10-2019

# Characterization of a Visual Environment to Support SCRUM ceremonies

Fidel Kussunga

Pedro Ribeiro

Nuno Santos

# Caracterização de um Ambiente Visual para apoiar as cerimónias do SCRUM

*Characterization of a Visual Environment to Support SCRUM ceremonies*

Fidel Kussunga, Departamento de Eletrónica Industrial, Universidade do Minho, Portugal, fikinacio@gmail.com

Pedro Ribeiro, Departamento de Sistemas de Informação, Universidade do Minho, Portugal, pmgar@dsi.uminho.pt

Nuno Santos, Centro da Computação Gráfica, Portugal, nuno.santos@ccg.pt

**Resumo**

Metodologias ágeis de desenvolvimento de software, como o *Scrum*, têm sido muito utilizadas 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, mas não fornece descrições detalhadas 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* utilizando as técnicas de modelação baseadas em *Unified Modeling Language* (UML).

**Palavras-chave:** Ambiente visual; Desenvolvimento de software; SCRUM; Modelação de software; Metodologias ágeis.

***Abstract***

Agile software development methodologies, such as Scrum, have been widely used today because they offer great benefits to their users, such as accelerating processes and resources to deal with instability in technological environments. The Scrum methodology focuses on the management and organization of software projects but does not provide detailed descriptions of how everything should be done in a project. Rapid customer feedback and support for volatile requirements result in a higher product value, however, initial requirement analysis using modeling techniques is not commonly used in agile processes such as Scrum to prepare the implementation phase of the project.

This article is intended to briefly describe the approach proposed to support Sprint and Product Backlog using Unified Modeling Language (UML), based modeling techniques.

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

## 1.INTRODUÇÃO

Os sistemas de software são cada vez mais complexos devido aos requisitos ambíguos e diversificados dos clientes (Yaohong & Jingtao, 2014). Ao mesmo tempo, o mercado é muito competitivo. 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


*ISSN 2183-489X*

adequados para projetos pouco estáveis. 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 (Yaohong & Jingtao, 2014). O surgimento de métodos ágeis nos últimos anos, atraíram cada vez mais atenção ao serem utilizados com sucesso para melhorar os processos de projetos de software.

As metodologias ágeis provenientes do Manifesto Ágil (Beck et al., 2001) 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 específicos.

O Scrum é uma metodologia para gerir o desenvolvimento de sistemas de informação que coloca uma forte ênfase na interação entre os intervenientes do processo de desenvolvimento. Além de gerir projetos de desenvolvimento de software, a metodologia é usada por equipas de suporte de software. Apresenta-se como uma abordagem para gerir o desenvolvimento e a manutenção de software (Doronina & Doronina, 2018). A metodologia ágil *Scrum* é baseada na divisão de projetos em iterações (*Sprints*). Uma visão geral do *Scrum* é apresentada na Figura 1.


Figura 1 - Visão Geral do *Scrum* (Mekni, Buddhavarapu, Chinthapatla, & Gangula, 2018)

A linguagem de modelação UML propõe um conjunto de diagramas utilizáveis na análise e design de sistemas de informação e é uma linguagem padrão no mercado de desenvolvimento de software. Esta ferramenta de modelação apoia as boas práticas no nível técnico em cada fase do desenvolvimento de software. Além disso, a modelação UML é independente de modelos de processo.

A abordagem de transformação arquitetural, denominada *Four Step Rule Set* (4SRS), é utilizada com bastante sucesso propondo sucessivas transformações da arquitetura de software (Santos, Pereira, et al., 2018), a fim de satisfazer os requisitos do utilizador elicitados. A técnica 4SRS é essencialmente baseada no mapeamento de diagramas de Casos de Uso em diagramas de objetos. A natureza iterativa da técnica e o uso de modelos gráficos são questões importantes para garantir que a arquitetura final reflita os requisitos do utilizador e garanta uma boa qualidade da solução (Santos, Pereira, et al., 2018).

O objetivo deste artigo é a proposta de um ambiente visual para apoiar o Scrum, utilizando as técnicas de modelação UML. A abordagem mapeia as técnicas de modelação UML nas práticas do Scrum e fornece descrições detalhadas de como pode ser aplicada em um projeto real.

## 2. REVISÃO DA LITERATURA

Esta secção analisa brevemente algumas das principais metodologias de desenvolvimento de software ágeis e técnicas de modelação de software.

### *2.1 Métodos ágeis*

Os métodos ágeis de desenvolvimento de software foram projetados para resolver o problema de entregar software de alta qualidade a tempo, sob constante e rápida mudança de requisitos em ambientes de negócio. O principal benefício do desenvolvimento ágil de software é permitir um processo adaptativo – no qual a equipa de desenvolvimento reage e lida com mudanças nos requisitos e especificações, mesmo no final do processo de desenvolvimento.

Desenvolvimento adaptativo de software (ASD) (Cockburn & Highsmith, 2001) substitui o tradicional ciclo em cascata por uma série repetida de ciclos de especulação, colaboração e aprendizado. Este ciclo dinâmico proporciona aprendizagem contínua e adaptação ao estado emergente do projeto. As características de um ciclo de vida do ASD são: 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 suprir a emergência e a criatividade.

*Extreme programming* (XP) (Beck, 1999; Beck et al., 2001) é uma coleção de práticas de engenharia de software bem conhecidas. O XP visa permitir o desenvolvimento bem-sucedido de software, apesar dos requisitos de software vagos ou em constante mudança. *Extreme Programming Explained* (Vliet, 2007) descreve XP como uma disciplina de desenvolvimento de software que organiza as pessoas para produzir software de maior qualidade de forma mais produtiva. O XP tenta reduzir o custo das mudanças nos requisitos, tendo múltiplos ciclos curtos de desenvolvimento, em vez de longos.

O Scrum é uma abordagem empírica baseada na flexibilidade, adaptabilidade e produtividade (Ken Schwaber & Sutherland, 2017). O Scrum permite que os desenvolvedores escolham técnicas, métodos e práticas específicas de desenvolvimento de software para o processo de implementação. A prática central no Scrum é a realização de reuniões diárias de 15 minutos para coordenação e integração. O Scrum está em uso há muitos anos e tem sido usado para entregar com sucesso uma ampla gama de produtos.

### *2.2 Técnicas de Modelação de Software*

A modelação é uma componente fundamental das atividades que levam à concretização de sistemas de software bem estabelecidos (Manuel & Ribeiro, 2008). Segundo o Guia SWEBOK (Fairley, Bourque & Keppler, 2014), 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; e pré-condições, pós-condições e invariantes. A criação de modelos de um problema do mundo real é fundamental para análise de requisitos de software.

A UML apresenta muitos tipos de diagrama e, portanto, suporta a criação de diversos tipos diferentes de modelos do sistema (Sommerville, 2011): Diagramas de atividades (*activity diagrams*), que mostram as atividades envolvidas num processo ou no processamento de dados; Diagramas de casos de uso (*use case diagrams*), que mostram as interações entre um sistema e o seu ambiente; Diagramas de sequência (*sequence diagrams*), que mostram interações entre os atores, o sistema e os componentes do sistema; Diagramas de classes (*class diagrams*), que mostram as classes de objetos no sistema e as associações entre essas classes; Diagramas de objetos (*object diagrams*), que permitem representar um conjunto de objetos e mostrar um exemplo de interação entre eles.

## 3. VISÃO GERAL DA ABORDAGEM PROPOSTA

Nesta secção, é apresentada a abordagem proposta para apoiar as cerimónias do *Scrum*. A abordagem (Figura 2) inclui dois blocos inovadores, um para transformação de modelos, ou seja, *User Stories* em Casos de Uso (US->UC) e outro para a transformação de Casos de Uso para o modelo Arquitetural (aplicação do 4SRS).

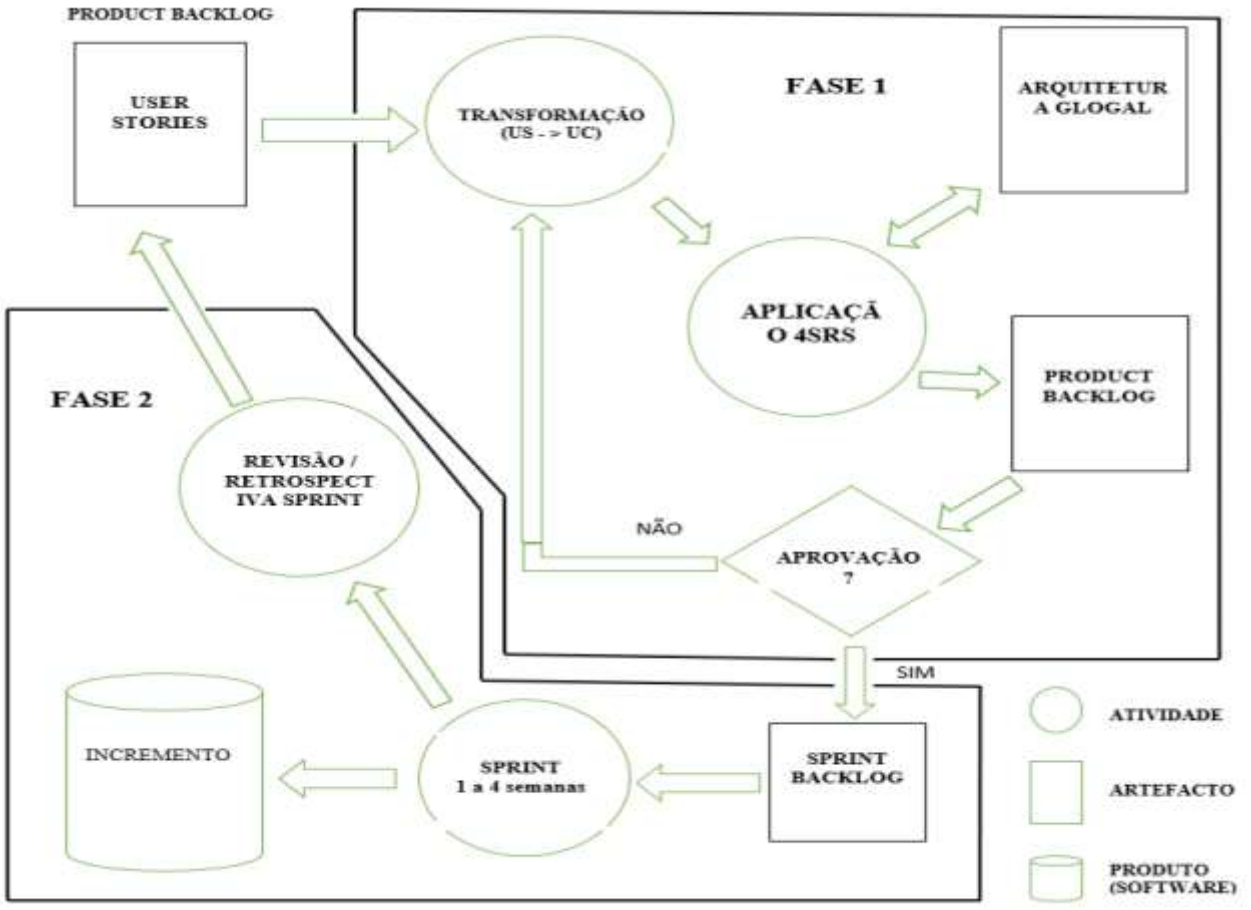


Figura 2 – Modelo global do ambiente visual

O ciclo de vida do ambiente visual é divido em duas fases principais, Planeamento do *Sprint* (Fase 1) e a Execução do *Sprint* (Fase 2). 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* (Ken Schwaber & Sutherland, 2017). Verifica-se que, a análise de requisitos de software é complexa, e é o segredo para o sucesso de qualquer desenvolvimento ágil de software (Yaohong & Jingtao, 2014). 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). A descrição de cada um desses blocos encontra-se na secção 4. Na Figura 2 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 (Yaohong & Jingtao, 2014).

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 (Yaohong & Jingtao, 2014), 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 (R. S. Madanayake, Dias & Kodikara, 2017).

## 4. DESCRIÇÃO DA ABORDAGEM PROPOSTA

Os detalhes da aplicação do ambiente visual para apoio às cerimónias do *Scrum*, incluindo atividades, artefactos e o modelo de implementação, serão discutidos com mais detalhe nesta secção.

### *4.1. Planeamento do Sprint (Fase 1)*

De acordo com o Guia do Scrum (Ken Schwaber & Sutherland, 2017), o trabalho a ser executado no *Sprint* é planeado no *Sprint Planning*. Este plano é criado envolvendo toda 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) Transformação (US->UC), (2) Aplicação do 4SRS, (3) Arquitetura Global 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 (Machado, & Gašević, 2012), mediante a aplicação de um conjunto de quatro passos.

O 4SRS recebe como entrada 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 saída, uma arquitetura lógica global (Ferreira et al., 2012).

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 entrada (Santos, Salgado, et al., 2018). 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" (I.Jacobson and Spence I., 2010). 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 a saída 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* é quem toma as decisões em relação a quais requisitos o produto terá. É ele que prioriza os requisitos e gere o *Product Backlog* (Ken Schwaber & Sutherland, 2017). 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 um recurso 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*.

### *4.2. 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 *story points* levará a equipa para concluir a tarefa. 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 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 (Yaohong & Jingtao, 2014). 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.

***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, no máximo 4 semanas) seja concluído. Quando todos os itens no *Sprint Backlog* são concluídos, uma nova iteração do sistema é entregue (Ken Schwaber & Sutherland, 2017). Para esta abordagem em específico, o *Sprint Backlog* é resultante do Planeamento do Sprint composto pelos blocos Transformação (US->UC), Aplicação do 4SRS, Arquitetura Global e Product Backlog.

**Revisão / Retrospetiva do Sprint**

No final de cada *Sprint* (Ken Schwaber & Sutherland, 2017), a equipa realiza a revisão e retrospetiva do *Sprint*, durante a qual as novas funcionalidades são demonstradas ao Product Owner ou a qualquer outra parte interessada que deseje fornecer feedback. Obter feedback que poderá influenciar o próximo Sprint é muito importante para maximizar os benefícios do produto. 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 também uma oportunidade para identificar qualquer aspeto para melhorar.

## 5. PROTÓTIPO DE SOFTWARE

Um diagrama contextual (corresponde ao bloco de transformação da abordagem proposta) da arquitetura do protótipo de software é apresentado na Figura 3, em conjunto com o processo manual de transformação de *User Stories* em *Epics* e o processo existente (Eclipse e PlantUML). O modelo do protótipo foi inspirado no trabalho de Madanayake et al., (2017).

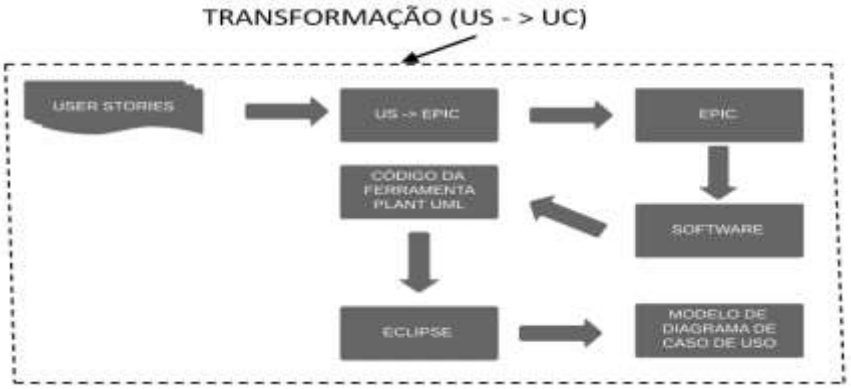


Figura 3 - Diagrama contextual da arquitetura do protótipo de software

### *5.1. EPIC*

Um *Epic* é uma *User Story* de alto nível, 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 em Madanayake, Dias, & Kodikara, (2016). Os *Epics* são obtidos a partir de um conjunto de *User Stories* num processo ainda não automático. Não são abordadas as *User Stories* nesta secção visto que as mesmas já foram discutidas anteriormente.

### *5.2. Código da ferramenta Plant UML*

Este bloco apresenta o código da ferramenta PlantUML gerado por intermédio de software desenvolvido. Este código da PlantUML implementado num ficheiro sem título no ambiente eclipse tem como saída o diagrama que representa o modelo de Caso de Uso desejado.

PlantUML é uma ferramenta que permite aos utilizadores gerar diagramas UML a partir de uma linguagem de texto simples (Plant UML in a Nutshell, http://plantuml.com/). PlantUML em si, é um software de código aberto, e há plug-ins de UML de fábrica para várias plataformas comuns, como Eclipse, NetBeans, Microsoft Word, LaTex, etc. (Plant UML in a Nutshell, http://plantuml.com/).

### *5.3. Eclipse*

O Eclipse é um ambiente de desenvolvimento para programas Java, porém é possível utilizar o Eclipse para desenvolvimento em outras linguagens de programação como C / C++ a partir de instalação de plug-ins. No Eclipse é possível criar um projeto, realizar a codificação e também verificar logs de execução.

### *5.4. Modelo de Diagrama de Caso de Uso*

Modelo de Diagrama de Caso de Uso é o resultado de transformação de *Epics* em código da ferramenta PlantUML. Este modelo, é o resultado que se pretende atingir com o desenvolvimento deste protótipo de

software. A Figura 4 apresenta o diagrama de Caso de Uso gerado automaticamente pelo protótipo assim como o respetivo *Epic* como forma da validação do conceito do processo de transformação de *User Stories* em Casos de Uso.

Descrição do Caso de Uso: "As an Operator I want to register customers, so customers can be counted".


Figura 4 - Modelo de caso de uso gerado pelo protótipo e respetivo *Epic*

## 6. ARQUITETURA LÓGICA

A Figura 5 apresenta o diagrama de objetos do *sprint* 1 do projeto Unified Hub for Smart Plants-UH4SP (projeto que serviu de caso de estudo), obtido a partir da aplicação do método 4SRS de acordo com abordagem proposta.

O modelo de objetos obtido para o *sprint* 1 representa um resultado parcial da validação da abordagem proposta e devido a limitação de espaço e no sentido de cumprir os requisitos exigidos para a publicação deste trabalho de investigação, não são apresentados os passos de como foi obtido o diagrama lógico, ficando esta tarefa para uma futura publicação completa.

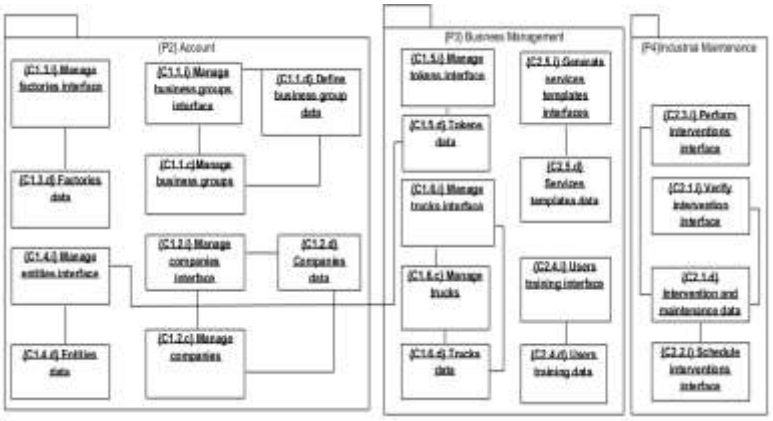


Figura 5 - Arquitetura lógica do *sprint* 1 com aplicação da abordagem proposta

## 7. CONCLUSÃO

Foi apresentado neste artigo o primeiro esboço de um ambiente visual para apoiar as decisões sobre o *Product Backlog* na metodologia ágil Scrum, com utilização de técnicas de modelação UML. Foi utilizado o método 4SRS que transforma modelos de requisitos funcionais do utilizador na forma de Casos de Uso para requisitos funcionais do sistema na forma de diagramas de componentes (arquitetura lógica de software). Por fim, *Epic* e o seu modelo de Caso de Uso gerado pelo protótipo de software foi sucintamente apresentado. Este é um trabalho em desenvolvimento, esperando-se uma automatização de todo o processo.

**Agradecimentos:** Este trabalho foi financiado por fundos nacionais através da FCT - Fundação para a Ciência e Tecnologia no âmbito do Projeto UID/CEC/00319/2019.

## REFERÊNCIAS

Beck, K. (1999). Change with Extreme Programming. *Ieee*, (c), 70–77. https://doi.org/10.1109/2.796139

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). *Manifesto for Agile Software Development Twelve Principles of Agile Software*. Retrieved from http://www.agilemanifesto.org

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *Computer*, *34*(11), 131–133. https://doi.org/10.1109/2.963450

Doronina, J., & Doronina, E. (2018). MODELS OF IT-PROJECT MANAGEMENT. *International Journal of Computer Science & Information Technology (IJCSIT)*, *10*(5). https://doi.org/10.5121/ijcsit.2018.10506

Fairley, R. E. D., Bourque, P., & Keppler, J. (2014). The impact of SWEBOK Version 3 on software engineering education and training. *2014 IEEE 27th Conference on Software Engineering Education and Training, CSEE and T 2014 - Proceedings*, 192–200. https://doi.org/10.1109/CSEET.2014.6816804

Ferreira, N., Santos, N., MacHado, R. J., & Gašević, D. (2012). 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, pp. 44–58). Springer-Verlag. https://doi.org/10.1007/978-3-642-31063-8_5

I.Jacobson and Spence I., 2010. (2010). Using Use Cases on Agile Projects - PDF. Retrieved March 24, 2019, from https://docplayer.net/21347812-Using-use-cases-on-agile-projects.html

Madanayake, R., Dias, G. K. A., & Kodikara, N. D. (2016). Use Stories vs UML Use Cases in Modular Transformation. *International Journal of Scientific Engineering and Applied Science (IJSEAS)*, (3). Retrieved from www.ijseas.com

Madanayake, R. S., Dias, G. K. A., & Kodikara, N. D. (2017). Transforming Simplified Requirement in to a UML Use Case Diagram Using an Open Source Tool. *International Journal of Computer Science and Software Engineering*, *6*(3), 2409–4285. Retrieved from www.IJCSSE.org

Manuel, A., & Ribeiro, N. (2008). *Um Processo de Modelação de Sistemas Software com Integração de Especificações Rigorosas, Universidade do Minho Escola de Engenharia*. Retrieved from https://core.ac.uk/download/pdf/55608897.pdf

Mekni, M., Buddhavarapu, G., Chinthapatla, S., & Gangula, M. (2018). Software Architectural Design in Agile Environments. *Journal of Computer and Communications*, *06*(01), 171–189. https://doi.org/10.4236/jcc.2018.61018

Plant UML in a Nutshell, http://plantuml. com. (n.d.). Commons:Wiki Loves Love 2019 - Wikimedia Commons. Retrieved February 7, 2019, from https://commons.wikimedia.org/wiki/Commons:Wiki_Loves_Love_2019

Santos, N., Pereira, J., Morais, F., Barros, J., Ferreira, N., & Machado, R. J. (2018). An agile modeling oriented process for logical architecture design. In *Lecture Notes in Business Information Processing* (Vol. 318, pp. 260–275). https://doi.org/10.1007/978-3-319-91704-7_17

Santos, N., Salgado, C. E., Morais, F., Melo, M., Silva, S., Martins, R., … Pereira, M. (2018). Decomposing monolithic to microservices architectures: a modeling approach, (August). https://doi.org/10.1145/3234152.3234180

Schwaber, K, & Sutherland, J. (2011). The Ccrum guide, *2*(July), 17. https://doi.org/10.1053/j.jrn.2009.08.012

Schwaber, Ken, & Sutherland, J. (2017). 2017 Scrum Guide, *19*(6), 504. https://doi.org/10.1053/j.jrn.2009.08.012

Sommerville, I. (2011). *SOFTWARE ENGINEERING Ninth Edition*. Retrieved from https://edisciplinas.usp.br/pluginfile.php/2150022/mod_resource/content/1/1429431793.203Software Engineering by Somerville.pdf

Vliet, H. van. (2007). Software Engineering: Principles and Practice. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.128.2614&rank=5

Wautelet, Y., Heng, S., Kolp, M., & Mirbel, I. (2014). Unifying and extending user story models. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 8484 LNCS, pp. 211–225). Springer, Cham. https://doi.org/10.1007/978-3-319-07881-6_15

Yaohong, X., & Jingtao, F. (2014). Research on Software Development Process Conjunction of Scrum and UML Modeling. https://doi.org/10.1109/IMCCC.2014.206