação, colaboração e aprendizagem. Este ciclo dinâmico proporciona aprendizagem contínua e adaptação ao estado emergente do projeto. As características de um ciclo de vida do ASD são que ele é focado na missão, baseado em recursos, iterativo, *time boxed*, orientado a riscos e tolerante a mudanças. O ASD afirma fornecer uma

estrutura com orientação suficiente para evitar que os projetos caiam no caos, mas não em demasia, o que poderia suprimir a emergência e a criatividade [34].

**Papéis e Responsabilidades**

O processo ASD tem origem em grande parte na cultura da organização e gestão e especialmente, na importância da colaboração nas equipas e no trabalho em equipa. Todas essas questões são muito consideradas [43]. A abordagem, no entanto, não descreve detalhadamente as estruturas da equipa. Da mesma forma, muito poucos papéis ou responsabilidades são listados. Um patrocinador de execução é nomeado como a pessoa com responsabilidade geral pelo produto em desenvolvimento. Os participantes de uma sessão conjunta de desenvolvimento de aplicações são os únicos outros papéis mencionados (um facilitador para planear e liderar a sessão, um escriba para elaborar atas, o gestor de projeto e representantes de clientes e desenvolvedores) [42].

**Práticas do ASD**

O ASD propõe muito poucas práticas para o trabalho de desenvolvimento de *software* do dia-a-dia. Basicamente, [29] menciona expressamente três: desenvolvimento iterativo, planeamento baseado em recursos (com base em componentes) e revisões usando *focus group* com o cliente. De facto, talvez o problema mais significativo com o ASD é que as suas práticas são difíceis de identificar e deixam muitos detalhes em aberto [44].

**Processo ASD**

Um projeto de Desenvolvimento de *Software* Adaptativo é realizado em ciclos de três fases. As fases dos ciclos são *Speculate*, *Collaborate* e *Learn* [42]. Na Figura 3 podem ser visualizadas as três etapas mencionadas.

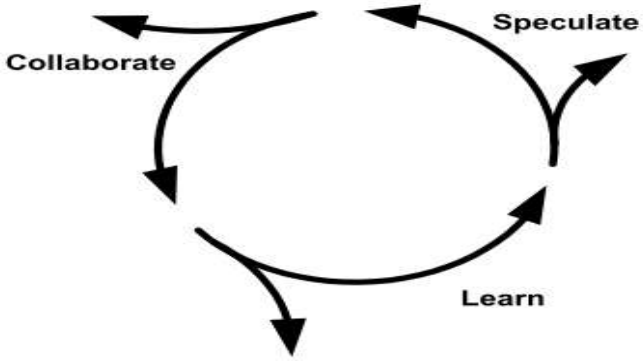



*Figura 3 - Ciclo de desenvolvimento ASD* [42]

As fases são nomeadas de forma a destacar o papel da mudança no processo. "*Speculate*" é usada em vez de "planeamento", já que um "plano" é geralmente visto como algo em que a incerteza é uma fraqueza e a partir do qual os desvios indicam falha. Da mesma forma, "*Collaborate*" destaca a importância do trabalho em equipa como meio de desenvolver sistemas com mudanças rápidas. "*Learn*" destaca a necessidade de reconhecer e reagir a erros e o facto de que os requisitos podem mudar durante o desenvolvimento [15].

### 2.2.3 *Feature Driven Development (FDD)*

FDD é uma abordagem ágil e adaptativo para o desenvolvimento de *software,* concentrando-se nas fases de projeto e construção [38][29]. Funciona também com outras atividades de um projeto de desenvolvimento de *software* [32] e não requer nenhum modelo de processo específico a ser usado. O FDD inclui o desenvolvimento iterativo com as melhores práticas encontradas para serem eficazes na indústria. O método FDD destaca os aspetos de qualidade em todo o processo e inclui entregas frequentes e tangíveis, juntamente com um acompanhamento rigoroso da evolução do projeto [42].

FDD baseia-se em cinco processos sequenciais (figura 4) e disponibiliza os métodos, técnicas e diretrizes necessárias aos *Stakeholders* do projeto para desenvolver o sistema. Além disso, o FDD junta funções, artefactos, metas e cronogramas necessários em projetos [32].

Diferente de algumas metodologias ágeis, o FDD considera-se adequado para o desenvolvimento de sistemas críticos [32].

A primeira vez que se falou do FDD foi no ano de 2000. Foi então desenvolvido com base no trabalho feito para um grande projeto de desenvolvimento de *software* por Jeff Luca, Peter Coad e Stephen [45].

**Papéis e Responsabilidades**

O FDD classifica os seus papéis em três categorias: papéis chave, papéis de apoio e funções adicionais [32]. Os seis principais papéis no projeto FDD são gestor de projeto, arquiteto chefe, gestor de desenvolvimento, programador chefe, proprietário de classe e especialistas de domínio. As cinco funções de suporte incluem o gestor de lançamentos, o *Language Lawyer/Language Guru*, o *Build Engineer*, o *Toolsmith* e o administrador do sistema. As três funções tradicionais necessárias em qualquer projeto são as dos testadores, *deployers* e *technical writers*. Um membro da equipa pode desempenhar várias funções e uma única função pode ser compartilhado por várias pessoas [32].

**Práticas do FDD**

O FDD consiste no conjunto de "melhores práticas" e os desenvolvedores do método afirmam que, embora as práticas selecionadas não sejam novas, a combinação específica desses ingredientes torna os cinco processos do FDD exclusivos para cada caso [44]. Palmer e Felsing [32] também defendem que todas as práticas disponíveis devem ser usadas para obter o máximo benefício do método, já que nenhuma prática isolada domina todo o processo. O FDD envolve as seguintes práticas [44]:

1. **Modelação de Objetos de Domínio**: Exploração e explicação do domínio do problema, resultando numa estrutura onde as *Features* são adicionados;
2. **Desenvolvendo pelas *Features***: Desenvolvendo e seguindo a evolução através de uma lista de pequenas funções decompostas e valorizadas pelo cliente;
3. **Propriedade Individual da Classe (*code*)**: Cada classe tem uma única pessoa nomeado para ser o responsável pela consistência, desempenho e integridade conceptual da classe;

4. **Equipas por *Features***: Refere-se a equipas pequenas e dinamicamente formadas;
5. **Inspeção**: Refere-se ao uso de mecanismos mais conhecidos de deteção de defeitos;
6. **Construções Regulares**: Refere-se a garantir que haja sempre um sistema em execução e demonstrável, disponível. Construções regulares formam a linha de base na qual novas *Features* são adicionadas;
7. **Gestão de Configurações**: Permite a identificação e a consulta do histórico das versões mais recentes de cada ficheiro de código fonte concluído;
8. **Relatório de Progresso**: O progresso é reportado com base no trabalho completo para todos níveis organizacionais necessários.

A equipa do projeto deve colocar em prática todas as práticas acima descritas, para cumprir com as regras de desenvolvimento do FDD. No entanto, a equipa pode adaptá-las de acordo com seu nível de experiência [44].

**Processos FDD**

Como referido anteriormente, o FDD consiste em cinco processos sequenciais durante os quais o *design* e construção do sistema são realizados (Figura 4). A parte iterativa dos processos do FDD (*Design* e Construção) oferece suporte ao desenvolvimento ágil, com adaptações rápidas a mudanças posteriores nos requisitos e necessidades de negócio. Normalmente, uma iteração de uma *Feature* envolve um período de trabalho de uma a três semanas à equipa.

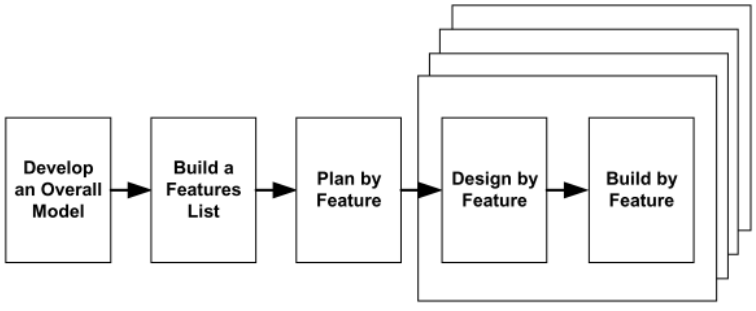



*Figura 4 - Processo FDD* [32]

A seguir são descritos cada um dos cinco processos de acordo com [32]

***Develop an Overall Model.*** Quando o desenvolvimento de um modelo geral começa, os especialistas do domínio já estão cientes do propósito, contexto e requisitos do sistema a ser construído[32], [35]. É presumível que existam requisitos documentados, como casos de uso ou especificações funcionais, nesta fase. Todavia, o FDD não aborda expressamente a questão de reunir e gerir os requisitos. Os especialistas do domínio apresentam um *walkthrough* no qual os elementos da equipa e o arquiteto chefe são informados da descrição de alto nível do sistema [38].

***Build a Features List***. As orientações, modelos de objetos e documentação de requisitos disponíveis fornecem um bom princípio para a criação de uma lista completa de *Features* para o sistema que está a ser desenvolvido. Na lista, a equipa de desenvolvimento apresenta cada uma das funcionalidades proposta pelo cliente incluídas no sistema. As funcionalidades são apresentadas para cada um dos setores do domínio e esses grupos de funcionalidades consistem nos chamados conjuntos principais de *Features*. Estes representam atividades diferentes dentro de áreas de domínio específicas. A lista de *Features* é revista pelos utilizadores e *sponsors* do sistema para sua validade e integridade.

***Plan by Feature***. O planeamento por *Feature* inclui a criação de um plano de alto nível no qual as várias *features* são sequenciadas de acordo com as suas prioridades e dependências e atribuídas aos programadores principais (ver secção 2.5.3). Além disso, as classes identificadas no processo (desenvolvimento de um modelo geral) são atribuídas a desenvolvedores individuais, ou seja, proprietários de classes. Também o agendamento e marcos principais podem ser definidos para as várias *features*.

***Design by Feature and Build by Feature***. Um pequeno grupo de *features* é selecionado do(s) conjunto(s) de *features* e as equipas de *features* necessárias para o desenvolvimento das *features* selecionados são formadas pelos proprietários da classe. Os processos de *design* por *feature* e construção por *feature* são procedimentos iterativos, durante os quais as *features* selecionadas são produzidos. Uma iteração deve durar de alguns de dias a no máximo duas semanas. Pode haver várias equipas de *features* a projetar e a criar simultaneamente o seu próprio conjunto de *features*. Esse processo iterativo inclui tarefas como inspeção de projeto, codificação, teste de

unidade, integração e inspeção de código. Após uma iteração bem-sucedida, as *features* concluídas são promovidos para construção principal, enquanto a iteração de criação começa com um novo conjunto de *features* retiradas da lista de *features*.

### 2.2.4 *Dynamic Systems Development Method* (DSDM)

Desde a sua origem, em 1994, o DSDM, tornou-se gradualmente no *frameword* número um para o desenvolvimento rápido de aplicações (RAD) no Reino Unido [30]. O DSDM é um *framework* sem fins lucrativos e não proprietária para o desenvolvimento RAD [46], mantida pelo consórcio DSDM.

*O* DSDM é um *framework* que agrega grande parte do conhecimento atual sobre gestão de projetos. O DSDM está enraizado na comunidade de desenvolvimento de *software*, mas a convergência de desenvolvimento de *software*, engenharia de processo e, consequentemente, projetos de desenvolvimento de negócios mudou a estrutura do DSDM para se tornar uma estrutura geral para tarefas complexas de solução de problemas [47].

A ideia principal por trás do DSDM é que, em vez de fixar a número de funcionalidades num produto e ajustar o tempo e os recursos para alcançar essas funcionalidades, é preferível fixar tempo e recursos e em seguida ajustar a quantidade de funcionalidade de acordo com [48]

**Papéis e Responsabilidades**

O DSDM define 15 funções para utilizadores e desenvolvedores. Os mais dominantes são listados a seguir [30].

**Desenvolvedo**r e **desenvolvedores seniores** são os únicos papéis de desenvolvimento. A antiguidade é baseada na experiência nas tarefas que o desenvolvedor realiza. O título de desenvolvedor sénior também indica um nível de liderança na equipa. As funções de desenvolvedor e desenvolvedor sénior abrangem toda a equipa de desenvolvimento, seja analista, projetistas, programadores ou testadores. Um **coordenador técnico** define a arquitetura do sistema e é responsável pela qualidade técnica do projeto. Das funções do utilizador, a mais importante é o de ***ambassador user***. Os respetivos deveres são trazer o conhecimento da comunidade de utilizadores para o projeto e disseminar informações sobre o pregresso do

projeto para outros utilizadores. O **visionário** é o participante do utilizador que tem a perceção mais precisa dos objetivos de negócios do sistema e do projeto. O **patrocinador executivo** é alguém da organização que é também utilizador e que possui a autoridade e responsabilidade financeira do projeto. O patrocinador executivo, portanto, tem o poder final na tomada de decisões.

**Práticas DSDM**

DSDM apresenta nove práticas que definem os princípios e a base para todas as atividades do DSMD. As práticas DSDM, chamadas princípios são listadas na Tabela 4.

*Tabela 4 - Práticas ESDM* [31]

| Práticas DSDM | Descrição |
|---|---|
| **Envolvimento ativo do utilizador é imperativo.** | Alguns utilizadores experientes devem estar presentes durante todo o desenvolvimento do sistema para garantir *feedback* oportuno e rigoroso. |
| **As equipas DSDM devem ser autónomas para tomar decisões.** | Processos de tomadas de decisão longos não podem ser tolerados em ciclos rápidos de desenvolvimento. Os utilizadores envolvidos no desenvolvimento têm conhecimento para dizer onde o sistema dever ir. |
| **O foco está na entrega frequente de produtos.** | Decisões erradas podem ser corrigidas, se o ciclo de entrega for curto e os utilizadores puderem fornecer *feedback* preciso. |
| **Aptidão para fins comerciais é o critério essencial para aceitação de produtos.** | Antes que as principais necessidades de negócios do sistema sejam satisfeitas, a excelência técnica em áreas menos importantes não deve ser procurada. |

| | |
|---|---|
| **O desenvolvimento iterativo e incremental é necessário para convergir numa solução de negócio precisa.** | Os requisitos do sistema raramente permanecem inalterados desde o início de um projeto até ao final. Ao permitir que os sistemas evoluam através do desenvolvimento iterativo os erros podem ser encontrados e corrigidos o mais cedo possível. |
| **Todas as alterações durante o desenvolvimento são reversíveis** | No decorrer do desenvolvimento, um caminho errado pode ser seguido. Usando iterações curtas e garantindo que os estados anteriores podem ser revertidos, o caminho errado pode ser corrigido com segurança. |
| ***A linha de base dos requisites é definida num nível alto*** | A linha de base dos requisitos principais deve ser feita apenas num nível alto, para permitir que os requisitos detalhados sejam alterados conforme necessário. |
| **O teste é integrado ao longo do ciclo de vida** | Com fortes restrições de tempo, os testes tendem a ser negligenciados quando deixados para o final do projeto. Portanto, todo componente do sistema deve ser testado à medida que são desenvolvidos. |
| **É essencial uma abordagem colaborativa e compartilhada por todos os *stakeholders*.** | Para que o DSDM funcione, a organização deve comprometer-se seriamente com ele. A escolha do que é entregue no sistema e do que é deixado de fora é sempre um compromisso e requer um acordo comum. |

**Processo DSDM**

O DSDM consiste em cinco fases: estudo de viabilidade, estudo de negócio, iteração de modelo funcional, iteração de *design* e construção e implementação (ver Figura 5). As primeiras duas fases são sequenciais e feitas apenas uma única vez. As três últimas fazes, durante as quais o trabalho de desenvolvimento real é feito, são iterativas e incrementais. O DSDM aborda iterações como *timeboxes.* Um *timebox* dura um período de tempo prefinido e a iteração tem que terminar dentro do *timebox*. O tempo permitido para cada iteração é planeado

antecipadamente, junto com os resultados que a iteração tem a garantia de produzir. No DSDM, uma duração típica do *timebox* é de alguns dias a algumas semanas [42].

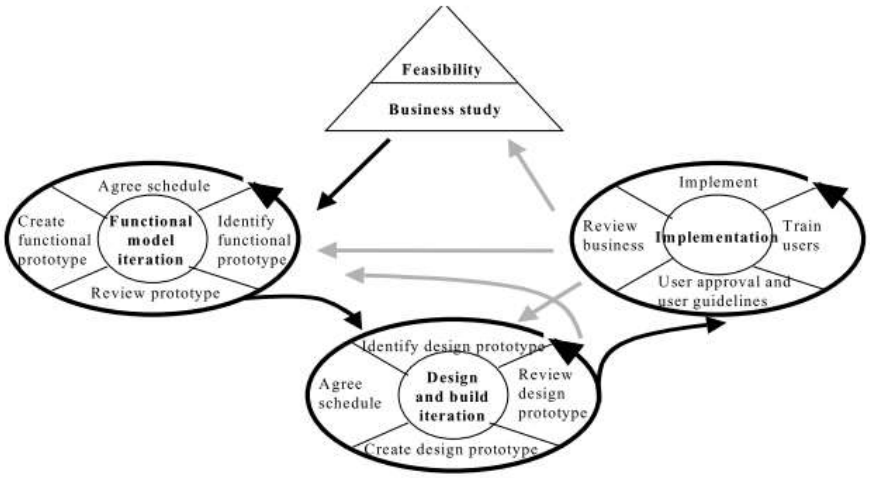



*Figura 5 - Diagrama de processo DSDM* [31]

A seguir, as fases são apresentadas sucintamente, com a respetiva documentação essencial [38].

Na fase de **estudo de viabilidade** a adequação do DSDM para o projeto em questão é avaliada. A julgar pelo tipo de projeto e acima de tudo, questões organizacionais e de pessoas, a decisão é tomada, seja para usar o DSDM ou não. Dois produtos de trabalho são preparados, um relatório de viabilidade e um plano preliminar para o desenvolvimento. A fase de estudo de viabilidade não deve demorar mais de que algumas semanas [42].

O **estudo de negócio** é uma fase em que as características necessárias do negócio e da tecnologia são analisadas. A abordagem recomendada consiste em reunir numa sala um número suficiente de especialistas do cliente, para poder considerar todas as facetas relevantes do sistema e concordar com as prioridades de desenvolvimento. Os processos e as classes de utilizadores afetados devem ser descritos. A identificação das classes de utilizadores afetadas ajuda a envolver o cliente, pois as pessoas chave na organização do cliente podem ser identificadas e envolvidas numa fase inicial [42].

A fase de **iteração do modelo funcional** é a primeira fase iterativa e incremental. Em cada iteração, o conteúdo e a abordagem da iteração são planeados, a iteração é executada e os resultados analisados para as iterações seguintes. Na iteração a análise e a codificação são realizadas, os protótipos são construídos e as experiências adquiridas com eles são usadas para melhorar os modelos de análise. Um modelo funcional é produzido como saída, contendo o código do protótipo e os modelos de análise. O teste é também uma parte essencial e contínua dessa fase. As funcionalidades priorizadas é uma lista priorizada das funcionalidades que serão entregues no final de iteração. Os documentos de revisão de prototipagem funcional recolhem os comentários dos utilizadores sobre o incremento atual, funcionando como entrada para as iterações subsequentes. Requisitos não-funcionais são listados, principalmente para serem tratados na próxima fase. A análise de risco do desenvolvimento adicional é um documento importante nesta fase, porque a partir da próxima fase (*design* e construção de iteração) em diante, os problemas encontrados serão mais difíceis de resolver.

Na **iteração de *design* e construção** o sistema é construído. A saída é um sistema testado que cumpre pelo menos o conjunto mínimo de requisitos acordados. O *design* e a construção são iterativas. Os protótipos funcionais e de *design* são revistos pelos utilizadores e os desenvolvimentos posteriores são baseado nos comentários dos utilizadores.

A fase final de **implementação** refere-se à transferência do ambiente de desenvolvimento para o ambiente de produção real. É dada formação aos utilizadores e o sistema é entregue. Além do sistema entregue, a saída da fase de implementação também inclui um manual do utilizador e um relatório de revisão do projeto.

### 2.2.5 *Scrum*

As primeiras referências na literatura ao termo *Scrum* apontam para o Takenchi e Nonaka [49] citado por [15], no qual é apresentado um processo de desenvolvimento de produto adaptativo, rápido e auto-organizado com origem no Japão [27]. O termo *Scrum* originalmente deriva de uma estratégia no jogo de rugby em que as equipas andam com a bola para a frente em pequenos passos para atingir objetivo, obter o ensaio. 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*. O s*print* pode ter duração de 2 a 4 semanas, fornecendo rapidamente valor de negócio para o cliente [50].

A abordagem *Scrum* foi desenvolvida para gerir o processo de desenvolvimento de sistemas. É uma abordagem empírica que aplica as ideias da teoria de controlo de processos industriais ao desenvolvimento de sistemas, resultando numa abordagem que reintroduz as ideias de flexibilidade, adaptabilidade e produtividade [27]. Não define nenhuma técnica específica de desenvolvimento de *software* para a implementação, mas antes concentra-se em como os membros da equipa devem funcionar para produzir o sistema de forma flexível num ambiente em constante mudança [51].

O *Scrum* é uma das metodologias ágeis mais populares atualmente sendo na verdade uma estrutura para gestão de processo de projetos de *software* [27]. O *Scrum* é frequentemente usado com outros métodos, como o XP [41], o *Rational Unified Process* (RUP) [52] ou outros. O motivo é que enfatiza a gestão de projetos e não fornece descrições completas e detalhadas de como tudo deve ser feito num projeto. Esses trabalhos são deixados para a equipa de desenvolvimento de *software Scrum* [1].

O *Scrum* ajuda a melhorar as práticas de engenharia existentes numa organização, pois envolve atividades de gestão frequentes, com o objetivo de identificar consistentemente quaisquer deficiências ou impedimentos no processo de desenvolvimento, bem como nas práticas utilizadas [53].

**Papéis e Responsabilidades**

Existem seis papéis identificáveis no *Scrum* que têm diferentes tarefas e propósitos durante o processo e suas práticas: *Scrum Master*, *Product Owner*, equipa *Scrum*, cliente, utilizador e gestão. A seguir, essas funções são apresentadas de acordo com as definições de Schwaber e Beedle [27]**.**

***Scrum Master***. O *Scrum Master* é um novo papel de gestão de projetos introduzido pelo *Scrum*. O *Scrum Master* é responsável por garantir que o projeto é realizado de acordo com as práticas, valores e regras do *Scrum* e que ele progride como planeado. O *Scrum Master* interage com a

equipa do projeto, bem como com o cliente e a gestão durante o projeto. Ele também é responsável por garantir que os impedimentos sejam removidos no processo para manter a equipa a trabalhar da maneira mais produtiva possível [54].

***Product Owner***. Segundo Schwaber e Beedle [27] , *Product Owner* é oficialmente responsável pelo projeto, gerindo, controlando e tornando visível a lista do *Product Backlog*. Ele é selecionado pelo *Scrum Master*, cliente e a gestão. Ele toma as decisões finais sobre as tarefas relacionados com o *Product Backlog*, participa na estimativa de esforço de desenvolvimento para itens do *Backlog* e transforma os problemas do *Backlog* em *features* a serem desenvolvidas.

**Equipa *Scrum***. A equipa *Scrum* é a equipa do projeto que tem autoridade para decidir sobre as ações necessárias e organizar-se para atingir as metas de cada *sprint*. A equipa S*crum* está envolvida, por exemplo, na estimativa de esforço, criando a *Sprint Backlog*, revendo a lista de *Product Backlog* e apresentando impedimentos que precisam ser removidos do projeto [27].

**Cliente**. O cliente participa das tarefas relacionadas com o *Product Backlog* para o sistema que está a ser desenvolvido.

**Gestão**. A gestão é responsável pela decisão final, juntamente com os padrões e convenções a serem seguidas no projeto. A gestão também participa na definição de metas e requisitos. Por exemplo, a gestão está envolvida na seleção do *Product Owner*, avaliando o progresso e reduzindo o *Backlog* com o *Scrum Master* [55].

**Práticas *Scrum***

De acordo com Schwaber e Deeble [3], o *Scrum* não requer ou fornece quaisquer métodos / práticas de desenvolvimento de *software* específicos a serem usados. Em vez disso, requer certas práticas e ferramentas de gestão nas várias fases do *Scrum* para evitar o caos causado pela imprevisibilidade e complexidade [26].

A seguir, apresenta-se a descrição das práticas do *Scrum* [25], [27], [56].

***Effort Estimation*.** A estimativa de esforço é um processo iterativo, no qual as estimativas de itens do *Backlog* vão-se tornando mais rigorosas, conforme mais informações estão disponíveis em relação aos itens do *Product Backlog*. O *Product Owner* juntamente com a equipa do *Scrum* são responsáveis por fazer a estimativa de esforço [25].

***Product Backlog***. O *Product Backlog* é uma lista ordenada de tudo o que é conhecido como necessário no produto [54]. É a única fonte de requisitos para quaisquer alterações a serem feitas no produto. O *Product Owner* é responsável pelo *Product Backlog*, incluindo o seu conteúdo, disponibilidade e pedido [25], [56].

À medida que um produto é usado e ganha valor, e o mercado fornece *feedback*, o *Product Backlog* torna-se numa lista maior e mais exaustiva. Os requisitos nunca param de mudar, portanto, o *Product Backlog* é um artefacto vivo. Alterações nos requisitos de negócios, condições de mercado ou tecnologia podem causar alterações no *Backlog* do Produto [27].

Esta prática inclui as tarefas para criar a lista de *Product Backlog* e controlá-la consistentemente durante o processo, adicionando, removendo, especificando, atualizando e priorizando os itens do *Product Backlog.* A Tabela 5 ilustra um exemplo de um *Product Backlog.*

*Tabela 5 - Exemplo de itens do Product Backlog, adaptado de Abrahamsson et al.* [42]

| **Prioridade** | **Item** | **Descrição** | **Tempo Estimado** | **Responsável** |
|---|---|---|---|---|
| Muito. Alta | | | | |
| | 1 | Conexão com Base de Dados | 40 | Fidel |
| | 2 | … | … | … |
| | 3 | Registo de Utilizadores | | Verónica |
| | 4 | … | | |
| Média | | | | |
| | 5 | Impressão de Relatórios | | |
| | 6 | … | … | … |

***Sprint***. Um *sprint* é uma unidade de planeamento na qual o trabalho a ser feito é avaliado, as *features* são selecionadas para desenvolvimento e o *software* é implementado [57]. Como já referido, cada *sprint* pode ter a duração de 2 a 4 semanas. No final de um *sprint*, a funcionalidade completa é entregue aos *stakeholders* [25]. As ferramentas de trabalho da equipa durante os *sprints* (ver figura 6) são: *Sprint Planning Meetings*, *Sprint Backlog* e *Daily Scrum Meetings* [54].

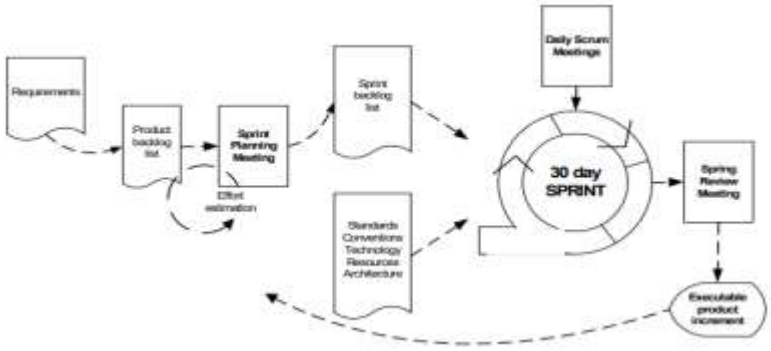



*Figura 6 - Práticas e inputs do sprint* [27]

***Sprint Planning meeting****.* Uma reunião de planeamento de *sprint* é uma reunião com duas fases organizada pelo *Scrum Master*. Os clientes, utilizadores, gestão, *Product Owner* e Equipa *Scrum* participam da primeira fase da reunião para decidir sobre os objetivos e as funcionalidades do próximo *sprint* (ver *Sprint Backlog* a seguir). A segunda fase da reunião é realizada pelo *Scrum Master* e a Equipa *Scrum*, focando-se na forma como o incremento do produto será implementado durante o *sprint* [25].

***Sprint Backlog****.* O *Sprint Backlog* é o ponto de partida para cada *sprint*. É uma lista de itens do *Product Backlog* selecionados para serem implementados no próximo *sprint*. Os itens são selecionados pela Equipa *Scrum* juntamente com o *Scrum Master* e o *Product Owner* na reunião de planeamento do *sprint*, com base nos itens priorizados e metas definidas para o *sprint*. Ao contrário do *Product Backlog*, o *Sprint Backlog* é estável até que o *sprint* (ou seja, 2 a 4 semanas) seja concluído. Quando todos os itens no *Sprint Backlog* são concluídos, uma nova iteração do sistema é entregue [56].

***Daily Scrum meeting****.* As reuniões diárias do *Scrum* são organizadas para acompanhar o progresso da equipa *Scrum* continuamente e também servem como reuniões de planeamento: o que foi feito desde a última reunião e o que deve ser feito antes da próxima. Também problemas e outros assuntos diversos são discutidos e apresentados nesta reunião curta (aproximadamente 15 minutos) realizada diariamente. Quaisquer deficiências ou impedimentos no processo de desenvolvimento ou nas práticas de engenharia são procurados, identificados e removidos para

melhorar o processo. O *Scrum Maste*r conduz as reuniões *Scrum*. Além da Equipa *Scrum* também a gestão, por exemplo, pode participar da reunião [56], [58].

***Sprint Review meeting****.* No último dia do *sprint*, a Equipa *Scrum* e o *Scrum Master* apresentam o resultado (ou seja, o incremento de produto de trabalho) do *sprint* para a gestão, clientes, utilizadores e o *Product Owner* numa reunião informal. Os participantes avaliam o incremento do produto e tomam a decisão sobre atividades seguintes. A reunião de revisão pode trazer novos itens do *Backlog* e até mesmo mudar a direção do sistema que está a ser construído [56], [58].

***Sprint Retrospective****.* A retrospetiva do *sprint* é uma reunião realizada após a revisão do *sprint* e antes da próxima reunião de planeamento do *sprint*. Os membros da equipa dão *feedback* sobre o *sprint* [56]. A equipa colabora na identificação de algumas melhorias para futuros *sprints* [58].

**Processos *Scrum***

O processo *Scrum* inclui três fases: pré-jogo, desenvolvimento e pós-jogo (Figura 7).

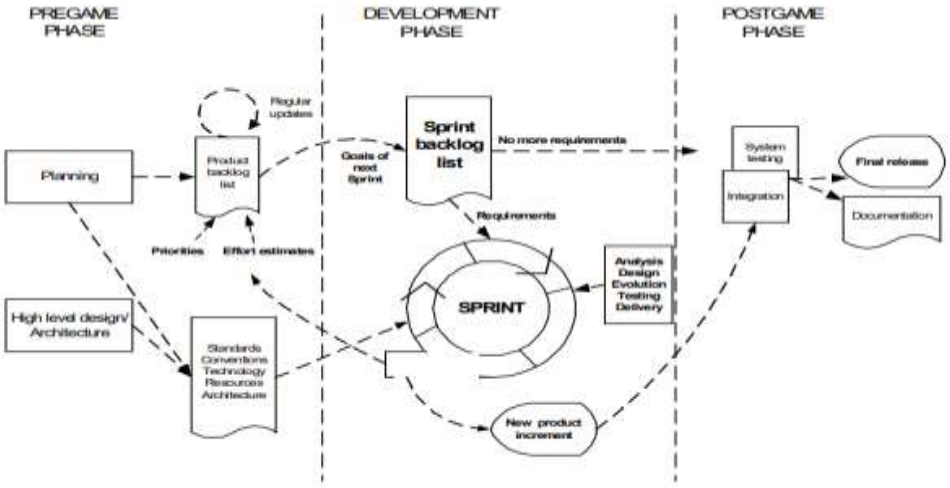



*Figura 7 - O processo scrum* [27]

Em seguida, as fases do *Scrum* são apresentadas de acordo com Schwaber [27]

**Pré-Jogo**. Esta fase inclui duas subfases: a primeira é o planeamento que consiste na definição do sistema que está a ser desenvolvido. É criado um *Product Backlog,* contendo todos os requisitos

atualmente conhecidos. Os requisitos podem ter origem no cliente, nas áreas envolvidas ou nos próprios desenvolvedores de *software*. Os requisitos são priorizados e o esforço necessário para sua implementação é estimado. O *Product Backlog* é constantemente atualizado com itens novos e mais detalhados, bem como com estimativas mais precisas e novas ordens de prioridade. O planeamento também inclui a definição da equipa do projeto, ferramentas e outros recursos, avaliação de riscos e questões de controlo e necessidades de formação. A cada iteração, o *Product Backlog* atualizado é revisto pela Equipa do *Scrum*, de modo a obter o seu comprometimento com a próxima iteração.

**Fase de desenvolvimento (também chamada de fase de Jogo).** Esta é a parte ágil da abordagem *Scrum*. Esta fase é tratada como uma "caixa preta" onde o imprevisível é esperado. As diferentes variáveis ambientais e técnicas (como prazo, qualidade, requisitos, recursos, tecnologias e ferramentas de implementação e até métodos de desenvolvimento) identificadas no *Scrum* e que podem mudar durante o processo, são observadas e controladas através de várias práticas *Scrum* durante os *sprints*. Na 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, mostrando-se a evolução do *software* a cada *sprint*.

**Fase Pós-Jogo**. Esta fase contém o encerramento e lançamento do *software*, em que todos os requisitos estão implementados não sendo permitido acrescentar novos requisitos.

**Funcionalidades do *Scrum***

A metodologia *Scrum* (Figura 8) é ideal para projetos com requisitos em rápida mudança ou altamente emergentes. O trabalho a ser feito num projeto *Scrum* está listado no *Product Backlog (*ver secção 2.2.5*)* [54]. No início de cada *sprint*, é realizada uma reunião de planeamento do *sprint*, durante a qual o *Product Owner* prioriza o *Product Backlog* e a Equipa *Scrum* seleciona as tarefas que podem ser concluídas durante o próximo *sprint* [56]. As tarefas são então movidas do *Product Backlog* para o *Sprint Backlog*. Durante o *sprint*, os desenvolvedores mantêm-se atualizados realizando breves reuniões diárias. No final de cada *sprint*, a equipa demonstra a funcionalidade concluída numa reunião de revisão do *sprint* [25].


*Figura 8 - Visão Geral do Scrum* [59]

**Teoria do *Scrum***

O *Scrum* é baseado na teoria empírica de controlo de processos, ou empirismo. O empirismo afirma que o conhecimento vem da experiência e da tomada de decisões com base no que é conhecido [26], [56]. O *Scrum* emprega uma abordagem incremental e iterativa para otimizar a previsibilidade e controlar o risco [41].

Para Ken Schwaber and Jeff Sutherland [25] são três os pilares que sustentam toda implementação do controlo de processo empírico: transparência, inspeção e adaptação.

**Transparência**. Aspetos significativos do processo devem ser visíveis para os responsáveis pelo resultado. A transparência exige que esses aspetos sejam definidos por um padrão comum, para que os observadores compartilhem um entendimento comum do que está a ser visto [25].

**Inspeção**. Os utilizadores do *Scrum* devem inspecionar frequentemente os artefactos do *Scrum* e progredir em direção a uma meta do *sprint* para detetar variações indesejáveis. A inspeção não deve ser tão frequente que atrapalhe o trabalho. Inspeções são mais benéficas quando executadas diligentemente por inspetores qualificados no local de trabalho [25].

**Adaptação**. Se um inspetor determinar que um ou mais aspetos de um processo se desviam fora dos limites aceitáveis e que o produto resultante será inaceitável, o processo ou o material que está a ser processado deve ser ajustado. Um ajuste deve ser feito o mais rápido possível para minimizar mais desvios [8].

## 2.3 Comparação de Métodos Ágeis

Nesta secção é feita uma comparação entre metodologias ágeis estudadas em termos de parâmetros considerados relevantes. A Tabela 6 ilustra a comparação entre várias abordagens. Destaque para os parâmetros da metodologia *Scrum*, que é uma metodologia que pode se adequar a um projeto de médio / grande dimensão, além de apresentar uma equipa inferior a 10 elementos e múltiplas equipas.

Feita uma análise às várias metodologias apresentadas é possível concluir que nenhum método apresenta todas as características necessárias para um processo de desenvolvimento de *software*. A adoção de uma ou outra metodologia depende do tipo do projeto que se pretende desenvolver, não existindo uma metodologia melhor ou pior.

*Tabela 6 - Comparação dos métodos ágeis estudados, adaptado de Ken Schwaber* [26]

| **Parâmetros de qualidade \ Ágil** | **XP** | **ASD** | **FDD** | **DSDM** | **SCRUM** |
|---|---|---|---|---|---|
| Práticas de engenharia | Sim | Sim | Sim | Sim | Não |
| Práticas de gestão de projetos | Não | Sim | Sim | Sim | Sim |
| Aceitação a mudança em cada iteração | Sim | Sim | Sim | Sim | Sim |
| *Refactoring* | Sim | NE | Não | Não | Não |
| Programação por pares | Sim | NE | Não | Não | Não |
| Priorização de requisitos | Sim | NE | Sim | Não | Não |
| Tamanho do projeto | Pequeno / médio | NE | Grande | Grande | Médio / grande |
| Desenvolvimento orientado a testes | Sim | NE | Não | Não | Não |
| Auto-organização | Não | NE | Sim | Sim | Sim |
| Teste unitários | Sim | NE | Sim | Sim | Não |
| Nível de documentação | Menos | NE | Mais | NE | Mais |
| Conceção | Centrada no código | NE | NE | NE | Centrada na conceção |
| Tamanho da equipa | <10 | NE | NE | NE | <10 e múltiplas equipas |

NE: Não Especificado

## 2.4 Técnicas de Modelação

A modelação é uma componente fundamental das atividades que levam à concretização de sistemas de *software* bem estabelecidos [60]. Segundo o Guia SWEBOK [61], "a modelação é uma técnica universal para ajudar engenheiros de software a entender, conceber e comunicar aspetos do software às partes interessadas". O SWEBOK divide a modelação nos seguintes tópicos: princípios de modelação; propriedades e expressão de modelos; sintaxe, semântica e pragmática de modelação; pré-condições, pós-condições e invariantes. A criação de modelos de um problema do mundo real é fundamental para análise de requisitos de software. O objetivo da modelação é ajudar a entender a situação em que o problema ocorre, bem como descrever uma solução [62].

Durante o desenvolvimento de *software*, muita comunicação ocorre, sendo apoiada por vários tipos de notações para transmitir a mensagem. Um esboço de um *layout* do ecrã pode suportar a comunicação entre um utilizador e um engenheiro de requisitos ou uma descrição muito mais formal de interfaces de classe pode suportar a comunicação entre um *designer* e um desenvolvedor [63].

Sommerville [15] define modelação de sistemas como um processo de desenvolvimento de modelos abstratos de um sistema, com cada modelo a apresentar uma visão ou perspetiva diferente desse sistema. Geralmente, a modelação de sistemas significa representar o sistema usando algum tipo de notação gráfica, que é quase sempre baseada em notações da *Unified Modeling Language* (UML).

As notações mais comuns usadas para dar suporte à comunicação entre os vários interessados no desenvolvimento de *software* usam muitas vezes algum tipo de Diagrama de Entidade e Relacionamento (DER). Por vezes, esses diagramas têm semântica muito informal [62]. Por exemplo, as entidades podem denotar partes do sistema, onde não está claro o que é exatamente um elemento. Uma entidade pode denotar um subsistema principal, outra entidade pode denotar o conjunto de medidas de segurança tomadas. Da mesma forma, linhas podem denotar uma relação de partes, uma relação de chamada, uma relação de uso e assim por diante [63].

Segundo Sommerville [22], os modelos são usados durante o processo de engenharia de requisitos para ajudar a recolher os requisitos de um sistema, durante o processo de *design*, para descrever o sistema aos engenheiros que o implementam e após a implementação para documentar a estrutura e o funcionamento do sistema.

O aspeto mais importante de um modelo de sistema é que ele não se foca nos detalhes. Um modelo é uma abstração do sistema em estudo e não uma representação alternativa desse sistema [60]. Idealmente, uma representação de um sistema deve manter todas as informações sobre a entidade que está a ser representada. Uma abstração propositadamente simplifica e seleciona as características mais importantes [22].

Nesta secção, são discutidas várias notações de modelação semiformais. Algumas usam um *layout* mais textual. Os diagramas e esquemas são geralmente desenvolvidos durante a engenharia de requisitos e o *design* [64]. Algumas servem principalmente a engenharia de requisitos. Por exemplo, os diagramas de casos de uso geralmente são utilizados durante a engenharia de requisitos. Os diagramas de estrutura de Jackson, por outro lado, são usados principalmente durante o *design*. Muitas notações de modelação servem para ambas as fases [63].

Atualmente, as principais notações de modelação são provenientes da *Unified Modeling Language (*UML). Muitos diagramas da UML, no entanto, são baseados ou derivados notações mais antigas. E certamente, em aplicações herdadas, é possível deparar-se com muitas dessas notações mais antigas [63].

Os fatores principais que influenciam na escolha da notação de modelação incluem [61]:

**A natureza do problema**. Alguns tipos de *software* requerem que determinados aspetos sejam analisados de forma particularmente rigorosa. Por exemplo, modelos paramétricos, que pertencem ao SysML, provavelmente serão mais importantes para *software* embebido do que para sistemas de informação, enquanto normalmente será o oposto para modelos de objetos e de atividades;

**A experiência do engenheiro de *software***. Geralmente, é mais produtivo adotar uma notação ou método de modelação com o qual o engenheiro de *software* tenha experiência;

**Os requisitos de processo do cliente**. Os clientes podem impor sua notação ou método preferido ou proibir qualquer um com o qual não estejam familiarizados.

A modelação de *software* é cada vez mais uma técnica difundida para ajudar os engenheiros de *software* a entender, projetar e comunicar aspetos essenciais do *software* aos *Stakeholders.* Os *Stakeholders* são pessoas ou partes com interesse explícito ou implícito no projeto de *software* (e.g., utilizador, comprador, fornecedor, arquiteto, avaliador, autoridade certificadora, desenvolvedor, engenheiro de *software* e outros). Embora existam muitas linguagens, notações, técnicas e ferramentas de modelação na literatura e na prática, existem conceitos gerais uniformes que se aplicam de alguma forma a todos eles [19], [62].

### 2.4.1 Princípios de Modelação

A modelação é uma abordagem organizada e sistemática para representar aspetos significativos do *software* em estudo, facilitando a tomada de decisões sobre o *software* ou seus elementos e facilitando a comunicação dessas decisões a outras pessoas na comunidade dos *Stakeholders*. Há três princípios gerais que orientam as atividades de modelação [61]:

1. ***Model the Essentials*:** bons modelos geralmente não representam todos os aspetos ou recursos de *software* sob todas as condições possíveis. A modelação envolve tipicamente o desenvolvimento de aqueles aspetos ou características do *software* que precisam de respostas específicas, omitindo qualquer informação não essencial;
2. ***Provide Perspective**:* a modelação fornece demonstrações do *software* em análise usando um conjunto definido de regras para expressão do modelo. Essa abordagem orientada por perspetiva fornece dimensionalidade ao modelo (e.g., visão estrutural, visão comportamental, visão temporal, visão organizacional e outras visualizações relevantes). Organizar as informações em visualizações concentra os esforços de modelação de *software* em preocupações específicas relevantes para essa visão, usando a notação, o vocabulário, os modelos e as ferramentas apropriados;
3. ***Enable Effective Communications:*** a modelação deve utilizar o vocabulário específico do domínio. Quando usada rigorosa e sistematicamente, essa modelação resulta numa

abordagem que facilita a comunicação efetiva de informações para as partes interessadas do projeto.

Um modelo é uma abstração ou simplificação do sistema, do *software* ou de um componente de *software*.

### 2.4.2 Técnicas de Modelação Clássicas

São apresentadas quatro técnicas clássicas de modelação [63], técnicas estas que já existem há um muito tempo:

**Diagrama de entidades e relacionamentos (ERD - *Entity Relationship Diagram***). É uma técnica de modelação de dados, iniciada por Chen nos anos setenta. Os diagramas de classe UML são baseados no ERD.

**Máquina de estados finitos (FSM – *Finite State Machine*).** É usada para modelar estados e transições de estado. Nos primeiros tempos do desenvolvimento de software, certos tipos de linguagens formais, por exemplo usadas em compiladores, eram modeladas como máquinas de estados finitos. Os diagramas de máquinas de estado do UML são baseados em máquinas de estados finitos.

**Diagramas de fluxo de dados (DFD – *Data Flow Diagram*).** Modelam um sistema como um conjunto de processos e fluxos de dados que conectam esses processos. É a notação usada na conceção do fluxo de dados.

**Cartões CRC (*Class-Responsibility-Collaboration*).** São uma ferramenta simples de obtenção de requisitos. Muitas das informações recolhidas em cartões CRC podem ser representadas em diagramas de comunicação UML.

Muitas outras notações de modelação clássicas existem. Muitas destas estão ligados a uma determinada abordagem ou método de desenvolvimento [63]. As notações detalhadas a seguir são independentes do método [63].

❖ **Diagrama de entidades e relacionamentos.** Em sistemas com uso intensivo de dados, a modelação da estrutura de dados é uma preocupação importante [63]. Até à década de 1970, as

técnicas de modelação de dados misturavam muito as preocupações de implementação com as preocupações decorrentes da estrutura lógica do UoD (*Universe of Discourse*).

O diagrama de entidades e relacionamentos (ERD), na forma pioneira de Chen, é direcionado a modelar a estrutura lógica semântica do UoD, em vez da sua realização em algum sistema de base de dados. Os ERDs representam modelos das entidades e os seus relacionamentos. Existem muitas variantes do ERD, que diferem nas suas notações gráficas e extensões da abordagem original de Chen [63]

De acordo Kufner e Vogt [63] uma entidade é algo que pode ser definido de maneira exclusiva. As entidades geralmente são representadas num ERD como retângulos.

As entidades possuem propriedades conhecidas como atributos. Por exemplo, alguns funcionários da biblioteca podem ter o nome "Manuel". Aqui, "Manuel" é o valor do atributo chamado "nome". Atributos são geralmente descritos como círculos ou elipses [63].

O diagrama de entidades e relacionamentos impõem restrições à cardinalidade dos relacionamentos. Na sua forma mais simples, os relacionamentos são 1 - 1, 1 - N ou N - M. Num ERD, essas restrições de cardinalidade são frequentemente indicadas por pequenos adereços das setas que ligam as entidades (Figura 9).

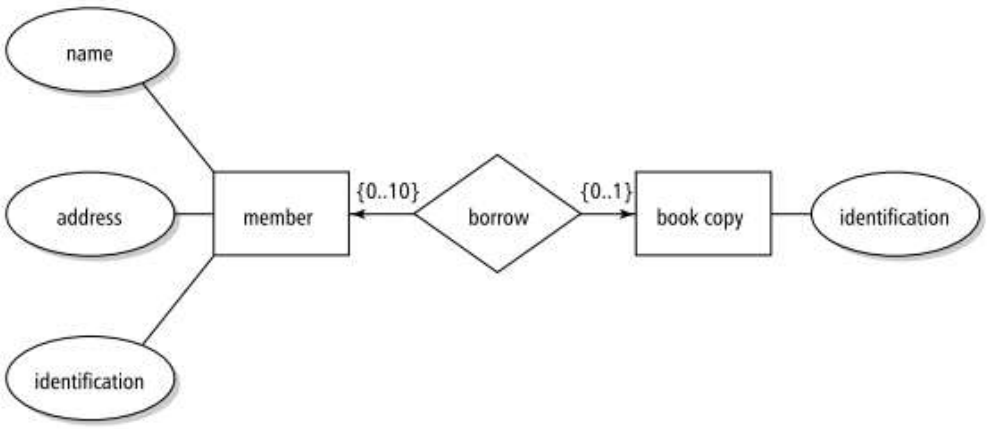



*Figura 9- Exemplo de um diagrama de entidades e relacionamento* [46]

O ERD atual tem muito em comum com técnicas de análise orientadas a objetos. Por exemplo, as relações de subtipo - supertipo entre tipos de entidade são incluídas em muitas técnicas de ERD. Por outro lado, o diagrama de classes da UML inclui muitos elementos do ERD [63].

❖ **Máquinas de estados finitos.** De acordo com Kufner e Vogt [63], as técnicas de especificação de requisitos que modelam um sistema em termos de estados e transições entre estados são chamadas técnicas de modelação baseadas em estado. Um formalismo simples para especificar estados e transições de estado é a Máquina de Estado Finitos (FSM). Uma FSM consiste num número finito de estados e um conjunto de transições de um estado para outro que ocorrem nos sinais de entrada de um conjunto finito de estímulos possíveis [63].

Com a evolução do sistema, durante o seu ciclo de vida, os objetos respondem a eventos, a interrupções, a invocação de operações ou apenas ao passar do tempo. Quando um evento sucede, desencadeia uma atividade que pode alterar o estado interno dos objetos em causa [60].

O estado inicial é um estado especialmente designado no qual a máquina é iniciada. Geralmente, um ou mais estados são designados como estados finais [63].

Em UML os diagramas de estado (Figura 10) são importantes na medida em que permitem fazer a descrição do comportamento dinâmico dos objetos. Esta informação não está presente nos outros diagramas, de forma tão direta e intuitiva [60].

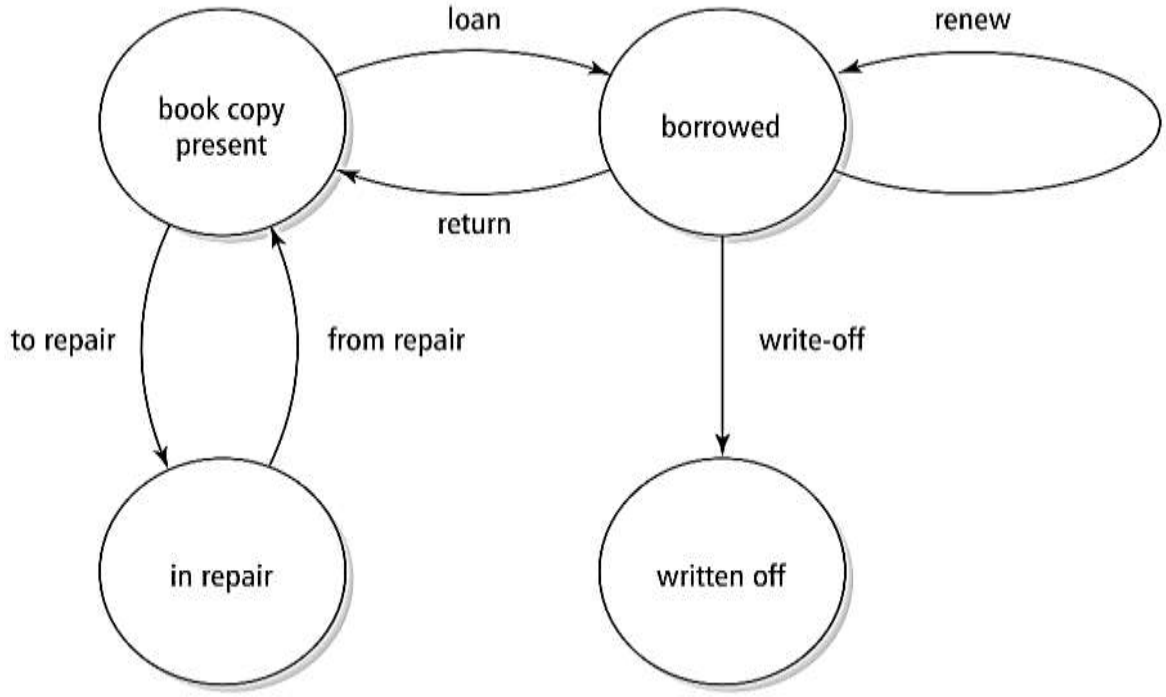



*Figura 10 - Exemplo de um diagrama de transição de estado* [46]

Kufner e Vogt [63] não aconselham a modelação de um sistema grande e monolítico usando FSM. Esta estrutura certamente será complexa e difícil de entender. Embora seja possível modelar o sistema numa série de FSMs, ainda há o problema de como integrá-los num só modelo [63].

Uma saída possível será permitir uma decomposição hierárquica dos FSMs. Isto é a essência de uma notação conhecida como *statecharts* [63]. Em gráficos de estado, grupos de estados podem ser vistos como uma entidade única num nível, para serem refinados no próximo nível de abstração. Na UML, os FSMs são modelados no diagrama de estados [60], [63].

❖ **Diagrama de Fluxo de Dados (DFD**). A técnica DFD teve origem no início dos anos 70 com Yourdon e Constantine [61]. Na sua forma mais simples, um *DFD* é apenas uma decomposição funcional em relação aos fluxos de dados [63]. Um componente (módulo) é uma caixa preta que transforma um fluxo de entrada num fluxo de saída

Os diagramas de fluxo de dados resultam de um processo de decomposição *top-down*. O diagrama do nível mais alto tem apenas um processo, denotando "o sistema". Em seguida, esse diagrama de nível superior é posteriormente decomposto em vários níveis com um maior detalhe [63].

❖ **Cartões CRC.** CRC significa, Classe – Responsabilidade – Colaboradores [63]. Um cartão CRC é simplesmente um cartão ordenado com três campos chamados Classe, Responsabilidade e Colaboradores. Os cartões CRC foram desenvolvidos em resposta a uma necessidade de documentar decisões de *design* colaborativo. Os cartões CRC são especialmente úteis nas primeiras fases do desenvolvimento de *software*, para ajudar a identificar componentes, discutir questões de *design* em equipas multidisciplinares e especificar componentes informalmente.

Os cartões CRC não são usados apenas em sessões de *design*. Na comunidade de padrões de *design*, por exemplo, eles são usados para documentar os elementos que participam num padrão [63].

Os cartões CRC podem ser usados para descrever qualquer elemento de *design*. A Figura 11 apresenta um exemplo de um modelo CRC para um componente Reservas numa biblioteca [63].

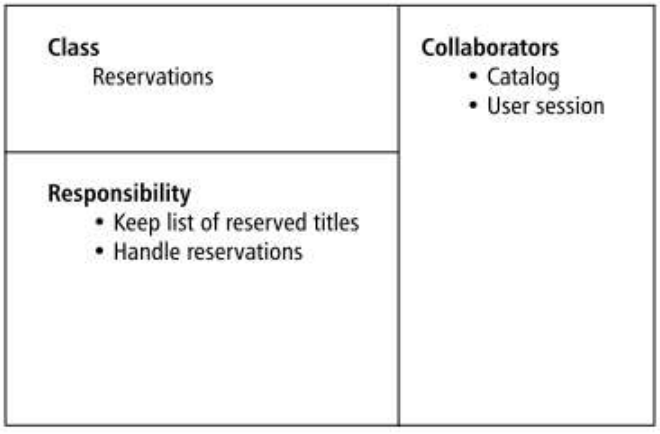



*Figura 11 - Exemplo de um cartão CRC* [46]

### 2.4.3 Tipos de Modelos

Um modelo típico consiste numa agregação de submodelos. Cada submodelo é uma descrição parcial e é criado para um propósito específico, pode ser composto por um ou mais diagramas. A coleção de submodelos pode aplicar várias linguagens de modelação ou uma única linguagem de modelação [19]. A *Unified Modeling Language* (UML) apresenta uma vasta coleção de diagramas de modelação. O uso desses diagramas, juntamente com os construtores da linguagem de modelação, traz três tipos de modelos mais usados segundo o SWEBOK [61]: modelos de informação, modelos comportamentais e modelos de estrutura.

**Modelos de Informação.** Os modelos de informação fornecem um foco central em dados e informação. Um modelo de informação é uma representação abstrata que identifica e define um conjunto de conceitos, propriedades, relações e restrições em entidades de dados. O modelo de informação semântica ou conceptual é frequentemente usado para fornecer algum formalismo e contexto no mundo real. Transformações subsequentes do modelo de informação semântica ou conceptual levam à elaboração de modelos de dados lógicos e físicos, posteriormente implementados no *software* [61].

**Modelos Comportamentais**. Os modelos comportamentais identificam e definem as funções do *software* que está a ser modelado. Os modelos comportamentais geralmente assumem duas formas básicas: máquinas de estado e modelos de fluxo de controlo de modelos de dados. Os modelos de fluxo de controlo descrevem como a sequência de eventos faz com que os processos sejam ativados ou desativados. O comportamento do fluxo de dados é tipificado como uma sequência de etapas em que os dados passam pelos processos em direção a armazenamentos de dados [61].

**Modelos de Estrutura.** Os modelos de estrutura ilustram a composição física ou lógica do *software* nas várias partes dos componentes. A modelação de estrutura estabelece o limite definido entre o *software* que está a ser implementado ou modelado e o ambiente no qual ele deve operar. Alguns construtores estruturais comuns usados na modelação de estrutura são composição, decomposição, generalização e especialização de entidades; identificação de relações relevantes e cardinalidade entre entidade; definição de processos ou interfaces

funcionais. Os diagramas de estrutura fornecidos pela UML para modelação de estrutura incluem diagramas de classes, componentes, objeto, implementação e diagrama de *packages* [61].

Em síntese, pode afirmar-se que as atividades de modelação influenciam decisivamente na qualidade do sistema final e que a própria escolha do processo de modelação e dos modelos utilizados estabelece uma orientação metodológica importante.

### 2.4.4 Modelação baseada em UML

Nesta secção, são discutidos os diagramas definidos na UML [65], que se tornou numa linguagem de modelação padrão para modelação orientada a objetos. A UML tem vários tipos de diagrama, e portanto, suporta a criação de muitos tipos diferentes de modelos do sistema [38], [60]:

1. Diagramas de atividades (*activity diagrams*), que mostram as atividades envolvidas num processo ou no processamento de dados;
2. Diagramas de casos de uso (*use case diagrams*), que mostram as interações entre um sistema e o seu ambiente;
3. Diagramas de sequência (*sequence diagrams*), que mostram interações entre os atores, o sistema e os componentes do sistema.
4. Diagramas de classes (*class diagrams*), que mostram as classes de objetos no sistema e as associações entre essas classes;
5. Diagramas de estados (*statechart diagrams*), que mostram como o sistema reage a eventos internos e externos.
6. Diagramas de objetos (*Object diagrams*), permite representar um conjunto de objetos e mostrar um exemplo de interação entre eles. Representa um momento na interação de objetos (um *snapshot*).

No sentido de se evitar a que esta secção se torne demasiado extenso, o foco é perceber como os seis tipos principais de diagramas UML são usados na modelação de sistemas. No entanto existem mais diagramas previstos na notação.

Ao desenvolver modelos do sistema, muitas vezes é preciso ser flexível na forma como a notação gráfica é usada. Nem sempre é preciso seguir de modo rígido os detalhes de uma

notação. O detalhe e o rigor de um modelo dependem de como se pretende usá-lo. Existem três formas pelas quais os modelos podem ser usados [22]:

1. Como meio para facilitar a discussão sobre um sistema existente ou proposto;
2. Como forma de documentar um sistema existente;
3. Como uma descrição detalhada do sistema que pode ser usada para desenvolver uma implementação do sistema.

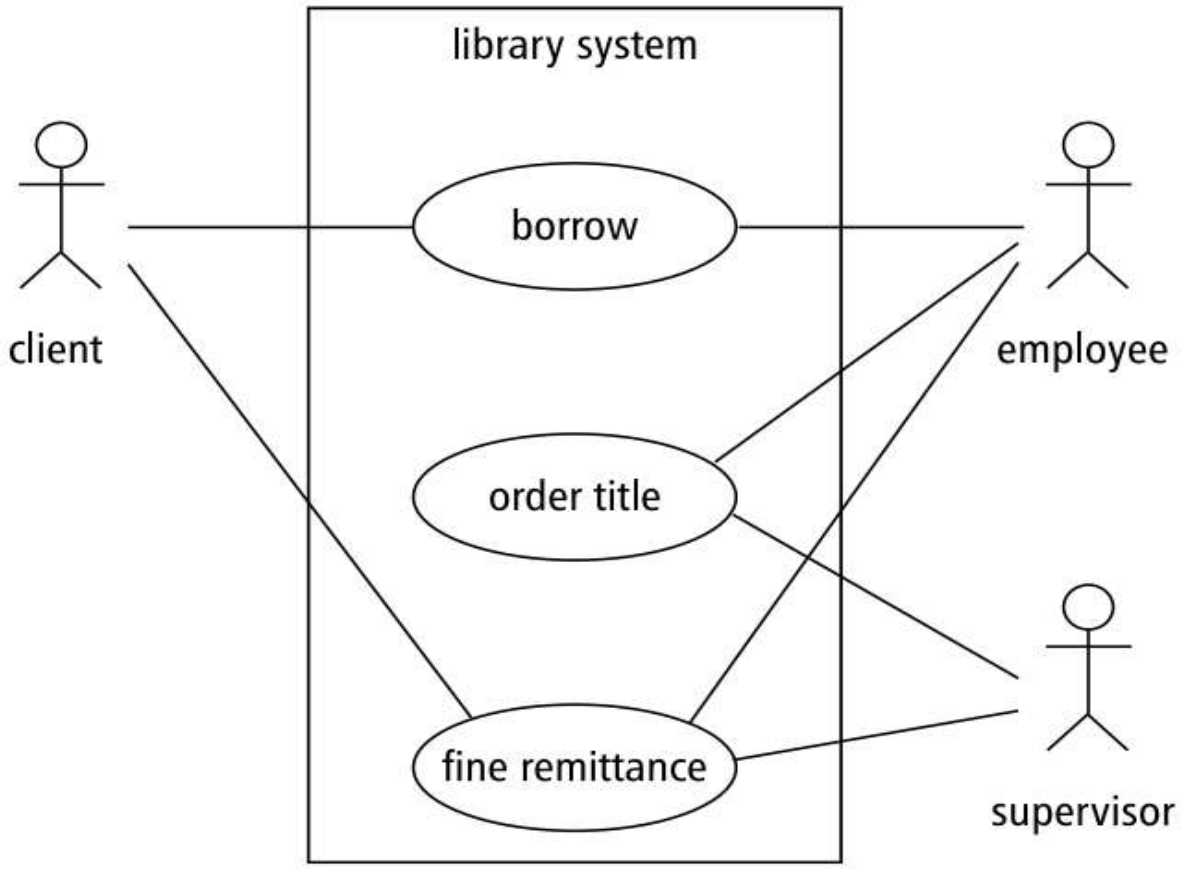
No primeiro caso, o objetivo do modelo é estimular a discussão entre os engenheiros de *software* envolvidos no desenvolvimento do sistema e o cliente. Os modelos podem estar incompletos (desde que cubram os pontos-chave da discussão) e podem usar a notação de modelação informalmente [15]. É assim que os modelos são normalmente usados na chamada "modelação ágil" [66]. Quando os modelos são usados como documentação, eles não precisam ser completos, pois pode-se desejar apenas desenvolver modelos para algumas partes de um sistema [15].

No terceiro caso, onde os modelos são usados como parte de um processo de desenvolvimento baseado num modelo, os modelos do sistema devem ser completos, formais e corretos. A razão para isso é que eles são usados como base para gerar o código-fonte do sistema. Portanto, é necessário ter o máximo rigor uma vez que a geração pode ser automatizada [22], [67].

**Diagramas UML**

Abordam-se de seguida os diagramas UML mais importantes, de forma concisa e breve, permitindo ilustrar os aspetos mais significativos dos mesmos.

❖ ***Diagrama de Casos de Uso.*** Um caso de uso (UC – *Use Case*) representa uma interação entre um utilizador e o sistema [65]. A identificação dos UCs é muito importante na medida em que desta forma se recolhem os requisitos funcionais do sistema que se vai modelar. Uma mais-valia deste processo de recolha de requisitos está relacionada com o facto de que é nesta fase que o cliente, ou utilizadores, mais intervêm no processo. É indicativo que a captura de requisitos seja feita utilizando o vocabulário do utilizador, permitindo assim diminuir significativamente a




distância entre a equipa de projeto e os seus conceitos, e a visão do sistema fornecida por parte de quem o vai utilizar [60].

*Figura 12 - Exemplo de um diagrama de casos de uso* [60]

Os diagramas de casos de uso (Figura 12) são utilizados para ilustrar as respostas de um sistema. Os diagramas de UC são especialmente importantes na organização e modelação de comportamentos esperados do sistema [65].

❖ **Diagrama de Sequência.** Um diagrama de sequência é um diagrama de interação que enfatiza a ordem do tempo das mensagens [60]. Um diagrama de sequência mostra um conjunto de objetos e as mensagens enviadas e recebidas por esses objetos [68]. Os objetos são geralmente

instâncias nomeadas ou anônimas de classes, mas também podem representar instâncias de outras coisas, como colaborações, componentes e nós. Os diagramas de sequência são usados para ilustrar a visualização dinâmica de um sistema [65].

A Figura 13 apresenta um diagrama de sequência onde se ilustra a maioria dos conceitos presentes nestes diagramas: objetos, mensagens, períodos de inatividade e anotações textuais [60].

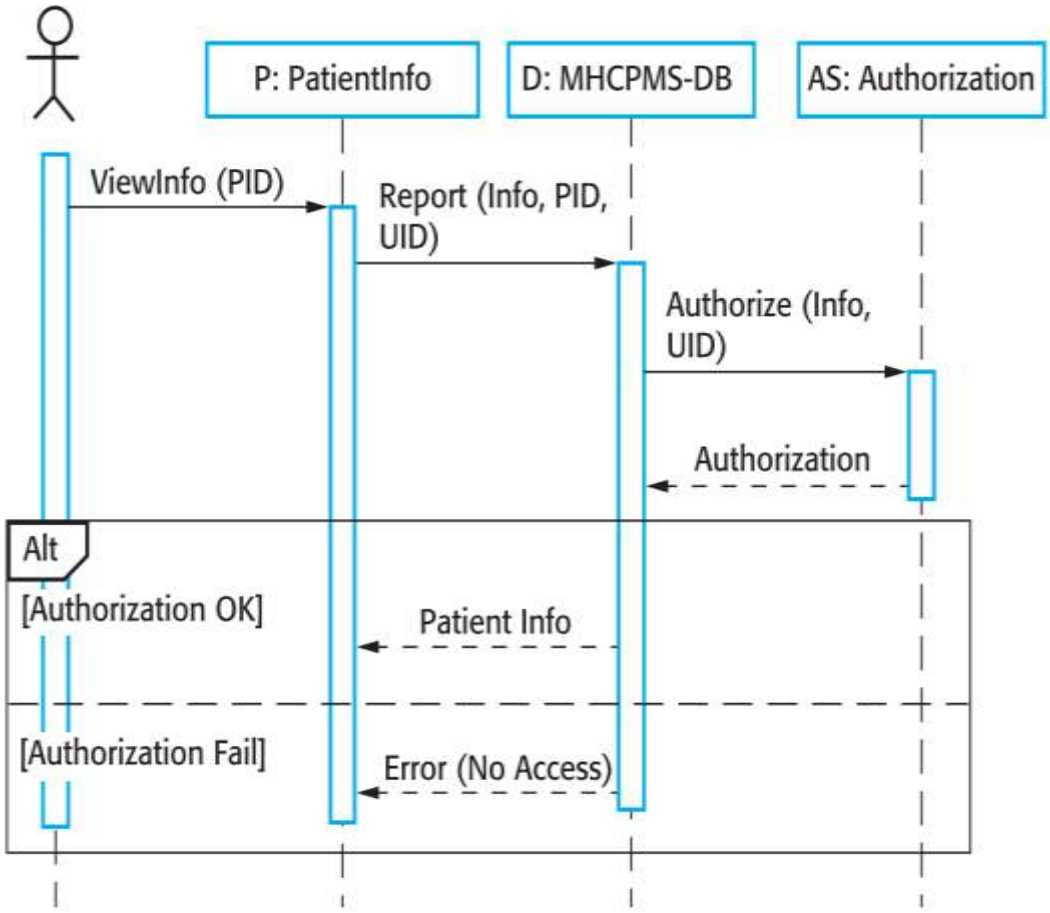



*Figura 13 - Exemplo de um diagrama de sequência* [15]

Os diagramas de sequência são utilizados sempre que se pretende detalhar o comportamento de vários objetos no contexto de um cenário de um caso de uso. Permitem estabelecer as características base da colaboração entre os objetos e são uma representação de alto nível do algoritmo que vai ser implementado [60].

Diagrama de interação é o nome dado a todos os diagramas de sequência juntamente com os diagramas de colaboração. Todos os diagramas de sequência e colaborações são diagramas de interação [65]

❖ **Diagrama de Classes.** Os diagramas de classe existentes em UML são semelhantes aos diagramas encontrados em todas as metodologias orientadas aos objetos. O conceito de classe é um conceito central no paradigma e qualquer metodologia orientada aos objetos privilegia esta vista pois está muito próxima dos conceitos do paradigma e das próprias linguagens de programação [60].

O diagrama de classes ilustra a componente estrutural do sistema e identifica claramente as classes, interfaces e respetivas relações existentes no sistema [65]. Os diagramas de classe são também um mecanismo necessário para a criação de outros diagramas, como sejam os de componentes e de implementação (*deployment*) [60].

Em UML existem genericamente cinco tipos diferentes de relacionamento entre as classes (Figura 14).

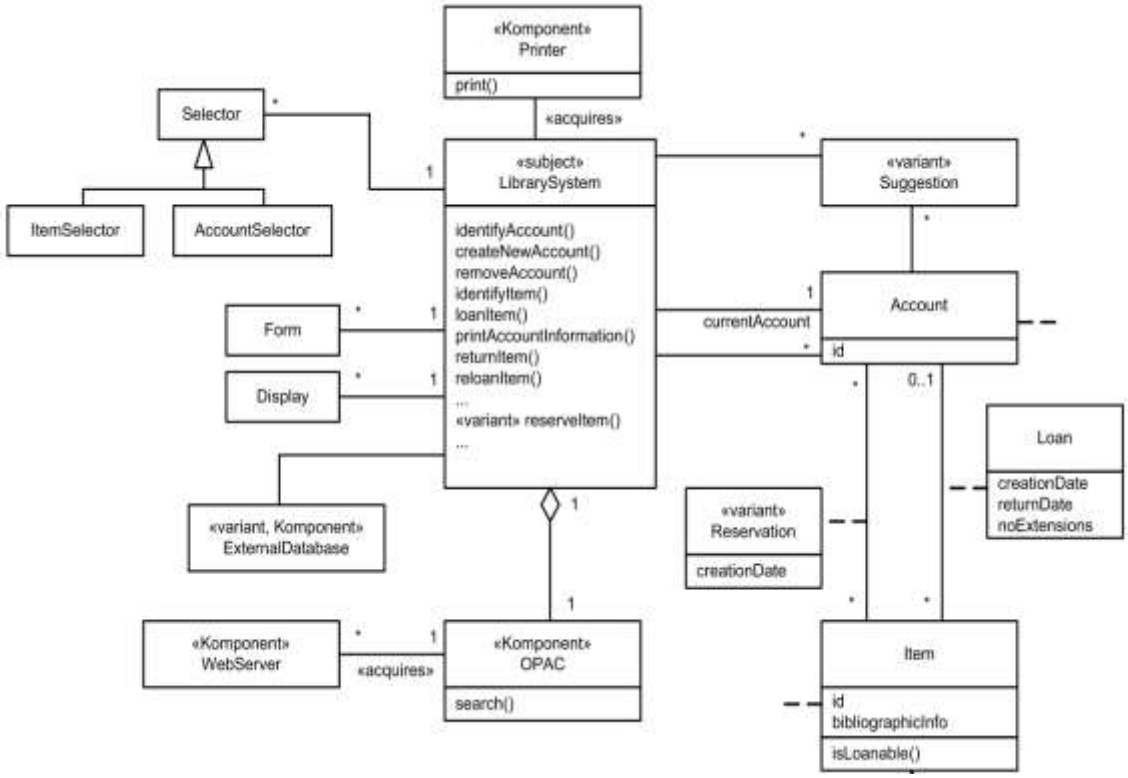



*Figura 14 - Diagrama de classes com relações de diverso tipo* [60]

Uma das vantagens dos diagramas de classes é o facto de permitirem que lhes seja adicionada informação sobre o estado dos objetos, seus tipos de dados e nível de visibilidade, sobre as operações e a sua assinatura. Permitem efetuar num nível visual o mesmo esforço que um programador faz a nível de uma linguagem de programação orientada a objetos, apenas não descrevendo os algoritmos das operações [60].

## 2.5 A Técnica *Four Step Rule Set – 4SRS*

A Técnica *Four Step Rule Set* - 4SRS [68], [70] é essencialmente baseada no mapeamento de diagramas de casos de uso em diagramas de objetos. Os diagramas de sequência, atividade e estado do UML e outros artefactos também podem ser considerados nas decisões de transformação.

Transformar casos de uso em modelos de arquitetura é uma tarefa difícil (conceção da solução), a técnica 4SRS (conjunto de quatro etapas) ajuda nessa tarefa. Em alto nível, a técnica 4SRS é organizada em quatro etapas (*Step 1 - Object creation, step 2 - Object elimination, step 3 - Object packing & aggregation e step 4 - Object association*) para transformar casos de uso em objetos [72].

### 2.5.1 *Step 1 - Object creation*

Nesta etapa, cada caso de uso deve ser transformado em três objetos (um de interface, um de dados e um de controlo). Cada objeto recebe a referência do seu respetivo caso de uso anexando-se um sufixo (i, d, c) que indica a categoria de objeto [72].

### 2.5.2 *Step 2 - Object elimination*

Nesta etapa, deve-se decidir qual dos três objetos deve ser mantido para representar totalmente, em termos computacionais, o caso de uso, levando em consideração todo o sistema e sem considerar cada caso de uso isoladamente. Essas decisões devem ser baseadas na descrição textual de cada caso de uso. Esta etapa tem como objetivo decidir quais dos objetos criados na

etapa anterior devem ser mantidos no modelo do objeto. Esta etapa também suporta a eliminação de redundância na solicitação do utilizador, bem como a descoberta de requisitos em falta [71].

### 2.5.3 *Step 3 - Object packing & aggregation*

Nesta etapa, os objetos não eliminados (aqueles que foram mantidos após a execução da etapa 2), para os quais existem vantagens em ser tratados de maneira unificada, devem dar origem a agregações ou pacotes de objetos semanticamente consistentes. Esta etapa suporta a construção de um modelo de objetos potencialmente coerente, pois ajuda na introdução de uma camada semântica adicional num nível de abstração mais alto, que funciona como uma "cola funcional" para os objetos [72].

O packing é uma técnica relativamente imatura, uma vez que introduz uma coesão semântica leve entre objetos. Essa coesão pode ser facilmente revertida na fase de *design*, sempre que necessário. Isso significa que o *packing* pode ser usada com flexibilidade para permitir a obtenção temporária de modelos de objetos mais abrangentes e compreensíveis [72].

Por outro lado, a agregação impõe uma forte coesão semântica entre objetos. O nível de coesão nas agregações é mais difícil de reverter nas próximas fases do projeto, o que sugere uma abordagem mais escrupulosa ao usar esse tipo de "cola funcional". Isso significa que a agregação deve ser usada apenas quando for assumido explicitamente que o conjunto de objetos considerados é afetado por uma decisão de *design* consciente [70].

### 2.5.4 *Step 4 - Object association*

Esta etapa final da técnica 4SRS suporta a introdução de associações no modelo de objetos, completamente baseadas nas informações existentes no modelo de caso de uso.

Em relação às informações do modelo de caso de uso, se as descrições textuais dos casos de uso possuírem dicas sobre o tipo de sequência em que os casos de uso são inseridos, essas informações devem ser usadas para suportar a inclusão de associações no modelo de objeto [72].

## 2.6 Conclusão

Neste capítulo pretendeu-se dar uma panorâmica sobre as metodologias ágeis de desenvolvimento de *software* e das técnicas de modelação de *software*. É importante perceber como é abordado o processo de desenvolvimento de sistemas de *software.*

Tendo em linha de conta os objetivos desta dissertação, apresentou-se as metodologias ágeis assim como as técnicas de modelação de *software*. Deu-se especial destaque à metodologia *Scrum* como sendo de importância vital tendo em conta ao tema deste trabalho. O capítulo que se segue faz um estudo detalhado dos trabalhos relacionados com o tema em discussão.

# 3. ESTUDO DAS ABORDAGENS EXISTENTES SOBRE AMBIENTES VISUAIS

Este capítulo pretende descrever alguns trabalhos que abordam o tema em discussão. Foram selecionados e estudados os trabalhos mais relevantes e dentre esses, são analisados com detalhe 4 trabalhos que apresentam segurança em relação à sua idoneidade (por exemplo, revista científica para onde foi publicado o artigo científico). Os modelos que serão apresentados transformam requisitos textuais, escritos em linguagem natural (NL), em modelos UML. É feita uma breve introdução sobre *user stories* na secção 3.1 e depois, nas secções seguintes são descritos os trabalhos selecionados. Na secção 3.6 é feita uma análise comparativa das abordagens estudadas no sentido de se avaliar as vantagens e desvantagens das mesmas. Na secção 3.7 são descritas outras abordagens não menos importantes, mas que não mereceram uma atenção especial como as outras 4 por não estarem muito relacionados com a metodologia *Scrum*.

## 3.1 *User Stories (US)*

As *user stories* são frases simples em linguagem natural para descrever com detalhe suficiente o conteúdo de uma *feature* a ser implementado. As frases geralmente contêm três elementos descritivos da funcionalidade: quem (*who*), o que (*what)*, porquê (*why*). É suficiente para descrever a frase com a seguinte estrutura: como um "quem", eu quero "o que" para que "porquê" [73].

O "quem" é o papel do utilizador do sistema. O "que" descreve a ação que deve ser possibilitada pelo sistema e permite ao utilizador atingir o objetivo mencionado no "porquê". O "porquê" tem a vantagem de garantir que a funcionalidade descrita responda a uma necessidade concreta.

As *user stories* devem ser independentes umas das outras, o que permitirá que a equipa as concretize e a entregue sem afetar o conjunto de entregas. A principal vantagem dessa abordagem é que o foco é o utilizador do sistema.

Espera-se que cada US, uma vez implementada, acrescente valor para o produto, independentemente da ordem de implementação[3].

De acordo com Lucassen [73], *user stories* são uma notação textual cada vez mais adotada para captura de requisitos no desenvolvimento de *software.* Os requisitos escritos em NL são fáceis de ler, mas apresentam uma grande desvantagem [74]: à medida que são escritas mais *US*, também aumenta a quantidade de conceitos envolvidos o que dificulta a construção de modelos precisos desses conceitos [75].

### 3.1.1 ***User Stories* não são requisitos**

A sociedade de Computação do Instituto de Engenheiros Elétricos e Eletrónicos (IEEE) publicou um conjunto de orientações sobre como escrever especificações de requisitos de *software* [76]. Este documento, conhecido como padrão IEEE 830, traz recomendações que abrangem temas como a forma de organizar o documento de especificação de requisitos, o papel dos protótipos, e as características dos bons requisitos. A característica mais distintiva do padrão IEEE 830 na especificação de requisitos é o uso da frase " O sistema deve [...]", que é a maneira recomendada pelo IEEE para escrever requisitos funcionais [37].

Um excerto típico de uma especificação IEEE 830 é semelhante ao seguinte [37]:

O sistema deve permitir que uma empresa pague por um anúncio de emprego com cartão de crédito.

- O sistema deve aceitar cartões Visa, MasterCard e American Express;
- O sistema deve cobrar o cartão de crédito antes que o lançamento do trabalho seja colocado no site;
- O sistema deve fornecer ao utilizador um número de confirmação exclusivo.

Enquanto os requisitos sugerem o que deve ser feito, as *user stories* focam nos objetivos, e isso torna a visão do produto completamente diferente. Ao concentrar-se nos objetivos do utilizador para o novo produto, ao invés de uma lista de atributos do novo produto, segundo Beck [3] pode-se projetar uma melhor solução para as necessidades do utilizador.

A diferença fundamental entre as *user stories* e o padrão IEEE 830 de requisitos que foi abordada em [37] refere-se ao custo, uma vez que um ou mais analistas passam dois ou três meses a desenvolver o documento de requisitos. Este documento é então entregue aos programadores, que potencialmente descobrem que o projeto levará 24 meses, ao invés dos seis meses previstos.

## 3.2 **Análise de trabalhos selecionados**

Esta secção é dedicada a descrever os trabalhos selecionados que abordam o tema em análise. Cada subsecção corresponde ao título do respetivo trabalho, propositadamente, decidiu-se manter os títulos dos trabalhos discutidos na sua língua original.

### 3.2.1 ***Model – Driven Architecture***

*Model – Driven Architecture* (MDA) é uma abordagem para *design*, desenvolvimento e implementação de *software* liderada pela *Object Management Group* (OMG) [77]. O MDA fornece diretrizes para estruturação de especificações de *software* que são expressos como modelos [78].

O MDA separa a lógica do negócio das aplicações da tecnologia. Modelos independentes da plataforma de uma aplicação ou comportamento e funcionalidade de negócios do sistema integrado, construídos usando UML e os outros padrões de modelação OMG associados, podem ser realizados através do MDA em praticamente qualquer plataforma aberta, incluindo *Web Sevices*, NET, CORBA R, J2EE, XML *Metadata Interchange* (XMI) e outros. Esses modelos independentes da plataforma documentam a funcionalidade e o comportamento comercial de uma aplicação separada do código específico da tecnologia que a implementa, isolando o núcleo

da aplicação da tecnologia, permitindo a interoperabilidade dentro e fora dos limites da plataforma [7], [79]. A Figura 15 apresenta a estrutura de como os sistemas são construídos.


*Figura 15 - Model Driven Architecture [72]*

Têm sido estudadas nos últimos anos, várias abordagens para a transformação de requisitos de linguagem natural em diagramas UML, mas são poucos os investigadores que deram especial atenção aos requisitos ágeis, e em particular, às *user stories*. A maioria são unicamente centradas no utilizador final como única parte interessada. Ao longo dos anos, alguns modelos (na forma de conceitos que relacionam as dimensões de Quem o Quê e Porquê numa frase) foram propostos por praticantes de métodos ágeis ou académicos para orientar a recolha de requisitos. No entanto, utilizar esses modelos pode ser problemático pois, de facto, nenhum deles define qualquer semântica relacionada a uma sintaxe específica, rigorosa e formal [80].

Por um lado, e segundo Herchi e Abdessalem [81], "a transição dos requisitos do utilizador para os diagramas UML é uma tarefa difícil para os *designers*, especialmente quando se lida com textos grandes que expressam essas necessidades". Por isso é necessário a adoção de técnicas / abordagens que facilitem na extração de diagramas UML e que possam ser úteis na definição de um ambiente visual para apoiar o *Scrum*.

### 3.2.2 *Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques*

Esta secção define o processo de geração de diagramas de casos de uso a partir de *user stories* usando *Tree Tagger*[1]. O modelo proposto por Elallaoui, Nafil e Touahni [5] consiste na transformação de *user stories* em diagramas de casos de uso baseando-se na técnica de transformação de modelos de acordo com a abordagem da *Model - Driven Architecture* (MDA). Para fazerem isso, os autores utilizaram a técnica de Processamento de Linguagem Natural (NLP – *Natural Language Processing*), com aplicação do analisador *TreeTagger.*

**Processo de Transformação**

Dado que as *US* são expressas em frases que representam as necessidades do cliente em alto nível e, para melhorar a compreensão e a colaboração entre empresa, *Product Owner*, desenvolvedores e testadores estas frases devem ser transformadas em diagramas de casos de uso UML. Para reduzir significativamente o tempo de criação destes diagramas, Elallaoui, Nafil e Touahni [5] propõem uma transformação automática de *user stories* em casos de uso UML.

A transformação de *user stories* em diagramas de casos apresenta os seguintes recursos principais: a primeira etapa consiste em pré – processamento de um ficheiro de texto que contém um conjunto de *user stories*. Isso é feito usando um algoritmo que remove todas as palavras desnecessárias. Em seguida, o novo ficheiro é analisado usando o analisador *TreeTagger*, que produz a árvore de análise para cada *US*, através do qual o substantivo (NN), nome próprio (NP), determinante (DT) e verbo (VV) podem ser selecionados. Esta árvore de análise facilita a extração de atores, casos de uso e respetivos relacionamentos de associações com base num *plugin*. O diagrama de casos de uso é construído usando a tecnologia Java e aplicando *Tags POS*. Uma *tag POS* [2](ou *tag* da parte da fala) é uma etiqueta especial atribuído a cada *token* (palavra) num corpo de texto para indicar a parte da fala e muitas outras categorias gramaticais, como tempo, número (plural / singular), etc. A Figura 16 apresenta um diagrama de classes que representa o meta-modelo do *plugin* implementado para esta proposta em análise.

[1] É uma ferramenta para anotação de texto com a parte da fala (*parti - of - spech* em Inglês).
[2] http://nlp.stanford.edu/software/tagger.shtml

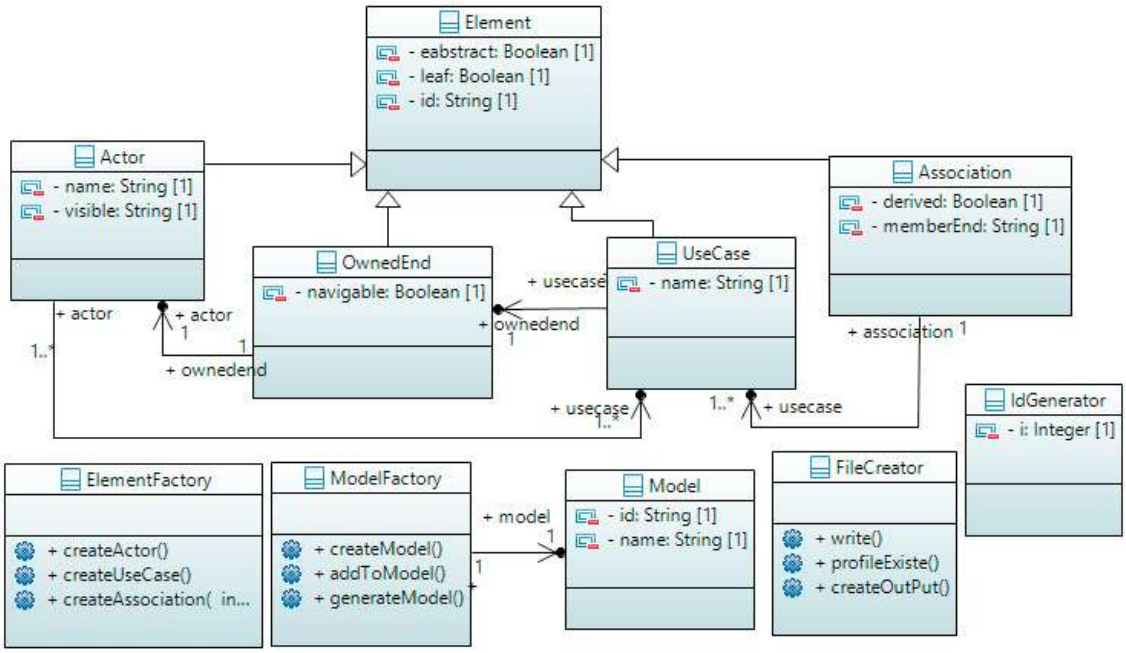



*Figura 16 - Meta-modelo do Plugin de caso de uso [4]*

Cada termo na *US* é classificado numa única parte da exposição. Como indicado na figura 17, foram implementados pelos autores, 10 classes em Java. A classe *FileCreato*r consiste na criação de ficheiros de saída física. A geração de Ids é executada usando a classe I*dGerator*. Para cada elemento do modelo, é inserido automaticamente um ID básico. A classe *ElementFactory* tem como objetivo criar todos os elementos do diagrama de casos de uso (criação de atores, casos de uso e suas associações). As associações foram apresentadas com fins (*sEnds*, *eEnd*), para associar o ator ao caso de uso correspondente (*sEnd = start End e eEnd = finish End*). A classe *ModelFactory* permite a criação de modelos, adição de elementos, que podem ser um ator, relação de associação, ou caso de uso, e finalmente gerar um modelo como ficheiro (.uml). Ator, relacionamento de associação e casos de uso descrevem o modelo de caso de uso. A classe Model contém um conjunto de atores, associações e casos de uso. A classe *OwnerEnd* especifica o final para o ator ou caso de uso. Na Figura 17 é apresentado o modelo de diagrama de casos uso de gerado pela abordagem descrita acima.

Para extrair diagramas de casos de uso UML automaticamente das *user stories*, o *plugin* apresentado por autores citados, não suporta frases que contenham mais de um substantivo composto (como “administrador de base de dados do administrador”).

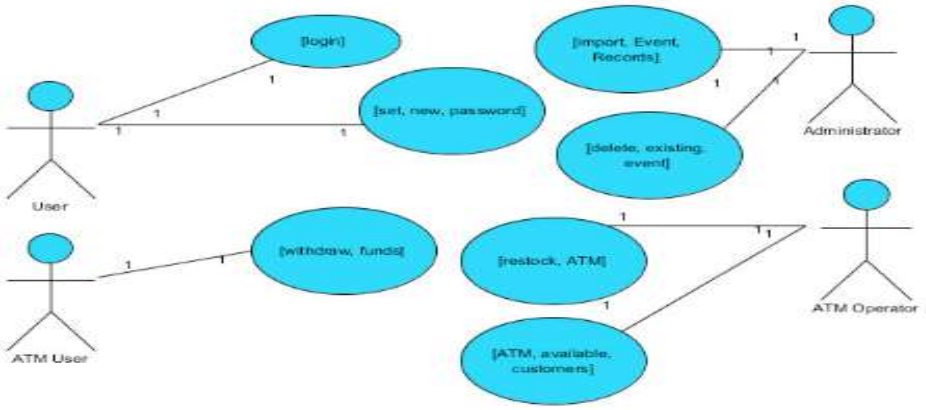



*Figura 17 - Diagrama de casos de uso gerado [4]*

O modelo apresentado por Elallaoui, Nafil e Touahni [5] embora apresente vantagens no que se refere a facilidade de análise das US e redução do tempo de geração de diagramas de casos de uso em relação à modelação manual, também apresenta algumas desvantagens como o não suporte de frases com substantivos compostos por parte do *plugin* e ainda, a abordagem não implementa outros relacionamentos importantes como generalização / especialização entre atores de casos de uso.

### *3.2.3 Automatic generation of UML sequence diagrams from user stories in Scrum process*

Elallaoui [82] apresentou uma abordagem que implementa um algoritmo para automatizar a transformação de US em diagramas de sequência na metodologia *Scrum*. O algoritmo implementado pode ler um ficheiro de texto contendo o conjunto de US, em seguida, gera um ficheiro XML para cada US. O formato do ficheiro XML resultante, é então transformado num diagrama de sequência usando o *Plugin* SDK da ferramenta UML 2 para o *Eclipse.*

**Processo de Transformação**

São discutidas as abordagens para automatização do processo de geração de diagramas de sequência UML para US na metodologia *Scrum*.

O primeiro passo é criar o *Product Backlog* (ver secção 2.2.5) que contém a lista de *user stories*, o *Product Owner* e a equipa realizam uma reunião de planeamento de *Sprint (Sprint Planning)* para planear o primeiro *Sprint* [56]. As US são escolhidas no *Product Backlog* e são transformadas (ver a Figura 18) em ficheiros XML. Os ficheiros XML resultantes são então transformados em diagramas de sequência.

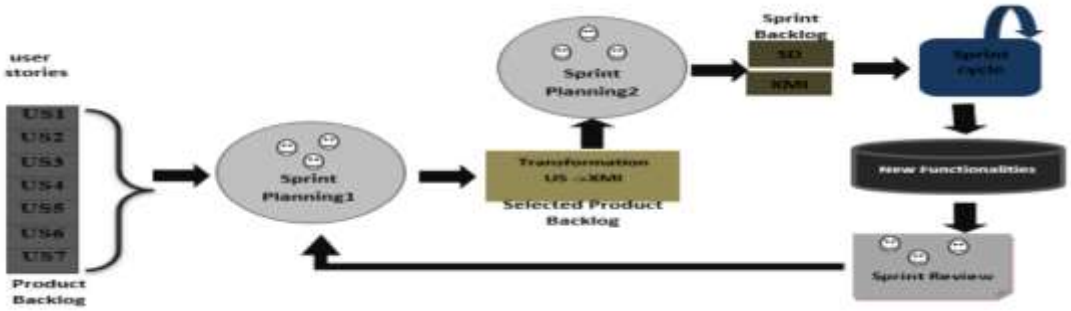



*Figura 18 - Automatização do processo de transformação de user stories [69]*

A técnica é baseada na seguinte sintaxe de user stories [74]: Eu (ator), quero (ação), de modo que (benefício). As palavras *ator* e *benefício* representam num diagrama de sequência, respetivamente Remetente e Mensagem.

Existem duas possibilidades para Ação:

Ação = Verbo + [Adjetivos possessivos] + Nome: neste caso, o verbo representa a ação e o nome representa o Recetor.

Ação = Verbo: neste caso, o verbo representa a ação e o sistema é recetor.

Vejamos a seguinte frase como exemplo que representa uma *user story*: "Como administrador e utilizador do sistema, quero recuperar o nome de utilizador ou senha da conta para executar a autenticação".

Analisando o texto acima, pode-se ver claramente que o verbo “executar “representa a ação, porque o objetivo da execução da autenticação é conseguir a autenticação no sistema.

A primeira etapa do algoritmo consiste em percorrer o ficheiro de texto que contém todas as frases no formulário de US, depois pesquisar o ator da primeira frase e inseri-lo na primeira linha do ficheiro XML criado. Em seguida, analisa a ação, se tiver a forma (verbo + adjetivo + nome), nesse caso o nome é inserido na segunda linha, caso contrário é o sistema que representa a linha. Essa técnica é feita para apresentar diagramas de sequência para cada US. Na figura 19 está o modelo que descreve o processo de transformação.

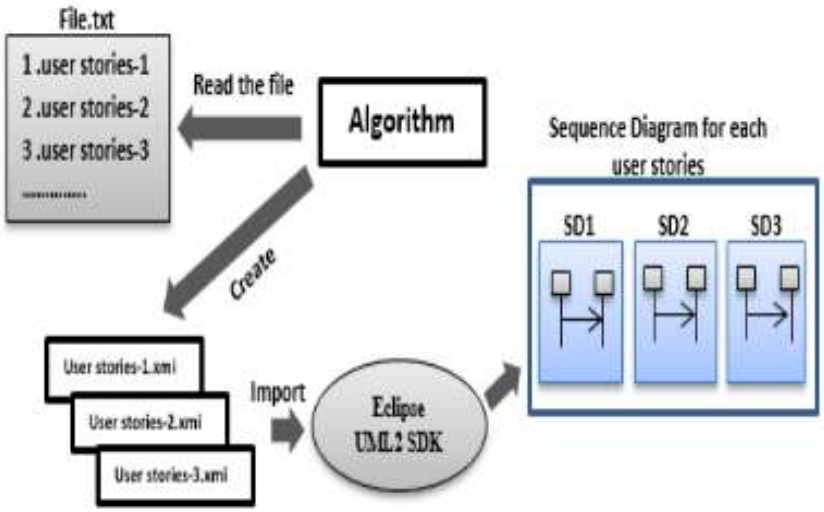



*Figura 19 - Transformação de user stories em diagramas de sequência [69]*

O formato de ficheiro XML resultante, é então, transformado num diagrama de sequência usando o *plugin* SDK da ferramenta UML 2 para Eclipse. Esta abordagem tem como seguintes vantagens: um especialista no domínio, ou um desenvolvedor, pode interpretar os diagramas de sequência de maneira precisa e explorável. Acredita-se que esta técnica poderá facilitar mais tarde a automação de testes na metodologia ágil *Scrum* [82].

### *3.2.4 Automatic builder of class diagram an application of UML generation from functional requirements*

Outra abordagem, é a proposta por Alattar e Norwawi [78]. Esta abordagem consiste em mapear os requisitos do utilizador para o diagrama de classes UML com base na abordagem *Model-Driven Architecture* (MDA). Os requisitos do utilizador são considerados como modelo de origem representado no texto processado (decomposição do texto em frases, palavras, categorias gramaticais, dependências sintáticas, etc.). O diagrama de classes UML é considerado como modelo de destino em conformidade com um ficheiro XML/XMI como meta-modelo de destino. As transformações de meta-modelos garantem um mapeamento do texto processado num ficheiro XMI.

Qualquer sistema baseado em MDA deve ter a capacidade de armazenar, gerir e publicar meta-dados no nível do sistema e aplicação, a fim de fornecer conversões precisas de PIM (*Platform Independent Model*) para PSIM (*Platform Specific Model*)[78].

No *framework* MDA, um modelo é sempre representado por um meta-modelo. O meta-modelo define a linguagem usada para explicar as relações entre os componentes do modelo [78]. A Figura 20 apresenta o processo de transformação MDA aplicado a este projeto.

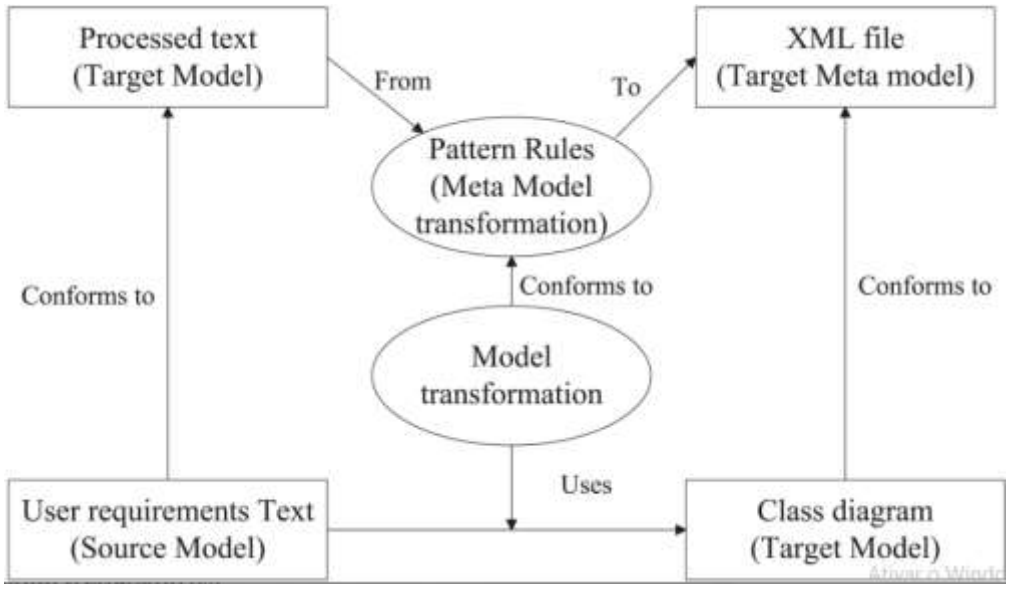



*Figura 20 - Processo de transformação MDA [73]*

**Processo de Transformação**

O processo de transformação do modelo proposto consiste em:

i. Processar o texto relacionado aos requisitos do utilizador (modelo de origem), o resultado é um texto processado (modelo de destino);
ii. Definição de regras de padrão (transformações de meta-modelo) que assegurem a transformação do texto processado num ficheiro XMI (meta-modelo-alvo);
iii. Gerar o diagrama de classes a partir do ficheiro XMI. O fluxo do processo do modelo em estudo nesta secção é apresentado na Figura 21.

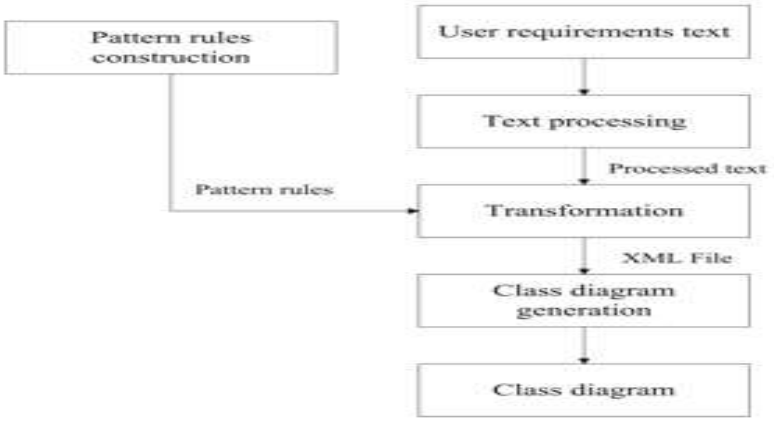



*Figura 21 - Fluxo do processo ABCD [73]*

Os autores consideram que a técnica proposta pode beneficiar tanto do poder de extração das US do utilizador como do trabalho de transformação já realizado na abordagem MDA. O benefício em aplicar uma abordagem MDA reside na sua capacidade de facilitar o trabalho da equipa de desenvolvimento e do *Product Owner.* Em geral, a técnica pode ser aplicada num processo de desenvolvimento de *software* e num contexto MDA particular [78].

### *3.2.5 An Automated Tool for Generating UML Models from Natural Language Requirements*

Deeptimahanti e Sanyal [83] descrevem uma ferramenta independente do domínio, denominada Uml *Model Generator Analysis of Requirements* (UMGAR), que gera modelos UML, como diagrama de casos de uso, diagrama de classes, diagrama de sequência e diagrama de colaboração. A UMGAR também fornece um analisador XMI genérico para gerar ficheiros XMI para visualizar os modelos gerados em qualquer ferramenta de modelação UML.

**Processo de Transformação**

O *Stanford Parse* [84] é usado para analisar fluxos básicos e alternativos no modelo de especificação de casos de uso para identificar remetente, destinatário e as mensagens entre eles. A UMGAR gera um modelo de classes a partir do diagrama de colaboração gerado, no qual atores e objetos identificados no diagrama de colaboração são considerados classes de *design* [85]. As mensagens entre objetos são extraídas como métodos que os associam as classes correspondentes usando o *Stanford Parser* e extraindo também relacionamentos de associação de sequências de fluxo de eventos. Finalmente, a UMGAR gera modelos de código baseado em Java usando o recurso de geração de código do *Enterprise Architect* para demonstrar a rastreabilidade entre requisitos e código usando o recurso de localização de conceito [86]. A Figura 22 mostra a arquitetura de processo da UMGAR.

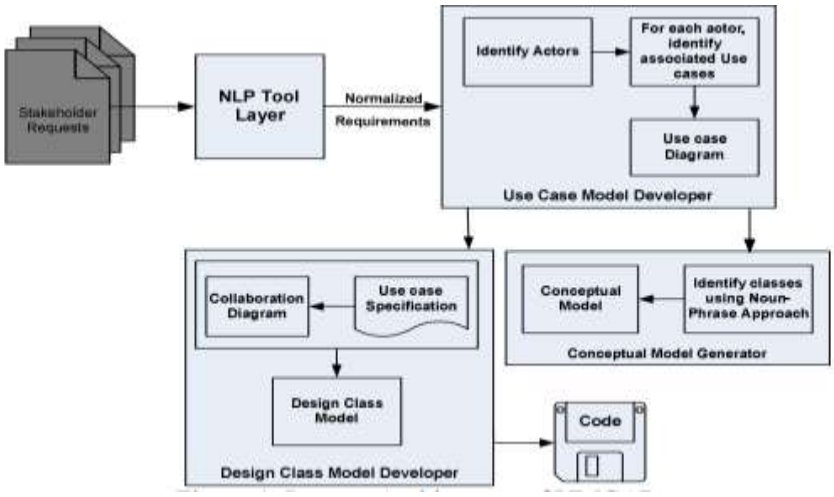



*Figura 22 - Arquitetura de Processo da UMGAR [78]*

As vantagens da UMGAR são a geração em pouco tempo de modelos UML com relacionamentos adequados e o processo de lidar com conhecimento de domínio usando ferramentas eficientes de NLP. A UMGAR é capaz de visualizar diagramas UML em qualquer ferramenta de modelação UML que possui o recurso de importação XMI. UMGAR apresenta algumas limitações, porque não pode ser utilizado em outros processos de modelação e está condicionada caso a ferramenta UML não apresente recurso de importação XMI.

## 3.3 Análise comparativa

Notou-se através do estudo de algumas abordagens, que tem havido esforços para explorar as tecnologias baseadas em NLP para automatizar a fase de análise de requisitos. Nesta secção, é fornecida uma breve análise comparativa das abordagens estudadas e suas respetivas limitações que fornecem a motivação para o ambiente visual que será proposto para esta dissertação.

*Tabela 7 - Comparação entre as abordagens estudadas, adaptado de Abdouli, Karaa and Ghezala* [85]

| Autor | Abordagem | Técnicas usadas | Saída | Vantagens | Desvantagens |
|---|---|---|---|---|---|
| Elallaoui, Nafil e Touahni (2018) | Abordagem Elallaoui, Nafil e Touahni. | NLP | Diagrama de casos de uso | -Facilidade de análise das US;<br>-Redução do tempo na geração de diagramas de casos de uso em relação à modelação manual;<br>-Geração automática de diagramas UML. | -Não suporta frases com substantivos compostos;<br>-Não implementa relacionamentos como generalização / especialização entre atores;<br>-Os atores, casos de uso e sua relação são identificados manualmente;<br>-Não suporta relacionamentos de inclusão e exclusão entre casos de uso. |

| | | | | | |
|---|---|---|---|---|---|
| Elallaoui (2015) | Abordagem Elallaoui. | Algorithm | Diagrama de sequência | - Geração automática de diagramas UML;<br>-Aceitar como entrada uma frase simples em linguagem natural, que respeite a sintaxe geral das US. | -Gera apenas diagramas de sequência;<br>-Não aceita na entrada frases demasiadas extensas. |
| Alattar e Norwawi (2016) | ABCD | NLP+ Pattern | - Diagrama de classe | -Transformação gera resultados corretos;<br>-A combinação de NLP e Pattern. | -Não gera todos os diagramas UML. |
| Deeptimahanti e Sanyal (2011) | UMGAR | NLP | Código | -Identificação automática de elementos OO | -Geração de diagramas de classes e diagramas de estado não bem realizados;<br>-Requer interação humana para eliminar classes irrelevantes e para identificar agregação / composição. |

Analisando as quatro técnicas apresentadas na tabela, e olhando com destaque para a abordagem proposta por Elallaoui, Nafil e Touahni, nota-se que esta abordagem apresenta vantagens como a facilidade de análise de *user stories* que é um aspeto importante para a abordagem a ser proposto. Outra vantagem desta abordagem é o tempo reduzido com que os casos de uso são gerados uma vez que o processo é automático o que não acontece no processo manual. Apesar das vantagens referidas, esta abordagem apresenta algumas desvantagens que a torna limitada, por exemplo a não implementação de relacionamento como generalização / especialização entre casos de uso. Outra abordagem a ser destacada é a proposta por Elallaoui em 2015. Esta abordagem gera automaticamente diagramas de sequência UML, mas o facto de gerar apenas diagramas de sequência torna esta abordagem limitada.

Em suma, o ambiente visual a ser proposto deve combinar e melhorar todas as características consideradas relevantes das quatro abordagens apresentadas.

## 3.4 Outras propostas

Esta secção tem como objetivo descrever sucintamente outras abordagens analisadas para além das 4 anteriores acima apresentadas. Estas abordagens foram preteridas das demais por não focarem o seu estudo em metodologias ágeis em geral, e muito menos na metodologia *Scrum* que é a metodologia pela qual o ambiente visual a propor pretende apoiar. Apresenta-se a seguir uma breve a descrição de cada abordagem.

Gulia e Choudhury [87] afirmaram que o principal problema que surge no ciclo de desenvolvimento de *software* surge durante a especificação de requisitos. Os erros encontrados durante a primeira fase do ciclo também migram para outras fases, o que resulta num processo mais dispendioso do que o especificado inicialmente. Para minimizar os erros que surgem no sistema existente, foi proposta uma técnica que aprimora a geração de modelos UML por meio de requisitos de linguagem natural, que podem facilmente fornecer assistência automática aos desenvolvedores. O foco foi a produção de diagramas de atividades e diagramas de sequência.

Herchi e Abdessalem [81] num outro trabalho, propõem uma abordagem para facilitar a extração de diagramas de classes a partir de requisitos textuais usando técnicas de NLP e ontologia de domínio. A razão desta proposta é que a transição de requisitos de utilizador para diagramas UML é uma tarefa difícil para o *design*, especialmente quando lida com textos grandes que expressam essas necessidades. A modelação de diagramas de classes deve ser executada com frequência, mesmo durante o desenvolvimento de uma simples aplicação.

SUGAR é uma abordagem proposta por Deeptimahanti e Babar [88], a proposta consiste em gerar casos de uso e diagramas de classe a partir de requisitos de NL.

More e Phalnikar [89] utilizaram um algoritmo e implementaram uma ferramenta protótipo chamada RAPID, para gerar diagramas UML a partir de especificações NL.

Moros *et al.* [90] propõem uma técnica para integrar requisitos de especificação textual na abordagem MDSD (*Model Driven Software Development*), usando técnicas próprias MDSD, meta-modelos e transformações. Os meta-modelos definidos pela abordagem incluem um para especificação de requisitos e outro para rastreabilidade entre requisitos.

## 3.5 Conclusão

O estudo das abordagens existentes sobre ambientes visuais mostrou que os investigadores estão a trabalhar no sentido de automatizar cada vez mais a geração de diagramas UML a partir dos requisitos especificados em linguagem natural. Mas as técnicas estudadas ainda não cobrem todas as necessidades. Verificou-se também que nenhuma das abordagens estudadas aborda ou faz referência à priorização de requisitos de negócio que são os objetivos pelos quais o projeto é realizado. O presente trabalho, a fim de contribuir para a solução do problema das limitações apresentadas pelas abordagens existentes, irá propor uma abordagem baseada no *framework* MDA para transformar especificações de requisitos em diagramas UML que atenda às necessidades da equipa *Scrum* assim como à priorização de requisitos.

# 4. DEFINIÇÃO DE UM AMBIENTE VISUAL PARA APOIAR O *SPRINT* E *PRODUCT BACKLOG*

É apresentada neste capítulo a abordagem de ambiente visual proposta para o cumprimento do objetivo desta dissertação. São descritas todas as características da abordagem e no final do capítulo a proposta é validada através de um estudo de caso. A proposta foi concebida com base nas abordagens apresentadas no capítulo anterior, tendo em conta as suas características relevantes. Foi ainda desenvolvido um protótipo de software para ajudar na transformação de *user stories* em diagramas de casos de uso. O principal objetivo do protótipo de *software* é a validação do conceito como ideia (*proof of concept*), não sendo, portanto, o foco principal deste trabalho.

## 4.1 Abordagem proposta

Após uma análise das propostas disponíveis sobre ambientes visuais (usando modelos da linguagem de modelação UML) no contexto das metodologias ágeis no geral e do *Scrum* em particular, foram identificados vários aspetos importantes, os quais foram considerados para a abordagem proposta. No entanto, também foi possível a identificação de aspetos menos positivos que se pretendem colmatar. Neste sentido, pretende-se que a abordagem proposta apresente as seguintes características principais: conversão de *user stories* (US) em casos de uso (UC) que apresenta os requisitos funcionais do sistema (seleção do *Product Backlog);* priorização de requisitos que agreguem valor para o negócio com base em fatores de influência, como custo, valor de negócio, risco, tempo, importância e outros; conversão de diagramas de casos de uso através da utilização do método *Four Step Rule Set* (4SRS) para obtenção de diagramas de objetos da arquitetura global como *output*. Para melhorar a compreensão e a colaboração entre cliente / utilizador, *Product Owner (PO)*, desenvolvedores e testadores e reduzir significativamente o tempo e permitindo a implementação eficiente do sistema, é justificável que o processo de transformação de *user stories* para casos de uso seja automático cumprindo assim com uma das

recomendações do *Scrum* de que o tempo máximo de execução de um *sprint* é não superior a 4 semanas. Mais ainda, recomenda-se que seja possível extrair diagramas de classes a partir de casos de uso no sentido de se auxiliar o PO na tomada de decisão através de análise do modelo de diagrama de classe resultante quando são introduzidos / gerados novos modelos de diagramas de casos de uso. Quando são implementados novos casos de uso, as mudanças no modelo de diagrama de classe devem ser observadas visualmente dando desta forma ao PO uma maior capacidade de análise às mudanças no diagrama de classes e assim poder tomar uma decisão na seleção de *user stories.*

Para geração automática de casos de uso, será necessário o desenvolvimento de um *software* (um protótipo) que terá a responsabilidade de gerar os casos de uso a partir de requisitos do sistema escritos como *user stories*.

Por automático, significa que os modelos (artefactos) são transformados usando uma linguagem de transformação ou com base em alguma ação que o modelador (utilizador da ferramenta) executa com a ferramenta (para a qual a ferramenta está programada para responder) ou até mesmo com base em regras pelas quais foi programada para responder a algum evento particular sem qualquer ação do modelador. Por semiautomática entende-se que a ferramenta suporta decisões que o modelador tem que fazer, permitindo-lhe representá-las nos diagramas [72].

O aumento da comunicação entre todos os *stakeholders*[3] de um projeto leva a uma identificação mais fiável das tarefas a serem executadas num *sprint*. A abordagem deve promover uma comunicação mais profunda sobre os casos de uso e tarefas entre membros da equipa.

O *Four Step Rule Set* - 4SRS é um método que permite a transformação dos requisitos do utilizador numa representação do modelo arquitetural [71],[91], mediante a aplicação de um conjunto de quatro passos. Estão disponíveis duas perspetivas do 4SRS: a orientada ao produto (conhecida como 4SRS *product level*) e a orientada ao processo (conhecida como 4SRS *process*

[3] É um grupo de pessoas que possuem uma participação num negócio. Uma pessoa como funcionário, cliente ou alguém que esteja envolvido com uma organização, sociedade etc. e, portanto, tenha responsabilidades em relação a ela e interesse no seu sucesso. (https://dictionary.cambridge.org/pt/dicionario/ingles/stakeholder)

*level*). A primeira utiliza os requisitos de um ou mais produtos, a segunda utiliza requisitos de um ou mais processos de negócio [92].

Para esta dissertação, utilizou-se a abordagem orientada ao processo (4SRS *process level*) que será designada neste documento de método 4SRS ou simplesmente 4SRS. A. L. Ferreira, Machado e Paulk [93] propõem o uso de uma perspetiva do nível de processo para definição de requisitos e produção do modelo lógico da arquitetura do sistema em detrimento da perspetiva do nível de produto definida tradicionalmente. Estes alegam que "usar uma perspetiva do nível de processo, em vez de uma perspetiva do nível de produto, contribui para uma definição mais precisa de requisitos do produto e melhorar a compreensão do projeto" [93].

O 4SRS recebe como *inputs* um conjunto de casos de uso que descrevem os requisitos para o (s) processo (s) específico (s) que aborda (m) a questão inicial, as atividades realizadas por pessoas ou máquinas, no contexto do sistema os casos de uso são refinados através de várias iterações do 4SRS. No final da execução de todas as etapas que constituem o método, obtém-se como *output*, uma arquitetura lógica global [71].

O ciclo de vida do ambiente visual é dividido em duas fases principais (Figura 23), Planeamento do *Sprint* (Fase 1) e a Execução do *Sprint* (Fase 2).

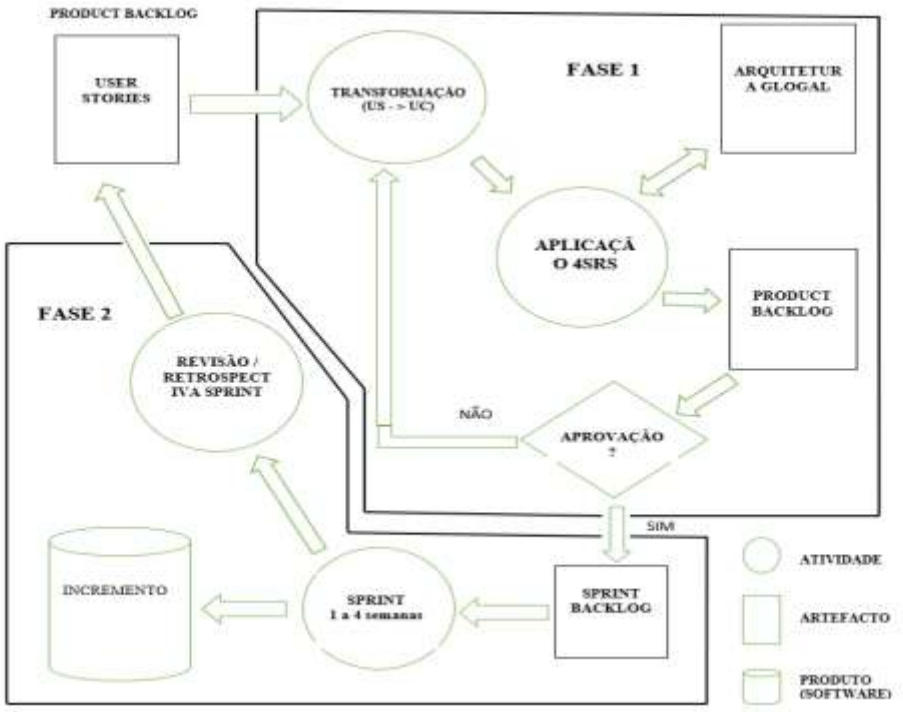



*Figura 23 - Definição de um ambiente visual para apoio as cerimónias do Scrum*

A metodologia *Scrum* apresenta as seguintes orientações: o *Product Backlog* (PL) é a base de todo o desenvolvimento; o PL é uma lista de *user stories*, ou seja, descrições curtas de todas as funcionalidades requeridas no produto de *software*; o *Product Owner* é o responsável pela priorização do *Product Backlog* [25]. Verifica-se que, a análise de requisitos de *software* é complexa e é o segredo para o sucesso de qualquer desenvolvimento ágil de *software* [1]. Assim, nesta proposta de um ambiente visual, introduziu-se a técnica / ferramenta de modelação de casos de uso na fase de análise de requisitos ágeis. A implementação do *Sprint*, a organização do projeto e as cerimónias do *Scrum* são seguidas. A primeira etapa (Planeamento do *Sprint*) é composta pelos artefactos, *Product Backlog* (lista de *user stories*), Arquitetura Global e pelas atividades Transformação – US->UC e Aplicação do 4SRS. A segunda fase (designada Execução do *Sprint*) é composta pelo artefacto *Sprint Backlog*, pelas atividades, *Sprint* e Revisão / Retrospetiva do *Sprint* e pelo incremento (*software*).

Na Figura 23 são ilustradas as atividades e artefactos dessas duas etapas. Os artefactos principais da fase de análise de requisitos ágeis são o *Product Backlog* e os modelos de casos de uso resultantes do processo de transformação de *user stories*. A segunda fase é implementada por meio de uma série de iterações denominadas *Sprints*. Cada *Sprint* é necessário para entregar um incremento de produto potencialmente utilizável [1].

A abordagem proposta pretende-se que seja holística, inspirada no *Scrum*, na *Unified Modeling Language* (UML) e orientada por *user stories*, por casos de uso e por classes, focada na arquitetura, além de iterativa e incremental. O desenvolvimento orientado a casos de uso é uma das práticas comumente adotadas no desenvolvimento de *software* [1], fornecendo uma excelente solução para requisitos imprecisos, incompletos ou inconsistentes. Nesta abordagem, a base é a visualização das *user stories* em casos de uso e a sua análise em relação à arquitetura global. É importante destacar a necessidade de uma recolha e análise eficiente e eficaz de requisitos durante todo o desenvolvimento [94].

O uso de diagramas de casos de uso para especificação de requisitos é obrigatório, pois o método 4SRS para derivar a arquitetura lógica utiliza casos de uso como *input* [95]. O processo de desenvolvimento da abordagem apresentado na Figura 23, incluindo funções, atividades, artefactos e métodos de implementação, serão discutidos em detalhe nas próximas secções.

### 4.1.1 Planeamento do *Sprint (Fase1)*

De acordo com o Guia do *Scrum* [25], o trabalho a ser executado no *sprint* é planeado no *Sprint Planning*. Este plano é criado envolvendo toda a equipa *Scrum*. A equipa prioriza os elementos do *Product Backlog* a serem implementados, e transfere estes elementos do *Product Backlog* para o *Sprint Backlog*, ou seja, a lista de funcionalidades a serem implementadas. O *Sprint Planning* nesta proposta é composto por 4 blocos: (1) Transformation (US->UC), (2) *Four Step Rule Set*, (3) *Global Architecture* e (4) *Product Backlog*. Em seguida apresenta-se a descrição de cada bloco.

**Transformação (US->UC)**

Nesta fase é feita a seleção dos itens do *Product Backlog* (escritos na forma de *user stories*) e convertidos em seguida de acordo com a prioridade em diagramas de casos de uso. A seleção de requisitos nesta fase é da responsabilidade do *Product Owner* que também faz a gestão do *Product Backlog*.

É lido um ficheiro de texto que contém a lista de itens de *Product Backlog* e com a utilização da ferramenta *PlantUML* instalado no ambiente eclipse é possível a conversão indireta (utilizando *Epics*) de *user stories* em casos de uso.

**Aplicação do 4SRS**

*Four Step Rule Set* - 4SRS é um método que permite a transformação dos requisitos do utilizador numa representação do modelo arquitetural [71], mediante a aplicação de um conjunto de quatro passos.

O 4SRS recebe como *inputs* um conjunto de casos de uso que descrevem os requisitos para os processos específicos que abordam as necessidades iniciais através das interações realizadas por pessoas ou máquinas. No contexto do desenvolvimento os casos de uso são refinados através de várias iterações do 4SRS. No final da execução de todas as etapas que constituem o método, obtém-se como *output*, uma arquitetura lógica global [71].

O uso de diagramas de casos de uso para especificação de requisitos é obrigatório, pois o método 4SRS para derivar a arquitetura lógica usa casos de uso como *input* [95]. O OMG (*Object Management Group*) define um caso de uso como: "a especificação de um conjunto de ações executadas por um sistema, que produz um resultado observável que é, tipicamente, de valor para um ou mais atores ou outras partes interessadas do sistema". Os casos de uso permitem expressar com simplicidade os requisitos do sistema e a ligação entre os atores do sistema e as funcionalidades.

**Arquitetura Global**

Arquitetura global é o *output* da aplicação do método 4SRS. A arquitetura global neste trabalho é o conjunto de vários diagramas de objetos que resultam do processo de transformação de diagramas de casos de uso no modelo arquitetural depois de executadas recursivamente as quatro etapas do método *Four Step Rule set.*

***Product Backlog***

*O Product Owner* (PO) é quem toma as decisões em relação a quais requisitos o produto terá. É ele que prioriza os requisitos e gere o *Product Backlog* [25], [56], [80]. Para fazer esse trabalho, a função do PO requer um indivíduo com certas competências e características, incluindo disponibilidade, experiência no negócio e habilidades de comunicação. Nesta abordagem, o processo de transformação automática de *user stories* em casos de uso e a aplicação do método 4SRS podem ajudar a reduzir a dificuldade e melhorar a eficiência do seu trabalho (pretende-se implementar a automatização do processo de extração de diagramas de classes a partir de casos de uso). Mais especificamente, um caso de uso para esta proposta é uma *feature* do *Product Backlog* e todos os diagramas de casos de uso que compõem o modelo de caso de uso global constituem o *Product Backlog*. A abordagem propõe um *Product Backlog* diferente dos tradicionais, uma vez que os casos de uso irão facilitar as decisões sobre o *Sprint Backlog.*

### *4.1.2* Execução do *Sprint (*Fase 2*)*

O objetivo de cada *sprint* é transformar as *user stories* do *Sprint Backlog* num incremento no estado "*done*". O *Sprint Backlog* pode ser mantido como um quadro de tarefas físicas na parede, a fim de obter resultados rápidos e frequentes. Durante o *sprint*, a equipa *Scrum* divide cada *user stories* em pequenas unidades chamadas tarefas, conforme necessário, e também estima quantos pontos de *user stories* (*Story Points*) cada tarefa levará para a equipa para concluir a tarefa. Cada membro da equipa decide realizar qualquer tarefa sozinho. A equipa é auto-

organizada e autogerida. Todos os membros da equipa contribuem da maneira que podem para completar o conjunto de trabalho que se comprometeram coletivamente a concluir no *sprint*.

Na implementação do *sprint*, as tecnologias da *Unified Modeling Language* (UML) são aplicadas. Tanto as características estáticas como as características dinâmicas do sistema são analisadas e, em seguida, a codificação pode ser realizada de forma correspondente [1]. Os diagramas podem ser desenhados de forma superficial no quadro ou até mesmo em papel, a fim de manter a agilidade e permitindo a programação o mais rápido possível. Para esta abordagem em concreto, são utilizados os modelos UML (diagramas de caso de uso e diagramas de classes)

***Sprint Backlog***

O *Sprint Backlog* é o ponto de partida para cada *sprint*. É uma lista de itens do *Product Backlog* selecionados para serem implementados no próximo *sprint*. Os itens são selecionados pela equipa *Scrum* juntamente com o *Scrum Master* e o PO na reunião de planeamento do *sprint*, com base nos itens priorizados e metas definidas para o *sprint*. Ao contrário do *Product Backlog*, o *Sprint Backlog* é estável até que o *sprint* (ou seja, no máximo 4 semanas) seja concluído. Quando todos os itens no *Sprint Backlog* são concluídos, uma nova iteração do sistema é entregue [25]. Para esta abordagem em específico, o *Sprint Backlog* é resultante do *Sprint Planning* composto pelas fases Transformação, 4SRS, Arquitetura Global e *Product Backlog* conforme a Figura 23.

**Revisão / Retrospetiva do *Sprint***

No final de cada *sprint* [25], a equipa realiza uma revisão do *sprint*, durante a qual as novas funcionalidades são demonstradas ao PO ou a qualquer outra parte interessada que deseje fornecer *feedback*. Obter *feedback* que poderia influenciar o próximo *sprint* é muito importante. O projeto é avaliado em relação à meta de *sprint* determinada durante *Sprint Planning*. Esse tipo de ciclo de *feedback* no desenvolvimento de *software* pode resultar na revisão ou adição de itens ao *Product Backlog*. É uma oportunidade para identificar qualquer aspeto para melhorar.

## 4.2 Processo de transformação de *user stories* em diagramas de casos de uso

Nesta secção são apresentados os módulos do protótipo de *software* desenvolvido em apoio ao processo de transformação automática de *user stories* para casos de uso.

### 4.2.1 Transformação

Em geral, pode-se dizer que uma definição de transformação consiste num conjunto de regras de transformação, que são especificações inequívocas da maneira pela qual (uma parte de) um modelo pode ser usado para criar (uma parte de) outro modelo. Com base nessas observações, pode-se agora definir transformação, regra de transformação e definição de transformação.

Klepper et al. [96] definem uma transformação como a geração automática de um modelo de destino a partir de um modelo de origem, de acordo com uma definição de transformação.

Uma definição de transformação é um conjunto de regras de transformação que, em conjunto, descreve como um modelo na linguagem de origem pode ser transformado num modelo na linguagem de destino [96].

### 4.2.2 Ferramentas e tecnologia

Neste tópico são apresentadas algumas ferramentas utilizadas para o desenvolvimento deste trabalho, porém com pouco detalhe já que não é o foco deste trabalho. A seguir são apresentadas de forma sucinta as ferramentas utilizadas.

***Lucidchart***

O *Lucidchart* é uma plataforma baseada na *cloud* servindo para gerar gráficos de fluxos para criar facilmente diagramas, esboços UML, fluxogramas e modelos ER. O *Lucidchart* é uma boa plataforma para criar fluxogramas de nível profissional e partilhar o trabalho criativo com outras pessoas. É ideal para diagramas de geração de ideias para programas e fluxogramas do nível de gestão de projetos. Sendo uma plataforma baseada na *cloud*, não é necessário instalar

nenhum *software* ou extensão. Para este trabalho foi utilizada a ferramenta *Lucidchart* para desenhar o ambiente visual proposto e também outros diagramas necessários para este trabalho.

**Java**

Java é uma linguagem de programação e foi escolhida para implementar o *software* protótipo responsável na conversão de *user stories* em casos de uso. Com esta ferramenta / tecnologia é possível executar e desenvolver programas Java com dois tipos de componentes, o JDK (*Java Development Kit*) que é utilizado para desenvolver programas e o JRE (*Java Runtime Environmet*) para executar. No utilizador é preciso instalar somente o JRE já que será necessário apenas executar o sistema de *software*.

**Eclipse**

O Eclipse é um ambiente de desenvolvimento para programas Java, porém é possível utilizar o Eclipse para desenvolvimento noutras linguagens de programação como C / C++ a partir de instalação de *plug-ins.* No Eclipse é possível criar um projeto, realizar as codificações e também verificar *logs* [4]de execução.

***Visual Paradigm***

O *Visual Paradigm* (Vp-UML) é uma ferramenta CASE UML que suporta UML 2, SysML e *Business Process Modeling Notation* (BPMN) do *Object Management Group* (OMG). É uma ferramenta muito utilizada no mercado de desenvolvimento de *software*. O *Visual Paradigm* foi utilizado neste trabalho para realizar a modelação de diagramas UML.

O *Visual Paradigm* suporta a gestão de requisitos, incluindo US*,* UC*,* diagramas de requisitos SysML e análise textual. Um diagrama de requisitos SysML especifica a *feature* ou condição que deve ser entregue no sistema de destino. O *Visual Paradigm* oferece suporte para outros tipos de diagramas UML.

[4] É uma lista de informações de aplicações, desempenho do sistema ou atividade do utilizador.

***PlantUML***

*PlantUML* é uma ferramenta que permite aos utilizadores gerar diagramas UML a partir de uma linguagem de texto simples [97]. A linguagem usada pelo *PlantUML* é chamada de linguagem específica da aplicação, pois só funciona para a ferramenta *PlantUML*. *PlantUML* em si, é um *software Open Source,* havendo *plug-ins* de UML de fábrica para vários *softwares* comuns, como *Eclipse*, *NetBeans*, *Microsoft Word*, *LaTex*, etc. [97].

*PlantUML* é a ferramenta de modelação adotada para o desenvolvimento do protótipo de *software*, por ser uma ferramenta baseada na descrição de texto legível basicamente em alto nível para desenhar os diagramas UML com maior facilidade e rapidez [97],[98].

### 4.2.3 Identificação de atores e caso de uso nas *user stories*

Ao analisar *user stories* e casos de uso, uma característica que causaria um problema seria que a função numa *user story* é semelhante a um ator no modelo de caso de uso, enquanto o propósito ou desejo numa *user story* é semelhante a um caso de uso. Segundo o estudo realizado em [99], em vez de estar diretamente relacionado às *user stories*, os modelos de caso de uso podem relacionar-se indiretamente com as US, por meio de *Epics*. *Epics* são grandes *user stories* que podem ser ainda mais decompostas em várias *user stories* menores com o intuito de simplificar o desenvolvimento de *Software* [99]. Um *Epic* é uma *user story* de maior dimensão que é grande demais para ser implementada numa única iteração, portanto, precisa ser dividida em pequenas *user stories*. Em [100] os autores perceberam que, mesmo que as *user stories* possam não ser diretamente compatíveis com caso de uso, os *Epics*, que são maiores e mais amplos no propósito, provavelmente são. Pode ser visto na Figura 24 a relação existente entre *Epics* e casos de uso.


*Figura 24 - User stories, Epics e correspondente diagrama de caso de uso* [94]

### 4.2.4 Identificação de relacionamento *include* de *user stories*

O exemplo apresentado a seguir na Figura 25 ilustra a ligação entre *user stories*, *Epics* e relacionamento de inclusão num modelo de caso de uso.

Por exemplo, numa Biblioteca, quando um utente quer reservar um livro, o Bibliotecário precisa fazer o seguinte [94]:

- Verifica se o utente é válido (utente registado);
- Verifica se o utente tem livros por devolver.

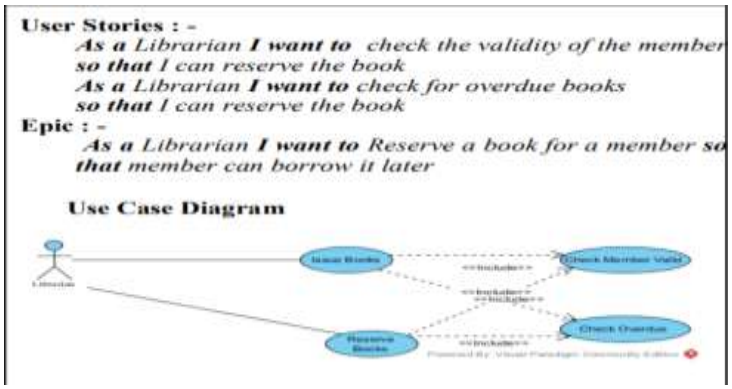



*Figura 25 - User stories, Epics e modelo de caso de uso com relacionamento de inclusão* [94]

### 4.2.5 Identificação de relacionamento *extend* de *user stories*

Para exemplificar o relacionamento *extend*, considerou-se o seguinte exemplo [94]. Se o Bibliotecário encontrou livros vencidos, ele precisa emitir uma multa. Como é um comportamento opcional, ele pode ser mostrado como um relacionamento de extensão.

É ilustrado na Figura 26 as *user stories*, *Epics* e modelo de caso de uso com um relacionamento de extensão.

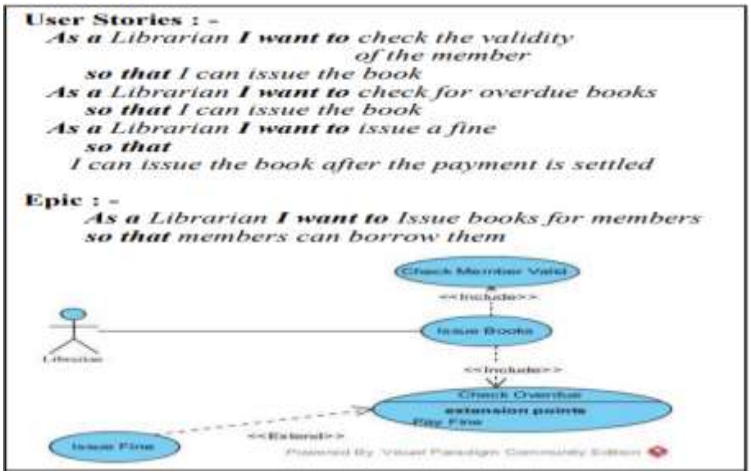



*Figura 26 - User stories, Epics e modelo de caso de uso com relacionamento de extensão* [94]

## 4.3 Protótipo para a transformação de *user stories* em casos uso (US-> UC)

Um diagrama contextual da arquitetura do protótipo de *software* é apresentado a seguir na Figura 27, em conjunto com o processo manual de transformação de *user stories* em *Epics* e o processo existente (*Eclipse* e *PlantUML*).

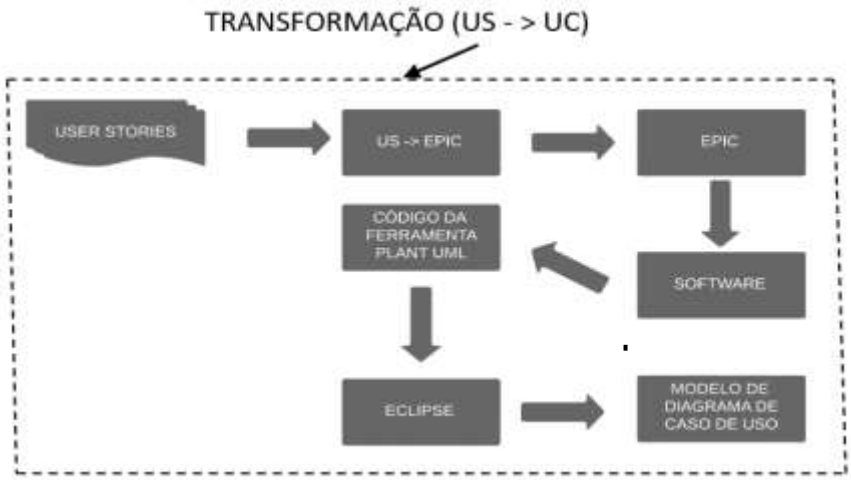



*Figura 27 - Diagrama contextual da Arquitetura do protótipo de software*

Em seguida serão apenas descritos alguns elementos visto que os outros já foram apresentados quando se abordou o tipo de ferramentas e tecnologias usadas (ver secção 4.2.2), evitando-se desta forma a repetição de informação.

***Software***

O *software* a desenvolver poderá transformar um grupo de *user stories* em diagramas de caso de uso UML. O objetivo principal deste protótipo de *software* é automatizar o processo de transformação de *user stories* para casos de uso o que ajudará na diminuição do tempo de execução dessa atividade.

***EPIC***

Como já definido, um *Epic* é uma *user stories* maior que é grande demais para ser implementado numa única iteração, portanto, precisa ser dividido em pequenas *user stories*. Neste trabalho foi adotado a utilização de *Epics* devido a relação direta que estes apresentam com casos de uso conforme o estudo apresentado por Madanayake, et al. [101]. Os *Epics* são obtidos a partir de um conjunto de *user stories* num processo ainda não automático.

**Código da Ferramenta *Plant UML***

Este bloco apresenta o código da ferramenta *Plant UML* gerado por intermédio de *software* a ser desenvolvido. Este código da *Plant UML* implementado num ficheiro sem título no ambiente eclipse tem como *output* o diagrama que representa o modelo de caso de uso desejado.

**Modelo de Diagrama de caso de uso *(Use Case Model)***

*Use Case Model* é o resultado de transformação de *Epics* em código da ferramenta *Plant UML*. Este modelo, é o resultado que se pretende atingir com o desenvolvimento deste protótipo de *software*.

### 4.3.1 Protótipo de *software*

Será usado o ambiente de desenvolvimento eclipse, para a qual o *plug-in* UML foi instalado. Nesta secção, é analisada brevemente a sintaxe do código da linguagem *Plant UML*. Cada bloco de código *Plant UML* começa com uma instrução @startuml e termina com instrução @enduml. O código entre essas instruções decidirá que tipo de diagrama UML pretendemos obter no final [97], [98]. A Figura 28 apresenta um exemplo de código da linguagem *Plant UML*.

```
@startuml
    left to right direction
Librarian --> (Issue Books)
@enduml
```

*Figura 28 - Código Plant UML que mostra um modelo de caso de uso simples [96]*

A *Plant UML* pode funcionar sem o Eclipse, mas é executada apenas na linha de comandos. Para torná-lo mais fácil de usar, foi instalado o Eclipse e importado o *Plant UM*L como um *plug-in* para ser executado no Eclipse. Isso é necessário, porque é preciso adicionar a aba *Plant UML* ao Eclipse aberto, indo em *Windows*> *ShowView*> *Other*> e selecionar *Plant UML* na janela *Show View* [97].

O protótipo de *software* desenvolvido mostrou ser capaz de converter *user stories* em casos de uso através de um processo indireto, ou seja, a partir de *Epics*. Apenas um modelo de caso de uso básico existe no momento, mas o protótipo atual serve como prova de conceito da ideia. Na Figura 29 pode ser visto o resultado desta operação.

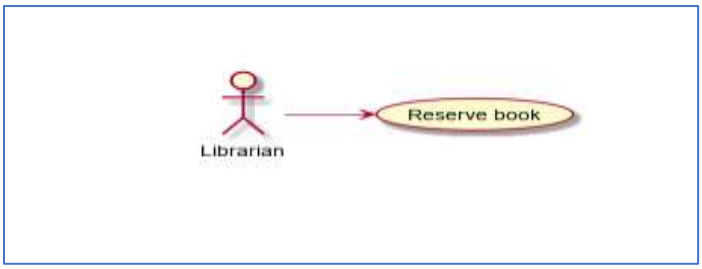



*Figura 29 - Diagrama de caso de uso gerado pelo protótipo*

## 4.4 Estudo de Caso para validação da proposta

Esta secção descreve o propósito do sistema que será usado como base de estudo para a caracterização de um ambiente visual para apoiar as cerimónias do *Scrum* e apresenta os diagramas de casos de uso e diagrama de componentes / objetos, utilizando a notação UML. Será ainda analisada a lista de requisitos fornecida para o projeto *Unified HUB for Smart Plants* ( UH4SP) [70]. Será apresentada a execução do método 4SRS e através de sucessivas iterações será derivada e refinada a arquitetura do sistema necessária para uma boa análise dos requisitos do utilizador.

### 4.4.1 Descrição do Projeto UH4SP

Garantir o produto correto, no tempo certo, quantidade exata, no destino programado pela pessoa autorizada, em perfeitas condições e ao melhor preço é o desafio atual das unidades industriais. Para tal, é necessário conceber um sistema de gestão e otimização logística baseada em conceitos tecnológicas, ferramentas e metodologias emergentes no contexto de sistemas industriais baseadas na internet.

A visibilidade, segurança e controlo nas operações de carga e descarga e na movimentação de produtos, viaturas e pessoas, automatização de processos e a colaboração com os diversos agentes ao longo da cadeia de valor, são vetores fundamentais na otimização logística das empresas industriais.

O estudo de caso trata de um sistema de pesagem industrial e *software* para automatização de processos operacionais de indústrias com estas necessidades, baseado em diversos domínios tecnológicos da indústria 4.0.

O projeto UH4SP tem como objetivo o desenvolvimento de uma arquitetura de *software* orientada a serviços e soluções tecnológicas, incorporando o paradigma de IoT (*Internet of Things*) e indústria 4.0, que promovam a visão corporativa e agregada de operações de unidades industriais dispersas por várias áreas, através de acessos remotos e locais. Engloba a construção de ferramentas colaborativas e transversais, otimização das operações e da experiência de utilização nas unidades industriais e por último, a fiabilidade do sistema.

Sendo assim, o foco nesta secção é a derivação da arquitetura lógica do projeto UH4SP que reflita os requisitos do utilizador. A Figura 30 ilustra as etapas que serão seguidas para a obtenção da arquitetura lógica.


*Figura 30 - Passos para conceção da arquitetura lógica do projeto UH4SP,adaptado de Machado et al.* [102]

Na secção 4.4.2 é abordado o processo de modelação de requisitos (1) e de seguida apresenta-se a execução do método 4SRS (2) e através de sucessivas iterações serão derivadas arquiteturas lógicas para cada *sprint* individualmente (3). No final, será analisada a arquitetura obtida e será feita uma análise comparativa com uma arquitetura existente do mesmo projeto

(obtida com um tratamento global de todos os requisitos) no sentido de se avaliar se há diferenças significativas em termos de robustez.

### 4.4.2 Requisitos de *software*

Esta secção aborda o processo de transformação da lista de requisitos em *Epics* e posteriormente em casos de uso.

Sendo assim, para representar os requisitos o ponto de partida é a análise e identificação de funcionalidades do sistema, assim como os *Stakeholders* que constituem o sistema, de acordo com a função de cada um, através de casos de uso. Os intervenientes são representados em forma de atores, que possuem uma ou mais funcionalidades, representadas em forma de casos de uso.

No sentido de facilitar o processo ou até lidar com casos de uso de maior complexidade na modelação, os casos de uso podem ser refinados através da decomposição dos mesmos, considerando-se as funcionalidades de mais alto nível, tipicamente designados de nível 0. A Figura 31 apresenta os níveis que poderão ser utilizados para definir as funcionalidades do sistema, no formato de uma "árvore". Na primeira camada (nível 0) são representados os requisitos mais abstratos que estão relacionados ao nível mais alto de refinamento, e assim em diante. Com este refinamento de alto nível, os requisitos são traduzidos nas funções de sistema e serão aqueles em que o fluxo de operações será descrito. No sentido de facilitar a rastreabilidade, sugere-se a utilização de numeração nos casos de uso e níveis dessa mesma numeração. Por exemplo, um caso de uso de nível alto {UC1}, a decomposição da mesma no nível um resultaria em {UC1.1}, {UC1.2} e ao nível 2 {UC1.1.1}, e assim em diante.

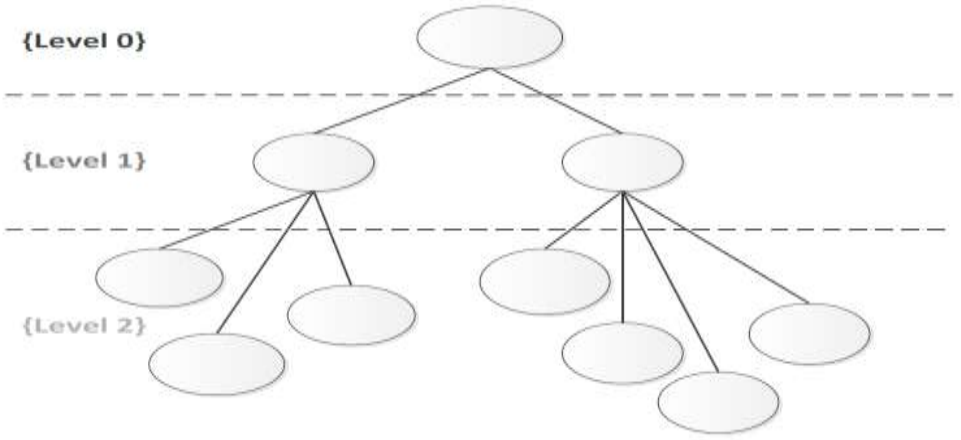



*Figura 31 - Decomposição de casos de uso em níveis* [102]

É também de salientar que os requisitos tratados no presente estudo de caso apenas dizem respeito aos requisitos funcionais.

### 4.4.3 *Story Points e* priorização de requisitos

*Story points* é uma unidade de medida para expressar uma estimativa do esforço geral que será necessário para implementar totalmente um item (uma *user story*) do *Product Backlog* [103].

Quando realizamos uma estimativa com *story points*, atribui-se um valor em pontos a cada *user story.* Os valores brutos que atribuímos não são importantes. O que importa são os valores relativos [103]. Projetos de *software* possuem um alto grau de incerteza. Mesmo que os desenvolvedores ou um consultor forneçam uma estimativa do projeto, até o desenvolvimento começar, é difícil saber exatamente o que é necessário para implementar as *user stories* planeadas para as versões. No entanto é certo que para garantir que o projeto será um sucesso, deve-se usar métodos de priorização, que ajudem a perceber quais *user stories* devem ser abordadas primeiro.

As metodologias ágeis de desenvolvimento de *software* tornam-se cada vez mais populares à medida que a palavra se espalha sobre os benefícios oferecidos em certas condições do projeto. Uma característica chave de qualquer abordagem ágil é o seu foco explícito na criação de valor de negócio para os clientes [103]. Essencialmente, em projetos ágeis de *software*, o

processo de desenvolvimento é um processo de criação de valor de negócio que depende da participação ativa do cliente. A criação de valor de negócios é assegurada tanto pelo produto final quanto pelo próprio processo. A priorização de *user stories* é uma tarefa difícil num ambiente ágil por causa da sua natureza volátil. A ignorância da criticidade das *user stories* resultará em vários problemas, como cliente insatisfeito e má qualidade do produto.

Os requisitos para este caso de estudo foram fornecidos na forma de *user stories.* No sentido de elicitação e priorização de requisitos, essas *user stories* foram priorizados como se verá ainda nesta secção. Na priorização foi considerada a importância e o esforço das *user stories,* mas também foi tido em conta a importância que algumas *user stories* têm, conforme o desejo do cliente.

Considerando a proporção de Importância atribuída pelo cliente e Esforço estimado pela equipa do projeto (I / E) (a equipa neste projeto em concreto é a autora destes valores).

Priorização de *user stories* = $\frac{Importância\ das\ user\ stories}{Esforço\ por\ user\ stories}$ [1]

A restrição de tempo é um grande problema para o cliente, bem como para o ambiente ágil. A libertação do produto é dada ao cliente de acordo com a *user story,* ou seja, o cliente informará qual *user story* deve ser feita antes, para iniciar o seu trabalho o mais rápido possível. O produto que é lançado cedo ou periodicamente é a melhor maneira de satisfazer as necessidades do cliente.

Dependências de *user stories* no *Product Backlog* têm um papel vital no ambiente ágil e foram tidos em conta neste caso de estudo (agrupar as *user stories* que apresentam a mesma dependência). As dependências entre *user stories* afetam a priorização no ambiente ágil. Combinar vários itens dependentes num grande *Epic* e dividir os itens de maneira diferente são duas técnicas comuns para lidar com *user stories* dependentes.

A segurança pode ser considerada como segurança de rede, segurança funcional, segurança de código, segurança de documentação e outros tipos, com base na lista dos requisitos do cliente. No momento de desenvolvimento do projeto, a segurança do ciclo de vida deve ser considerada um fator de grande prioridade.

Os pré-requisitos / disponibilidade de recursos são orçamento, pessoas, material, tecnologia, espaço e outros ativos que são necessários para uma operação eficaz. A disponibilidade de pré-requisitos tem um papel vital no ambiente ágil porque a tarefa poderá ficar atrasada se o recurso não estiver disponível naquele momento.

Nesta fase, a importância e o esforço da *user story* foram calculados e, em seguida, a proporção da importância e do esforço (I/E) para cada *user story* foi usada para a priorização. Como exemplo foi considerado o *sprint* 1 (primeiros 10 *user stories*), enquanto a tabela para as restantes *user stories* pode ser consultada no Anexo I.

*Tabela 8 - User Stories*

| **User Stories, Importance and Effort Factor** | | | | |
|---|---|---|---|---|
| **ID** | **User Story** | **I** | **E** | **I/E** |
| **7** | As a System Administrator, I want to CRUD an industrial group in order to manage business groups. | 90 | 68 | 1,32 |
| **8** | As a System Administrator or a corporate manager, I want to CRUD a group company in order to manage group companies. | 100 | 70 | 1,43 |
| **12** | As a System administrator, Corporate manager, Company manager or Factory manager, I want to CRUD factories in order to manage factories | 95 | 70 | 1,36 |
| **13** | As a Forwarder admin or systems admin, I want to CRUD trucks in order to manage trucks. | 80 | 67 | 1,19 |
| **24** | As a user I want to perform login, in order to access collaborative web app. | 85 | 78 | 1,09 |
| **25** | As a user I want to recover login credentials, in order to access collaborative web app. | 83 | 79 | 1,05 |
| **29** | As a user I want to perform login, in order to access collaborative web app. | 81 | 79 | 1,03 |
| **30** | As a user I want to recover login credentials, in order to access collaborative web app. | 78 | 78 | 1,00 |
| **32** | As a System Administrator I want to CRUD work tokens in order to manage work tokens. | 96 | 70 | 1,37 |
| **37** | As an Entity (forwarder, client or supplier) manager I want to request, read, update and disable work tokens in order to manage work tokens to my entity. | 98 | 70 | 1,40 |

Considerando o *Product Backlog* do projeto UH4SP, o gráfico Importância / Esforço (Figura 32) apresenta a ordem de priorização das *user stories* mostrando o resultado dos cálculos realizados para cada *user story.*

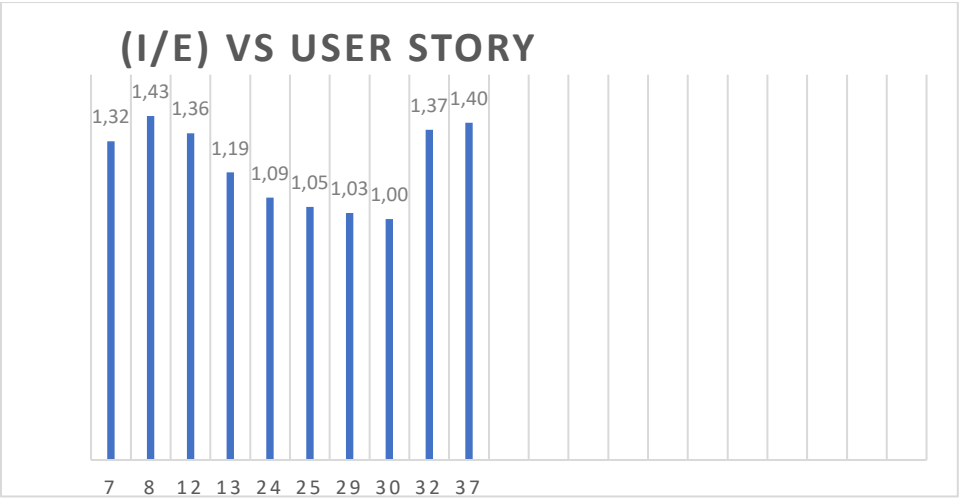



*Figura 32 - Priorização por cálculo de Importância / Esforço (sprint 1)*

Depois de priorizadas as *user stories*, foi desenhado o gráfico entre a importância para a relação esforço (E/I) para cada *user story*. Inicialmente procurou-se encontrar a menor *user story* na ótica do autor que representa toda equipa do projeto. A menor *user story (user story* número "5") encontrada foi-lhe atribuído o valor de "I" igual a 1 e depois seguiu-se encontrar a *user story* de maior prioridade segundo o autor, que neste caso é a *user story* número "8" e recebeu o valor máximo de "100". Os resultados dos demais podem ser visualizados na Figura 32. Depois da priorização das *user stories*, a fase seguinte foi a criação de *sprints.*

### 4.4.4 *Sprint Backlog*

Após priorizar o *Product Backlog* como mostrado acima, partiu-se para a fase de criação de *sprints*. O *Sprint Backlog* é o resultado de itens do *Product Backlog* considerados prioritários para agregação de valor ao cliente. O *Sprint Backlog* do projeto UH4SP é composto de 5 *sprints* de acordo com a análise feita*,* com 4 *sprints* de 10 *user stories* cada e um *sprint* composto de um *Epic*. O processo de transformação de *user stories* em *Epics* é um processo manual, enquanto o processo de transformação de *Epics* para casos de uso pode ser automático ou manual como ficou provado com o protótipo de *software* desenvolvido ao longo deste trabalho que valida este conceito. Para o presente caso de estudo, foi considerada apenas a transformação manual de *Epics* em casos

de uso e todas as *user tories* foram analisadas no sentido de se avaliar a existência de um valor de negócio / funções / objetivos / metas para poderem ser extraídos atores e casos de uso correspondentes. Um caso de uso corresponde a um objetivo / valor de negócio num *Epic* e uma subtarefa num *Epic* corresponde a um objetivo de uma *user story,* ou seja, o objetivo de uma *user story* corresponde a uma subtarefa num *Epic*. O papel / função num *Epic* corresponde a um caso de uso.

A utilização de ferramentas *Plant UML* apresentadas nas secções 4.2 e 4.5 permite que seja possível transformar o papel / função e objetivos / valor de negócio / metas de um *Epic* em modelo de casos de uso, ou seja, ator e caso de uso respetivamente. O primeiro passo para transformar *user story* num *Epic*, é analisar a *user story* e extrair o ator e objetivo da mesma. No sentido de se evitar que o texto seja demasiado extenso, para o caso de demonstração foi considerado apenas o *sprint* 1 e que foi analisado com mais detalhes, enquanto para os restantes 4 *sprint* a demonstração do processo de transformação explicada neste parágrafo segue a mesma lógica.

***Sprint1***

Neste primeiro *sprint* de 10 user *stories* considerados prioritários, foram selecionadas as *user stories* números "7,8,12,13,24,25,29,30,32,37" conforme o gráfico da Figura 32.

***User Stories: -***
*8.* ***As a*** *System Administrator, I want to CRUD an industrial group in order* ***to*** *manage business groups.*
*7.* ***As a*** *System Administrator or a corporate manager,* ***I want to*** *CRUD a group company in order* ***to*** *manage group companies.*
*12.* ***As a*** *System administrator, Corporate manager, Company manager or Factory manager,* ***I want to*** *CRUD factories in order* ***to*** *manage factories.*
*13.* ***As a*** *System Administrator* ***I want to*** *CRUD work tokens in order* ***to*** *manage work tokens.*
*24.* ***As a*** *Forwarder admin or systems admin,* ***I want to*** *CRUD trucks in order to manage trucks.*
*25.* ***As an*** *Entity (forwarder, client or supplier) manager****, I want to*** *request, read, update and disable work tokens in order to manage work tokens to my entity.*
***EPIC****: -* ***As a*** *System Administrator, Corporate manager, Company manager, Forwarder admin, Entity,* ***I want*** *manage business groups, manage group companies, manage work tokens, manage trucks and manage work tokens to entity* ***to*** *….*
*Use Case: - {UC1} Manage accounts.*

*User Stories: -*
*29.* ***As a*** *user* ***I want to*** *perform login, in order* ***to*** *access collaborative web app.*

*30. **As a** user I want to recover login credentials, in order **to** access collaborative web app.*
*32. **As a** user I want to perform login, in order **to** access collaborative web app.*
*37. **As a** user I want to recover login credentials, in order **to** access collaborative web app.*
*EPIC: - As a System Administrator, Operator, Driver, Local Manager and IT Manager, I want to access collaborative web app to…*
*Use. Case: - {UC2} Manage local platform*

*Figura 33 - User stories, Epics e casos de uso do sprint 1*

Inicialmente as *user stories* foram agrupadas em função da relação existente entre elas no que tange aos objetivos de negócio. O primeiro subgrupo é composto de *user stories 8*, 7, 12, 13, 24 e 25. Analisando a *user story* número "8", pode-se ver que o objetivo de negócio / meta é a frase após "**to**", ou seja, "*manage business groups*". A mesma análise é feita para outras *user stories* desse grupo. Como o objetivo / meta numa *user story* corresponde a uma subtarefa (subtarefa é a frase entre "**As a / an**" e "***I want to***") num *Epic*, facilmente extraímos o *Epic* desse subgrupo de *user stories*. A subtarefa do *Epic* resultante desse processo é o conjunto de todos objetivos de todas as *user stories*, isto é: "*manage business groups*, *manage group companies*, *manage work tokens*, *manage trucks* e *manage entity*" como se pode visualizar na Figura 33. Analisando cada um desses objetivos, foi possível concluir que todos esses objetivos fazem parte do mesmo caso de uso, que neste caso é o {UC1} *Manage accounts.* A análise feita no primeiro grupo, é valida também para o segundo grupo de *user stories* que resultou no *Epic* com subtarefa "*access collaborative web app*" e consequentemente o caso de uso resultante é o {UC2} *Manage local platform.*

### 4.4.5 Arquitetura Lógica (*Logical Architecture*)

A arquitetura lógica representa os componentes derivados a partir das necessidades recolhidas nos clientes. Assim, estas necessidades, tipicamente apenas refletem os processos necessários para o negócio, e nem sempre têm em conta necessidades técnicas para uma solução concebida para este contexto de forma adequada.

Após o processo de transformação de *user stories* em casos de uso, estão reunidas as condições necessárias para desenvolver e executar o método *Four Step Rule* (4SRS), que irá

suportar a conceção da arquitetura lógica do sistema. O método 4SRS está segmentado em vários passos e micro passos, suportados em forma de tabela, como será possível observar ao longo desse capítulo. Está dependente do trabalho feito na secção 4.4.2, visto que apenas os casos de uso mais específicos são utilizados no método. Estes casos de uso são designados de "casos de uso folha". Assim sendo, nesta secção o método 4SRS será explicado e aplicado de maneira a permitir a compreensão de como será feita a derivação da arquitetura lógica.

O caso de uso {UC1} *Manage accounts* é refinado em {UC1.1} *Manage business groups,* {UC1.2} *Manage companies*, {UC1.3} *Manage factories*, {UC1.4} *Manage entities*, {UC1.5} *Manage tokens* e por último {UC1.6} *Manage trucks.* O segundo caso de uso resultou nos seguintes casos de uso refinados: {UC2.1} *Manage local IT resources,* {UC2.2} *Schedule interventions,* {UC2.3} *Perform interventions,* {UC2.4} *Provide users training* e {UC2.4} *Generate templates.* Por questões de demonstração e interpretação apenas os casos de uso {UC2.1} e {UC2.2} serão exemplificados nas tabelas abaixo, enquanto os restantes poderão ser consultados no Anexo III. Como já referido na descrição da técnica 4SRS (ver secção 4.1.1*)*, esta apresenta uma tabela dividida em 4 passos, na qual o primeiro passo "*Component creation*" se direciona aos casos de uso derivados dos componentes de *software*. Para cada "caso de uso folha" são criados automaticamente, três componentes / objetos de *software*, associados a uma categoria (i – interface, d – dados e c – controlo) para depois serem validados. Os componentes / objetos de interface dizem respeito a interfaces com utilizadores, *software* ou outras entidades; os componentes de dados relacionam-se com os repositórios genéricos de dados que contêm a informação armazenada numa base de dados e os componentes de controlo referem-se a um componente de processamento, incluindo a computação dos processos de negócio. De forma a facilitar a rastreabilidade dos componentes para o caso de uso, a utilização da mesma numeração do caso de uso, e o sufixo relativo à categoria, tal como, para o caso de uso {UC2.1}, os componentes serão numerados como {C2.1.i}, {C2.1 d} e {C2.1.c}.

É apresentado a seguir o trabalho realizado para a derivação da arquitetura lógica do projeto UH4SP. É descrita a execução de cada um dos passos e micro – passos que constituem o método 4SRS, assim como são apresentados alguns exemplos dos resultados obtidos na execução de cada uma destas etapas. No final, é apresentada a arquitetura lógica do projeto em análise.

O uso de tabelas permite que um conjunto de ferramentas seja criado e construído para que as transformações possam ser parcialmente automatizadas. Essas representações tabulares constituem o mecanismo principal para automatizar um conjunto de etapas de transformação de modelo assistida por decisão. O 4SRS tem sido usado tanto nas instituições de ensino como na indústria [104] e tem demonstrado ser ágil para ajudar os engenheiros de *software* a encontrar e refinar os requisitos de arquitetura, com base nos requisitos do utilizador introduzidos.

**Arquitetura Lógica para *sprint* 1**

Para a derivação da arquitetura lógica para o *sprint* 1, foram utilizados os casos de uso apresentados na Figura 33, nomeadamente os casos de uso {UC1} e {UC2} resultantes das primeiras 10 *user stories* priorizadas.

**Passo 1 (*Step* 1) – *Object creation / Component creation***

Na Tabela 9 apresenta-se o passo 1 do 4SRS, no exemplo a transformação dos casos de uso {UC2.1} *Manage local IT* resources e {UC2.2} *Schedule interventions* em componentes. A execução do passo 1 do caso de uso {UC2.1} resultou na criação de três componentes: {C2.1.c}, {C2.1.d} e {C2.1.i}, bem como a respetiva descrição.

*Tabela 9 - Demonstração da execução do passo 1 do 4SRS*

| Step 1 - Component creation | |
|---|---|
| **Use Case** | **Description** |
| {UC2.1} | Manage local IT resources |
| {C2.1.c} | Generated C |
| {C2.1.d} | Generated C |
| {C2.1 i} | Generated C |
| {UC2.2} | Schedule interventions |
| {C2.2.c} | Generated C |
| {C2.2.d} | Generated C |
| {C2.2.i} | Generated C |

**Passo 2 *(Step 2) – Object elimination / Component elimination***

Neste passo, todos os elementos "C" criados no passo 1 são submetidos a processo de eliminação. Durante a execução deste passo, são criados "C" e eliminados "C". Este passo é o mais crítico e mais extenso dos 4 passos que compõem o método 4SRS, pois, para além de eliminação, também serão validados neste passo os componentes definidos no passo 1 (*Component creation).* É neste passo que também os componentes validados estão representados na arquitetura lógica do projeto UH4SP que será construída.

Este passo é decomposto em 8 micro passos, nomeadamente: (2i) – *Use case identification,* (2ii) – *Local elimination,* (2iii) – *Component naming,* (2iv) – *Component description,* (2v) – *Object representation,* (2vi) – *Global elimination,* (2vii) – *Component renaming e* (2viii) – *Component specification.* Baseando-se no caso de uso escolhido para o contexto de demonstração e pelo tipo (descrição textual de caso de uso), são validados os objetos, do passo 1 (*Step* 1), que garantem a execução do caso de uso não considerando os restantes – micro passos 2i e 2ii, correspondentes à primeira validação, assim como mostrado na tabela 10, é definido uma designação para cada objeto validado (micro passo 2iii) e depois, uma descrição tendo em conta o seu comportamento (micro passo 2iv). A tabela 10 apresenta os micro passos 2i, 2ii, 2iii e 2iv, correspondendo à classificação e eliminação de Componentes, bem como a nomeação e descrição (dos não eliminados) dos casos uso {UC2.1} e {UC2.2}.

*Tabela 10 - Demonstração dos micro passos 2i, 2ii, 2iii e 2iv*

| **Step 2 – Component elimination** | | | |
|---|---|---|---|
| **2i** | **2ii** | **2iii** | **2iv** |
| di | | | *(UC2.1 - Manage local IT resources)* |
| | T | | |
| | F | *Intervention and maintenance data* | O objeto guarda todos os dados relacionados com a manutenção ou intervenção. |
| | F | *Verify intervention interface* | Define a interface que permite que o administrador do sistema e os gestores de IT |

| | | | |
|---|---|---|---|
| | | | verifiquem as necessidades de intervenção / manutenção que podem ser agendadas. |
| di | | | *(UC2.2* - Schedule interventions*)* |
| | T | | |
| | F | *Store interventions data* | Armazena os dados das intervenções. |
| | F | *Schedule interventions interface* | Define a interface que permite ao administrador do sistema e ao gestor IT programarem intervenções que permitem gerir assistência aos recursos de IT das unidades industriais. |

A primeira coluna do passo 2 (*Step2*) corresponde à execução do micro passo 2i. Neste micro passo, foram classificados os dois casos de uso em análise como uma das 8 combinações ou padrões diferentes (Ø, i, c, d, ic, di, icd). A ideia por detrás dessa classificação é ajudar na transformação de cada caso de uso em componentes. Essa classificação forneceria dicas sobre quais categorias de componentes usa e como conectar esses componentes / objetos. Para o caso de demonstração, {UC2.1} *Manage local IT resources* foi classificado como tipo "di", o que significa que são mantidos os componentes (interface e dados) e {UC2.2} *Schedule interventions* foi classificado como tipo "di" também, o que significa que apenas o componente do tipo controlo será eliminado no micro passo 2ii, enquanto os componentes do tipo interface e dados serão mantidos.

A segunda coluna do passo 2 corresponde à execução do micro passo 2ii. O objetivo deste micro passo é responder se cada componente criado no passo 1 (Tabela 9) faz sentido no domínio do problema, uma vez que a criação de componentes no passo 1 foi executada cegamente, não considerando o contexto do sistema para a criação dos componentes. Os componentes a serem eliminados são marcados com "T - *True*" e os componentes que devem ser mantidos são marcados com "F - *False*" como se pode ver na Tabela 10 acima. Para o caso de demonstração {UC2.1} eliminou-se apenas o componente do tipo controlo, pois o mesmo não faz sentido no domínio do problema.

A terceira coluna do passo 2 corresponde à execução do micro passo 2iii. Neste micro passo, os componentes que não foram eliminados no micro passo anterior receberam um nome próprio que reflete tanto o caso de uso no qual ele é originado quanto o papel específico do componente, levando em conta o seu componente principal. Para o caso de uso de demonstração, os componentes {C2.1.d} e {C2.1.i}, por exemplo, foram denominados de *Intervention and maintenance* data e *Verify intervention interface* respetivamente.

No micro passo 2iv, cada componente nomeado resultante do micro passo anterior foi descrito, para que os requisitos do sistema que representam sejam incluídos no modelo de componentes. Essas descrições basearam-se nas descrições do caso de uso original correspondente.

Na quinta coluna do passo 2 (Tabela 11) corresponde a execução do micro passo 2v subdividido em duas colunas (*represented by* e *represent*). Este é o micro passo mais crítico da técnica 4SRS, uma vez que suporta a eliminação de redundância na definição de requisitos do utilizador, bem como a descoberta de requisitos em falta. A coluna "*represented by*" guarda a referência do componente que representará o componente em análise. Se o componente analisado for representado por ele mesmo {C2.1.d}, a coluna correspondente "*represented by*" deve se referir a ele mesmo. A coluna "*represent*" guarda as referências dos componentes que o componente analisado irá representar e não delega em outros componentes a sua representação (i.e., é representado por si mesmo) e adicionalmente representa uma lista considerável de outros componentes (cada um desses componentes deve referir-se a componente que os representa nas suas colunas "*represented by*") (ver na tabela 11, a célula a cinza significa que é representada pela célula a laranja).

*Tabela 11 - Demonstração da execução dos micro passos 2v - 2viii*

| **Step 2- *Component elimination*** | | | | |
|---|---|---|---|---|
| **2v** | | **2vi** | **2vii** | **2viii** |
| | | | | *(UC2.1 - Manage local IT resources)* |
| | | | | |
| {C2.1.d} | {C2.2. d} {C2.3.c} | F | *Intervention and maintenance data* | O objeto guarda todos os dados relacionados com a manutenção ou intervenção. |
| {C2.1.i} | | F | *Verify intervention interface* | Define a interface que permite que o administrador do sistema e os gestores de IT verifiquem as necessidades de intervenção / manutenção que podem ser agendadas. |
| | | | | *(UC2.2* - Schedule interventions*)* |
| | | | | |
| {C2.2.d} | | T | | |
| {C2.2.i} | | F | *Schedule interventions interface* | Define a interface que permite ao administrador do sistema e ao gestor IT programarem intervenções que permitem gerir assistência aos recursos de IT das unidades industriais. |

O micro passo 2vi é totalmente "automático", já que é baseado nos resultados do anterior (micro passo 2v). Os componentes representados por outros devem ser eliminados, pois os seus requisitos do sistema são cumpridos por outros componentes.

A oitava coluna do passo 2 corresponde à execução do micro passo 2vii. Este micro passo tem como finalidade a renomeação dos componentes que não foram eliminados no micro passo anterior e que representam componentes adicionais. Os novos nomes devem refletir a plenitude dos requisitos do sistema. Para demonstração, o objeto {C2.1.d} foi renomeado como *Intervention and maintenance data,* visto que as atividades e as tarefas são idênticas e que os atores envolvidos são os mesmos. O passo seguinte é semelhante à execução do passo 2ii. Uma vez que estes agora representam casos de uso adicionais para além dele mesmo, estes agora serão nomeados (coluna 4 da tabela 11) e reescritos (coluna 5), sendo que devem refletir o contexto global do sistema.

**Passo 3 (*Step 3*) – *Object packaging & aggregation / Component packaging & aggregation***

A décima coluna corresponde à execução do passo 2. Neste passo, os componentes que não foram eliminados na execução do passo 2 deram origem a agregação ou pacotes e componentes semanticamente consistentes.

Para a criação das Agrações foi considerado o modelo como um todo. Foram criados pacotes que agregam os componentes que colaboram para a execução de uma mesma tarefa do projeto UH4SP, situada num nível de abstração superior.

Como apresentado na Tabela 12, verifica-se que os componentes que não foram eliminados após a execução do passo 2 deram origem ao pacote {P3} *Industrial maintenance*. Como o caso de uso {C2.1.d} *Store interventions data* não sobreviveu ao passo 2v, não se preenchem os campos seguintes.

*Tabela 12 - Demonstração da execução dos passos 3 e 4 do método 4SRS*

| **Step 3 - Packing & Aggregation** | **Step 4 - Component Association** | |
|---|---|---|
| | **4i** | **4ii** |
| | | |
| | | |
| {P3} Industrial maintenance | {C2.1. i}<br>{C2.2. i}<br>{C2.3. i} | |
| {P3} Industrial maintenance | {C2.1. d} | |
| | | |
| | | |
| | | |
| {P3} Industrial maintenance | {C2.1. d} | |

Na Tabela 12 a coluna 2 corresponde à execução do passo 4. Para o caso de demonstração, as associações foram derivadas da classificação de casos de uso executado no passo 1. Ou seja, como verificado na tabela acima, a segunda coluna representa as associações diretas enquanto a terceira coluna (na tabela geral do método 4SRS e primeira coluna na tabela acima) representa as associações derivadas da classificação dos casos de uso (não está representada nenhuma associação derivada de classificação de caso de uso neste exemplo). A classificação de {U2.1} como tipo “di” sugere a existência das seguintes associações em relação aos componentes

gerados a partir do mesmo caso de uso: o componente {C2.1.i} está relacionado ao componente {C2.1.d} e este por sua vez está relacionado aos componentes {C2.2.i} e {C2.3.i}. Para mais detalhes ver a arquitetura lógica resultante.

No final da execução da técnica 4SRS, o caso de uso encontra-se funcionalmente decomposto, o que vai servir como base para a modelação de arquitetura lógica. A Tabela 13 apresenta a tabela geral resultante das transformações feitas para os casos de uso {UC2.1) e {UC2.2}. A tabela 13 inclui as decisões tomadas para os casos de uso da demonstração.

*Tabela 13 - Resumo das 4 tabelas que apresentam a execução da técnica 4SRS*

| Step 1 - Component creation | | Step 2 - Component Elimination | | | | | | | | | Step 3 - Packing & Aggregation | Step 4 - Component association |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Use Case** | **Description** | **2i** | **2ii** | **2iii** | **2iv** | **2v** | | **2vi** | **2vii** | **2viii** | | |
| {UC2.1} | Manage local IT resources | di | | | | | | | | | | |
| {C2.1.c} | Generated C | | T | | | | | | | | | |
| {C2.1.d} | Generated C | | F | Intervention and maintenance data | … | {C2.1. d} | {C2.2. d} {C2.3.c} | F | Intervention and maintenance data | … | {P3} Industrial maintenance | {C2.1. i} {C2.2. i} {C2.3. i} |
| {C2.1.i} | Generated C | | F | Verify intervention interface | … | {C2.1. i} | | F | Verify intervention interface | … | {P3} Industrial maintenance | {C2.1. d} |
| {UC2.2} | Schedule interventions | di | | | | | | | | | | |
| {C2.2.c} | Generated C | | T | | | | | | | | | |
| {C2.2.d} | Generated C | | F | Store interventions data | … | {C2.1. d} | | T | | | | |
| {C2.2.i} | Generated C | | F | Store interventions interface | … | {C2.2. i} | | F | Schedule interventions interface | … | {P3} Industrial maintenance | {C2.1. d} |

A Tabela resultante da aplicação da técnica 4SRS para o primeiro *sprint* pode ser consultada no Anexo III. Esta tabela contém todas as decisões tomadas durante o processo da aplicação da técnica. Com base nestas decisões foi derivada a arquitetura lógica do *sprint* 1 do projeto UH4SP, constituída pelos componentes de *software* que restaram após o passo 2, agregados em pacotes, conforme definido após o passo 3 e com fluxos de informação entre eles.

A composição da arquitetura do *sprint* 1 é de 11 casos de uso que resultaram em 23 objetos / componentes e 3 pacotes. Os 3 pacotes representam alguns processos do projeto UH4SP, designadamente: (P2) "*Accounts*", (P3) "*Business Management*" e (P4) "*Industrial Maintenance*".

Uma arquitetura lógica pode ser considerada, segundo Azevedo [91], como "uma revisão de um sistema composto por conjunto de abstrações de problemas específicos que suportam os requisitos funcionais".

Assim sendo, a arquitetura lógica do UH4SP é composta por um conjunto de abstrações de atividades que compõem os requisitos funcionais deste projeto.

A Figura 34 apresenta a arquitetura lógica do primeiro *sprint* do projeto UH4SP, apresentando pacotes que agregam todos os objetos da arquitetura lógica implementada. Estes pacotes representam as atividades de alto nível de abstração do UH4SP, como tal podem ser considerados como o primeiro nível de abstração.

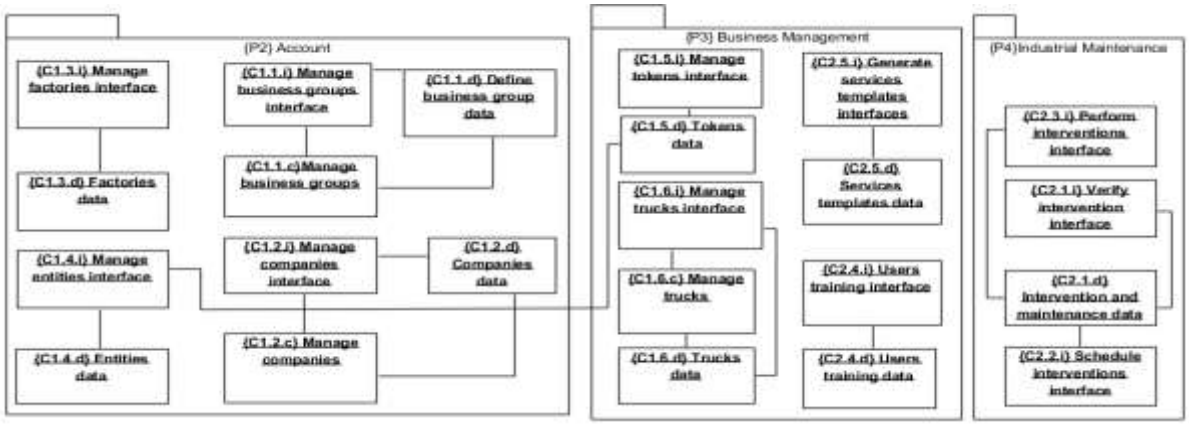



*Figura 34 - Arquitetura lógica do sprint 1*

O modelo lógico apresentado acima representa o tratamento das primeiras 10 *user stories* selecionadas e posterior conversão destas em casos de uso. Estes casos de uso foram refinados num nível arquitetónico e executados explicitamente num nível de desenvolvimento de serviço baseado em componentes.

Depois de obter esse novo modelo de objeto / componentes refinados de arquitetura, os serviços subjacentes podem ser descritos por um conjunto de diagramas para especificar os componentes arquitetónicos correspondentes e projetar um diagrama de classes para caracterização estática do componente de serviço.

## Arquiteturas Lógicas dos *prints* 2 a 5

Os passos para o processo de execução do método 4SRS com objetivo da obtenção do diagrama lógico, são os mesmos para os restantes *sprints* pelo que não serão demonstrados os passos para estes *sprints*, limitando-se apenas a explicar os pontos essenciais e na apresentação dos diagramas de componentes resultantes.

### *Sprint* 2

Tal como no *sprint* 1, o *sprint* 2 resultou da segunda seleção das *user stories* que consideramos serem mais importante tendo em conta ao valor de negócio para o cliente. Assim sendo, e seguindo os mesmos passos já explicados, procedeu-se a extração do *Epic* correspondente às *user stories* números "14, 15, 21, 22, 23, 27, 35, 36, 39 e 40". As *user stories* foram depois agrupadas de acordo com a relação que a mesma mantém uma com a outra. O primeiro subgrupo é composto por 4 *user stories* todas relacionadas e destas resultou o *Epic* na Figura 35 e do *Epic* foi extraído o caso de uso correspondente.

***EPIC**: - **As a** System Administrator, Corporate Manager, Client, Supplier, Forwarder, IT Manager and Factory Admin, **I want to** give trucks, develop dashboards, configure dashboards and data source, **to…***
***Use Case: -** {UC1.7} **Configure users' profile***

*Figura 35 - Epic e respetivo caso de uso para o sprint 2*

O caso de uso obtido depois foi decomposto em casos de usos {UC1.7.1} *Manage profiles* e {UC1.7.2} *Assign permissions*.

A *user story* "***As a*** *Forwarder admin or systems admin,* ***I want to*** *CRUD trailers in order* ***to*** *manage trailers*" faz parte do segundo subgrupo ficando sozinha por não apresentar explicitamente uma relação com as outras *user stories* selecionadas para este *sprint*. A Figura 36 apresenta o *Epic* e caso de uso resultante dessa *user story*.

| |
|---|
| ***EPIC: - As a*** *Factory admin or system admin****, I want to*** *manage trailers****, to…*** |
| ***Use Case: -*** *{UC1.8}* ***Manage trailers*** |

Figura 36. *Epic* e caso de uso do segundo subgrupo de *user stories*

O terceiro subgrupo é composto pelas *user stories* apresentadas na Figura 37, assim como o *Epic* resultante e respetivo caso de uso.

| |
|---|
| ***As a*** *System administrator, Corporate manager, Company manager, and Factory manager, Forwarder, Client or Supplier,* ***I want to*** *configure dashboards settings in order to configure dashboards.*<br>***As a*** *Company manager* ***I want to*** *request, read, update and disable work tokens in order to manage work tokens* ***to*** *my Company factories.*<br>***As a*** *Factory manager* ***I want to*** *request, read, update and disable work tokens in order* ***to*** *manage work tokens to my factory.*<br>***As an*** *Entity (forwarder, client or supplier) manager* ***I want to*** *assign drivers and trucks* ***to*** *work tokens that were associated to my entity in order to manage work tokens.*<br>***As a*** *System admin* ***I want to*** *receive a notification when a stakeholder/entity manager request work tokens in order* ***to*** *validate work tokens.*<br>***EPIC****: -* ***As a*** *System Administrator, Corporate manager, Company manager, Factory manager, Forward manager, Client or Suppler, I want* ***to*** *manage work tokens, company factories and validate tokens, to…* |
| ***Use Case: -*** *{UC1.3.1}* ***Manage work tokens*** |

*Figura 37 - User stories, Epic e caso de uso*

O caso de uso {UC1.3.1} *Manage work tokens* foi refinado através do processo da decomposição resultando nos seguintes casos de uso: (1) {UC1.3.1.1} *Validate tokens*, (2)

{UC1.3.1.2} *Associate entity*, (3) {UC1.3.1.3} *Associate factory,* (4) {UC1.3.1.4} *Request tokens* e por fim o caso de uso (5) {UC1.3.1.5} *Assign tokens.*

Após concluir o processo de transformação e obtidos os casos de uso necessários, seguiu-se para a aplicação do método 4SRS executado passo a passo (*sprint*). A tabela de transformação desse *sprint* pode ser consultado no Anexo III enquanto a arquitetura resultante da aplicação do método é apresentada na Figura 38.

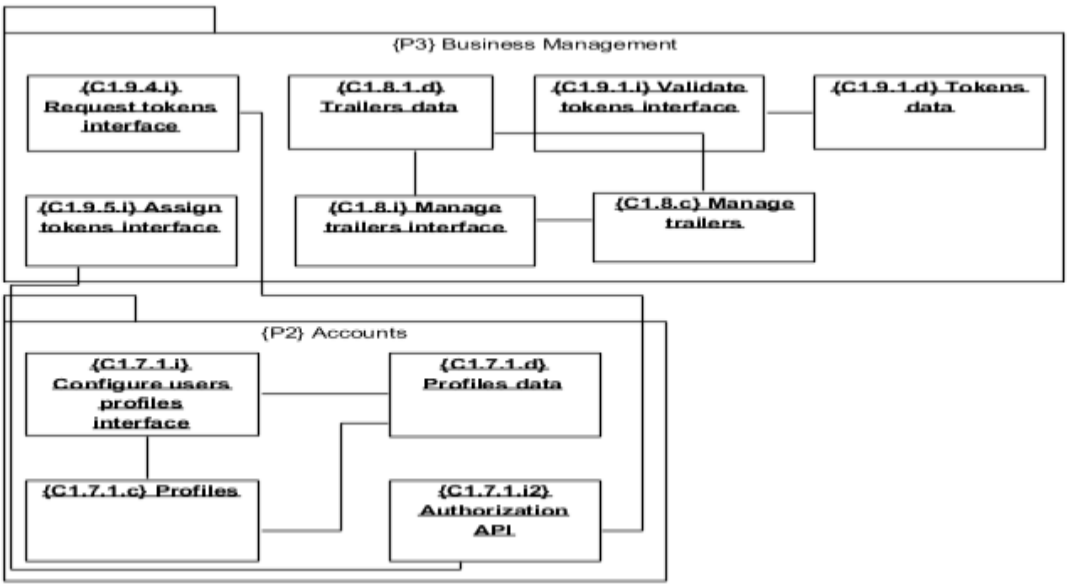



*Figura 38 - Diagrama da arquitetura lógica do sprint 2*

O *sprint* 2 resultou em 8 casos de uso depois do refinamento, 11 componentes / objetos e dois pacotes, nomeadamente o pacote {P2} *Accounts* e o pacote {P3} *Business Management*.

***Sprint* 3**

A semelhança dos *sprints* anteriores, procedeu-se à extração de *Epics* e casos de uso selecionadas para este *sprint*.

O primeiro subgrupo de *user stories* deu origem ao *Epic* "***As a/an** application, system administrator, I want to assign a token, Perform authentication and configure application, **to**…*" e ao caso de uso "{UC1.10} *Manage applications*". Deste caso de uso foram derivados outros casos

de uso, nomeadamente {UC1.10.1} *Configure application*, {UC1.10.2*}* *Perform authentication* e {UC1.10.3} *Assign a token*. O segundo subgrupo gerou o *Epic* "***As a*** *system Administrator, Stakeholders / Entity manager, I want to manage work tokens"* e o caso de uso *"{UC1.5} Manage work tokens"*. O terceira e último subgrupo, resultou no *Epic* "***As a*** *system Administrator, Corporate manager, I want to manage client companies, manage companies, group companies and factories, associate group* ***to…"*** e o caso de uso "{UC1.2} *Manage Stakeholders*". O caso de uso foi refinado dando origem ao caso de uso {UC1.2.1} *Manage companies* e este deu originou aos casos de uso {UC1.2.1.1} *Manage clientes companies* e {UC1.2.1.2} *Associate group.*

À semelhança dos *sprints* 1 e 2, foi executado o método 4SRS para a obtenção da arquitetura do *sprint* 3 (ver Figura 39). A tabela resultante com todas as decisões tomadas para este *sprint*, pode ser consultado no Anexo III.

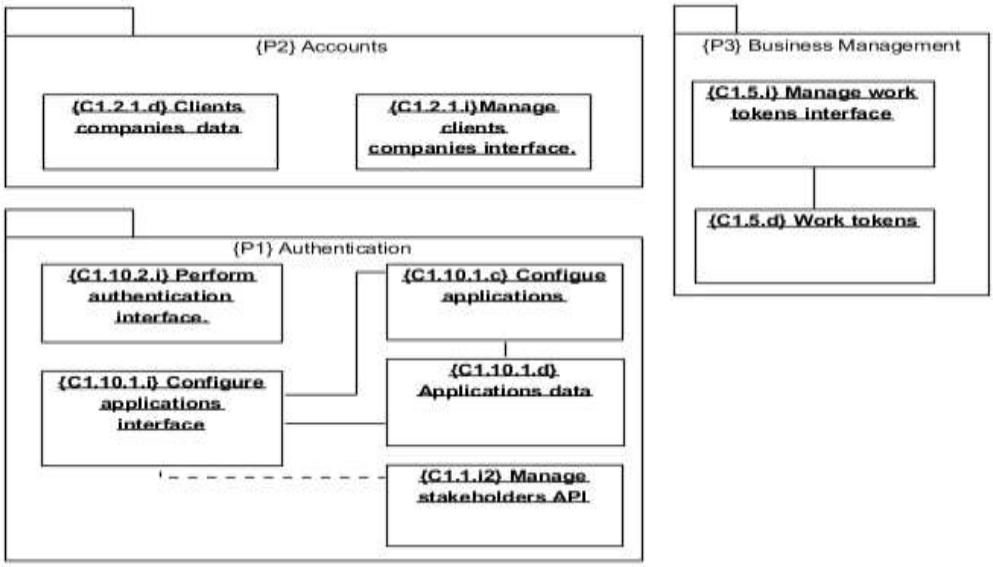



*Figura 39 - Arquitetura lógica do sprint 3*

O terceiro *sprint* é composto por 6 casos de uso refinados, 9 componentes e 2 pacotes, estando os componentes {C1.2.1.i} e {C2.2.1.d} contidos no pacote {P1} *Accounts*, enquanto os restantes componentes do diagrama estão conectados ao pacote {P3} *Authentication.*

***Sprint 4***

O quarto *sprint* é representado como um *Epic*. Como já definido anteriormente, um *Epic* descreve requisitos que ainda precisam de ser divididas em *user stories,* ou seja*,* um *Epic* representa uma ou mais *user stories*. Além disso, é discutido que um caso de uso contém um conjunto de *user stories* inter-relacionadas. Uma proposta comumente aceite de tais relações é apresentada por Cohn [37]. Para a abordagem proposta, um caso de uso é derivado diretamente de um *Epic*. Este *Epic* tem o mesmo nome que o caso de uso correspondente, ou seja, "*Epic: Consult SLA* e caso de uso "{UC1.8} *Consult SLA*". A Tabela resultante da aplicação do método 4SRS pode ser consultado no Anexo III, enquanto o diagrama lógico que resultou dessa tabela está representado na Figura 40.

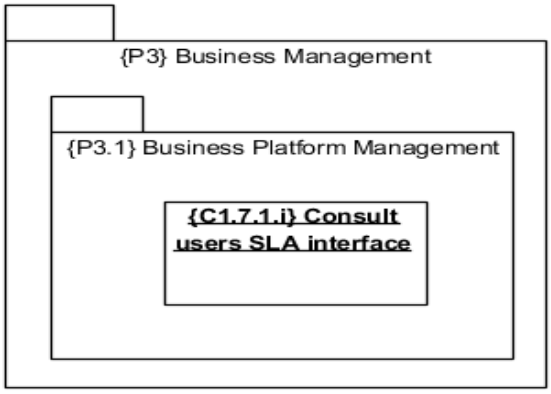



*Figura 40 - Arquitetura lógica do sprint 4*

Como se pode ver na Figura, o componente {C1.7.1.i} está conectado ao subpacote {P3.1} *Business Platform Management* que pertence ao pacote {P3} *Business Management*.

***Sprint* 5**

Seguindo a mesma lógica dos outros *sprints* procedeu-se ao agrupamento das *user stories* que compõem este *sprint* em subgrupos tendo em conta as inter-relações existentes entre elas. Sendo assim, o primeiro subgrupo é composto de 3 *user stories* que mantêm uma relação, depois

de feita a análise já explicada nas secções anteriores, procedeu-se à extração do *Epic* correspondente assim como do seu caso de uso como se pode ver na Figura 41.

***User Stories: - As a** System Administrator, Corporate manager, company manager, factory manager, forwarder manager, client manager and supplier manager, **I want to** associate a user account to a stakeholder or entity, **to** configure user account.*
***As a** System Administrator, Corporate manager, company manager, factory manager, forwarder manager, client manager and supplier manager, **I want to** associate a user account **to** a stakeholder or entity, **to** configure user account.*
***As a** System Administrator, Corporate manager, company manager, factory manager, forwarders, client and supplier, **I want to** CRUD a user account, **to** configure user account.*
***EPIC:** - As a System, Corporate manager, Company manager, Factory manager, Forwarder manager, Client manager and Supplier manager**, I want to** configure user account **to**…*
***Use. Case**: - {UC1.1.1} **Configure user account**.*

*Figura 41 - User stories, Epic e caso de uso do subgrupo 1 do sprint 5*

Do caso de uso {UC1.1.1} *Configure account,* resultou o caso de uso {UC1.1.1.1} *Create user account* e este por sua vez foi decomposto em {UC1.1.1.1.1} *Change user account,* {UC1.1.1.1.2} *Desable user account* e {UC1.1.1.1.3} *Consult user account*. O segundo subgrupo de *user stories* deu origem ao ***Epic*** "***As a** System Administrator, Corporate manager, Company manager, Factory manager, Forwarders, Client and Suppler, **I want to** a user profile and configure profile, **to**…*" e ao caso de uso "{UC1.7} *Manage user profiles*". O terceiro subgrupo é composto de uma única *user story* "***As a** Corporate manager, Company manager, Factory manager, Client, Supplier and Forwarders, I want to consult a contract information* in order to *consult SLAs*" dando origem ao ***Epic*** "*Consult SLA*" enquanto que o caso de uso corresponde é o "{UC3.1} *Define SLA*". Do quarto subgrupo resultou o ***Epic*** "***As a** System Administrator, Forwarder and Supplier, I want to forwarder companies and manage suppler companies **to**…*" e o caso de uso "{UC1.2} *Manage companies*". Este caso de uso foi depois decomposto em dois casos de uso "{UC1.2.1} *Manage forwarders companies*" e "{UC1.2.2} *Manage supplier companies*". O quinto e último subgrupo, gerou o *Epic* "***As a** System Administrator and user, I want to Perform authentication **to**…*" e o caso

de uso "{UC1.10.2} *Perform authentication*" e este por sua vez foi refinado dando origem aos casos de uso "{UC1.10.2.1} *Authentication*" e "UC1.10.2.2} *Recover account*".

Após terminar o processo de transformação, à semelhança dos *sprints* anteriores, está tudo a postos para a aplicação do método 4SRS através de execução de passos e micro passos representando as decisões tomadas. Estas decisões têm reflexo na arquitetura a ser gerada de acordo com a tabela de transformação resultante da aplicação deste método. Para o caso do *sprint* 5 em concreto, esta tabela pode ser consultada no Anexo III enquanto a Figura 42 apresenta a arquitetura lógica modelada usando a notação UML.

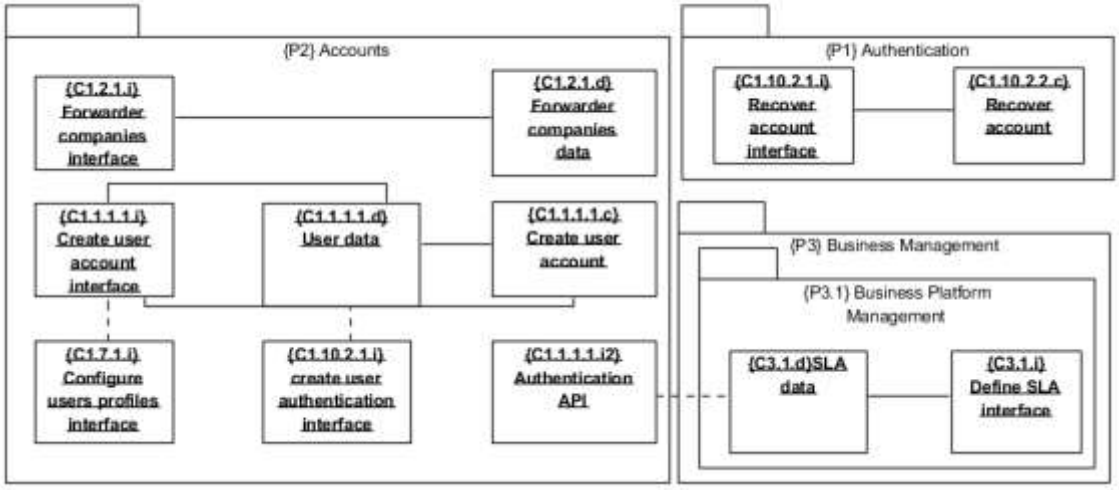



*Figura 42 - Arquitetura lógica do sprint 5*

Neste *sprint* foram implementados 9 casos de uso refinados, 12 componentes e 3 pacotes como se pode visualizar na Figura acima.

***Arquitetura Global***

Os modelos de diagrama de componentes / objeto obtido ao longo da execução dos 5 *sprints*, constituem em conjunto o modelo da arquitetura global do projeto UH4SP, adotando para cada *sprint* um refinamento funcional complementar no nível arquitetónico. Esse refinamento arquitetónico foi executado explicitamente num nível de desenvolvimento de serviço baseado em componentes. Depois de obter os modelos de diagramas dos 5 *sprints*

individualmente, agora é possível juntar todos os diagramas para a obtenção da arquitetura global do sistema. A Figura 43 apresenta a arquitetura global (com a abordagem proposta) do projeto UH4SP, representando a conexão dos componentes assim como os pacotes que pertencem.

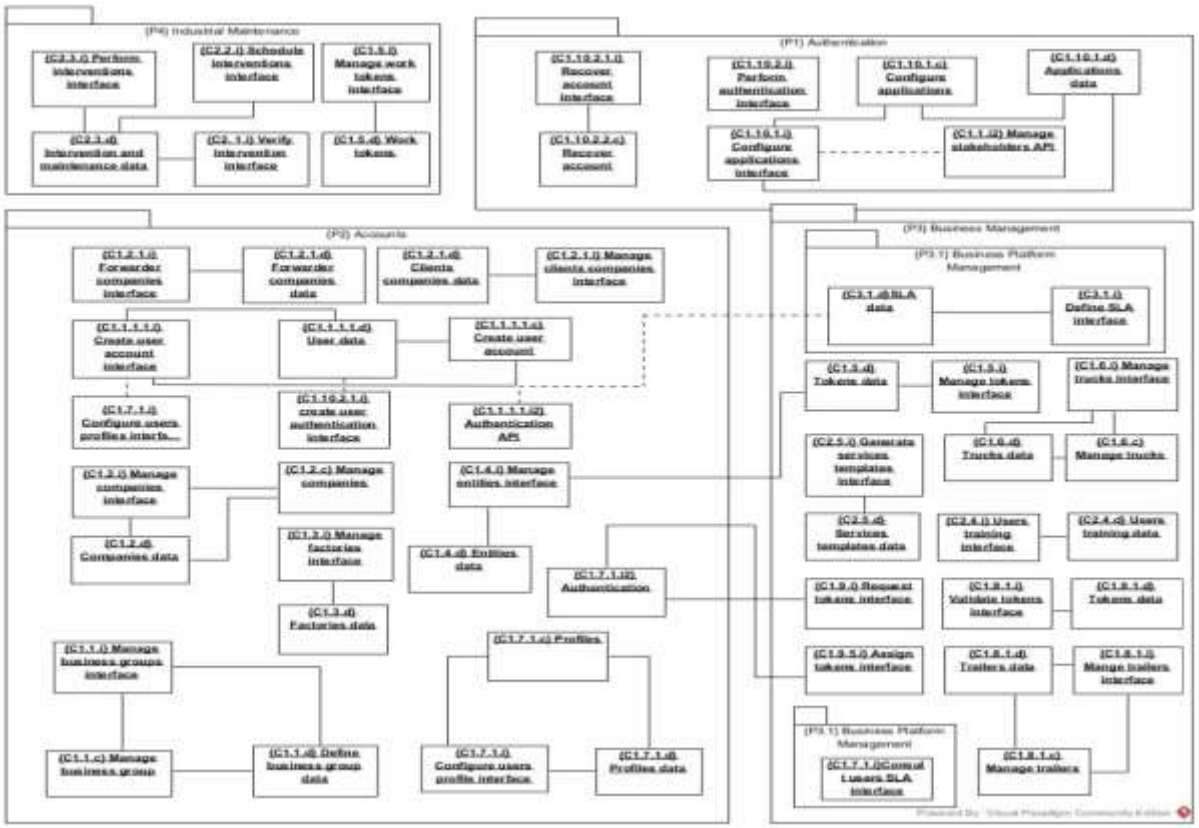


*Figura 43 -Arquitetura global do projeto UH4SP com a utilização da abordagem proposta*

A arquitetura representada acima, é o resultado da junção das arquiteturas refinadas através da aplicação do método 4SRS em cada *sprint*. Esta arquitetura é constituída por 4 pacotes e 56 componentes.

O pacote "{P1} *Authentication*" permite que o utilizador aceda à plataforma UH4SP de forma segura, garantindo assim a sua identidade, assim como fazer a recuperação ou alteração das suas credencias.

O pacote "{P2} *Accounts*" executa uma série de funcionalidades relacionadas com a configuração de contas e perfis dos utilizadores, assim como também faz a gestão dos *stakeholders*.

O pacote "{P3} *Business Management*" permite ao utilizador fazer a gestão de negócios das organizações assim como também é feita neste pacote a gestão de plataformas do projeto UH4SP.

Outro pacote que aparece no diagrama lógico modelado, é o "{P4} *Industrial Maitentance*" que é o responsável pela gestão da plataforma que contém os requisitos relacionados com a monitorização e manutenção das fábricas, nomeadamente a manutenção das máquinas que dão origem à informação que depois é utilizada pela plataforma.

A arquitetura apresenta também as associações entre componentes / objetos de *software* de acordo com a etapa número 4 do método 4SRS. As associações representadas em linhas tracejadas representam as associações de diagramas de casos de uso e as representadas em linhas retas, são as representantes das associações diretas.

## 4.5 Comparação de arquiteturas

Esta secção tem como objetivo comparar a arquitetura lógica global (Diagrama de pacotes) apresentada acima com a arquitetura resultante do processo tradicional (*Waterfall*). A Figura 44 apresenta as duas arquiteturas juntas lado a lado.

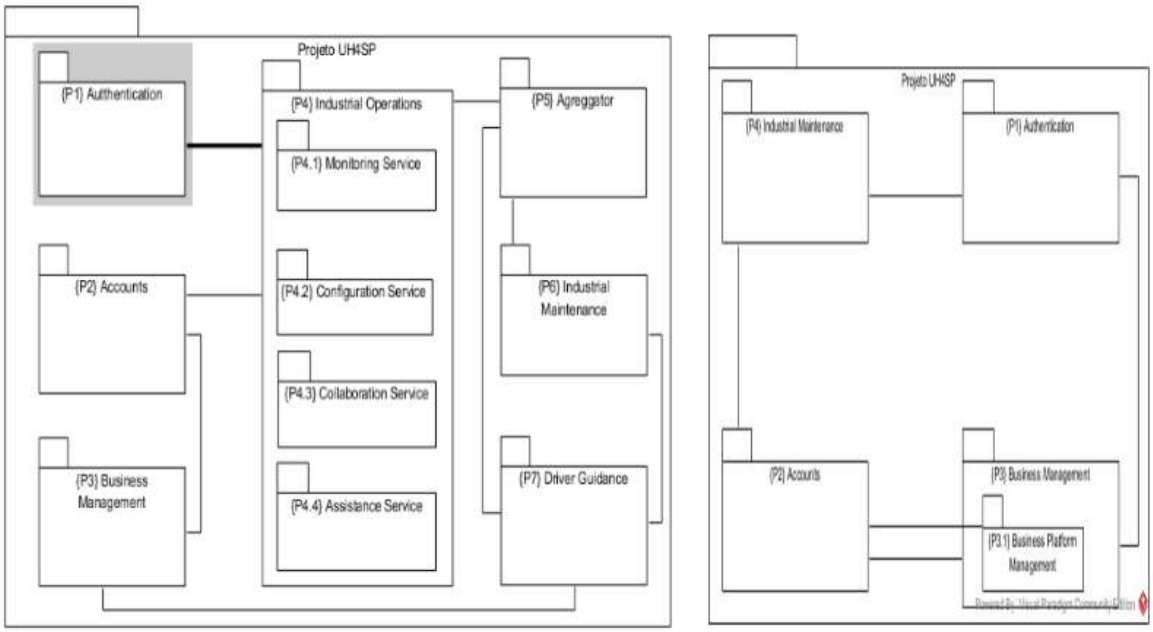



*Figura 44 - Comparação de arquiteturas lógicas (Diagrama de pacotes)*

Observando as duas arquiteturas, facilmente notamos que a arquitetura da direita (arquitetura obtida com aplicação da abordagem) está muito próxima em termos de resultados com a arquitetura da esquerda (arquitetura existente, obtido por processo tradicional). A arquitetura obtida apresenta um resultado semelhante em comparação com a obtida com aplicação do método tradicional (*Waterfall*) o que torna a abordagem proposta uma alternativa para a resolução do problema da complexidade nos sistemas de *software*, uma vez que a arquitetura resulta de casos de uso refinados através da utilização do método 4SRS que elimina os componentes redundantes assim como adiciona componentes em falta.

## 4.6 Conclusão

Neste capítulo foi abordada a questão central desta dissertação, apresentando-se tudo aquilo que foi realizado. Sendo que o objetivo principal foi a caracterização de um ambiente visual de apoio as cerimónias do *Scrum* (baseada em técnicas de modelação) começou-se por fazer a

priorização dos requisitos do sistema recolhidos em forma de *user stories* e depois de priorizadas, as *user stories* foram convertidas em casos de uso através da utilização do método indireto apresentado na secção 4.2, os casos de uso obtidos serviram como entrada para o método 4SRS.

Como resultado das atividades anteriores, isto é, levando em conta todos os requisitos modelados e refinados, procedeu-se à derivação das arquiteturas particulares de cada *sprint* e no final essas arquiteturas individuais foram agrupadas numa única arquitetura global. No total foram tratados 35 casos de uso, 4 pacotes e 56 componentes.

Com a validação da abordagem no projeto UH4SP, verificou-se a existência de benefícios, tais como o resultado das arquiteturas parciais obtidos de cada *sprint* torna fácil fazer a análise do valor de negócio das *user stories* selecionadas para aquele *sprint* e a potencial qualidade na arquitetura global (em termos de resultados esperados). A validação foi considerada um sucesso, na medida em que todos *sprints* geraram arquiteturas lógicas aceitáveis e deste modo diminuindo a complexidade no projeto. A arquitetura global foi comparada com a resultante do processo tradicional (*Waterfall*) e concluiu-se que a arquitetura obtida com a aplicação da abordagem proposta apresenta um resultado próximo da arquitetura obtida com a abordagem tradicional.

# 5. CONCLUSÕES E TRABALHO FUTURO

Neste capítulo apresenta-se as principais conclusões deste trabalho de investigação. Estas conclusões abordam os principais resultados e contribuições deste projeto. São apresentadas ainda a seguir, as principais limitações deste estudo. O capítulo termina, com sugestões de trabalhos futuros.

## 5.1 Conclusões

O presente trabalho foi realizado com o objetivo de propor um ambiente visual para apoiar as cerimónias do *Scrum,* baseado nas técnicas de modelação UML. Este trabalho pretende contribuir para fornecer um entendimento comum e melhorado a todos os intervenientes no processo de desenvolvimento de *software*, cumprindo as exigências da abordagem *Scrum*.

A temática de gestão de projetos de *software*, com todos os contextos associados, não é uma questão binária de métodos ágeis contra métodos tradicionais. Embora exista uma clara disrupção de pensamentos entre ambas as tendências, existem pontos de contacto e de correlação que devem ser explorados e adaptados de modo a conjugar o melhor dos dois mundos. A necessidade de especificação formal e de definição de arquiteturas sólidas na fase de *design* dos métodos ágeis. A proposta de um ambiente visual tenta introduzir a variável referente à agilidade numa técnica (4SRS) mais vocacionada à sua integração nos métodos mais tradicionais e formais.

A metodologia *Scrum* é uma metodologia tão leve e desprovida de referências relacionadas com a construção do produto que nem sempre é considerada uma metodologia. *Scrum* é um *framework* no qual as pessoas podem abordar problemas adaptativos complexos, ao mesmo tempo em que entregam, produtiva e criativamente, produtos do mais alto valor possível. No seu referencial, o *Scrum* indica eventos, papéis e artefactos que, se utilizados de forma

correta, conseguem praticamente autogerir uma equipa que tenha um elevado grau de comprometimento relacionado com o trabalho a executar.

Relativamente aos objetivos propostos neste trabalho, foram analisadas as metodologias ágeis de desenvolvimento de *software*, com destaque para o *Scrum* que foi analisado com extensivo detalhe, oferecendo uma panorâmica de funcionamento de ciclo iterativo, ou *sprint*, explicando as responsabilidades de cada elemento na equipa *Scrum*, detalhando os artefactos que a mesma pode utilizar para tornar o *sprint* mais produtivo, e ainda explicando a cadência e estrutura de cada evento necessário para completar o ciclo.

Foram ainda estudadas as técnicas de modelação de *software* com especial destaque para as técnicas baseadas na notação UML. Foram explicados os principais diagramas que compõem a notação UML, nomeadamente o diagrama de casos de uso, diagrama de classe, diagramas de sequências, componentes e outros. Entre todos, deu-se maior atenção aos diagramas de casos de uso e de classes (diagrama de objetos) por serem os diagramas que foram utilizados no âmbito deste trabalho (requisitos e arquitetura).

No capítulo 3 foram analisadas algumas propostas de ambientes visuais existentes na literatura com o objetivo de fornecerem uma visão abrangente do ambiente visual que seria proposto no capítulo seguinte. Foram estudas as principais características e no final foi feita uma comparação no sentido de se avaliar as vantagens e desvantagens de cada uma das abordagens.

Adicionalmente, foi desenvolvido um protótipo de *software* que confere uma certa automatização à abordagem proposta. O protótipo converte *user stories* em diagramas de casos de uso. Também foi especificado um modelo de processo que permite a transformação de casos de uso no modelo de arquitetura lógica compatível com as equipas de *Scrum*, formato em diagramas de componentes. O processo de transformação de *user stories* em caso de uso consiste no seguinte: as *user stories* são transformadas em *Epics* através de um processo totalmente manual e os *Epics* dão origem a casos de uso que servem de *input* para o método 4SRS.

A análise do estudo de caso permitiu a aplicação do método 4SRS num processo ágil (*Scrum*), dando uma série de contribuições para a abordagem. Uma contribuição valiosa do método 4SRS é a sua capacidade de refinar artefactos de *design* (arquitetura lógico). Portanto, o método 4SRS é apropriado para contextos multi projetos e / ou multi equipas. O refinamento de

arquiteturas de *software* lógico também é relevante para reduzir a complexidade na atividade de modelação de sistemas de *software* em larga escala (justamente o que se pretendia com aplicação deste na abordagem proposta).

Outra contribuição deste método na abordagem proposta é a sua capacidade de exigir remoção de requisitos redundantes e a descoberta de requisitos em falta (este último tanto no nível de componentes de arquitetura lógica quanto no nível de casos de uso).

Ainda sobre o trabalho desenvolvido, foram aceites dois artigos científicos em duas conferências internacionais.

## 5.2 Limitação da proposta

Embora os objetivos a que esta dissertação se propôs tenham sido alcançados, essas respostas não surgiram desprovidas de limitações que necessitam de trabalho suplementar para conferir melhorias à abordagem proposta.

**Adaptação semântica e de conteúdo**. O ambiente visual proposto apresenta algumas limitações que resultaram das dificuldades próprios da realização de um trabalho de final do curso. A priorização de *user stories* decorreu sob certas limitações. Primeira, a falta de experiência na tarefa de priorização de requisitos. Segundo, a diversidade (funções, opiniões, tarefas) dos *stakeholders* poderia ter melhorado os resultados da arquitetura lógica, se esta tarefa fosse realizada num ambiente próprio de um projeto.

**Validação da proposta no terreno.** Embora exista a explicação no contexto real da abordagem proposta, a sua aplicação ainda não saiu do papel, precisando de uma validação no terreno, com acompanhamento constante sobre adequabilidade do processo de transformação de *user stories* em casos de uso e do processo de geração da arquitetura lógica a partir desses casos de uso. Esta falta de validação pode provocar alterações na estrutura dos casos de uso e consequentemente da arquitetura lógica, caso se verifique que existe informação prescindível, ou que, pelo contrário, falte informação que facilitaria o processo de análise de requisitos agregando valor ao projeto.

Inclusivamente, a aplicação da abordagem por vários analistas de sistemas experientes, com conhecimento relativamente à gestão de requisitos em projetos ágeis, poderia significar um conjunto de importantes contributos que, além de validar o resultado final da aplicação da abordagem, poderia resultar em várias sugestões que simplificassem e concedessem outras valências à proposta apresentada.

## 5.3 Trabalho Futuro

De forma a ultrapassar as limitações indicadas existe um conjunto de tarefas futuras que, quando executadas, poderão conferir uma melhoria na abordagem proposta.

**Automatização.** Uma vez que a abordagem depende de informação que pode ser estruturada e de diagramas lógicos e do processo de transformação de *user stories* em casos de uso, torna-se indispensável a automação do processo. Isso pouparia bastante esforço de consulta de documentação e agregação da informação.

**Geração automática de casos de uso a partir de *user stories***. De modo a melhorar a geração automática de casos de uso a partir de *user stories*, propõe-se a continuação do processo de desenvolvimento do protótipo de *software* iniciado nesta dissertação.

Outro importante trabalho a ser feito nesta abordagem, é a conversão de *user stories* em casos de uso usando a técnica de aprendizagem de máquina (*machine learning)*.

# REFERÊNCIAS


[1] X. Yaohong and F. Jingtao, “Research on Software Development Process Conjunction of Scrum and UML Modeling,” 2014.

[2] M. Ferreira, A. Tereso, P. Ribeiro, G. Fernandes, and I. Loureiro, “Project Management Practices in Private Portuguese Organizations,” *Procedia Technol.*, vol. 9, pp. 608–617, 2013.

[3] M. Brian, M. R. C, M. Steve, S. Ken, S. Jeff, and T. Dave, “Manifesto for Agile Software Development Twelve Principles of Agile Software,” 2001.

[4] J. Doronina and E. Doronina, “MODELS OF IT-PROJECT MANAGEMENT,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 10, no. 5, 2018.

[5] M. Elallaoui, K. Nafil, and R. Touahni, “Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques,” *Procedia Comput. Sci.*, vol. 130, pp. 42–49, 2018.

[6] M. Cohn, *User Stories Applied: For Agile Software Development (Addison Wesley Signature Series)*. Addison-Wesley, 2004.

[7] A. Kriouile, N. Addamssiri, and T. Gadi, “Abdelouahed Kriouile, Najiba Addamssiri, Taoufiq Gadi. An MDA Method for Automatic Transformation of Models from CIM to PIM,” *Am. J. Softw. Eng. Appl.*, vol. 4, no. 1, pp. 1–14, 2015.

[8] S. Grapenthin, S. Poggel, M. Book, and V. Gruhn, “Improving task breakdown comprehensiveness in agile projects with an Interaction Room,” *Inf. Softw. Technol.*, vol. 67, pp. 254–264, 2015.

[9] J. F. Nunamaker, M. Chen, and T. D. M. Purdin, “Systems Development in Information Systems Research,” *J. Manag. Inf. Syst.*, vol. 7, no. 3, pp. 89–106, 2015.

[10] M. Berndtsson, J. Hansson, B. Olsson, and B. Lundell, *Thesis Projects*. London: Springer London, 2008.

[11] Adomavicius, Bockstedt, Gupta, and Kauffman, “Making Sense of Technology Trends in the Information Technology Landscape: A Design Science Approach,” *MIS Q.*, vol. 32, no. 4, p. 779, 2017.

[12] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A Design Science Research

Methodology for Information Systems Research," *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–77, 2008.

[13] B. Kuechler and S. Petter, "Design Science Research in Information Systems," *Des. Sci. Res. Inf. Syst.*, pp. 1–66, 2017.

[14] E. F. Cruz, "Design Science Research em Sistemas de Informação," no. June, 2011.

[15] I. Sommerville, *SOFTWARE ENGINEERING Nith Edition, Addison-Wesley*, vol. 35, no. 2. 2009.

[16] P. . Roger R. Pressman, *Software Engineering Practicioner's Approach Fifth Edition, Mc Graw Hill*. 2001.

[17] R. S. Pressman, "Engenharia de Software," 2011. [Online]. Available: https://pt.scribd.com/document/347525214/Engenharia-de-Software-Pressman-2011-pdf. [Accessed: 05-Nov-2018].

[18] G. Hurlburt and J. Voas, "Software is driving software engineering?," *IEEE Softw.*, vol. 33, no. 1, pp. 101–104, 2016.

[19] L. N. Raha, "A Guide for Building the Knowledgebase for Software Entrepreneurs, Firms, and Professional Students - IEEE Conference Publication," *2018 IEEE 16th Int. Conf. Softw. Eng. Res. Manag. Appl.*, pp. 165–171, 2018.

[20] J. Nandhakumar and D. E. Avison, "The fiction of methodological development: A field study of information systems development," *Inf. Technol. People*, vol. 12, no. 2, pp. 176–191, 1999.

[21] D. L. P. A. P. C. CLEMENTS, "A Rational Design Process: How and Why to Fake IT." 1986.

[22] I. Sommerville, "Software Process Models SPECIFICATION-BASED MODELS," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 269--271, Mar. 1996.

[23] P. Meso and R. Jain, "Agile software development: Adaptive systems principles and rest practices," *Inf. Syst. Manag.*, vol. 23, no. 3, pp. 19–30, 2006.

[24] A. Silveira Campanelli and F. S. Parreiras, "The Journal of Systems and Software Agile methods tailoring-A systematic literature review," *J. Syst. Softw.*, vol. 110, pp. 85–100, 2015.

[25] K. Schwaber and J. Sutherland, "2017 Scrum Guide," vol. 19, no. 6, p. 504, 2017.

[26] K. Schwaber, “SCRUM Development Process,” 1994.

[27] K. Schwaber, M. Beedle, and M. B. KEN SCHWABER, “Agile Software Development with Scrum,” p. 158, 2002.

[28] K. Beck, “Change with Extreme Programming,” *Ieee*, no. c, pp. 70–77, 1999.

[29] A. Cockburn and J. Highsmith, “Agile software development: The people factor,” *Computer (Long. Beach. Calif).*, vol. 34, no. 11, pp. 131–133, 2001.

[30] J. Stapleton, *DSDM, dynamic systems development method : the method in practice*. Addison-Wesley, 1997.

[31] J. Stapleton and DSDM Consortium., *DSDM : business focused development*. Addison-Wesley, 2003.

[32] S. R. (Stephen R. Palmer and J. M. Felsing, *A practical guide to feature-driven development*. Prentice Hall PTR, 2002.

[33] T. Chow and D.-B. Cao, “A survey study of critical success factors in agile software projects,” *J. Syst. Softw.*, vol. 81, no. 6, pp. 961–971, 2008.

[34] B. Choudhary and S. K. Rakesh, “An approach using agile method for software development,” in *2016 1st International Conference on Innovation and Challenges in Cyber Security, ICICCS 2016*, 2016, pp. 155–158.

[35] D. Stankovic, V. Nikolic, M. Djordjevic, and D.-B. B. Cao, “A survey study of critical success factors in agile software projects in former Yugoslavia IT companies,” *J. Syst. Softw.*, vol. 86, no. 6, pp. 1663–1678, 2013.

[36] J. Shore and S. Warden, *The art of agile development*. O’Reilly Media, Inc, 2008.

[37] M. Cohn, *User stories applied : for agile software development*. Addison-Wesley, 2004.

[38] G. Crepaldi, *Ninth edition of the G.B. Morgagni Awards Program*, vol. 35, no. 2. 2005.

[39] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods Review and analysis,” 2002.

[40] M. Qasaimeh and A. Abran, “Extending Extreme Programming User Stories to Meet ISO 9001 Formality Requirements,” *J. Softw. Eng. Appl.*, vol. 04, no. 11, pp. 626–638, 2011.

[41] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, “Empirical Study of Agile Software

Development Methodologies," *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 1, pp. 1–6, 2015.

[42] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New directions on agile methods: A comparative analysis," *Proc. - Int. Conf. Softw. Eng.*, no. June, pp. 244–254, 2003.

[43] J. Highsmith, C. Consortium, and A. Cockburn, "Development : The Business of Innovation," pp. 120–122, 2001.

[44] L. Williams, "Agile Software Development (class 1 handout)," 2002.

[45] P. Kettunen, "Adopting key lessons from agile manufacturing to agile software product development—A comparative study," *Technovation*, vol. 29, no. 6, pp. 408–422, 2009.

[46] H. Kufner and M. Vogt, "Betreuung Von Drogenabhangigen in Bauerlichen Familien," *Sucht*, vol. 41, no. 2. pp. 98–99, 1995.

[47] B. J. j. Voigt, "Dynamic System Development Method," Zürich, Switzerland, 2004.

[48] P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and architecture: Can they coexist?," 2010.

[49] Takeuchi and Nonaka, "Scrum Godfathers: Takeuchi and Nonaka - Scrum Inc." [Online]. Available: https://www.scruminc.com/scrum-godfathers-takeuchi-and-nonaka/. [Accessed: 18-Jan-2019].

[50] E. M. Simao and E. M. Simão, "Comparison of Software Development Methodologies based on the SWEBOK," Univesidade do Minho, 2011.

[51] L. Rising and N. S. Janoff, "The Scrum Software Development Process for Small Teams," *Software, IEEE*, vol. 17, Issue, no. August, pp. 26–32, 2000.

[52] D. Nguyen-Cong and D. Tran-Cao, "A review of effort estimation studies in agile, iterative and incremental software development," *Proc. - 2013 RIVF Int. Conf. Comput. Commun. Technol. Res. Innov. Vis. Futur. RIVF 2013*, pp. 27–30, 2013.

[53] G. H. Williams, M. L. Tuck, J. M. Sullivan, R. G. Dluhy, and N. K. Hollenberg, "Parallel adrenal and renal abnormalities in young patients with essential hypertension," *Am. J. Med.*, vol. 72, no. 6, pp. 907–914, 1982.

[54] C. Xing and D. M. Isaacowitz, "SCRUM," *Motiv. Emot.*, vol. 30, no. 3, pp. 243–250, 2006.

[55] N. P. Jeldi and V. K. M. Chavali, “Software Development Using Agile Methodology Using Scrum Framework,” *Int. J. Sci. Res. Publ.*, vol. 3, no. 4, pp. 3–5, 2013.

[56] K. Schwaber and J. Sutherland, “The Scrum guide,” vol. 2, no. July, p. 17, 2011.

[57] C.F.R.Sandro, “Recomendações para a adoção de prá cas ágeis no desenvolvimento de so ware : es- tudo de casos,” no. 104, 2016.

[58] I. Kayes, M. Sarker, and J. Chakareski, “Product Backlog Rating : A Case Study On Measuring Test Quality In Scrum,” 2014.

[59] M. Mekni, G. Buddhavarapu, S. Chinthapatla, and M. Gangula, “Software Architectural Design in Agile Environments,” *J. Comput. Commun.*, vol. 06, no. 01, pp. 171–189, 2018.

[60] A. M. N. Ribeiro, “Um Processo de Modelação de Sistemas Software com Integração de Especificações Rigorosas Tese, Universidade do Minho,” 2008.

[61] P. Bourque and R. E. Fairley, “SWEBOK v.3 - Guide to the Software Engineering - Body of Knowledge.,” 2014.

[62] R. E. D. Fairley, P. Bourque, and J. Keppler, “The impact of SWEBOK Version 3 on software engineering education and training,” *2014 IEEE 27th Conf. Softw. Eng. Educ. Training, CSEE T 2014 - Proc.*, pp. 192–200, 2014.

[63] D. Youll, “Software engineering: Principles and practice H van Vliet Wiley (1993) 558 pp £19.95 ISBN 0 471 93611 1,” *Inf. Softw. Technol.*, vol. 36, no. 1, p. 58, 1994.

[64] B. Rumpe, “Agile Modeling with the UML 1 Portfolio of Software Engineering Techniques,” *Radic. Innov. Softw. Syst. Eng. Futur. 9th Int. Work.*, no. October 2002, pp. 297–309, 2004.

[65] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, vol. 3. 1998.

[66] Scott W.Ambler, “Modelagem Ágil ( AM ),” 2002. [Online]. Available: http://www.agilemodeling.com/.

[67] I. Sommerville, “Software Process Models,” 1996.

[68] M.Fowler, “Unified Modeling Language,” vol. 1, no. December, 2017.

[69] R. J. Machado, J. M. Fernandes, P. Monteiro, and H. Rodrigues, “Transformation of UML Models for Service-Oriented Software Architectures,” *12th IEEE Int. Conf. Work. Eng. Comput. Syst.*, pp. 173–182, 2005.

[70] N. Santos *et al.*, “Specifying software services for fog computing architectures using recursive model transformations,” *Fog Comput. Concepts, Fram. Technol.*, pp. 153–181, 2018.

[71] N. Ferreira, N. Santos, R. J. MacHado, and D. Gašević, “Derivation of process-oriented logical architectures: An elicitation approach for cloud design,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7343 LNCS, Springer-Verlag, 2012, pp. 44–58.

[72] S. Azevedo, R. J. Machado, and R. S. P. Maciel, “On the use of model transformations for the automation of the 4SRS transition method,” *Lect. Notes Bus. Inf. Process.*, vol. 112 LNBIP, pp. 249–264, 2012.

[73] G. Lucassen, *Understanding User Stories*. eboren op 10 juni 1991 te Voorburg, 2017.

[74] M. Elallaoui, “Automatic generation of UML sequence diagrams from user stories in Scrum process,” 2015.

[75] G. Lucassen, F. Dalpiaz, J. M. E. M. Van Der Werf, and S. Brinkkemper, “Visualizing user story requirements at multiple granularity levels via semantic relatedness,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9974 LNCS, pp. 463–478, 2016.

[76] B. W. Boehm and B. W., “A spiral model of software development and enhancement,” *Computer (Long. Beach. Calif).*, vol. 21, no. 5, pp. 61–72, May 1988.

[77] OMG, “Unified Modeling Language : Superstructure,” *Career Anal. Des.*, no. August, 2005.

[78] B. Alattar and N. M. Norwawi, “A personalized search engine based on correlation clustering method,” *J. Theor. Appl. Inf. Technol.*, vol. 93, no. 2, pp. 345–352, 2016.

[79] OMG, “Unified Modeling Language: Infrastructure,” no. March, 2006.

[80] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, “Unifying and extending user story models,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8484 LNCS, pp. 211–225.

[81] H. Herchi and W. Ben Abdessalem, “From user requirements to UML class diagram,” 2012.

[82] M. Elallaoui, K. Nafil, and R. Touahni, “Automatic generation of TestNG tests cases from UML sequence diagrams in Scrum process,” *Colloq. Inf. Sci. Technol. Cist*, pp. 65–70, 2017.

[83] D. K. Deeptimahanti and R. Sanyal, *Semi-automatic Generation of UML Models from*

*Natural Language Requirements*. 2011.

[84] Klein and Manning, “The Stanford Natural Language Processing Group,” 2007. [Online]. Available: https://nlp.stanford.edu/software/lex-parser.shtml. [Accessed: 27-Dec-2018].

[85] M. Abdouli, W. B. A. Karaa, and H. Ben Ghezala, “Survey of works that transform requirements into UML diagrams,” *2016 IEEE/ACIS 14th Int. Conf. Softw. Eng. Res. Manag. Appl. SERA 2016*, pp. 117–123, 2016.

[86] A. Marcus, A. Sergeyev, V. Rajlieh, and J. I. Maletic, “An information retrieval approach to concept location in source code,” in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2004, pp. 214–223.

[87] S. Gulia and T. Choudhury, “An efficient automated design to generate UML diagram from Natural Language Specifications,” *Proc. 2016 6th Int. Conf. - Cloud Syst. Big Data Eng. Conflu. 2016*, pp. 641–648, 2016.

[88] D. K. Deeptimahanti and M. A. Babar, “An automated tool for generating UML models from natural language requirements,” *ASE2009 - 24th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 680–682, 2009.

[89] P. More and R. Phalnikar, “ISSN : 2249-0868 Foundation of Computer Science FCS,” 2012.

[90] B. Moros, A. Toval, F. Rosique, and P. Sánchez, “Transforming and tracing reused requirements models to home automation models,” *Inf. Softw. Technol.*, vol. 55, pp. 941–965, 2013.

[91] S. M. Fevereiro De Azevedo, “Refinement and Variability Techniques in Model Transformation of Software Requirements,” 2014.

[92] C. Campos, “Estudo de Interdependências e Rastreabilidade no Processo RUP: Análise do Micro-Processo V-Model e do Método 4SRS,” 2013.

[93] A. L. Ferreira, R. J. Machado, and M. C. Paulk, “An approach to software process design and implementation using transition rules,” *Proc. - 37th EUROMICRO Conf. Softw. Eng. Adv. Appl. SEAA 2011*, pp. 330–333, 2011.

[94] R. S. Madanayake, G. K. A. Dias, and N. D. Kodikara, “Transforming Simplified Requirement in to a UML Use Case Diagram Using an Open Source Tool,” *Int. J. Comput. Sci. Softw. Eng.*, vol. 6, no. 3, pp. 2409–4285, 2017.

[95] N. Santos *et al.*, “Decomposing monolithic to microservices architectures: a modeling approach,” no. August, 2018.

[96] A. Kleppe, J. Warmer, W. Bast, and A. Wesley, “Addison Wesley - MDA Explained, The Model Driven Architecture, Practice and Promise - April 2003,” 2003.

[97] Plant UML a free UML Diagramming tool., “Commons:Wiki Loves Love 2019 - Wikimedia Commons,” 2019. [Online]. Available: https://commons.wikimedia.org/wiki/Commons:Wiki_Loves_Love_2019. [Accessed: 07-Feb-2019].

[98] http://plantuml. com. Plant UML in a Nutshell, “Use case Diagram syntax and features,” 2019. [Online]. Available: http://plantuml.com/use-case-diagram. [Accessed: 07-Feb-2019].

[99] How can we map use cases to stories?:, “How can we map use cases to stories? – Practitioners Agile Guidebook,” *18/06/2008*, 2008. [Online]. Available: https://agilefaq.wordpress.com/2008/06/18/how-can-we-map-use-cases-to-stories/. [Accessed: 10-Feb-2019].

[100] User Stories: An Agile Introduction, “User Stories: An Agile Introduction,” 2016. [Online]. Available: http://www.agilemodeling.com/artifacts/userStory.htm. [Accessed: 08-Mar-2019].

[101] R. Madanayake, G. K. A. Dias, and N. D. Kodikara, “Use Stories vs UML Use Cases in Modular Transformation,” *Int. J. Sci. Eng. Appl. Sci.*, vol. 3, no. 1, pp. 2395–3470, 2016.

[102] R. J. Machado, J. M. Fernandes, P. Monteiro, and H. Rodrigues, “LNCS 4034 - Refinement of Software Architectures by Recursive Model Transformations,” 2006.

[103] R. Popli, N. Chauhan, and H. Sharma, “Prioritising user stories in agile environment,” *Proc. 2014 Int. Conf. Issues Challenges Intell. Comput. Tech. ICICT 2014*, pp. 515–519, 2014.

[104] J. M. Fernandes and R. J. Machado, “From Use Cases to Objects: An Industrial Information Systems Case Study Analysis,” 2001.

# ANEXOS

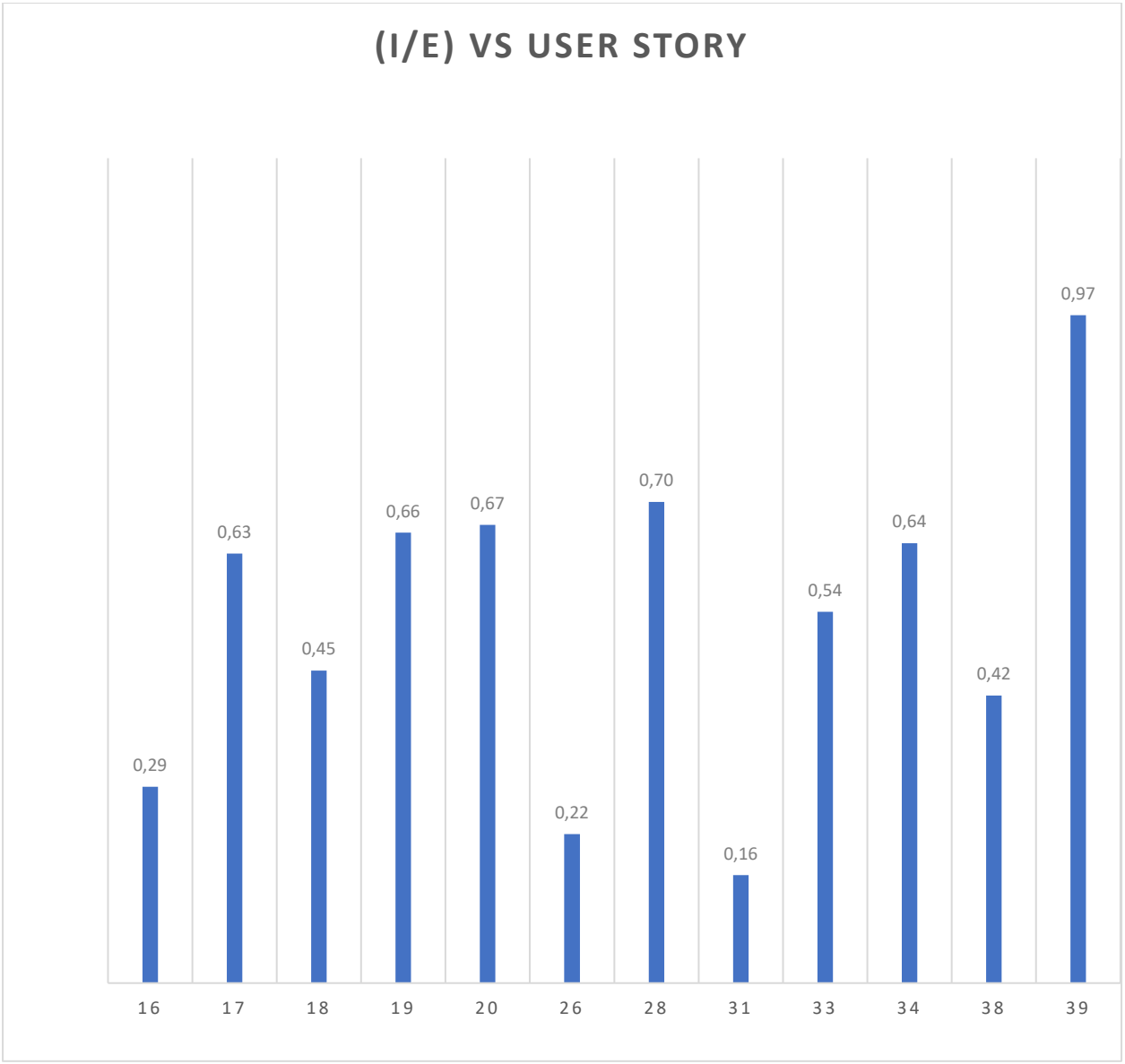
## ANEXO I – Tabela de Priorização de *User Stories*

| User Stories, Importance and Effort Factor | | | | |
|---|---|---|---|---|
| **ID** | **User Story** | **I** | **E** | **I/E** |
| **1** | As a System Administrator, Corporate manager, company manager, factory manager, forwarders, client and supplier, I want to CRUD a user account, to configure user account. | 5 | 70 | 0,07 |
| **2** | As a System Administrator, Corporate manager, Company manager, Factory manager, Forwarders, Client and Supplier, I want to CRUD a user profile to configure user profile. | 3 | 80 | 0,04 |
| **3** | As a System Administrator, Corporate manager, Company manager, Factory manager, Forwarders, Client and Supplier, I want to assign permissions to a user profile | 8 | 80 | 0,10 |
| **Epic** | Consult SLA | **17** | 70 | 0,24 |
| **4** | As a Corporate manager, Company manager, Factory manager, Client, Supplier and Forwarders, I want to consult a contract information in order to consult SLAs. | 12 | 70 | 0,17 |
| **5** | As a System administrator and user, I want to Insert username and password to perform authentication | 1 | 50 | 0,02 |
| **6** | As a System administrator and user, I want to recover account username or password to perform authentication | 7 | 80 | 0,09 |
| **9** | As a System Administrator or a forwarder, I want to CRUD a forwarder company in order to manage forwarder companies. | 2 | 60 | 0,03 |
| **10** | As a System Administrator or a client admin, I want to CRUD a client companies in order to manage client companies, and last manage companies. | 17 | 65 | 0,26 |
| **11** | As a System Administrator or a supplier, I want to CRUD a supplier company in order to manage supplier companies. | 10 | 80 | 0,13 |
| **12** | As a System administrator, Corporate manager, Company manager or Factory manager, I want to CRUD factories in order to manage factories | 95 | 70 | 1,36 |
| **14** | As a Forwarder admin or systems admin, I want to associate trailers to a given trucks. | **70** | 71 | 0,99 |
| **15** | As a Forwarder admin or systems admin, I want to CRUD trailers in order to manage trailers. | **60** | 70 | 0,86 |
| **16** | As a System Administrator, I want to CRUD an application account, to configure application account. | 20 | 70 | 0,29 |
| **17** | As an application, I want to send an app_gid and a GPS location to perform authentication, in order to access the UH4SP WebAPIs. | 30 | 48 | 0,63 |
| **18** | As a System Administrator, I want to assign or refresh a token to an application. | 25 | 55 | 0,45 |
| **19** | As a System administrator, Corporate manager, Company manager, and Factory manager, Forwarder, Client or Supplier, I want to visualize graphical indicators about my group, companies, factories or to resources or data that I am associated. | 38 | 58 | 0,66 |
| **20** | As a System administrator, Corporate manager, Company manager, and Factory manager, Forwarder, Client or Supplier, I want to visualize system operations about my groups, companies, factories or to resources or data that I am associated. | 40 | 60 | 0,67 |
| **21** | As a web application, I want to receive a JSON file in order to develop dashboards. | **75** | 65 | 1,15 |
| **22** | As a System administrator, Corporate manager, Company manager, and Factory manager, Forwarder, Client or Supplier, I want to configure dashboards settings in order to configure dashboards. | **65** | 70 | 0,93 |

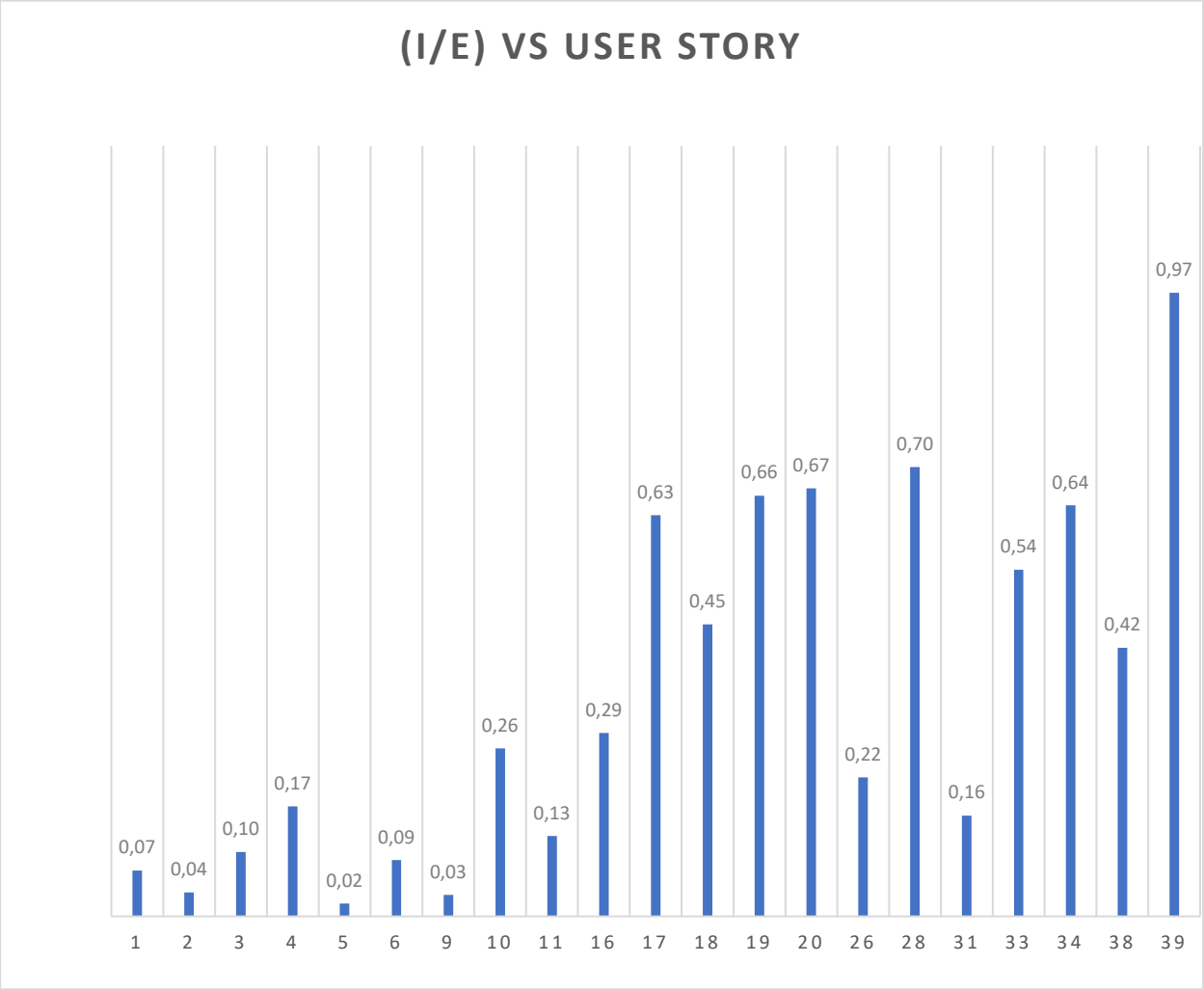
| | | | | |
|---|---|---|---|---|
| **23** | As a service, I want to get users permissions to access to a particular data source. | **55** | 70 | 0,79 |
| 26 | As a System Administrator, Corporate manager, company manager, factory manager, forwarder manager, client manager and supplier manager, I want to associate a user account to a stakeholder or entity, to configure user account. | 13 | 60 | 0,22 |
| **27** | As a System administrator, Corporate manager, Company manager, and Factory manager, Forwarder, Client or Supplier, I want to configure dashboards settings in order to configure dashboards. | **72** | 71 | 1,01 |
| **28** | As a service, I want to get users permissions to access to a particular data source. | 42 | 60 | 0,70 |
| **31** | As a System Administrator, Corporate manager, company manager, factory manager, forwarder manager, client manager and supplier manager, I want to associate a user account to a stakeholder or entity, to configure user account. | 11 | 70 | 0,16 |
| **33** | As a System Administrator I want to validate work tokens that was requested by managers in order to manage work tokens | 27 | 50 | 0,54 |
| **34** | As a Corporate manager I want to request, read, update and disable work tokens in order to manage work tokens to my group companies and factories | 32 | 50 | 0,64 |
| **35** | As a Company manager I want to request, read, update and disable work tokens in order to manage work tokens to my Company factories. | **63** | 70 | 0,90 |
| **36** | As a Factory manager I want to request, read, update and disable work tokens in order to ***manage work tokens to my factory.*** | **53** | 70 | 0,76 |
| **38** | As a Stakeholder/Entity manager I want to receive a notification when a given work tokens were associated to my entity in order to manage work tokens. | 23 | 55 | 0,42 |
| **39** | As an Entity (forwarder, client or supplier) manager I want to assign drivers and trucks to work tokens that were associated to my entity in order to manage work tokens. | **68** | 70 | 0,97 |
| **40** | As a System admin I want to receive a notification when a stakeholder/entity manager request work tokens in order to validate work tokens. | **47** | 65 | 0,72 |

## ANEXO II – Gráficos de Priorização de *Sprints*

*Sprint 2*



*Sprints 3, 4 e 5*



## ANEXO III - Tabela *Four Step Rule Set*

4SRS – *Sprint* 1

| Step 1 - Component Creation | | | | | | Step 2 - Component Elimination | | | | | Step 3 - Packing & Aggregation | Step 4 - Component Association | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Use Case | Description | 2i | 2 ii | 2iii | 2iv | 2v | | 2vi | 2vii | 2viii | | 4i | 4 ii |
| {UC1.4.1} | Manage business groups | cdi | | | | | | | | | | | |
| {C1.1.c} | Generated C | | F | Manage business groups | ... . | {C1.1.c} | | F | Manage business groups | .... | {P2} Accounts | {C1.1.d} {C1.1.i} | |
| {C1.1.d} | Generated C | | F | Define business group data | | {C1.1.d} | | F | Define business group data | | {P2} Accounts | {C1.1.c} {C1.1.i} | |
| {C1.1.i} | Generated C | | F | Manage business groups interface | | {C1.1.i} | | F | Manage business groups interface | | {P2} Accounts | {C1.1.c} {C1.1.d} | |
| {C1.1.i2} | | | F | Manage stakeholder's API | | {C1.1.i2} | | F | Manage stakeholder's API | | {P3} Business Management | {C1.1.d} | |
| {UC1.4.2} | Manage companies | cdi | | | | | | | | | | | |
| {C1.2.c} | Generated C | | F | Manage companies | | {C1.2.c} | | F | Manage companies | | {P2} Accounts | {C1.2.d} {C1.2.i} | |
| {C1.2.d} | Generated C | | F | Companies data | | {C1.2.d} | | F | Companies data | | {P2} Accounts | {C1.2.c} {C1.2.i} | |
| {C1.2.i} | Generated C | | F | Manage companies interface | | {C1.2.i} | | F | Manage companies interface | | {P2} Accounts | {C1.2.c} {C1.2.i} | |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| {UC1.3} | Manage factories | di | | | | | | | | | | | |
| {C1,3.c} | Generated C | | T | | | | | | | | | | |
| {C1,3.d} | Generated C | | F | Factories data | | {C1,3.d} | | F | Factories data | | {P2} Accounts | {C1,3.i} | |
| {C1.3.i} | Generated C | | F | Manage factories interface | | {C1,3.i} | | F | Manage factories interface | | {P2} Accounts | {C1,3.d | |
| {UC1.4} | Manage entities | di | | | | | | | | | | | |
| {C1.4.c} | Generated C | | T | | … | | … | | | …. | | | |
| {C1.4.d} | Generated C | | F | Entities data | … | {C1.4.d} | | F | Entities data | …. | {P2} Accounts | {C1.4.i} | |
| {C1.4.i} | Generated C | | F | Manage entities interface | | {C1.4.i} | | F | Manage entities interface | | {P2} Accounts | {C1.4.d} | |
| {UC1.5} | Manage tokens | di | | | | | | | | | | | |
| {C1.5.c} | Generated C | | T | | | | | | | | | | |
| {C1.5.d} | Generated C | | F | Tokens data | | {C1.5.d} | | F | Tokens data | | {P3} Business Management | {C1.5.i} | |
| {C1.5.i} | Generated C | | F | Manage tokens interface | | {C1.5.i} | | F | Manage tokens interface | | {P3} Business Management | {C1.5.d} | |
| {UC1.6} | Manage trucks | cdi | F | | | | | | | | | | |
| {C1.6.c} | Generated C | | F | Manage trucks | | {C1.6.c} | | F | Manage trucks | | {P4} Industrial maintenance | {C1.6.d} {C1.d.i} | |
| {C1.6.d} | Generated C | | F | Trucks data | | {C1.6.d} | | F | Trucks data | | {P4} Industrial maintenance | {C1.6.c} {C1.d.i} | |
| {C1.d.i} | Generated C | | F | Manage trucks interface | | {C1.d.i} | | F | Manage trucks interface | | {P4} Industrial maintenance | {C1.6.c} {C1.6.d} | |
| {UC2.1} | Manage local IT resources | di | | | | | | | | | | | |
| {C2.1.c} | Generated C | | T | | | | | | | | | | |
| {C2.1.d} | Generated C | | F | Intervention and maintenance data | | {C2.1.d} | {C2.2.d} {C2.3.c} | F | Intervention and maintenance data | | {P4} Industrial maintenance | {C2.1.i} {C2.2.i} {C2.3.i} | |
| {C2.1.i} | Generated C | | F | Verify intervention interface | | {C2.1.i} | | F | Verify intervention interface | | {P4} Industrial maintenance | {C2.1.d} | |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| {UC2.2} | Schedule interventions | di | | | | | | | | | | | |
| {C2.2.c} | Generated C | | T | | | | | | | | | | |
| {C2.2.d} | Generated C | | F | Store interventions data | | {C2.1.d} | | T | | | | | |
| {C2.2.i} | Generated C | | F | Store interventions interface | | {C2.2.i} | | F | Schedule interventions interface | | {P4} Industrial maintenance | {C2.1.d} | |
| {UC2.3} | Perform interventions | di | | | | | | | | | | | |
| {C2.3.c} | Generated C | | T | | | | | | | | | | |
| {C2.3.c} | Generated C | | F | Store interventions data | | {C2.1.d} | | T | | | | | |
| {C2.3.c} | Generated C | | F | Perform interventions interface | … | {C2.3.c} | | F | Perform interventions interface | … | {P3} Industrial maintenance | {C2.1.d} | |
| {UC2.4} | Provide users training | di | | | | | | | | | | | |
| {C2.4.c} | Generated C | | T | | | | | | | | | | |
| {C2.4.d} | Generated C | | F | Users training data | | {C2.4.d} | | F | Users training data | | {P3} Business Management | {C2.4.i} | |
| {C2.4.i} | Generated C | | F | Users training interface | | {C2.4.i} | | F | Users training interface | | {P3} Business Management | {C2.4.d} | |
| {UC2.5} | Generate templates | di | | | | | | | | | | | |
| {C2.5.c} | Generated C | | T | | | | | | | | | | |
| {C2.5.d} | Generated C | | F | Services template data | | {C2.5.d} | | F | Services template data | | {P3} Business Management | {C2.5.i} | |
| {C2.5.i} | Generated C | | F | Generate service templates interface | …. | {C2.5.i} | | F | Generate service templates interface | …. | {P3} Business Management | {C2.5.d} | |

4SRS – *Sprint* 2

| Step 1 - Component Creation | | | | | | Step 2 - Component Elimination | | | | | Step 3 - Packing & Aggregation | Step 4 - Component Association | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Use Case** | **Description** | **2i** | **2ii** | **2iii** | **2iv** | **2v** | | **2vi** | **2vii** | **2viii** | | **4i** | **4ii** |
| {UC1.7.1} | Manage profiles | cdi | | | | | | | | | | | |
| {C1.7.1.c} | Generated C | | F | Manage Profiles | …. | {C1.7.1.c} | | F | Profiles | | {P1} Accounts | {C1.7.1.d} {C1.7.1.i} | |
| {C1.7.1.d} | Generated C | | F | Profiles data | | {C1.7.1.d} | | F | Profiles data | | {P1} Accounts | {C1.7.1.c} {C1.7.1.i} | |
| {C1.7.1.i} | Generated C | | F | Manage profiles interface | ….. | {C1.7.1.i} | | F | Configure users' profiles interface | | {P1} Accounts | {C1.7.1.c} {C1.7.1.d} | |
| {C1.7.1.i2} | Generated C | | F | Authorization API | | {C1.7.1. i2 | | F | Authorization Service API | | {P1} Accounts | {C1.7.1.d} | |
| {UC1.7.2} | Assign permissions | i | | | | | | | | | | | |
| {C1.7.2.c} | Generated C | | T | | | | | | | | | | |
| {C1.7.2.d} | Generated C | | T | | | | | | | | | | |
| {C1.7.2.i} | Generated C | | F | Assign permissions interface | …. | {C1.7.1.i} | | T | | | | | |
| {UC1.8} | Manage trailers | cdi | | | | | | | | | | | |
| {C1.8.c} | Generated C | | F | Manage trailers | | {C1.8.c} | | F | Manage trailers | … | {P2} Business Management | {C1.8.d} {C1.8.i} | |
| {C1.8.d} | Generated C | | F | Trailers data | | {C1.8.d} | | F | Trailers data | | {P2} Business Management | {C1.8.c} {C1.8.i} | |
| {C1.8.i} | Generated C | | F | Manage trailers interface | | {C1.8.i} | | F | Manage trailers interface | …. | {P2} Business Management | {C1.8.c} {C1.8.c} | |
| {UC1.9.1} | Validate tokens | di | | | | | | | | | | | |
| {C1.9.1.c} | Generated C | | T | | | | | | | | | | |
| {C1.9.1.d} | Generated C | | F | Tokens data | | {C1.9.1.d} | | F | Tokens data | | {P2} Business Management | {C1.9.1.i} | {C1.1.i} |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| {C1.9.1.i} | Generated C | | F | Validate tokens interface | | {C1.9.1.i} | {C1.9.2.i} {C1.9.3.i} | F | Validate tokens interface | . | {P2} Business Management | {C1.9.1.d} | |
| {UC1.9.2} | Associate entity | i | | | | | | | | | | | |
| {C1.9.2.c} | Generated C | | T | | | | | | | | | | |
| {C1.9.2.d} | Generated C | | T | | | | | | | | | | |
| {C1.9.2.i} | Generated C | | F | Associate entity interface | | {C1.9.1.i} | | T | | | | | |
| {UC1.9.3} | Associate factory | i | F | | | | | | | | | | |
| {C1.9.3.c} | Generated C | | T | | | | | | | | | | |
| {C1.9.3.d} | Generated C | | T | | | | | | | | | | |
| {C1.9.3.i} | Generated C | | F | Associate factory interface | | {C1.9.1.i} | | T | | | | | |
| {UC1.9.4} | Request tokens | i | | | | | | | | | | | |
| {C1.9.4.c} | Generated C | | T | | | | | | | | | | |
| {C1.9.4.d} | Generated C | | T | | | | | | | | | | |
| {C1.9.4.i} | Generated C | | F | Request tokens interface | | {C1.9.4.i} | | T | Request tokens interface | …. | {P2} Business Management | | {C1.7.1.i2} {C1.1.i2} |
| {UC1.9.5} | Assign tokens | i | | | | | | | | | | | |
| {C1.9.5.c} | Generated C | | T | | | | | | | | | | |
| {C1.9.5.d} | Generated C | | T | | | | | | | | | | |
| {C1.9.5.i} | Generated C | | F | Assign tokens interface | | {C1.9.5.i} | | T | Assign tokens interface | | {P2} Business Management | | {C1.7.1.i2} {C1.1.i2} |

4SRS – *Sprint* 3

| Step 1 - Component Creation | | Step 2 - Component Elimination | | | | | | | | | Step 3 - Packing & Aggregation | Step 4 - Component Association | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Use Case | Description | 2i | 2ii | 2iii | 2iv | 2v | | 2vi | 2vii | 2viii | | 4i | 4ii |
| {UC1.2.1} | companies | di | | | | | | | | | | | |
| {C1.2.1.c} | Generated C | | T | | ... | | | | | | | | |
| {C1.2.1.d} | Generated C | | F | Clients companies data | | {C1.2.1.d | | F | Clients companies data | | {P1} Accounts | {C1.2.1.i} | |
| {C1.2.1.i} | Generated C | | F | Manage clients companies interface | .... | {C1.2.1.i} | {C1.2.2.i} | F | Manage clients companies interface. | | {P1} Accounts | {C1.2.1.d} | |
| {UC1.2.2} | Associate group | i | | | | | | | | | | | |
| {C1.2.2.c} | Generated C | | T | | | | | | | | | | |
| {C1.2.2.d} | Generated C | | T | | | | | | | | | | |
| {C1.2.2.i} | Generated C | | F | Associate groups interface | | {C1.2.1.i} | | T | | | | | |
| {UC1.5} | Manage work tokens | di | | | | | | | | | | | |
| {C1.5.c} | Generated C | | T | | | | | | | | | | |
| {C1.5.d} | Generated C | | F | Work tokens data | | {C1.5.d} | | | Work tokens data | | | {C1.5.i} | |
| {C1.5.i} | Generated C | | F | Mnage work tokens interface | | {C1.5.i} | | F | Manage work tokens interface | | {P2} Business Management | {C1.5.d} | |
| {UC1. 10.1.1} } | onfigure application | cdi | | | | | | | | | | | |
| {C1.10.1.c} | Generated C | | F | Configue applications | | {C1.10.1.c} | | F | Configue applications | | {P4} Authentication | {C1.10.1.d} {C1.10.1.i} | |
| {C1.10.1.d} | Generated C | | F | Applications data | dados das | {C1.10.1.d} | | F | Applications data | dados da | {P4} Authentication | {C1.10.1.c} {C1.10.1.i} | {C1.1.i2} |
| {C1.10.1.i} | Generated C | | F | Configue applications interface | | {C1.10.1.i} | | F | Configure applications interface | | {P4} Authentication | {C1.10.1.c} {C1.10.1.d} | |
| {UC1.10.2} | Perform authentica | i | | | | | | | | | | | |
| {C1.10.2.c} | Generated C | | T | | | | | | | | | | |
| {C1.10.2.d} | Generated C | | T | | | | | | | | | | |
| {C1.10.2.i} | Generated C | | F | Perform authentication interface. | | {C1.10.2.i} | | | Perform authentication interface. | | {P4} Authentication | | {C1.1.i2} |
| {UC1.10.1.3} | Assign a token | i | | | | | | | | | | | |
| {C1.10.3.c} | Generated C | | T | | | | | | | | | | |
| {C1.10.3.d} | Generated C | | T | | | | | | | | | | |
| {C1.10.3.i} | Generated C | | F | Assign a token interface | | {C1.10.1.i} | | T | | | | | |

4SRS – *Sprint* 4

| Step 1 - Component Creation | | Step 2 - Component Elimination | | | | | | | | | Step 3 - Packing & Aggregation | 4 - Component Association | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Use Case | Description | 2i | 2ii | 2iii | 2iv | 2v | | 2vi | 2vii | 2viii | | 4i | 4ii |
| {UC1.8} | Consult SLA | cdi | | | | | | | | | | | |
| {C1.8.c} | Generated C | | T | | | | | | | | | | |
| {C1.8.d} | Generated C | | T | | | | | | | | | | |
| {C1.8.i} | Generated C | | F | Consult users SLA interface | ..... | {C1.8.i} | | F | Consult users SLA interface | ... | {P2} Business Management<br>{P2.1} Business Platform Management | | |

4SRS – *Sprint 5*

| | | | | | Step 2 - Component Elimination | | | | | Step 3 - Packing & Aggregation | Step 4 - Component Association | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Description | 2i | 2 ii | 2iii | 2i v | 2v | | 2 vi | 2vii | 2v iii | | 4i | 4ii |
| Create user accounts | c di | | | | | | | | | | | |
| Generated C | | F | Create user account | … . | {C1.1.1.1.c} | | | Create user account | | {P1} Accounts | {C1.1.1.1.d}<br>{C1.1.1.1.i} | |
| Generated C | | F | User data | | {C1.1.1.1.d} | {C1.1.1 .3.d}<br>{C1.1.1 .2.d} | F | User data | | {P1} Accounts | {C1.1.1.1.c}<br>{C1.1.1.1.i} | |
| Generated C | | F | Create user account interface | | {C1.1.1.1.i} | | F | Create user account interface | | {P1} Accounts | {C1.1.1.1.c}<br>{C1.1.1.1.d} | {C1.7.1.i} |

| Generated C | | F | Authentication API | | {C1.1.1.1.i2} | | F | Authentication API | | {P2} Authentication | {C1.1.1.1.d} | {C1.1.1.3.i} {C1.10.2.2.c} {C1.10.2.i} |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Change user account | di | | | | | | | | | | | |
| Generated C | | T | | | | | | | | | | |
| Generated C | | F | Store user data | | {C1.1.1.1.d} | | T | | | | | |
| Generated C | | F | Edit user interface | | {C1.1.1.1.i} | | T | | | | | |
| Disable user account | di | | | | | | | | | | | |
| Generated C | | T | | | | | | | | | | |
| Generated C | | F | Store user data | | {C1.1.1.1.d} | | T | | | | | |
| Generated C | | F | Disable user interface | | {C1.1.1.1.i} | | T | | | | | |
| Consult user account | i | | | | | | | | | | | |
| Generated C | | T | | | | | | | | | | |
| Generated C | | T | | | | | | | | | | |
| Generated C | | F | Consult user interface | ... | {C1.1.1.1.i} | | T | | | | | |
| Manage forwarders companies | c di | | | | | | | | | | | |
| Generated C | | F | | | | | | | | | | |
| Generated C | | F | Forwarder companies data | | {C1.2.1.d} | {C1.2.2.d} | F | Forwarder companies data | | {P1} Accounts | {C1.2.1.i} | |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generated C | | F | Forwarder companies interface | | {C1.2.1.i} | {C1.2.2.i} | F | Forwarder companies interface | | {P1} Accounts | {C1.2.1.d} | |
| Manage suppler companies | di | | | | | | | | | | | |
| Generated C | | T | | | | | | | | | | |
| Generated C | | F | Suppler company's data | | {C1.2.1.d} | | T | | | | | |
| Generated C | | F | Suppler companies' interface | | {C1.2.1.i} | | T | | | | | |
| Authentication | i | | | | | | | | | | | |
| Generated C | | T | | | | | | | | | | |
| Generated C | | T | | | | | | | | | | |
| Generated C | | F | create user authentication interface | | {C1.10.2.1.i} | | F | create user authentication interface | | {P3} Authentication | | {C1.1.1.1.d} |
| Recover account | ci | | | | | | | | | | | |
| Generated C | | F | Recover account | | {C1.10.2.2.c} | | F | Recover account | | {P3} Authentication | {C1.10.2.2.i} | {C1.1.1.1.d} |
| Generated C | | T | | | | | | | | | | |
| Generated C | | F | Recover account interface | | {C1.10.2.2.i} | | F | Recover account interface | | {P2} Authentication | {C1.10.2.2.c} | |
| Define SLA | di | | | | | | | | | | | |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generated C | | T | | | | | | | | | | |
| Generated C | | F | SLA data | | {C3.1.d} | | F | SLA data | | {P4} Business Platform Management | {C3.1.i} | {C1.1.1.1.i2} {C1.7.2.i2} |
| Generated C | | F | Define SLA interface | … | {C3.1.i} | | F | Define SLA interface | …. | {P4} Business Platform Management | {C3.1.d} | |

## ANEXO IV – Transformação de *User Stories* em Casos de Uso

**Sprint 3**

***User Stories: -***
*As an application, I want to send an app_gid and a GPS location to perform authentication, in order to access the UH4SP WebAPIs.*
*As a System Administrator, I want to assign or refresh a token to an application.*
*As a System Administrator, I want to CRUD an application account, to configure application account.*
***EPIC: -***
***As a/an** application, system administrator, **I want to** assign a token, perform authentication and configure application, **to**…*
***Use Case**: - **{UC1.10} Manage applications***
*{UC1. 10.1.1} Configure application*
*{UC1. 10.1.2} Perform authentication*
*{UC1. 10.1.3} Assign a token*
***User Stories: -***
*As a System Administrator I want to validate work tokens that was requested by managers in order to manage work tokens.*
*As a Stakeholder/Entity manager I want to receive a notification when a given work tokens were associated to my entity in order to manage work tokens.*
***EPIC: - As a** system Administrator, Stakeholders / Entity manager, **I want to** manage work tokens.*
***Use Case: - {1.5} Manage work tokens***
*User Stories: -*
*As a System Administrator or a client admin, I want to CRUD a client companies in order to manage client companies, and last manage companies.*
*As a Corporate manager I want to request, read, update and disable work tokens in order to manage work tokens to my group companies and factories.*
*As a service, I want to get users permissions to access to a particular data source.*
*As a System administrator, Corporate manager, Company manager, and Factory manager, Forwarder, Client or Supplier, I want to visualize system operations about my groups, companies, factories or to resources or data that I am associated.*
*As a System administrator, Corporate manager, Company manager, and Factory manager, Forwarder, Client or Supplier, I want to visualize graphical indicators about my group, companies, factories or to resources or data that I am associated.*
*EPIC: - As Service, system Administrator, Corporate manager, Company manager, Factory manager, Forwarder, Client / Suppler, I want to resource data I am associated and a particular data source.*
***EPIC: -***
***As a** system Administrator, Corporate manager, I want to manage client companies, manage companies, group companies and factories, associate group*
***Use Case: - {UC1} Manage Stakeholders***
*{UC1.2} Manage companies*
*{UC1.2.1} Manage client's companies*

## ANEXO V – *Publicação Científica – Proposal of a Visual Environment to Support Scrum*

**Autores**: Fidel Kussunga, Pedro Ribeiro
**Editora**: Elsevier
Conferência: ProjMan 2019
**Estado**: Aceite

**Abstract**: Agile methodologies are increasingly considered important in the current context because of their focus on business benefit, strong stakeholder involvement and rapid incorporation of changing requirements. The Scrum methodology presents a generic and agile model for software development, but there are some gaps regarding solution architecture and requirements modeling and traceability. Fast customer feedback and support for volatile requirements result in higher product value, however, initial requirements modelling using modelling techniques are not commonly used in agile processes such as Scum to prepare the phase of Implementation of the software project.
This article presents a first proposal for a visual environment to support the product / sprint backlog, helping to prioritize, track and decide on the most valuable, aligned and quickly deployable requirements.

**Keywords**: *Visual environment; Software development; SCRUM; Software modeling; Agile methodologies*.

## ANEXO VI - *Publicação Científica* - Caracterização de um Ambiente Visual para Apoiar as Cerimónias do *Scrum*

**Autores**: Fidel Kussunga, Pedro Ribeiro, Nuno Santos
**Editora**: Scopus
Conferência: CAPSI 2019
**Estado**: Aceite
**Sumário:** Metodologias ágeis de desenvolvimento de *software*, como o *Scrum*, têm sido muito utilizados hoje em dia por oferecerem grandes benefícios para os seus utilizadores, como aceleração de processos e recursos para lidar com instabilidade de ambientes tecnológicos. A metodologia *Scrum* centra-se na gestão e organização do projeto de *software* e não fornece descrições de como tudo dever ser feito em um projeto. O feedback rápido do cliente e o suporte para requisitos voláteis resultam num valor de produto mais alto, no entanto, a modelação de requisitos iniciais usando técnicas de modelação, não são usados comumente em processos ágeis como *Scrum* para preparar a fase de implementação do projeto de *software*.
Este artigo pretende descrever de uma forma sumária, a abordagem proposta para apoiar o *Sprint e Product Backlog*. A linguagem de modelação *Unified Modeling Language* (UML) foi utilizada para apoiar na priorização de requisitos, melhorar a qualidade da solução e facilitar a manutenção.

**Palavras – chave:** Ambiente visual, desenvolvimento de *software*, metodologias ágeis, modelação de *software*, *SCRUM.*

## ANEXOS VII – Casos de Uso

A seguir são apresentadas as descrições dos diagramas de casos de uso do projeto UH4SP.

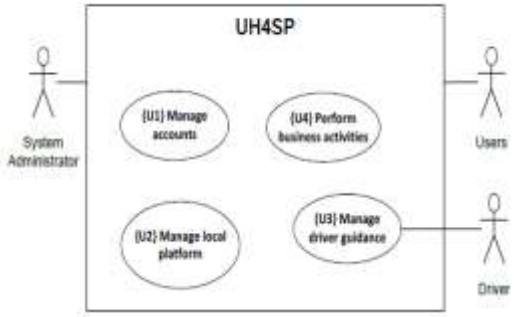



*Figura 45 - Diagrama de caso de uso do projeto UH4SP*

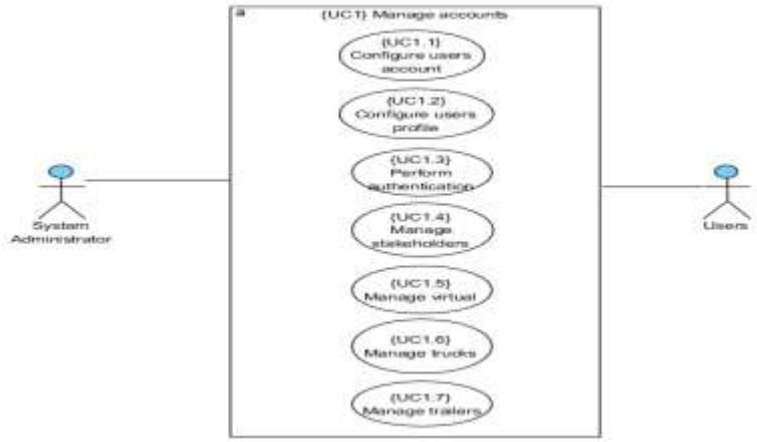



*Figura 46 - Diagrama de caso de uso {UC1} "Manage accounts"*

*Tabela 14 - Descrição textual de caso de uso {UC1} "Manage accounts"*

| | |
|---|---|
| **Referência** | {UC1} |
| **Nome** | *Manage accounts* |
| **Pequena descrição** | Este caso de uso descreve as atividades que permitem configurar contas dos utilizadores, configuração de perfis, a gestão de *stakeholders* bem como a gestão de camiões e trailers. |
| **Ator (es)** | *System Administrator, Users* |
| **Fluxo básico** | Este caso de uso inicia quando o ator pretende fazer a gestão de contas.<br>1.O ator faz a gestão de contas.<br>2.O resultado do passo 1 e guardado. |
| **Fluxo alternativo** | |
| **Pré-condições** | |
| **Pós-condições** | |

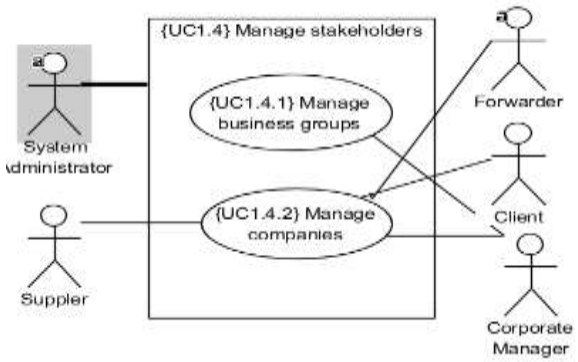



*Figura 47 - Diagrama de caso de uso {UC1.4} "Manage stakeholders"*

*Tabela 15 - Descrição textual do caso de uso {UC1.4} "Manage stakeholders"*

| **Referência** | {UC1.4} |
|---|---|
| **Nome** | *Manage stakeholders* |
| **Pequena descrição** | Este caso de uso descreve a atividade que consiste na gestão dos stakeholders. |
| **Ator (es)** | System Administrator, Suppler, Forwarder, Client e Corporate Manager |
| **Fluxo básico** | Este caso de uso inicia quando o ator pretende fazer a gestão dos stakeholders.<br>1.O ator faz a gestão de camiões. Criar, atualizar, desativar e consultar.<br>2.O resultado do passo 1 e guardado. |
| **Fluxo alternativo** | |
| **Pré-condições** | |
| **Pós-condições** | |

*Tabela 16 - Descrição textual de caso de uso {UC1.4.1} "Manage business groups"*

| **Referência** | {UC1.4.1} |
|---|---|
| **Nome** | *Manage business groups* |
| **Pequena descrição** | Este caso de uso permite que o administrador de sistemas ou gestor corporativo executem operações como criar, alterar, desativar e consultar, relativas a grupos industriais. |
| **Ator (es)** | Corporate Manager |
| **Fluxo básico** | Este caso de uso inicia quando o ator pretende fazer a gestão de grupos de negócio. |
| **Fluxo alternativo** | |
| **Pré-condições** | |
| **Pós-condições** | |

*Tabela 17 - Descrição textual de caso de uso {UC1.4.2} "Manage companies"*

| | |
|---|---|
| **Referência** | {UC1.4.1} |
| **Nome** | *Manage companies* |
| **Pequena descrição** | O caso de uso permite que o administrador, o fornecedor, o transportador,<br>O cliente ou o gestor corporativo executem operações como criar, alterar, desativar,<br>Consultar, relativas as companhias. |
| **Ator (es)** | System Administrator, Forwarder, Suppler, Client e Corporate Manager |
| **Fluxo básico** | Este caso de uso inicia quando o ator pretende fazer a gestão de companhias.<br>. |
| **Fluxo alternativo** | |
| **Pré-condições** | |
| **Pós-condições** | |

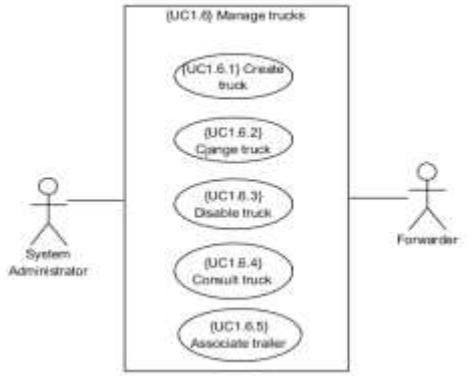



*Figura 48 - Descrição textual de caso de uso {UC1.6} "Manage trucks"*

*Tabela 18-Descrição textual de caso de uso {UC1.6} "Manage trucks"*

| | |
|---|---|
| **Referência** | {UC1.6} |
| **Nome** | *Manage trucks* |
| **Pequena descrição** | Este caso permite que o administrador de sistema e o transportador possam fazer a gestão de camiões. |
| **Ator (es)** | System Administrator, Forwarder |
| **Fluxo básico** | Este caso de uso inicia quando o ator pretende gerir camiões.<br>1.O ator faz a gestão de camiões. Criar, atualizar, desativar e consultar.<br>2.O resultado do passo 1 e guardado. |
| **Fluxo alternativo** | |
| **Pré-condições** | |
| **Pós-condições** | |

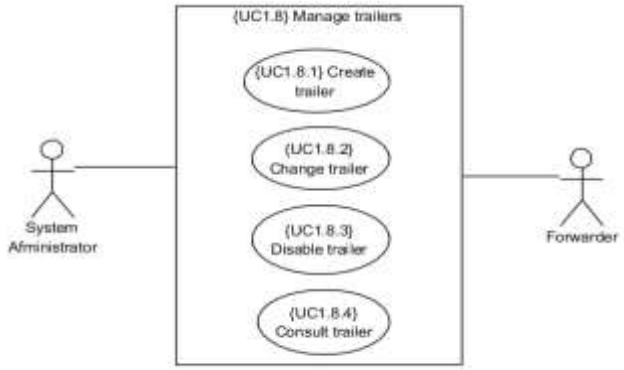



*Figura 49 - Diagrama de caso de uso {UC1.8} "Manage trailers"*

{UC1.7} “Manage trailers”: permite que o administrador de sistemas e o transportador possam fazer a gestão de reboques, com o objetivo de criar, atualizar, desativar ou consultar.

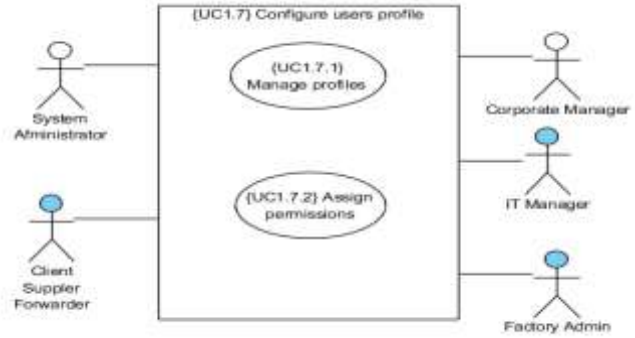



*Figura 50 - Diagrama de caso de uso {UC1.7} "Configure profile"*

{UC1.7.1} “*Manage profiles*”: este caso de uso define o tipo de perfil que terá acesso ao *backoffice* da plataforma. Enquanto o caso de uso {UC1.7.2} “*Assign permisssions*” permite aos atores contidos no diagrama possam atribuir permissões a um determinado perfil.

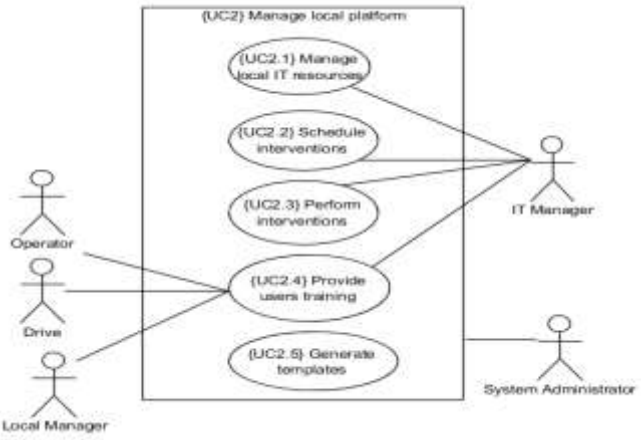



*Figura 51 - Diagrama de caso de uso {UC2} "Manage local platform"*

{UC2.1} "*Manage local IT resources*": este caso de uso permite que o administrador de sistemas e o gestor de infraestruturas verifiquem as necessidades de intervenção ou manutenção que possam ser marcadas.

O caso de uso {UC2.2} "*Schedule intervention*" permite que o gestor de infraestrutura e ao administrador de sistemas que programem assistência aos recursos de infraestruturas (assistência remota com dispositivos inteligentes).

{UC2.3} *"Perform intervention*": faz assistência remota a recursos de infraestruturas por meio de óculos inteligentes e outros dispositivos inteligentes. Essa assistência pode ser agendada em {UC2.2} "*Schedule intervention"*.

{UC2.4} "*Provide users training"* permite ao administrador de sistemas gerir a formação dos utilizadores, de modo a verificar as necessidades dos mesmos e agendar a formação do utilizador. Ao introduzir novos dispositivos inteligentes por exemplo, nos quais os utilizadores necessitam de formação para que seja manipulado corretamente. Também fornece tutoriais sobre recursos do sistema. o gestor de infraestruturas introduz as necessidades de formação e acesso a tutoriais e documentação de ajuda. O operador, motorista e o gestor local consultam os tutoriais e a documentação de suporte.

{UC2.5} "*Generate templates":* este caso de uso permite que o administrador de sistemas faça a gestão de modelos para implementações e testes rápidos em sistemas de informações locais que, têm um conjunto de serviços pré-configurados, perfis, *browsers*, para tornar rápido o processo em caso de aparecer um novo serviço local.

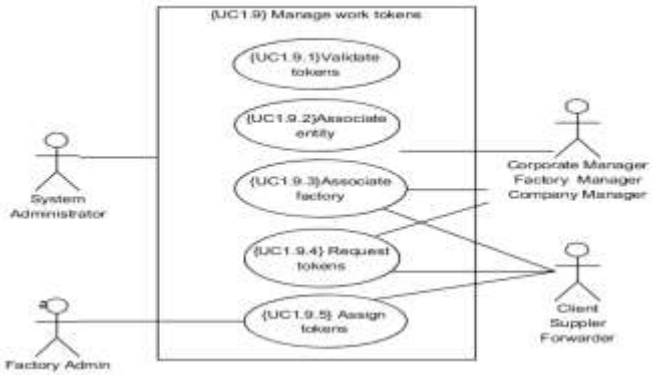



*Figura 52 -Diagrama de caso de uso {UC9} "Manage work tokens"*

{UC1.9.1} "*Validate tokens":* o caso de uso permite que o administrador do sistema valide solicitações de *tokens* de gestores corporativos.

Outro caso de uso é o {UC1.9.2} Associate *entity*": permite ao gestor corporativo associar uma entidade aos *tokens* solicitados.

O caso de uso {UC1.9.3} "*Associate factory",* permite ao gestor corporativo associar fábricas aos *tokens* solicitados. O gestor corporativo pode associar muitas fábricas do grupo e depois em seguida poder inserir diferentes quantidades de *tokens* em cada uma delas.

{UC1.9.4} "*Request tokens*": permite ao gestor corporativo solicitar *tokens* para companhias. Ainda pode solicitar diferentes quantidades de fichas para cada um.

O quinto caso de uso {UC1.9.5} "*Assign tokens*", permite aos x atribuir *tokens* aos seus motoristas.

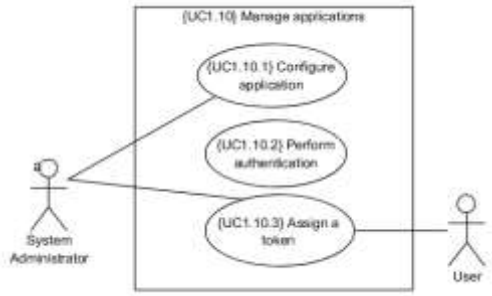



*Figura 53 - Diagrama de caso de uso {UC1.10} "Manage applications"*

{UC1.10.1} "*Configure application*": permite ao administrador de sistemas configura aplicações para criar, atualizar, consultar ou desativar aplicações.

{UC1.10.2} “Perform authentication”: permite que as aplicações executem a autenticação, com o objetivo de aceder aos WebAPIs do UH4SP.

{UC1.10.3} *“Assign a token*”: permite que o administrador de sistemas atribua ou atualize um token para uma aplicação.

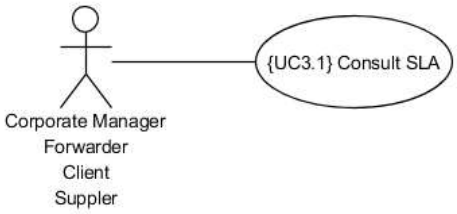



*Figura 54 - Diagrama de caso de uso {UC3.1} “Consult SLA”*

{UC1.8} “Define SLA”: este caso de uso inclui a definição de SLA entre o administrador de sistemas e um consumidor da *cloud*. O contrato pode ser consultado em {UC3.1} “*Consult SLA”.*

## ANEXO VIII – Arquitetura Global do Projeto UH4SP obtido pelo método tradicional