Universidade do Minho
Escola de Engenharia
Departamento de Sistemas de Informação

José Pedro Macedo Martins

Proposta de um Processo de Engenharia de Requisitos no Âmbito da Gestão do Ciclo de Vida de Sistemas de Informação na Bosch Car Multimédia Portugal, S.A.

Dissertação de Mestrado
Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação do(s)
Professor Doutor Ricardo J. Machado
Engenheiro José Carlos Martins

Guimarães, Janeiro de 2017

DECLARAÇÂO

Nome: José Pedro Macedo Martins

Endereço eletrónico: pedromartins21.03@gmail.com Telefone: 918528428

Bilhete de Identidade/Cartão de Cidadão: 13944606

Título de dissertação: Proposta de um Processo de Engenharia de Requisitos no Âmbito da Gestão do Ciclo de Vida de Sistemas de Informação na Bosch Car Multimédia Portugal, S. A.

Orientador/a/es:

Professor Doutor Ricardo Jorge Silvério de Magalhães Machado

Engenheiro José Carlos Costa Martins

Ano de conclusão: 2017

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Universidade do Minho, __/__/_____

Assinatura:_____

AGRADECIMENTOS

A realização desta dissertação era impossível sem o apoio emocional e contributo de determinadas pessoas. Portanto, quero agradecer a todos aqueles, que de alguma forma foram importantes na perpetração deste trabalho, nomeadamente:

Ao meu orientador científico, Professor Doutor Ricardo J. Machado, pela sua orientação, pela sua disponibilidade, pelas suas sugestões e pelas suas críticas construtivas.

Ao meu co-orientador e responsável na Bosch Car Multimédia Portugal, S.A., Engenheiro José Carlos Martins, pela sua orientação, pelo seu apoio no ambiente organizacional, pelas suas sugestões, pelas suas críticas construtivas e pela oportunidade de desenvolver este trabalho.

Ao Rui Abreu, por se disponibilizar a experimentar o processo concebido nesta dissertação de mestrado, por aceitar sempre as minhas sugestões e críticas construtivas com bastante ânimo e tolerância, e pelas suas sugestões.

Ao Professor Doutor Francisco Duarte, por possibilitar a experimentação do processo no projeto "ALR" e pelas suas críticas construtivas.

À minha namorada, Catarina Lima, pelo incentivo e apoio numa fase crítica do desenvolvimento da dissertação, e por se ter disponibilizado para ler o documento, com a finalidade de detetar lapsos literários e melhorar a escrita deste.

Aos meus pais, António Martins e Aldina Macedo, pelo incentivo, pelo apoio incondicional e por perceberem a importância deste trabalho.

Aos meus amigos mais chegados, pelos momentos de Epicurismo.

À equipa do departamento CI/CWR1 da Bosch Car Multimédia Portugal, S.A., onde fiz bons amigos, pelo bom ambiente proporcionado para o desenvolvimento deste trabalho.

À equipa do departamento QMM7 da Bosch Car Multimédia Portugal, S.A., por me ter recebido sempre bem e pelo bom ambiente na experimentação relativa ao projeto "GEIME".

À Bosch Car Multimédia Portugal, S.A. por me facultar a oportunidade e as condições para a realização deste trabalho.

(Página intencionalmente deixada em branco)

RESUMO

No departamento de CI-CWR1, da organização Bosch Car Multimédia Portugal, S.A., deteta-se a dificuldade em preservar requisitos de projetos para projetos. Para mitigar isso, é necessário um mecanismo que introduza no ambiente aplicacional a atividade de gestão de requisitos, com a funcionalidade de gerir o ciclo de vida dos requisitos em ciclos de vida de sistemas de informação coexistentes. A gestão de requisitos é uma atividade do processo de engenharia de requisitos que tem a particularidade de auxiliar as demais atividades e incorpora funcionalidades como, rastreabilidade, gestão de mudança, e rastreio de versões e estados de requisitos.

Tendo em vista a resolução do problema apresentado nesta dissertação de mestrado, concebe-se um processo para a engenharia de requisitos otimizado de forma a possuir uma gestão de requisitos eficaz. Um processo apresenta normalmente três constituintes: atividades, papeis e artefactos. As atividades do processo concebido dividem-se por duas componentes. Na componente de *development* incluem-se sete atividades para se obterem os requisitos de um sistema de informação. A componente de *management* é constituída por quatro atividades, onde se faz uma gestão dos requisitos inerentes a um sistema de informação, e a gestão do ciclo vida dos requisitos associados ao domínio aplicacional. Em adição identificam-se oito papéis para os atores realizarem as atividades do processo, com recurso a oito artefactos projetados pelo autor ou aproveitados de outras metodologias.

Na experimentação do processo são sugeridos dois projetos com características adaptáveis às funcionalidades deste. No projeto "GEIME" faz-se a análise de requisitos, da reabilitação de um sistema de informação para a gestão de calibração de dispositivos, através de duas iterações utilizando a *reverse engineering* para se levantarem requisitos da versão do sistema obsoleta. No projeto "ALR" faz-se a análise de requisitos de um sistema de informação para a avaliação de lotes do processo produtivo Bosch.

Palavras-chave: Requisito, Engenharia de Requisitos, Gestão de Requisitos, Ciclo de vida, Sistema de Informação, Projeto.

(Página intencionalmente deixada em branco)

ABSTRACT

In the department of CI-CWR1, of Bosch Car Multimédia Portugal, S.A., the difficulty in preserving requirements between projects can be detected. In order to mitigate this, a mechanism is necessary to introduce in the applicational environment, the management of requirements activity to manage the life cycle of requirements, in life cycles of coexisting information systems. The management of requirements is an activity in the process of requirements engineering which has the special feature of helping the other activities and incorporates features such as traceability, change management and tracking of requirements versions and states.

Owing to solve the problem above, in this Master's thesis, a process for requirements engineering is conceived to optimize an effective management of requirements. A process normally presents three constituents, that is, activities, roles and artifacts. The activities in the conceived process are divided by two components. In the development component, seven activities are included in order to achieve the requirements of an information system. The management component comprises four activities, where the management of the requirements is inherent to an information system and the management of the life cycle of the requirements associated to the application domain are made. In addition, eight roles are identified for the agents to do the activities of the process, using eight artifacts designed by the author or used from other methodologies.

In the experiment process, two projects are suggested with characteristics adaptable to its functions. In the 'GEIME' project, a requirements analysis of rehabilitation of an information system is made, for the management of the devices calibration, through two iterations using reverse engineering to raise requirements from the obsolete version of the system. In the 'ALR' project, the requirements analysis of an information system for the evaluation of the batches in Bosch's productive process is made.

Keywords: Requirement, Requirements Engineering, Requirements Management, LifeCycle, Information System, Project.

(Página intencionalmente deixada em branco)

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

(Página intencionalmente deixada em branco)

LISTA DE ACRÓNIMOS

AHP - Analytical Hierarchy Process

DSR – Design Science Research

IEEE – Institute of Electrical and Electronics Engineers

IEC – International Electrotechnical Commission

IIBA – International Institute of Business Analysis

ISO – International Organization for Standardization

RDEM - Requirement Driven Evolution Model

REPEAT - Requirements Engineering Process At Telelogic

SLO – Software Lifecycle Object

NIH – Not Invented Here

NAH – Not Applicable Here

GEIME – Gestão de Equipamento, Inspeção, Medição e Ensaio

ALR – Active Lot Release

UML – Unified Modeling Language

BPMN – Business Process Modeling Language

IBM – International Business Machines

(Página intencionalmente deixada em branco)

1 INTRODUÇÃO

O primeiro capítulo desta dissertação de mestrado tem o propósito de dar a conhecer o tema de estudo e os objetivos que dele advêm, assim como apresentar estratégias relevantes.

Primeiramente, expõe-se a motivação para a realização desta dissertação de mestrado e enquadra-se o leitor na temática desta. De seguida são identificados os objetivos e consequentes resultados esperados resultantes deste trabalho. De entre diversas metodologias de investigação, uma é selecionada, sendo esta escolha devidamente justificada. Posteriormente a metodologia escolhida é detalhada e as estratégias para a realização da revisão de literatura são expressas. De modo a concluir o capítulo, a estrutura desta dissertação de mestrado é descrita.

1.1 Motivação e Enquadramento

É irrefutável que ultimamente os sistemas de informação têm vindo a desempenhar um papel importante no alavancamento das organizações. Isto deve-se ao facto de surgirem necessidades no seio de empresas, em atividades que só podem ser desempenhadas com o auxílio das tecnologias de informação.

A complexidade e o tamanho dos sistemas de informação tendem a aumentar ano após ano. Com o aumento do tamanho, e da complexidade dos sistemas é natural surgirem problemas que não podem existir em sistemas pequenos (Jalote, 2005). Schneider (2005) afirma que 54 % dos erros detetados em projetos de desenvolvimento de software são detetados após a fase de desenvolvimento e 45 % destes erros são originados durante o processo da engenharia de requisitos.

Outra realidade que deve estar intrínseca a esta dissertação de mestrado é que, quanto mais tarde estes erros são detetados, mais custosos são de resolver. Em média, resolver um erro na fase dos requisitos custa 2 pessoas-hora, resolver um erro na fase da conceção custa 5 pessoas-hora, se um erro for identificado na fase de implementação a sua resolução custa 15 pessoas-hora, na fase de teste custa 50 pessoas-hora e na fase de operação/manutenção custa 150 pessoas-hora (Jalote, 2005). Perante estes valores, não há dúvidas que se deve olhar para o processo da engenharia de requisitos com seriedade. Uma boa especificação de requisitos é o reflexo de um bom sistema (Jalote, 2005).

É universal que nada é eterno, tudo tem um estado inicial, operacional e final. O grande desafio das organizações é manter o ciclo de vida das entidades que lhes são benéficas e terminar o ciclo de vida daquelas que são lhes são prejudiciais (Blythe, 2005). A gestão, num modo geral tem o propósito de garantir a assertividade de algo, analisando o estado atual e estabelecendo limites para uma evolução sustentada (F. W. . Taylor, 1995).

Um problema que surge nos projetos de desenvolvimento de software é os requisitos morrerem com o encerramento deste. Frequentemente surgem outros projetos com problemas já solucionados em contextos diferentes pela a equipa de desenvolvimento. As questões que se colocam aqui são: Não é produtivo aproveitar certas características que são da natureza de todos os domínios? Não é contra-produtivo levantar constantemente os mesmo requisitos?

Este é um problema que se tem vindo a evidenciar na secção CI-CWR12, do departamento de CI-CWR1[1] na Bosch Car Multimédia Portugal S.A. A missão da secção de CI-CWR12 na Bosch Car Multimédia Portugal S.A. é o desenvolvimento de sistemas de informação de apoio à organização.

Com base no conhecimento do estado de arte (ver capítulo 2), já há a possibilidade de garantir a manutenção dos requisitos ao longo de um projeto, através da gestão de requisitos. Se esta gestão de requisitos for alargada para além das fronteiras do projeto inerente, é possível solucionar o problema desta dissertação de mestrado.

1.2 Objetivos da Dissertação e Resultados Esperados

Nesta secção apresentam-se os objetivos e os resultados que se pretende atingir com a realização deste trabalho.

O problema associado à realização desta dissertação de mestrado prende-se com o facto de os requisitos especificados no âmbito de projetos de desenvolvimento de sistemas de informação, no departamento de CI-CWR1, da Bosch Car Multimédia Portugal S.A., desaparecerem com implementação da solução. A abordagem atualmente adotada pelo departamento referido é contra-produtiva, na medida em que, muitos requisitos são levantados repetidamente e, consequentemente, fazem-se esforços que podem ser poupados caso a abordagem seja mais orientada aos requisitos.

Com isto, esta dissertação de mestrado pretende responder à seguinte questão de investigação: "**De que forma se pode gerir o ciclo de vida de requisitos, de sistemas de informação existentes na Bosch Car Multimédia Portugal, S.A.?**".

O presente trabalho tem um carácter teórico e prático. Sobre o ponto de vista teórico é realizada uma revisão de literatura e é aproveitada a informação que dela advém. Do ponto de vista prático, pretende-se estudar o domínio aplicacional para moldar o conhecimento adquirido à organização.

Assim, de modo a responder à questão formulada são identificados e descritos os objetivos para este trabalho:

1 CI-CWR1 é o departamento de informática da Bosch Car Multimédia Portugal S.A, Braga, Portugal

1. Criar um processo de engenharia de requisitos para a gestão do ciclo de vida de sistemas de informação da organização Bosch: Este processo deve estar preparado para garantir que os requisitos não são esquecidos com a conclusão de um projeto de desenvolvimento de sistemas de informação. Basicamente, quando surge a necessidade de criar uma nova solução, o processo iniciará a sua execução até à extinção do domínio aplicacional. Alguns requisitos poderão surgir fora de fases de desenvolvimento, este processo deve estar preparado para garantir que estes não se perdem. Este processo deve ser desenvolvido iterativamente, e espera-se que haja uma evolução entre iterações aproveitando informação da iteração prévia e da revisão de literatura. Este deve ser desenvolvido com o propósito de ser implementado no departamento CI-CWR1, da Bosch Car Multimédia Portugal S.A., logo deve estar alinhado com os processos já existentes.
2. Sugerir ou criar artefactos de suporte ao processo criado: Estes artefactos devem ser documentos (*templates*) desenvolvidos pelo departamento de qualidade da Bosch Car Multimédia Portugal S.A. Caso não sejam encontrados artefactos que suportem uma determinada atividade do processo a desenvolver, deve ser feita uma pesquisa de forma a encontrar artefactos em metodologias existentes. Caso seja necessário criar artefactos, estes devem seguir as boas práticas de modelos de referência.
3. Experimentar o processo criado no departamento CI-CWR1 da Bosch Car Multimédia Portugal S.A.: Após o processo estar criado, é necessário que este seja experimentado no departamento de CI-CWR1 da Bosch Car Multimédia Portugal S.A. Por fim é necessário medir o impacto que a experimentação do processo teve.

Atingidos todos os objetivos, espera-se que o resultado desta dissertação de mestrado seja: Um processo de engenharia de requisitos, experimentado no departamento de CI-CWR1 da Bosch Car Multimédia Portugal S.A., para a gestão do ciclo de vida de sistemas de informação na organização Bosch.

1.3 Abordagem Metodológica

Neste capítulo é selecionado e explanado um método de pesquisa para esta dissertação e de seguida, é descrita a estratégia a utilizar na revisão de literatura.

Em projetos de investigação é essencial utilizar um método de investigação adequado às necessidades, uma vez que, este contribui consideravelmente para alcançar os objetivos estabelecidos.

> "O uso de um método sistemático é a alma da investigação." (Berndtsson, Hansson, Olsson, & Lundell, 2008)

1 Introdução

Um método de pesquisa inclui (1) recolha dados, (2) formulação de uma hipótese ou proposição, (3) teste de uma hipótese, (4) interpretação de resultados, e (5) indicação das conclusões que podem ser posteriormente avaliadas de forma independente por outros (Berndtsson *et al.*, 2008).

Dos métodos de abordagem de investigação existentes identificam-se os seguintes (Neiva, 2013):

- *Exploratory Research*;
- *Design Science Research* (DSR);
- *Empirical Research*;
- *Quantitative Positivism Research*:
  - *Fields Experiments*;
  - *Laboratory Expirements*;
  - *Surveys*.
- *Qualitative Research*:
  - *Action Research*;
  - *Case Study Research*;
  - *Ethnography*;
  - *Grounded Theory*.

Olhando para os métodos de investigação apresentados no parágrafo anterior, existem dois métodos que possuem características compatíveis com as necessidades desta dissertação de mestrado.

O primeiro é o *Design Science Research*, por ser um processo iterativo que dá indicações ao investigador para identificar um problema, e definir os objetivos para a solução do mesmo numa primeira fase, e conceber um artefacto com base no conhecimento científico de uma revisão de literatura, para resolver o problema identificado numa segunda fase. Posteriormente, vai-se afinando o artefacto avaliando o desempenho deste num contexto adaptável de resolução do problema (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007).

O segundo é o Action Research, que permite a um investigador observar o contexto onde se verifica o problema, e de seguida procurar na bibliografia métodos que vão resolvendo o problema iterativamente. Um exemplo vulgar do método de investigação em ação é um detetive a tentar resolver um mistério de assassinato (Ogland, 2009).

Apesar de os dois métodos referidos a cima serem muito parecidos (Ogland, 2009), chega-se a conclusão que o método de investigação mais adequado para este projeto é o *Design Science Research*

(DSR), porque o problema está associado à ausência de um mecanismo no domínio aplicacional para a gestão do ciclo de vida de requisitos. Consequentemente, a melhor abordagem é criar um artefacto de raiz e ir melhorando-o, avaliando a sua eficácia e eficiência em experimentações no departamento de CI-CWR1.

1.3.1 *Design Science Research*

Peffers *et al* (2007) apresentam um modelo de processo para o desenvolvimento de investigação recorrendo ao método DSR. Este modelo é composto por seis atividades (ver Ilustração 1): (1) identificação do problema e motivação; (2) definição dos objetivos para a solução; (3) conceção e desenvolvimento; (4) demonstração; (5) avaliação e (6) comunicação (Neiva, 2013).

1. **Identificação do problema e motivação**: Nesta fase, define-se o problema específico de investigação e justifica-se o valor da solução. A definição do problema será usada para o desenvolvimento de um ou mais artefactos que poderão fornecer uma solução efetivamente. A justificação do valor da solução e da sua relevância para a questão requer um bom conhecimento do estado de arte na área do problema e da importância da sua solução (Neiva, 2013; Peffers *et al.*, 2007).
2. **Definição dos objetivos para a solução**: Após a identificação do problema e do estado de arte na área do problema, devem-se explicar os objetivos para a solução. Os objetivos podem ser quantitativos, tais como os termos em que a solução desejável seria melhor do que as atuais, ou qualitativos, como a descrição de como um novo artefacto deverá suportar soluções para problemas até então não abordados. Os objetivos devem ser inferidos racionalmente a partir do trabalho realizado na fase anterior, do conhecimento do estado da arte na área do problema e das soluções atuais, bem como da sua eficiência para a resolução do problema (Neiva, 2013; Peffers *et al.*, 2007).
3. **Conceção e desenvolvimento**: Nesta fase, concebe-se o artefacto. Concetualmente, um artefacto pode ser qualquer objeto concebido no qual existiu alguma contribuição da investigação para o seu desenvolvimento. Esta atividade inclui a definição das funcionalidades desejadas para o artefacto e da sua arquitetura, e em seguida, a criação do artefacto (Neiva, 2013; Peffers *et al.*, 2007). Os artefactos desenvolvidos podem ser de quatro tipos: construtos, modelos, métodos, ou instanciações (Hevner, March, Park, & Ram, 2004; Neiva, 2013).

4. **Demonstração**: Neste momento faz-se a demonstração da utilização do artefacto para resolver uma ou mais instâncias do problema. A demonstração pode envolver a utilização do artefacto em experiências, simulações, casos de estudo, entre outras (Neiva, 2013; Peffers *et al.*, 2007).
5. **Avaliação**: A utilidade, a qualidade e a eficácia de um artefacto devem ser rigorosamente demonstradas através de métodos de avaliação bem executados (Hevner *et al.*, 2004; Neiva, 2013). Nesta fase, avalia-se, observando e medindo, se o artefacto desenvolvido fornece uma solução para o problema. Esta atividade envolve a comparação entre objetivos e os resultados reais produzidos pelo artefacto na demonstração. Nesta fase, é necessário conhecer as métricas relevantes e as técnicas de análise (Neiva, 2013; Peffers *et al.*, 2007). Dependendo da natureza do problema, a avaliação pode assumir muitas formas (Neiva, 2013):
    a) Comparação das funcionalidades do artefacto com os objetivos para a solução apresentados na segunda atividade (Peffers *et al.*, 2007);
    b) Recurso a medidas quantitativas para medir o desempenho do artefacto, por exemplo, através de simulações ou medidas do tempo de resposta e disponibilidade (Peffers *et al.*, 2007);
    c) Estudo de caso: Estudar o artefacto em profundidade no ambiente empresarial (Hevner *et al.*, 2004);
    d) Estudo de campo: Monitorizar o uso do artefacto em vários projetos (Hevner *et al.*, 2004);
    e) Experiências controladas: Estudar o artefacto em ambientes controlados para avaliar as suas qualidades, como por exemplo, a usabilidade (Hevner *et al.*, 2004).
6. **Comunicação**: Na última fase, comunica-se o problema e a sua importância, o artefacto, a sua utilidade e novidade, o rigor da sua conceção, e a sua eficácia (Neiva, 2013).

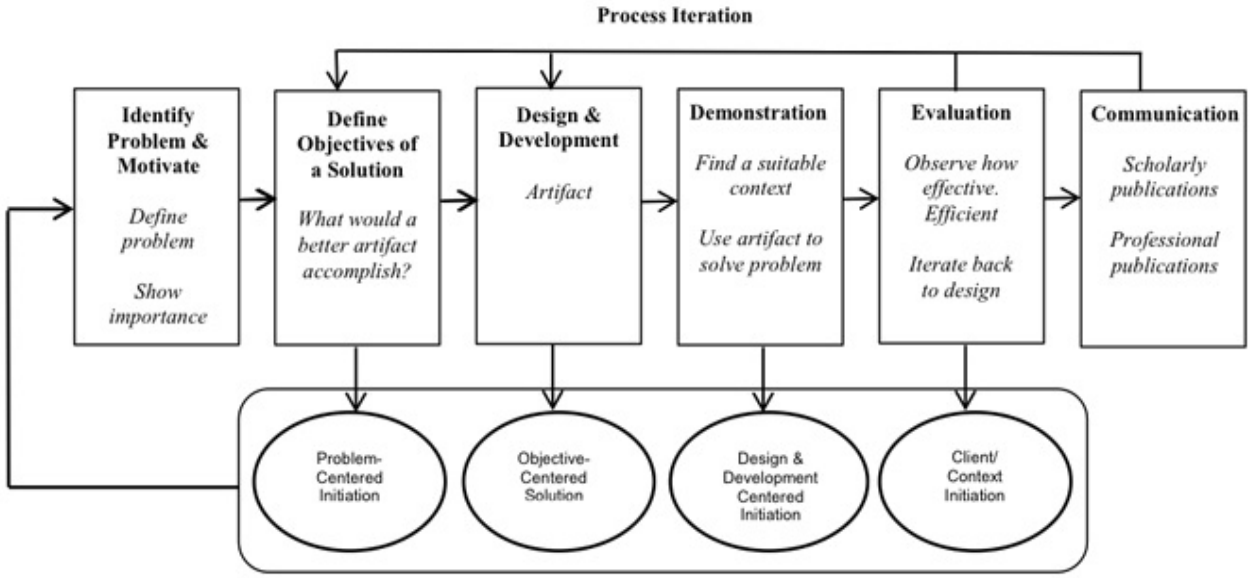


Ilustração 1: Design Science Research, adaptado de (Peffers et al., 2007)

1.3.2 Estratégia para a Revisão de Literatura

Na revisão de literatura são efetuadas as seguintes atividades:

- Identificação dos conceitos e palavras-chave relacionados com tema;
- Seleção das fontes de pesquisa científica a serem utilizadas na revisão de literatura:
  - Google Scholar;
  - Web of Science;
  - Springer;
  - Scopus;
  - Repositórium;
  - IEEE Xplore Digital Library;
  - Science Direct.
- Pesquisa nas fontes direcionadas para pesquisa científica para encontrar artigos científicos, livros, dissertações de mestrado ou teses de doutoramento sobre a área do tema;
- Seleção dos artigos científicos com base nos seguintes critérios:
  - Credibilidade do autor (se é uma referência na área ou não);
  - Credibilidade da revista onde o artigo está publicado;
  - Número de citações do artigo.

- Ler o resumo dos artigos científicos, capítulos de livros, dissertações de mestrado e teses de doutoramento e fazer uma segunda seleção dos mais relevantes;
- Pesquisa na referência bibliográfica dos artigos, livros, dissertações e teses selecionados de maneira a obter mais objetos de estudo com informação relevante;
- Recolha de informação relevante em artigos, livros, dissertações e teses selecionados;
- Redação da revisão de literatura com base na informação recolhida.

1.4 Estrutura do Documento

Esta secção descreve os constituintes deste documento e explica como este está organizado.

No primeiro capítulo faz-se um enquadramento do problema e motivação para a realização desta dissertação de mestrado. Segue-se a apresentação dos objetivos e dos resultados esperados deste trabalho. Também é selecionado um método de investigação científica para suportar o desenvolvimento, seguido de uma explanação deste. Como conclusão deste capítulo, e com o objetivo de dar a conhecer ao leitor os diferentes tópicos do presente trabalho, é apresentada a estrutura do documento.

No segundo capítulo analisa-se o estado de arte referente ao tema. Esta análise é constituída por três temas fundamentais para resolver o problema em questão:

1. Requisitos;
2. Engenharia de requisitos;
3. Ciclos de vida.

No terceiro capítulo concebe-se um processo ajustável ao departamento CI-CWR1 da Bosch Car Multimédia Portugal S.A. para solucionar o problema que surge neste ambiente industrial. Primeiramente, as atividades associadas a este processo são explicadas e justificadas detalhadamente. De seguida, determinam-se os atores necessários para execução deste processo.

No quarto capítulo é experimentado o processo concebido no capítulo 3 utilizando problemas manifestados na organização Bosch Car Multimédia Portugal S.A. São realizadas duas experimentações onde se verificam o problema, o processo em execução para resolver o problema e uma discussão dos resultados que advêm da experimentação. Entre as duas experimentações surge uma divulgação de resultados efetuada na organização.

No quinto capítulo, conclui-se a dissertação de mestrado comparando aquilo que foi planeado com o efetuado. De seguida apresentam-se as limitações e trabalho futuro inerentes a este assunto.

2 REQUISITOS, ENGENHARIA DE REQUISITOS E CICLOS DE VIDA

2.1 Introdução

Como é referido no capítulo 1, o problema da presente dissertação prende-se com o facto de haver uma lacuna no departamento CI-CWR1, da organização Bosch Car Multimédia S.A., relativamente à preservação de requisitos de projetos de desenvolvimento de sistemas de informação.

No presente capítulo deste documento é apresentada uma análise do "estado de arte" que identifica e trata os principais tópicos que fundamentam a solução para o problema deste objeto de investigação. As temáticas peculiares a esta revisão de literatura estão divididas por três secções de maior relevância.

No sub-capítulo 2.2, são abordados os conceitos relacionados com a noção de requisito. Primeiramente, o conceito de requisito é definido segundo diversos autores. Após isto, o requisito é representado numa classificação quando ao seu nível e tipo. Para finalizar a sub-secção, a classificação dos requisitos quanto ao seu tipo é pormenorizada.

No sub-capítulo 2.3, a teoria respeitante à disciplina de engenharia de requisitos é desmistificada. Inicialmente é apresentada a definição de engenharia de requisitos. De seguida, esta é estruturada num processo, constituído pelas atividades, que são genericamente efetuadas neste âmbito.

No sub-capítulo 2.4, são apresentados os ciclos de vida de instâncias relacionadas com o tema desta dissertação de mestrado, ou seja, tem o propósito de ilustrar um ciclo de vida convencional para o produto, o sistema/software e o requisito.

2.2 Requisitos

Neste sub-capítulo é explanado o conceito mais elementar da temática desta dissertação de mestrado, o requisito. Inicialmente, o termo requisito é definido segundo diversas perspetivas, de seguida é mencionada uma classificação de requisitos quanto ao seu nível e tipo. Posteriormente a classificação de requisitos quanto ao seu tipo é estudada com detalhe.

2.2.1 Definição de Requisito

De um modo geral, um requisito pode ser definido como qualquer coisa que alguém deseja (Fernandes & Machado, 2015). Sendo assim, quando falamos de requisitos, assumimos na forma mais elementar, que há uma entidade (pessoa ou objeto) que sente a necessidade de satisfazer uma limitação.

Admitindo que uma entidade está sempre num sistema, e este sistema pode ter a maior dimensão, como é o caso do sistema solar, ou a menor, onde pode ser considerado um sistema de informação de pequenas proporções, as necessidades estão por natureza inerentes a este domínio. Segundo Sommerville (2010) "os requisitos de um sistema são descrições do que o sistema deve fazer, os serviços que o sistema fornece e as suas restrições". O conceito de requisito pode ser definido de muitas formas, devido à sua transversalidade, no entanto, é relevante apresentar a definição proposta pelo IEEE (1990) :

1. Uma condição ou uma capacidade que alguém necessita para resolver um problema ou para alcançar um objetivo;
2. Uma condição ou uma capacidade que deve ser verificada ou possuída por um sistema ou por um componente do sistema, para satisfazer um contrato, norma, especificação, ou outros documentos formais impostos;
3. Uma representação documentada de uma condição ou capacidade, em (1) e (2).

Enquanto as clausulas (1) e (2) providenciam definições muito razoáveis do que é um requisito, a cláusula (3) está claramente a definir aquilo que pode ser entendido por "representação documentada" (Kaindl & Svetinovic, 2010).

Haskings (2006) fez um bom trabalho para evitar esta confusão e definiu requisitos como "caraterísticas que identificam os níveis de cumprimento necessários para alcançar objetivos específicos para um dado conjunto de condições". A definição de Davis (2005), "caraterística observável externa a um sistema desejado", evita qualquer confusão na distinção entre requisitos e as suas representações (Kaindl & Svetinovic, 2010).

Um engenheiro de software não usa o termo "requisito" no mesmo sentido que a definição da palavra num dicionário: Algo exigido ou obrigatório, uma necessidade (Wiegers & Beatty, 2013). Segundo o IEEE (2014) um requisito de software no seu estado mais básico, é uma propriedade que precisa de ser exibida em alguma encomenda para resolver algum problema do mundo real.

Um cliente quando pretende fazer um investimento num software para resolver um conjunto de problemas, assume que este será a resolução de todas as suas restrições. O engenheiro de software deve lembrar o cliente que software com um elevado número de caraterísticas, e de elevada qualidade exige um vasto investimento. É neste momento que começam a aparecer os problemas críticos e os menos prioritários.

As pessoas tendem a questionar se há mesmo necessidade de priorizar requisitos, pois se um requisito tem pouca prioridade, provavelmente não será implementado. Se um requisito não é verdadeiramente necessário, então não é um requisito, reivindicam eles (Wiegers & Beatty, 2013). Um

requisito tem uma componente temporal associada, um requisito que é altamente prioritário hoje, pode ser menos prioritário numa fase avançada, consequentemente um requisito menos prioritário hoje, pode ser extremamente prioritário numa fase diferente do ciclo de vida de um produto. Um requisito sendo menos prioritário, não deixa de ser um requisito.

Requisitos podem ser levantados de diferentes fontes e podem satisfazer diferentes necessidades. O IEEE (1990) introduziu a primeira classificação para requisitos, separando estes em necessidades do utilizador e capacidade do sistema. Com isto é legítimo afirmar que "um **requisito** é uma capacidade que um sistema deve possuir, para satisfazer necessidades de utilizadores" (Fernandes & Machado, 2015).

2.2.2 Classificação de Requisitos

O termo "requisito" não é utilizado consistentemente na indústria de software. Em alguns casos, um requisito é simplesmente uma afirmação abstrata e de alto nível de um serviço que um sistema deve fornecer ou restringir num sistema, ou inversamente, é uma definição detalhada e formal de uma função de um sistema (Sommerville, 2011). Devido à existência de diversos tipos de informação de requisitos, surge a necessidade de criar conjuntos consistentes de adjetivos para mudar o termo "requisito", devido à sua vasta usabilidade (Wiegers & Beatty, 2013).

Os requisitos podem ser classificados quanto a seu nível ou tipo. O nível e tipo de requisitos estão por norma intimamente associados. Pode-se afirmar que um requisito é alto nível quando o seu detalhe é reduzido, e baixo nível quando tem um grande detalhe.

Aurum e Wohlin (2005) distinguem requisitos primários e requisitos derivados, ou seja, há um requisito primário que é derivado em outros requisitos para detalhar uma dada funcionalidade que será implementada num dado sistema. Aqui vemos um exemplo do nível associado ao tipo de requisito, dado que um requisito primário tem pouco detalhe, logo este é alto nível. Contrariamente, um requisito derivado tem grande detalhe, com isto pode-se afirmar que este é baixo nível.

Classificar os requisitos quanto ao seu nível como alto e baixo, é apenas uma forma trivial de um fazer. Wiegers e Beatty (2013) diferenciam o nível de requisitos em (desde o alto nível até ao grande detalhe): requisitos de negócio; requisitos de utilizador; requisitos funcionais. Krishnamurthy e Saran (2008) entendem que os requisitos podem ser organizados numa hierarquia, aliando à classificação anterior os "requisitos detalhados", que constituem um nível inferior dos requisitos funcionais.

O IIBA (2009) identifica os seguintes níveis de requisitos:

- Requisitos de negócio – Declarações alto nível das balizas, objetivos, ou necessidades da empresa;
- Requisitos de *stakeholders* – Declarações da necessidade de um *stakeholder*[2] ou de uma classe de *stakeholders*;
- Requisitos da solução – Caraterísticas de uma solução que conhece os requisitos de negócio e de *stakeholders* (é neste nível que encaixam os requisitos funcionais e não funcionais);
- Requisitos de implementação – Descreve capacidades que a solução deve ter para facilitar a transição do estado atual da empresa para um estado futuramente desejado.

Os requisitos são frequentemente agrupados em categorias que alguns autores classificam quanto ao seu nível, enquanto outros classificam estes quanto ao seu tipo, ou então, uma classe pode alinhar um nível com um tipo. Por exemplo, Wiegers e Beatty (2013) entendem que os requisitos funcionais podem ser uma classe de nível, enquanto Sommerville (2010) compreende que requisitos funcionais são uma classe de tipo que constitui uma classe de tipo maior, os requisitos de sistema.

Relativamente à classificação dos requisitos quanto ao seu tipo, é importante afirmar que não há um consenso entre os diversos autores. Krishnamurthy e Saran (2008) mencionam duas categorias: requisitos funcionais e não funcionais. Em adição às duas categorias anteriormente referidas, estes autores sugerem que os requisitos podem ser desejos do cliente e necessidades que o sistema deve possuir.

Sommerville (2010) distingue os requisitos em requisitos de utilizador (abstrações mais alto nível das intenções do cliente), e requisitos de sistema (descrições mais baixo nível do que o sistema deve fazer). Posteriormente, divide os requisitos de sistema em requisitos funcionais e não funcionais.

Wiegers e Beatty (2013) também sugerem algumas definições de tipos de requisitos (ver Tabela 1).

2 Um *stakeholder* é uma parte interessada ou interveniente num projeto.

Tabela 1: Tipos de requisitos, adaptado de (Wiegers & Beatty, 2013)

Termo	Definição
Requisito de negócio	Um objetivo de negócio alto nível da organização que constrói um produto ou de um cliente que o adquire.
Regra de negócio	Uma política, diretriz, norma ou regulação que define ou restringe algum aspeto do negócio. Não é um requisito de software na sua forma natural, mas origina vários tipos de requisitos de software.
Restrição	Uma restrição que é imposta sobre as escolhas disponíveis para o desenvolvedor para o conceção e construção de um projeto.
Requisito de interface externa	Uma descrição de uma conexão entre um sistema de software e um utilizador, outro sistema de software, ou um dispositivo de hardware.
Caraterística	Uma ou mais capacidades logicamente relacionadas de um sistema que fornece valor para um utilizador, e é descrita por um conjunto de requisitos funcionais.
Requisito funcional	Uma descrição de um comportamento que um sistema exibirá sobre condições especificadas.
Requisito não funcional	Uma descrição de uma propriedade ou caraterística que um sistema deve exibir ou uma restrição que este deve respeitar.
Atributo de qualidade	Um tipo de requisitos não funcionais que descreve um serviço ou uma característica de desempenho de um produto.
Requisito de sistema	Um requisito de alto nível para um produto que contém múltiplos subsistemas, que podem ser todas as instâncias de software e hardware.
Requisito de utilizador	Um objetivo ou tarefa que especifica classes de utilizadores que devem poder executar o sistema, ou um atributo do produto desejado.

Chemuturi (2013) sugere ainda 3 tipos de classificação alternativos:

- Classificação de requisitos baseada em considerações de funcionalidade – Constituída por requisitos de funcionalidades nucleares e requisitos de funcionalidade auxiliar (equivalente aos requisitos funcionais e não funcionais);
- Classificação de requisitos baseada em considerações da construção do produto – Constitui características que irão minimizar defeitos ao longo da vida do produto;
- Classificação de requisitos baseada na fonte dos requisitos – Identifica as fontes onde os requisitos são levantados.

Os requisitos são classificados em níveis e tipos. Os níveis de um requisito dividem-se em alto nível, que significa pouco detalhe, e baixo nível que está associado a um elevado detalhe dos requisitos. Os tipos de requisitos podem ser requisitos de utilizador, requisitos de sistema, requisitos funcionais e requisitos não

funcionais (este tipo de requisito é constituído por vários subtipos (ver capítulo 2.2.2, classificação de requisitos)).

Requisitos de Utilizador e de Sistema

Uma forma de diferenciar os requisitos quanto ao seu tipo, é separá-los pelo tipo de abordagem a que são sujeitos. Há requisitos que podem ter um papel mais importante em atividades mais alto nível, como é o caso da negociação de requisitos (ver capítulo 2.3.2, atividades da engenharia de requisitos), enquanto outros são mais voltados para atividades de desenvolvimento do sistema. Inerentes a estas atividades estão *stakehollders* que esperam especificações de requisitos compatíveis com as suas competências (ver Ilustração 2).

Frequentemente estas abordagens são feitas por utilizadores, ou por desenvolvedores do sistema, assim nascem os requisitos de utilizador e os requisitos de sistema.

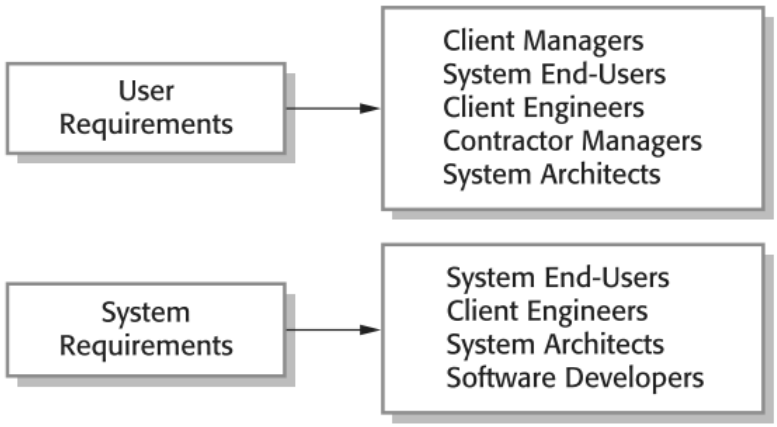


Ilustração 2: Leitores dos diferentes tipos de requisitos, adaptado de (Sommerville, 2011)

Segundo Fernandes e Machado (2015), um requisito de utilizador representa uma funcionalidade que é esperada que o sistema providencie aos seus utilizadores, ou uma restrição que é aplicável a uma operação de um sistema. Estes requisitos são direcionados para o domínio do problema e são normalmente expressos sem grande rigor matemático. Um requisito de sistema é uma especificação mais detalhada de um requisito, sendo que, é geralmente um modelo formal do sistema desejado. Estes requisitos são orientados para o domínio da solução, dando informações aos engenheiros de forma a ajudá-los na conceção e construção do sistema. Os requisitos de sistema encontram-se numa etapa entre os requisitos de utilizador e a conceção do sistema.

Para Sommerville (2010), requisitos de utilizador são afirmações, em linguagem natural aliadas de diagramas, respeitantes aos serviços que os utilizadores esperam do sistema, e as suas restrições. Requisitos de sistema são descrições mais detalhadas das funções, serviços, e restrições operacionais do sistema. O documento dos requisitos do sistema deve definir exatamente o que é para ser implementado.

Recuperando os requisitos primários e derivados de Aurum e Wohlin (2005), é possível fazer uma associação entre estes e os requisitos de utilizador e sistema. No fundo, requisitos do sistema derivam de requisitos de utilizador. É expectável que um requisito de utilizador gere vários requisitos de sistema. Segue um exemplo de um requisito de utilizador derivado em diversos requisitos de sistema, adaptado de (Fernandes & Machado, 2015):

Requisito de utilizador:

- Um utilizador manipula ficheiros criados por ele e por outros utilizadores.

Requisitos de sistema:

1. Os tipos de ficheiro e os respetivos ícones são definidos pelos utilizadores;
2. Cada tipo de ficheiro é representado por um ícone distinto;
3. Cada tipo de ficheiro está associado a um programa que processa e manipula os ficheiros correspondentes;
4. Quando um utilizador clica num ícone do ficheiro, o ficheiro será automaticamente aberto por um programa associado.

Neste exemplo, é possível verificar o nível a que cada tipo de requisito está associado. O **requisito de utilizador** é concebido para contrariar o problema de um utilizador, que neste caso é manipular ficheiros. Contudo, há requisitos que se não são incluídos não há possibilidade de satisfazer a necessidade do utilizador, respeitantes mais à tecnologia utilizada para desenvolver a solução do utilizador, são estes os **requisitos de sistema**. Por exemplo, assumindo que um requisito de um indivíduo é adquirir um carro, devido à dificuldade de se deslocar para o seu emprego, ele não deve mencionar que o carro deve ter uma chave para ligar e desligar, uma vez que ele assume a chave como um constituinte natural de um automóvel.

Requisitos Funcionais

Este tipo de requisitos é reconhecido por comunidades de engenheiros de software. Endres e Rombach (2003) afirmam que a ênfase principal do sistema está nos requisitos funcionais.

Aurum e Wohlin (2005) definem um requisito funcional como o que o sistema irá fazer.

O IEEE (2014) reconhece requisitos funcionais como sendo descrições de funções que o software tem para executar. Eles são também conhecidos como capacidades ou caraterísticas.

O IIBA (2009) entende por requisito funcional, uma descrição do comportamento e informação que a solução irá gerir. Descreve também capacidades que o sistema irá ser capaz de executar em termos de comportamentos e operações, uma ação específica do sistema ou uma resposta.

Segundo Krishnamurthy e Saran (2008), um requisito funcional é o que genericamente chamamos de requisito. Os requisitos relacionados com o processamento lógico do sistema são requisitos funcionais.

Laplante (2007) afirma que requisitos funcionais descrevem os serviços que o sistema deve fornecer. Requisitos funcionais podem ser alto nível, gerais ou detalhados, expressando entradas, saídas, exceções e por aí adiante.

Sommerville (2010) refere que requisitos funcionais descrevem o que o sistema deve fazer. Quando expressados como requisitos de utilizador, os requisitos funcionais são normalmente descritos num contexto abstrato que pode ser entendido pelos utilizadores do sistema. Requisitos funcionais descrevem as funções do sistema, as suas entradas, as suas saídas e exceções em detalhe.

Wiegers e Beatty (2013) reconhecem um requisito funcional como uma descrição de um comportamento que o sistema irá exibir em condições específicas.

Fernandes e Machado (2015) compreendem que um requisito funcional descreve uma funcionalidade a ser disponibilizada para os utilizadores do sistema, caracterizando parcialmente os seus comportamentos como uma resposta a estímulos a que são sujeitos. Este tipo de requisitos não deve mencionar qualquer tipo de questão tecnológica, consequentemente deve ser independente das fases de conceção e implementação.

Um **requisito funcional** é uma descrição de uma funcionalidade, das suas entradas e das suas saídas que irão constituir um sistema, sem mencionar qualquer tipo de questão tecnológica, ou seja, deve ser independente das fases de conceção e implementação.

Fernandes e Machado (2015) propõe ainda uma sub-classificação dos requisitos funcionais, são eles os requisitos implícitos e explícitos. Um requisito implícito é um requisito que o analista inclui, baseado no conhecimento do domínio que possui, sendo que este não foi levantado em *stakeholders* solicitados. Contrariamente, um requisito explícito é solicitado por um *stakeholder*.

Requisitos Não Funcionais

Este tipo de requisitos, como o nome sugere, não estão diretamente relacionados com serviços específicos prestados pelo sistema para os seus utilizadores (Sommerville, 2011). Aurum e Wohlin

(2005) definem requisitos não funcionais como restrições nos tipos de soluções que vão de encontro aos requisitos funcionais. Estes enumeram ainda alguns exemplos: precisão, desempenho, segurança e modificabilidade.

Segundo Endres e Rombach (2003), requisitos não funcionais são requisitos subentendidos e podem incluir alguns critérios de qualidade como portabilidade, confiabilidade, eficiência, usabilidade, testabilidade, manutenção e capacidade de reutilização. Este tipo de requisitos tem um maior impacto no custo do que os requisitos funcionais.

O IEEE (2014) entende que os requisitos não funcionais são aqueles que restringem a solução, e podem ser conhecidos também por requisitos de qualidade ou de restrição. Alguns exemplos deste tipo de requisitos podem ser: requisitos de desempenho, requisitos de manutenção, requisitos de segurança, requisitos de confiabilidade, requisitos de interoperabilidade, entre outros.

Para o IIBA (2009), requisitos não funcionais capturam condições que não se relacionam diretamente com o comportamento ou funcionalidade da solução, mas descrevem melhor condições ambientais para que a solução se mantenha eficaz, ou qualidades que o sistema deve ter. Em adição, estes requisitos podem ser denominados de requisitos de qualidade ou requisitos suplementares.

Krishnamurthy e Saran (2008) referem que requisitos não funcionais podem ser entendidos como requisitos auxiliares, e podem ter um impacto direto na conceção e arquitetura da solução.

Laplante (2007) afirma que requisitos não funcionais são impostos pelo ambiente em que o sistema vai existir, e estes podem constituir restrições de tempo, propriedades de qualidade, normas aderidas, linguagens de programação para serem usadas, conformidades com leis, entre outros.

Sommerville (2010) compreende que requisitos não funcionais são restrições nos serviços ou funções oferecidas pelo sistema. Eles incluem restrições de tempo, restrições no processo de desenvolvimento, e restrições impostas por normas. Contrariamente aos requisitos funcionais, estes são frequentemente aplicados ao sistema como um todo.

Wiegers e Beatty (2013) assumem que um requisito não funcional é “uma descrição de uma propriedade ou caraterística que um sistema deve exibir ou uma restrição que ele deve respeitar”.

Fernandes e Machado (2015) reivindicam que um requisito não funcional corresponde a um conjunto de restrições impostas no sistema para ser desenvolvido, estabelecendo por exemplo o quão atrativo, útil, rápido ou confiável ele é.

Um **requisito não funcional** é uma caraterística de um sistema, expressa na forma de uma qualidade ou restrição, que ajuda a definir uma ou várias funcionalidades. Por exemplo, um computador

providencia diversas funcionalidades, contudo, se este não é minimamente tolerante a falhas, a utilização deste equipamento torna-se incómoda.

Requisitos não funcionais podem ser classificados. Sommerville (2010) divide os requisitos não funcionais em 3 categorias, são elas (ver Ilustração 3):

- Requisitos de produto: Estes requisitos especificam ou restringem o comportamento do software. Exemplos: requisitos de desempenho, requisitos de confiabilidade, requisitos de segurança e requisitos de usabilidade;
- Requisitos organizacionais: Estes requisitos são requisitos de sistema amplos derivados de políticas e procedimentos da organização do cliente e do desenvolvedor. Exemplos: requisitos do processo operacional, requisitos do processo de desenvolvimento e requisitos de ambiente;
- Requisitos externos: Todos os requisitos que são derivados de fatores externos ao sistema e do seu processo de desenvolvimento. Eles podem incluir requisitos regulamentais, requisitos legislativos e requisitos éticos.

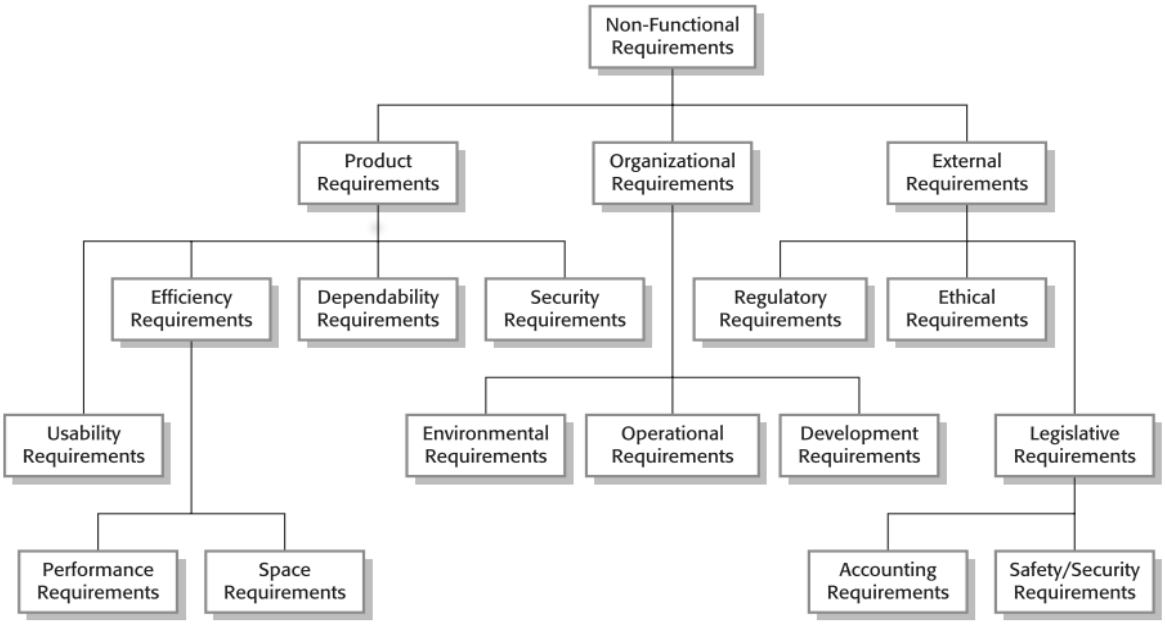


Ilustração 3: Classificação de requisitos não funcionais, adaptado de (Sommerville, 2011)

Robertson e Robertson (2012) classificam os requisitos em 8 tipos:

1. Requisitos de aparência: aspeto visual do sistema;
2. Requisitos de usabilidade: facilidade de utilização do sistema;
3. Requisitos de desempenho: definem aspetos como a velocidade do sistema ou a capacidade de armazenamento;

4. Requisitos operacionais: caraterísticas do ambiente onde o sistema irá funcionar;
5. Requisitos de manutenção e apoio: mudanças esperadas e especificação do suporte dado ao sistema;
6. Requisitos de segurança: segurança e confidencialidade do sistema;
7. Requisitos culturais e políticos: cultura e costumes do pessoal relacionado com a operação do sistema;
8. Requisitos legais: Leis e normas aplicáveis ao sistema.

Há outras formas de classificar os requisitos não funcionais, contudo esta última classificação é considerada a mais adequada no contexto desta dissertação de mestrado.

Requisitos de Aparência

Este tipo de requisito não funcional é muitas vezes decisivo na satisfação de um cliente relativamente a um novo produto. Muitas organizações têm cores, logótipos ou fontes inerentes à sua filosofia, com isto é frequente um cliente requerer um novo produto com a sua identidade. Por exemplo, muitas marcas têm atualmente loja online para evitar que os seus clientes visitem as lojas físicas. É obrigatório que os clientes dessa marca se sintam identificados com a marca, quando estão a efetuar uma compra neste meio. Por exemplo, a *Apple*[3] tem como logótipo uma maçã que tem um significado relevante relativo ao marketing da marca, é fundamental esta maçã estar em destaque na sua loja online para os clientes sentirem que aquele serviço não é uma fraude.

Um exemplo de requisito de aparência pode ser:

- O produto deve utilizar as cores, logótipo e fontes da organização onde vai atuar.

Requisitos de Usabilidade

Um engenheiro de software deve considerar sempre as habilitações do utilizador final. Muitas vezes o acesso aos utilizadores finais é limitado e isto leva a uma substituição destes por outros *stakeholders* (Krishnamurthy & Saran, 2008), estas situações devem ser evitadas. Há pessoas mais incompatíveis com uma determinada tecnologia que outras, como consequência disto surge a necessidade de evitar ambiguidades, mesmo no mais óbvio dos cenários. Por exemplo, no departamento de informática da Bosch

3 A Apple é uma empresa multinacional que atua no âmbito das novas tecnologias fundada por Steve Jobs, Steve Wozniak e Ronald Waine

Car Multimédia Portugal S.A.[4] faz-se um esforço para perceber quais são os utilizadores finais em todos os projetos. Se o produto for satisfazer um problema no departamento dos recursos humanos há a necessidade deste fazer uma interface muito mais interativa. Muitas vezes é referido que um sistema só fica estável após um período considerável nas mãos do utilizador final.

Um exemplo de requisito de usabilidade pode ser:

- O produto deve ter uma interface compatível com pessoas portadoras de deficiência motora.

Requisitos de Desempenho

O desempenho refere a capacidade de um sistema responder a estímulos, como por exemplo, o tempo necessário para responder a eventos ou o número de eventos processados por uma unidade de tempo (Fernandes & Machado, 2015).

Fernandes e Machado (2015) referem que os requisitos de desempenho devem referir caraterísticas do sistema como:

- Precisão: precisão dos cálculos feitos pelo sistema;
- Disponibilidade: percentagem do tempo que um sistema opera corretamente. Endres e Rombach (2003) expressam a disponibilidade na seguinte fórmula:
- *Availability* = $\frac{MTBF}{MTBF + MTTR}$ onde MTBF é o tempo médio de falha e MTTR é o tempo médio de reparação;
- Tolerância a falhas: capacidade que um sistema tem para um nível aceitável de operação em circunstâncias indesejáveis;
- Capacidade de armazenamento de dados: quantidade de dados que o sistema é capaz de processar e armazenar;
- Escalabilidade: habilidade do sistema manter qualidade do serviço, quando exposto um grande número de pedidos.

Um exemplo de requisito de desempenho pode ser:

- O produto deve abrir um ficheiro do tipo xls[5] em 1 segundo.

4 Bosch Car Multimédia Portugal S.A. é uma organização internacional instalada em Braga, Portugal

5 Xls é a extensão de ficheiro utilizada pelo Microsoft Excel

Requisitos de Operacionalidade

Requisitos de operacionalidade descrevem aspetos relativos ao ambiente onde o sistema irá trabalhar. Há produtos que são desenvolvidos para funcionar em ambientes adversos. Por exemplo um relógio para um mergulhador tem de funcionar numa profundidade considerável. Estes requisitos também referem outros sistemas que interagem com o sistema a desenvolver, por exemplo o libreoffice[6] é um programa que funciona em ambiente linux[7], é necessário um protocolo para o sistema operativo poder executar o programa.

Um exemplo de requisito de operacionalidade é:

- O produto deve ser resistente à água a uma profundidade de 100 metros.

Requisitos de Manutenção e Apoio

A manutenção do sistema normalmente é dividida em 4 tipos: preventivo, corretivo, perfetivo e adaptativo (Fernandes & Machado, 2015). É neste tipo de requisitos que se prevê mudanças no futuro e se pensa o sistema para ser afetado no menor número de módulos possível numa situação destas. Os requisitos de apoio estão relacionados com o tipo de suporte que se dá aos utilizadores finais em forma de menus de ajuda e tutoriais.

Um exemplo de requisito de manutenção e apoio pode ser:

- O produto deve estar preparado para ser traduzido para qualquer linguagem.

Requisitos de Segurança

Estes requisitos devem compreender caraterísticas como a confidencialidade e integridade do sistema. Confidencialidade protege dados alocados para o produto de acessos não autorizados. A integridade atua sobre a confiabilidade dos dados do sistema ao longo do seu ciclo de vida, ou seja, garante que os dados não são apagados ou alterados de forma negligenciada, ou através de um ataque de uma fonte exterior ou interior ao sistema.

Um exemplo de requisito de segurança pode ser:

- O produto deve rejeitar a introdução de letras no campo do contacto telefónico.

---

6 Libreoffice é um conjunto de ferramentas de escritório de código aberto

7 Linux é o núcleo de um conjunto de sistemas operativos de código aberto desenvolvido por Linus Torvalds

Requisitos Culturais e Políticos

Requisitos culturais estão relacionados com os valores dos utilizadores finais. Pessoas têm crenças e costumes do dia-a-dia que se lhes são negados, podem criar uma grande insatisfação. A religião, a língua ou a cozinha são aspetos que definem a cultura de uma pessoa. Há aspetos que restringem a forma de atuar de certas pessoas, ou que criam preconceitos. Por exemplo nenhuma religião aceita o casamento entre pessoas do mesmo sexo, uma instituição extremamente religiosa ficaria insatisfeita com uma referência à homossexualidade. Requisitos políticos englobam fatores relacionados com a estratégia e poderes da organização. É preciso ter em atenção que um novo produto não vai prejudicar uma pessoa com poder na organização, para evitar situações de inércia social ou de contra-implementação (Keen, 1981).

Um exemplo de requisito cultural ou político pode ser:

- O produto deve mostrar os feriados religiosos num calendário.

Requisitos Jurídicos

No capítulo 2.2.1 é referido que sistema pode ter dimensões astronómicas como é o caso do sistema solar porém, até este sistema tem de respeitar regras da física, como a gravidade que obriga os planetas a girar em torno do sol. Para um sistema mais pequeno, como é o caso de um sistema de informação, é preciso ver que em diversos municípios existem leis que obrigam os elementos do seu sistema a comportarem-se de uma determinada forma. Por exemplo, o ataque organizado a uma base de dados confidencial é considerado crime em Portugal. Para todos os contextos também existem instituições credíveis que auxiliam uma determinada atividade. Por exemplo, o IEEE tem várias normas que ajudam no desenvolvimento de software. Para um novo sistema é preciso garantir que não irá infringir nenhuma lei para garantir o seu sucesso.

Um requisito jurídico pode ser:

- A qualidade do produto deve estar definida com base na norma ISO/IEC 9126.

2.2.3 Sumário

Um requisito é uma capacidade que um sistema deve possuir para satisfazer necessidades de utilizadores. Os requisitos podem ser classificados quanto a seu nível e quanto ao seu tipo. O nível dos requisitos pode ser alto nível e baixo nível. Pode-se classificar os requisitos quanto ao seu tipo em: requisitos de utilizador, requisitos de sistema, requisitos funcionais e requisitos não funcionais. Um requisito do utilizador é um requisito que é feito para atividades alto nível, logo deve ser percetível tanto por utilizadores como por

desenvolvedores do produto, já um requisito de sistema deve ter um nível de detalhe apropriado para auxiliar fases avançadas como a conceção e o desenvolvimento. Um requisito funcional descreve uma funcionalidade, as suas entradas e as suas saídas que irão constituir um sistema, sem mencionar qualquer tipo de questão tecnológica, ou seja, deve ser independente das fases de conceção e implementação. Requisitos não funcionais são caraterísticas do sistema expressas em forma de qualidades ou restrições que auxiliam as funcionalidades do sistema. Estes dividem-se em requisitos de aparência, requisitos de usabilidade, requisitos de desempenho, requisitos de operacionalidade, requisitos de manutenção e apoio, requisitos de segurança, requisitos culturais e políticos, e requisitos jurídicos.

2.3 Engenharia de Requisitos

Este sub-capítulo integra a teoria relevante da atividade de engenharia de requisitos. Inicialmente define-se o conceito de engenharia de requisitos, posteriormente este é transformado num processo que constitui atividades de modo a garantir o seu sucesso. Seguidamente, estas atividades são exploradas e explicadas.

2.3.1 Definição de Engenharia de Requisitos

Geralmente, quando uma pessoa entra em contacto com a engenharia de software, nomeadamente um aluno que se encontra numa fase inicial de cursos que incorporam esta disciplina, tende a desprezar a subárea de engenharia de requisitos. Uma vez que, o seu conteúdo é teórico comparando com outras subáreas, cria-se um estereótipo de que a engenharia de requisitos é trivial, e a sua atividade é normalmente atribuída a pessoas pouco talentosas no domínio das tecnologias de programação. É muito fácil para quem nunca foi confrontando com as atividades da engenharia de requisitos afirmar que é fácil levantar requisitos. Basicamente, o cliente refere as suas necessidades, o engenheiro de requisitos documenta estas e já está. Esta última afirmação até podia fazer sentido num universo onde o cliente primeiro sabe realmente quais são as suas necessidades e segundo sabe comunicá-las sem ambiguidades. É muito frequente ouvir pessoas experientes da área do negócio afirmarem que o cliente tem sempre razão. Na ótica da engenharia de requisitos esse ditado só faz sentido se estivermos a tratar de ficção científica, pois para um engenheiro de requisitos, um cliente raramente tem razão.

A engenharia de requisitos é uma subárea da engenharia de software como foi esclarecido contudo, é transversal a muitas áreas. É necessário entender que todos os produtos, sejam de software ou não, têm requisitos. Por exemplo, uma cadeira tem o requisito funcional de garantir que uma pessoa se consegue

sentar nela, mas também tem o requisito não funcional que irá dar garantias que esta suportará o peso que uma pessoa pode ter.

Para perceber o propósito desta subárea, é necessário definir esta segundo diversos autores. Pressman (2005) entende que engenharia de requisitos constrói uma ponte entre o projeto e a construção. Para ele, a engenharia de requisitos fornece o mecanismo apropriado para entender o que o cliente deseja, analisando as necessidades, avaliando a exequibilidade, negociando uma condição razoável, especificando a solução sem ambiguidades, validando a especificação e gerindo os requisitos à medida que eles são transformados em um sistema.

Segundo Fernandes e Machado (2015), a engenharia de requisitos é um conjunto de atividades que num contexto de desenvolvimento de um sistema, através de um processo de engenharia, permite levantar, negociar, e documentar as funcionalidades e restrições de um sistema.

Cheng e Atlee (2009) argumentam que a engenharia de requisitos é o processo onde os requisitos são levantados, modelados, analisados e documentados.

Pohl (1995) define engenharia de requisitos como um processo sistemático para desenvolver requisitos, através de um processo cooperativo e iterativo de analisar o problema, documentar as observações resultantes numa variedade de formatos de representação, e verificar a assertividade da compreensão adquirida.

Para Sommerville (2010), a engenharia de requisitos é o processo de entender e definir que serviços são requeridos pelo sistema, e de identificar as restrições na operação e desenvolvimento do sistema.

Aurum e Wohlin (2005) referem que engenharia de requisitos é o processo pelo qual os requisitos para produtos de software são recolhidos, analisados, documentados, e geridos ao longo do ciclo de vida da engenharia de software. Além disso, a engenharia de requisitos está preocupada em interpretar e perceber os objetivos, necessidades e crenças dos *stakeholders*.

Zave (1997) afirma que engenharia de requisitos é um ramo da engenharia de software que está preocupado com os objetivos, funções e restrições dos sistemas de software. A engenharia de requisitos também envolve a relação destes fatores para especificações precisas do comportamento do software, e para a sua evolução ao longo do tempo.

Hull, Jackson e Dick (2006) dividem a engenharia de requisitos aplicando esta ao domínio do problema e ao domínio da solução. A engenharia de requisitos no domínio do problema promove o levantamento de requisitos dos *stakeholders*. A engenharia de requisitos no domínio da solução transforma requisitos de utilizador em requisitos de sistema.

Gilb (1997) tenta perceber como os requisitos e a engenharia se relacionam, e conclui que a junção destes dois conceitos permite uma comparação entre as ideias da conceção e os requisitos.

A **engenharia de requisitos** é uma disciplina transversal a diversas áreas, desde que estas tenham o propósito de desenvolver sistemas. É também um processo que permite levantar, negociar, documentar e gerir requisitos ao longo do ciclo de vida do sistema.

2.3.2 Processo de Engenharia de Requisitos

A engenharia de requisitos, independentemente do contexto onde estiver inserida, tem atividades que garantem o seu sucesso. Dentro da engenharia de software há diferentes métodos de desenvolvimento, como é o caso do desenvolvimento tradicional (ou em cascata) ou o desenvolvimento ágil (Pressman, 2005), mas é possível subir o nível e incorporar a engenharia de requisitos na engenharia de sistemas (Nuseibeh & Easterbrook, 2000).

Grande parte das organizações (normalmente pequenas empresas) cuja missão é o desenvolvimento de sistemas, necessitam de executar o processo de engenharia de requisitos, porém fazem-no numa abordagem completamente *ad-hoc*[8]. Organizações de desenvolvimento qualificado de sistemas têm propensão para seguir modelos de referência propostos por instituições de normalização, e adaptar estes à sua natureza (Aurum & Wohlin, 2003). Exemplos de instituições de normalização podem ser: ISO, IEC, IEEE, etc. Atualmente, a norma aceite para a engenharia de sistemas e software é a ISO/IEC/IEEE 29148 e nesta é possível encontrar boas práticas para a execução do processo de engenharia de requisitos (ISO/IEC/IEEE, 2011).

Pohl (1995) apresenta diversos métodos para definir o processo de engenharia de requisitos, contudo, argumenta que estes estão obsoletos devido a ausência de atividades que considera indispensáveis, ou usam uma abordagem *top down*[9] ou *botton up*[10] (fornecendo só algumas vantagens), ou desprezam a possibilidade de este processo ser iterativo. Ele entende que o processo da engenharia de requisitos deve possuir todas

8 *Ad-hoc* é um termo latino utilizado para definir atividades informais, ou seja, sem uma projeção prévia

9 Abordagem *top down* é uma estratégia de processamento de informação onde o seu início é no maior nível de abstração

10 Abordagem *button up* é uma estratégia de processamento de informação onde o seu início é o mais próximo da realidade

estas características e deve constituir atividades como: (1) levantamento de requisitos, (2) negociação de requisitos, (3) especificação ou documentação de requisitos e (4) validação ou verificação de requisitos.

Pressman (2005) entende que a proposta de Pohl (1995) ainda não satisfaz o propósito da engenharia de requisitos adequadamente, portanto sugere que este processo deve ter as seguintes atividades: (1) início do processo, (2) levantamento de requisitos, (3) desenvolvimento de requisitos, (4) negociação de requisitos, (5) documentação de requisitos, (6) validação de requisitos e (7) gestão de requisitos (ver Ilustração 4).

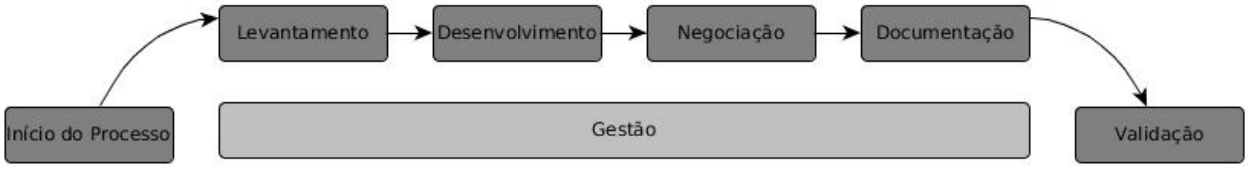


Ilustração 4: Atividades da engenharia de requisitos, adaptado de (Fernandes & Machado, 2015)

Apesar do carácter sequencial da Ilustração 4, é necessário referir que esta deve ser vista com alguma relatividade, com frequência ocorrem exceções no processo padrão (Fernandes & Machado, 2015). A gestão de requisitos é uma atividade que deve ser alinhada paralelamente a todas a outras. Desde o momento em que termina a primeira iteração do processo, a gestão de requisitos começa a interagir com todas as outras atividades (ver capítulo 2.3.2, atividades da engenharia de requisitos).

Início do processo

Todos os processos necessitam de um estímulo que ditará o seu início. O processo de engenharia de requisitos não é exceção, e precisa de ser iniciado através de uma necessidade ou uma expetativa de negócio, portanto nesta atividade é mandatório identificar o propósito do projeto (Fernandes & Machado, 2015). Também é feito um estudo de viabilidade, onde o objetivo é analisar o âmbito do sistema como um todo e deve-se evitar detalhar o problema. (Fernandes & Machado, 2015).

De modo a facilitar esta atividade, sugerem-se três grupos de questões (Fernandes & Machado, 2015; Pressman, 2005):

1. Questões direcionadas para o propósito do projeto: Recomendam-se as seguintes questões para este tipo (Fernandes & Machado, 2015):
    - Quem está por detrás do pedido para o sistema?
    - Quem irá usar o sistema?
    - Quais são os benefícios económicos do sistema?

- Há outras fontes que é necessário usar/consultar?

2. Questões para perceber o problema do cliente: Estas questões ajudam a entender melhor qual é o problema do cliente. Questões sugeridas (Fernandes & Machado, 2015):
    - Que características desejas ver incluídas nos *outputs* produzidos pelo sistema?
    - Que problemas são resolvidos com a solução?
    - Podes descrever o ambiente empresarial onde o sistema irá trabalhar?
    - Será o sistema afetado por alguma questão ou restrição de desempenho?
3. Questões para perceber a efetividade da comunicação entre atores (meta-questões): Como o próprio nome indica, este tipo de questões ajuda os atores a perceberem se a comunicação entre eles está a ser bem-sucedida. Questões propostas (Fernandes & Machado, 2015):
    - Consideras-te a pessoa correta para responder às minhas questões?
    - Achas que as minhas questões são pertinentes para o problema que tens?
    - Estou a colocar um número excessivo de questões?
    - Há questões que supostamente eu deveria estar a colocar-te?
    - Há mais alguém que pode providenciar informações adicionais?

Levantamento de Requisitos

O levantamento de requisitos é a atividade da engenharia de requisitos onde são recolhidos os requisitos das fontes através de uma técnica de levantamento (Fernandes & Machado, 2015). Christel e Kang (1992) afirmam que a primeira tarefa do levantamento de requisitos é identificar parceiros que são fontes de requisitos. As fontes de requisitos podem ser (IEEE, 2014):

1. Objetivos: Também denominado de interesse de negócio ou fator crítico de sucesso. Refere-se aos objetivos de alto nível de um sistema de informação;
2. Conhecimento do domínio: O conhecimento do domínio dá ao engenheiro de requisitos de software o discernimento necessário para discutir com *stakeholders* sobre o domínio de atuação do sistema de informação resultante do projeto, logo são uma grande ajuda nas entrevistas de levantamento de requisitos (ver abaixo). Os modelos de referência contém informação fidedigna fornecida por entidades regulamentares de um determinado domínio (Frank, 1999; C. Taylor & Probst, 2003), logo aconselha-se a recolha do conhecimento do domínio desta bibliografia;

3. *Stakeholders*: Os *stakeholders* são humanos que influenciam (positivamente ou negativamente) ou são influenciados por um projeto. O problema que origina o projeto é frequentemente identificado neste tipo de fonte (futuros utilizadores), contudo há outros tipos de *stakeholders* que não têm um impacto tão direto no projeto mas se não são considerados, podem restringir o sucesso da solução. Fernandes e Machado (2015) identificam os seguintes tipos de *stakeholders*:
    - Utilizador: Pessoa que opera e interage diretamente com o sistema;
    - Cliente: Alguém que paga para adquirir um sistema, quando está disponível para ser usado;
    - Especialista: Pessoa que providencia um conhecimento profundo de um dado domínio ou assunto;
    - Desenvolvedor: Profissional que executa atividades que contribuem para o desenvolvimento e manutenção de um dado sistema técnico;
    - Inspetor: Indivíduo que supervisiona ou inspeciona algo;
    - *Stakeholder* negativo: Alguém que deseja que o sistema não seja desenvolvido, nem colocado em operação.
4. Regras de negócio: Estados que definem ou restringem algum aspeto da estrutura ou comportamento do negócio;
5. Ambiente operacional: Requisitos que são derivados do ambiente em que o sistema de informação irá ser executado. Por exemplo: restrições de tempo;
6. Ambiente organizacional: O engenheiro de requisitos de software deve ser sensível com o facto de o sistema de informação não forçar mudanças não planeadas em processos de negócio. Além disto, também deve ter em consideração que o sistema de informação será condicionado pela estrutura, cultura e políticas internas à organização.

De modo a levantar corretamente requisitos das fontes, existem técnicas que auxiliam a atividade. Segundo Fernandes e Machado (2015), as técnicas de levantamento de requisitos podem ser agrupadas em três categorias: individuais, grupos de pessoas e artefactos.

Técnicas de levantamento de requisitos individuais:

- Entrevista: Colocar questões (ter em consideração que a qualidade das questões irá influenciar a qualidade das respostas) a pessoas que irão lidar com o sistema presencialmente;
- Pesquisa: Técnica adotada em vários domínios que usa questionários para recolher e manusear informação obtida de múltiplos respondentes;

- Introspeção: O engenheiro de software coloca-se na pele do cliente ou utilizador final e define os requisitos sem consultar outras fontes (normalmente o engenheiro tem um grande conhecimento do domínio onde a aplicação irá operar) (Goguen & Charlotte, 1993);
- Etnografia: Estudo do comportamento de pessoas no seu ambiente natural.

Técnicas de levantamento de requisitos baseadas em grupos de pessoas:

- *Brainstorming*: Técnica que congrega um grupo de 5 a 12 pessoas de modo a facilitar a geração de ideias;
- Foco de grupo: Convocatória de um grupo de pessoas para discutir um ponto específico;
- Aplicação conjunta da conceção: Tem o objetivo de juntar peritos do problema e engenheiros com o objetivo de se obter mais informação do domínio do problema;
- Trabalho cooperativo: trabalho que resulta de uma interação distribuída e assíncrona. Para auxiliar esta técnica existe software que facilita o trabalho em grupo executado cooperativamente.

Técnicas de levantamento baseadas em artefactos:

- Análise do domínio: Analisar o domínio onde o sistema estará localizado de maneira a encontrar elementos comuns a todos os sistemas do domínio;
- Análise orientada a objetos: Estudar os requisitos na perspetiva de objetos do mundo real e das suas classes;
- Prototipagem: Técnica valiosa para clarificar requisitos ambíguos (IEEE, 2014);
- Cenário (ou história de utilizador): Pequena história que descreve o comportamento funcional de um sistema e ilustra uma sequência específica de ações e eventos necessários para a execução;
- Modelação de objetivos: Técnica que estrutura os objetivos hierarquicamente;
- Personagem: Técnica criada por Cooper (1999) onde se pretende fazer uso de uma pessoa fictícia que representa um tipo importante de utilizadores do produto em desenvolvimento.

Para um levantamento de requisitos bem-sucedido é necessário identificar as fontes de requisitos, de seguida é necessário escolher as técnicas mais eficazes nas fontes identificadas, e por fim levantar os requisitos. O produto desta atividade é uma lista de requisitos, por norma ambíguos, que serão evoluídos na atividade de desenvolvimento de requisitos.

Desenvolvimento dos Requisitos

Esta atividade, muitas vezes denominada de análise de requisitos, foca-se em analisar e classificar os requisitos levantados na atividade anterior (Fernandes & Machado, 2015). Os requisitos levantados apresentam sempre problemas, por isso é preciso manusear estes para se aproximarem mais da solução. O IEEE (2014) divide o desenvolvimento de requisitos nas seguintes tarefas:

- Detetar e resolver conflitos entre requisitos;
- Descobrir os limites do sistema e como ele deve interagir com o seu ambiente operacional e organizacional;
- Desenvolver requisitos de sistema (normalmente derivados de requisitos de utilizador).

O resultado final da fase de desenvolvimento de requisitos é um modelo de análise que define o domínio do problema informacional, funcional e comportamental (Pressman, 2005).

Negociação de Requisitos

É um dado adquirido que um humano quer sempre o mais vantajoso para si. Isso implica muitas vezes o desejo de um número de características para o produto incompatíveis com o número de recursos disponíveis para o projeto concedente (Pressman, 2005). Como se já não bastasse os utilizadores terem desejos irrealistas, um produto tem normalmente diversos utilizadores finais, e nem sempre são estes quem fornecem os recursos. É muito frequente o cliente não ser o utilizador final.

Enquanto um cliente quer sempre despender o menor número de recursos, os utilizadores querem um produto que satisfaça da melhor forma as suas necessidades. Frequentemente até são requisitadas funcionalidades que são meras excentricidades, com isto é certo que um produto nunca vai agradar na plenitude a todos os *stakeholders.* Porém, é necessário encontrar uma solução que agrade minimamente a todos, é precisamente este o objetivo da negociação de requisitos.

A negociação é tradicionalmente vista como interações atuais entre participantes que os levam a um acordo comum, e começa quando os participantes começam a comunicar os seus objetivos, e acaba (bem-sucedida) quando acordam um contrato específico (Aurum & Wohlin, 2005).

A negociação de requisitos compreende as seguintes tarefas (Aurum & Wohlin, 2005; Fernandes & Machado, 2015):

1. Pré-negociação: Nesta fase faz-se a definição do problema, identificação de *stakeholders*, levantamento de objetivos e análise desses objetivos;

2. Negociação: Os *stakeholders* chegam a um acordo sobre as questões em conflito;
3. Pós-negociação: Os *stakeholders* analisam e avaliam as saídas da negociação e sugerem uma renegociação se necessário.

Em alguns projetos, os requisitos são tão díspares que a negociação de requisitos não é suficiente, isto é, os diversos *stakeholders* inerentes ao projeto não conseguem chegar a um acordo sobre quais são os requisitos que o novo produto deve incluir, considerando os recursos disponíveis. Nestes casos, é necessário alguma técnica que aplique um maior critério, é necessário uma técnica de prioritização de requisitos para produzir o produto mínimo viável (Ries, 2011).

Prioritização de Requisitos

A prioritização de requisitos é uma técnica que auxilia na identificação dos requisitos fundamentais e pode ser vista como um processo que ordena um conjunto de requisitos, de acordo com vários critérios (Fernandes & Machado, 2015). De acordo com Aurum e Wohlin (2005) a prioritização de requisitos dá apoio às seguintes atividades:

- Ajudar os *stakeholders* a decidir os requisitos principais para o sistema;
- Planear e selecionar um conjunto ordenado e otimizado de um conjunto de requisitos para a implementação de lançamentos sucessivos;
- Balancear o benefício de negócio de cada requisito com o seu custo;
- Escolher apenas o subconjunto de requisitos que irá produzir um sistema que irá satisfazer o cliente;
- Estimar a satisfação do cliente esperada;
- Manusear requisitos contraditórios, focar no processo de negócio e resolver desacordos entre *stakeholders*.

Em adição, a prioritização pode servir como técnica de auxílio à gestão de requisitos (Cheng & Atlee, 2007; Davis, 2005).

O processo da atividade de prioritização de requisitos constitui os seguintes passos (Fernandes & Machado, 2015; Karlsson, Wohlin, & Regnell, 1998):

1. Preparação: Uma pessoa estrutura os requisitos de acordo com o princípio da técnica a ser usada. É selecionada um equipa e um líder de equipa, e a informação necessária é-lhes fornecida;

2. Execução: Os *stakeholders* conduzem a prioritização dos requisitos baseada na informação que é previamente fornecida. O critério deve estar acordado antes de iniciar a prioritização;
3. Apresentação: Apresentação dos resultados implicados no processo de prioritização aos *stakeholders*.

Como está acima referido, para executar o processo de prioritização de requisitos, é necessário escolher uma técnica de prioritização. Estas são as técnicas de prioritização mais utilizadas (Aurum & Wohlin, 2005; Fernandes & Machado, 2015; Karlsson *et al.*, 1998):

- *Top*-10: Cada *stakeholder* escolhe os 10 requisitos que considera mais importantes;
- *Ranking*: É usada uma escala e os requisitos são ordenados por importância (o mais importante fica em primeiro, o menos importante em último);
- *Grouping*: Os requisitos são colocados em grupos (por exemplo: distribuir os requisitos por: crítico, importante, bom de se ter);
- 100-*unit test*: Cada *stakeholder* distribui 100 pontos pelos requisitos;
- AHP (*analytical hierarchy process*): Os requisitos são colocados numa matriz onde se compara o nível de importância entre requisitos (por exemplo: R1 > R3 e R1 < R2 logo R2 > R1 > R3).

Documentação de Requisitos

Nesta atividade (que também pode ser denominada de especificação de requisitos), espera-se que o produto da engenharia de requisitos seja criado. O produto da engenharia de requisitos pode ser um documento ou vários, escritos unicamente em linguagem natural ou auxiliada de diagramas, com um grande nível de formalidade ou não, dependendo das características do sistema (Fernandes & Machado, 2015).

Pandey, Suman e Ramani (2010), entendem que a atividade de documentação de requisitos é dividida em duas tarefas:

1. Identificação dos requisitos: Atribuição de um identificador único a cada requisito;
2. Especificação de requisitos: O documento de especificação é produzido após os requisitos estarem devidamente identificados.

Para assegurar uma boa estrutura do documento de requisitos, os requisitos devem ser distinguidos pelo seu tipo (Fernandes & Machado, 2015). Se o âmbito do projeto for as tecnologias de informação, o documento de requisitos chama-se "especificação de requisitos de software" (IEEE, 1993). Pressman (2005) sugere que para a realização deste documento, o engenheiro de requisitos deve seguir um padrão. O

IEEE (1998), fez um bom trabalho nesta matéria e padronizou as boas práticas a seguir para redação deste documento.

Validação de Requisitos

A atividade de validação é definida como o processo de avaliar sistemas no fim do seu processo de desenvolvimento para assegurar a conformidade com os requisitos (Boehm, 1984; IEEE, 1990), portanto é discutível se esta pertence à engenharia de requisitos ou à fase de teste (Fernandes & Machado, 2015).

É necessário evitar confusões na diferença entre as atividade de validar e verificar. Enquanto a atividade de validar pressupõe que o projeto está a desenvolver o produto correto, a atividade de verificar questiona se o produto está a ser bem desenvolvido (Boehm, 1984).

O objetivo da validação de requisitos é assegurar que os requisitos definem o sistema desejado pelo cliente (Fernandes & Machado, 2015), são declarados de modo não ambíguo, e as suas inconsistências, omissões e erros são detetados e corrigidos (Pressman, 2005).

Gestão de Requisitos

Se os conceitos estudados até agora causaram uma falta de consenso entre os autores identificados no âmbito desta dissertação, este é ainda mais crítico. Pressman (2005) define gestão de requisitos como um conjunto de atividades que ajudam a equipa de desenvolvimento a identificar, controlar, rastrear e modificar requisitos.

Fernandes e Machado (2015) usam a definição de Pressman (2005) neste contexto. No entanto, acrescentam que a gestão de requisitos é uma atividade paralela a todas as atividade da engenharia de requisitos, e esta inicia no fim da primeira iteração de todas as outras atividades, desde que haja uma mudança dos requisitos ao longo do processo de desenvolvimento.

Segundo van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal e Bijlsma (2006) a gestão de requisitos acarreta as atividades de recolher, identificar e rever os requisitos chegados, e organizar estes para ter esclarecidas as dependências mútuas, ativos principais existentes, linhas de produtos e temas.

Davis (2005) argumenta que a gestão de requisitos é um conjunto de atividades que consiste em recolher requisitos e identificar os corretos, para os satisfazer e documentar. Há 3 subconjuntos importantes de atividades da gestão de requisitos:

- Levantamento de requisitos: recolha de requisitos candidatos de clientes, utilizadores, especialistas do domínio e outros *stakeholders*;

- Triagem de requisitos: determinar quais os requisitos que devem ser satisfeitos quando analisados dentro do contexto dos recursos de desenvolvimento disponíveis, tempo de mercado, objetivos de receita, e retorno de investimento;
- Especificação de requisitos: documentar o comportamento externo do sistema desejado.

Para Cheng e Atlee (2009), a gestão de requisitos é uma atividade "guarda-chuva" que compreende um número de atividade relacionadas com a gestão de projeto ou da fase de requisitos. Estas atividades incluem rastreabilidade, análise de impacto, estimação de custo, gestão de risco e a gestão de variação de requisitos.

Leffingwell e Widrig (2003) referem que a gestão de requisitos é uma abordagem sistemática para levantar, organizar e documentar os requisitos do sistema, e um processo que estabelece e mantêm o acordo entre o cliente e a equipa de projeto na mudança de requisitos do sistema.

Haskings (2006) reivindica que a gestão de requisitos preocupa-se em coletar, analisar e validar os requisitos com todas as comunicações e negociações inerentes ao trabalho com o pessoal.

Wiegers e Beatty (2013) dividem a gestão de requisitos na seguintes atividades:

1. Controlo de versão:
    a) Definir uma identificação de versão;
    b) Rastrear versões de requisitos individuais;
    c) Rastrear versões de conjuntos de requisitos.
2. Controlo de mudança:
    a) Propor mudanças;
    b) Analisar impacto;
    c) Fazer decisões;
    d) Atualizar requisitos individuais;
    e) Atualizar conjuntos de requisitos;
    f) Atualizar planos;
    g) Medir volatilidade de requisitos.
3. Rastreio do estado de requisitos:
    a) Definir estados possíveis dos requisitos;
    b) Guardar o estado de cada requisito;
    c) Rastrear a distribuição do estado de todos os requisitos.
4. Rastreabilidade de requisitos:

a) Definir ligações com outros requisitos;

b) Definir ligações com outros elementos do sistema.

Sommerville (2010) compreende que a gestão de requisitos é o processo de perceber e controlar mudanças nos requisitos do sistema.

O IIBA (2009) alerta para a necessidade da existência da gestão de requisitos em projetos de elevada complexidade, sendo requerido gerir aprovações formais, o patamar e rastreamento de diferentes versões de documentos de requisitos, e rastrear os requisitos desde a origem até à implementação.

Aurum e Wohlin (2005) reconhecem que a gestão de requisitos está preocupada em gerir grandes quantidades de informação relacionadas com os requisitos levantados durante o processo de engenharia de software. A gestão de requisitos inclui, entre outras coisas, manter o rastreio e manutenção da decomposição dos requisitos.

A **gestão de requisitos** é uma atividade de engenharia de requisitos que é paralela a todas as atividades deste domínio. Desde que é feita a primeira iteração do processo de engenharia de requisitos, ou é planeada uma mudança em algum requisito, a gestão de requisitos começa a estabelecer relações com todas as atividades da engenharia de requisitos. É necessário constatar que os requisitos mudam durante fases mais avançadas do ciclo de vida do produto, consequentemente a gestão de requisitos é uma atividade que é iniciada no final da primeira iteração do desenvolvimento dos requisitos e mantêm-se até ao final do ciclo de vida do produto. A gestão de requisitos divide-se nas seguintes atividades:

1. Rastreio de versões de requisitos;
2. Gestão da mudança de requisitos;
3. Rastreio de estados de requisitos;
4. Rastreabilidade de requisitos.

Modelos de referência na gestão de requisitos

Um modelo de referência consiste em um conjunto mínimo de conceitos unidos, axiomas e relações dentro de um programa de domínio particular, e é independente de normas específicas, tecnologias, implementações, ou de outros detalhes concretos (Brown, Metz, & Hamilton, 2005). Dos modelos de referência existentes, no domínio da gestão de requisitos, destacam-se os seguintes:

1. ISO/IEC/IEEE 29148:2011 (ISO/IEC/IEEE, 2011);
2. BABOK (Guide to the Business Analysis Body of Knowledge) (IIBA, 2009);
3. CMMI (Capability Maturity Model – Integration) (CMMI Product Team, 2010);

4. V-Modell XT (Apache, 2006).

Rastreabilidade de requisitos

A rastreabilidade de requisitos permite gravar e seguir o ciclo de vida dos requisitos para frente ou para trás (Fernandes & Machado, 2015; Gotel & Finkelstein, 1993). Campos (2013) identifica 6 tipos de rastreabilidade de requisitos:

1. Rastreabilidade para trás: capacidade de seguir as ligações de rastreabilidade de um requisito de volta às suas origens ou de onde foi derivado;
2. Rastreabilidade para frente: capacidade de rastreabilidade para os requisitos que foram derivados do requisito em consideração;
3. Pré-rastreabilidade: aspetos da existência de um requisito antes de ser incluído na especificação de requisitos;
4. Pós-rastreabilidade: aspetos da existência de um requisito a partir do momento em que foi incluído na especificação de requisitos;
5. Rastreabilidade horizontal: relaciona versões ou variantes do mesmo tipo de informação, por exemplo, entre requisitos ou entre componentes do sistema;
6. Rastreabilidade vertical: rastreia informação entre anteriores e subsequentes fases no processo de desenvolvimento, isto é, entre objetos de informação de diferentes tipos. Por exemplo, uma relação entre o requisito e um elemento da conceção.

Na ilustração 5 está esplícito que a rastreabilidade horizontal e vertical são suportadas pela rastreabilidade para trás e para frente.

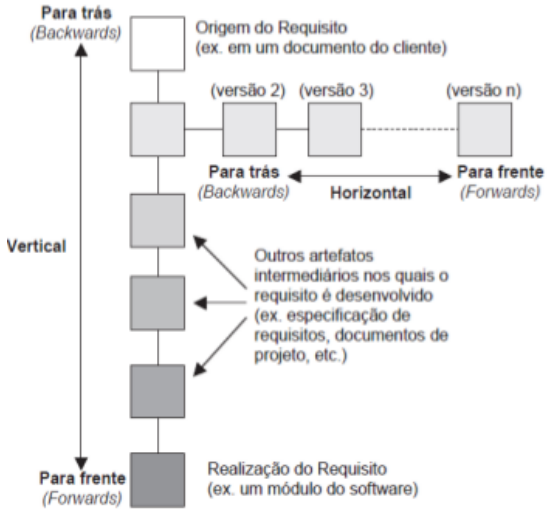


Ilustração 5: Rastreabilidade horizontal e vertical, adaptado de (Campos, 2013)

Interdependências de requisitos

As relações de requisitos, podem ser expressas em interdependências de requisitos (Aurum & Wohlin, 2005; Carlshamre, Sandahl, Lindvall, Regnell, & Dag, 2001).

Carlshamre *et al.* (2001) identificam as seguintes interdependências de requisitos:

- "R1 AND R2": R1 precisa de R2 e R2 precisa de R1 para ambos funcionarem;
- "R1 REQUIRES R2": R1 precisa de R2 para funcionar, mas não se verifica o vice-versa. Se vista do panorama inverso, pode ser denominada também de "is prerequisite of", traduzido para o português como é "pré-requisito de";
- "R1 TEMPORAL R2": R1 foi implementado antes de R2, ou vice-versa;
- "R1 CVALUE R2": R1 afeta o valor de R2 para o cliente positivamente ou negativamente;
- "R1 ICOST R2": R1 afeta o custo de implementação de R2 positivamente ou negativamente;
- "R1 OR R2": Apenas um dos dois requisitos pode ser implementado.

A proposta de Aurum e Wohlin (2005) para a identificação de interdependências de requisitos não é muito diferente da sugestão apresentada anteriormente. Estes autores dividem as interdependências em 3 tipos (ver Ilustração 6):

1. Interdependências estruturais:
    - "Refined_to": Um requisito de negócio é transformado em diversos requisitos de software. Estas interdependências podem ser interpretadas como derivações de requisitos;

- "Changes_to": Um requisito muda para outro requisito se uma nova versão do requisito é desenvolvida e substitui a antiga;
- "Similar_to": Um requisito é parecido com, ou está a sobrepor-se a outro.

2. Interdependências restritivas:
   - "Requires": A realização de um requisito depende da realização de outro;
   - "Conflicts_with": Um requisito está em conflito com outro se eles não poderem existir ao mesmo tempo, ou se aumentando a satisfação de um, diminuir a satisfação de outro.
3. Interdependências de custo/valor:
   - "Increases/Decreases_cost_of": Se um requisito é selecionado para a implementação, então o custo de implementar outro requisito aumenta ou diminui;
   - "Increases/Decreases_value_of": Se um requisito é selecionado para a implementação, então o valor, para o cliente, de outro requisito aumenta ou diminui.

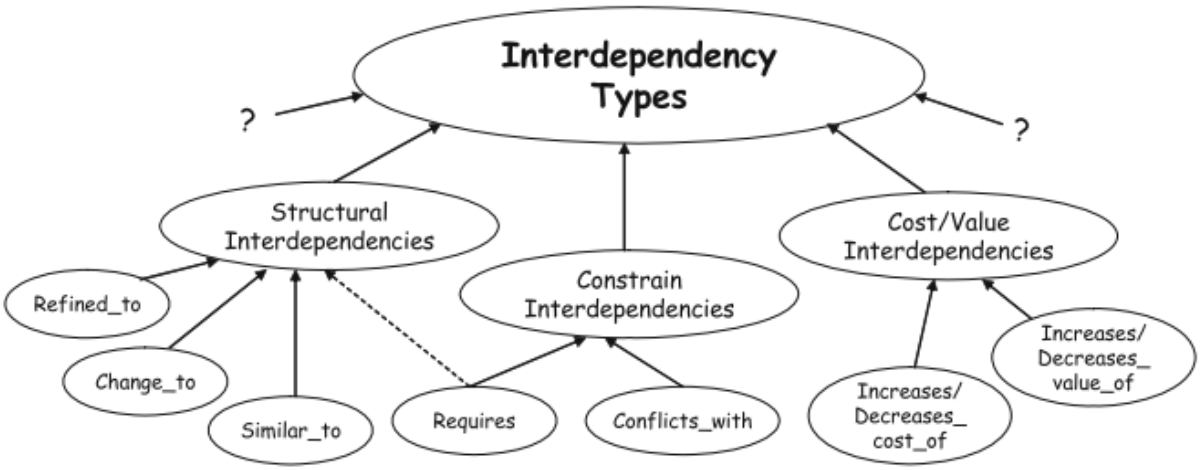


Ilustração 6: Uma classificação do tipo de interdependências, adaptado de (Aurum & Wohlin, 2005)

Gestão de mudança de requisitos

Se fosse possível criar um conjunto de requisitos para um sistema uma vez e apenas uma vez, a vida seria muito mais fácil, e não seria necessário esta secção (Leffingwell & Widrig, 2003). Porém, uma mudança não é má, é algo necessário (Wiegers & Beatty, 2013). O processo da gestão de requisitos só pode ser útil se ele reconhecer e incorporar as questões de mudança(Leffingwell & Widrig, 2003).

Há diversas razões para a inevitabilidade de mudanças nos requisitos. Muitas das razões são fatores internos e podem estar sobre o controlo da organização, contudo há fatores externos que estão fora do controlo dos desenvolvedores e dos utilizadores (Leffingwell & Widrig, 2003).

Fatores externos normalmente estão associados ao aparecimento de uma solução mais eficiente, ou a mudanças no mercado. Fatores internos resultam em erros relacionados com o levantamento de requisitos, por exemplo, os requisitos serem levantados de falsas fontes de requisitos e a solução não resolver o problema do utilizador (Leffingwell & Widrig, 2003).

O processo da gestão de mudanças de requisitos é constituído pelas seguintes atividades (Wiegers & Beatty, 2013):

1. Descrição do propósito e âmbito;
2. Atribuição de papéis e responsabilidades;
3. Atualização do estado do pedido de mudança;
4. Introdução do critério;
5. Realização das tarefas para a mudança de requisitos:
    a) Avaliação do pedido de mudança;
    b) Tomar uma decisão de mudança;
    c) Implementação da mudança;
    d) Verificação da mudança.
6. Encerramento do critério (quando o estado do pedido é rejeitado, fechado, ou cancelado);
7. Relatar o estado da gestão da mudança.

A tarefa de "avaliação do pedido de mudança" precede de um procedimento para analisar o impacto da mudança (Wiegers & Beatty, 2013). Aurum & Wohlin (2005) definem a análise de impacto como "a atividade de identificar as potenciais consequências, incluindo efeitos colaterais e efeitos cascata, para uma mudança, ou estimar o que precisa de ser modificado para realizar uma mudança antes de esta ser efetuada". Já Wiegers & Beatty (2013) afirmam que a análise de impacto envolve três passos:

1. Perceber as implicações possíveis de fazer uma mudança. Uma mudança num requisito produz com frequência um grande efeito cascata, conduzindo a modificações em outros requisitos, arquiteturas, conceções, código, e testes. Mudanças podem conduzir a conflitos com outros requisitos ou podem comprometer atributos de qualidade, como o desempenho ou segurança;
2. Identificar todos os requisitos, ficheiros, modelos, e documentos que poderiam ter de ser modificados se a equipa incorporar uma mudança pedida;
3. Identificar as tarefas pedidas para implementar a mudança, e estimar o esforço necessário para completar essas tarefas.

Reutilização de Requisitos

A reutilização de requisitos é uma sub-disciplina da engenharia de requisitos onde é frequentemente inserida na gestão de requisitos, que nos últimos anos tem sido abordada por diferentes autores.

Wiegers & Beatty (2013) argumentam que a "reutilização é um objetivo eterno para aqueles que procuram aumentar a produtividade de software. O pessoal pensa com mais frequência em termos de reutilização de código, mas muitos outros componentes de projetos de software também devem ter potencial de reutilização. Reutilizar requisitos pode aumentar a produtividade e aumentar a qualidade, assim como conduzir a uma maior consistência entre os sistemas relacionados".

O IIBA (2009) entende que o propósito da reutilização dos requisitos é "aumentar a eficiência do desenvolvimento e implementação de software, e de melhorias na solução depois do desenvolvimento, reutilizando requisitos existentes".

A reutilização de requisitos tem três dimensões (Wiegers & Beatty, 2013):

1. Dimensão da reutilização: Determina o que é possível reutilizar num determinado cenário. Um cenário ideal será reutilizar o pacote completo de requisitos, modelos, componentes da conceção, código e testes. Contudo, na maioria dos casos é preciso proceder a algumas alterações no conteúdo existente;
2. Dimensão da modificação: Determina a modificação que será necessária para tornar os requisitos existentes reutilizáveis no novo projeto;
3. Mecanismo de reutilização: Determina que mecanismo irá suportar a reutilização de requisitos. O mecanismo de reutilização pode ir desde ao rudimentar "copiar – colar" de outras especificações à criação de um repositório de requisitos, podendo ser uma folha de cálculo, uma base de dados ou uma ferramenta especializada.

Os sistemas substitutos ou sujeitos à reengenharia reutilizam sempre requisitos da incarnação original, mesmo se estes "requisitos" nunca foram escritos. A engenharia reversa permite extrair informação de sistemas velhos para reutilização, criando assim uma abordagem alternativa de reutilização de requisitos (Wiegers & Beatty, 2013).

Wiegers & Beatty (2013) identificam barreiras e fatores de sucesso para a reutilização de requisitos. Estas são as barreiras à reutilização de requisitos:

- Maus requisitos ou em falta: A barreira mais frequente à reutilização de requisitos são requisitos de projetos prévios não documentados, mal escritos, incompletos, ou mal ajustados para circunstâncias atuais;

- "Síndromes de NIH e NAH": A síndrome de NIH, sigla para "not invented here", traduzido para "não inventado aqui", está relacionado com reutilizar requisitos de outras organizações, o que resulta na possibilidade de ser difícil o seu entendimento. A síndrome de NAH, sigla para "not applicable here", traduzido para "não aplicável aqui", está relacionado com a resistência à aplicação de um novo processo ou abordagem na organização;
- Estilo de escrita: Requisitos escritos em linguagem natural estão, por norma, cheios de ambiguidades, informação perdida e suposições escondidas, o que resulta numa redução do potencial de reutilização;
- Organização inconsistente: Os autores dos requisitos organizam os requisitos os seus requisitos em muitos sítios;
- Tipo de projeto: Os requisitos estão demasiado dependentes do projeto concedente;
- Propriedade: Se os requisitos estiverem associados a um produto feito para um cliente específico, a propriedade intelectual destes pertence ao cliente. O engenheiro de requisitos pode não ter os direitos legais de reutilizar estes requisitos nestas condições.

Já os fatores de sucesso para a reutilização de requisitos são:

- Repositório: Ninguém pode reutilizar algo que não pode encontrar. Uma ferramenta que impulsiona a reutilização de requisitos é um repositório pesquisável onde se podem alocar a informação dos requisitos. Um repositório pode ter a seguintes formas:
  - Uma pasta em rede que contem documentos de requisitos prévios;
  - Uma coleção de requisitos alocados numa ferramenta de gestão de requisitos;
  - Uma base de dados que aloca conjuntos de requisitos selecionados de projetos prévios.
- Qualidade: Ninguém quer reutilizar lixo. Reutilizadores precisam de confiança na qualidade da informação alocada;
- Interações: Quando se seleciona um requisito para reutilizar, é necessário estar consciente das suas dependências;
- Terminologia: Utilizar sempre os mesmos termos técnicos aumenta o potencial de reutilização;
- Cultura organizacional: A organização pode contribuir para encorajar a reutilização de duas perspetivas: contribuir com componentes de alta qualidade com potencial de reutilização real, e reutilizar eficazmente artefactos existentes.

2.3.3 Sumário

A engenharia de requisitos pode ser definida como uma disciplina transversal a diversas áreas, desde que estas tenham o propósito de desenvolver sistemas. Pode ser também um processo que permite levantar, negociar, documentar e gerir requisitos ao longo do ciclo de vida do sistema.

As atividades que constituem o processo de engenharia de software são:

1. Início do processo: Perceção e descrição do âmbito;
2. Levantamento de requisitos: Fase onde o engenheiro de requisitos faz o levantamento de requisitos organizadamente;
3. Desenvolvimento de requisitos: Atividade que foca-se em analisar e classificar os requisitos levantados na atividade anterior;
4. Negociação de requisitos: Abordagem iterativa para eliminar, combinar ou modificar requisitos de modo a atingir um grau de satisfação dos intervenientes;
5. Documentação de requisitos: O documento é o produto final da engenharia de requisitos. Este documento deve alinhar modelos com linguagem natural;
6. Validação de requisitos: Revisão do documento produzido de modo a evitar ambiguidades neste;
7. Gestão de requisitos: Conjunto de atividades que ajudam a equipa de desenvolvimento a identificar, controlar e rastrear requisitos e modificações nos requisitos.

2.4 Ciclos de Vida

Neste sub-capítulo percebe-se o ciclo de vida de instâncias relacionadas com o tema desta dissertação de mestrado. Primeiro são sugeridas as fases de um ciclo de vida genérico para um conceito que pode ser aplicável a diversos contextos, ou seja, o produto. De seguida sugerem-se modelos aceitáveis para ciclos de vida de sistemas e software. Por fim, são propostos dois ciclos de vida para requisitos.

2.4.1 Ciclo de Vida do Produto

O ciclo de vida do produto é um conceito que descreve como os produtos progridem desde a introdução no mercado até a sua obsolescência (Blythe, 2005). Poli e Cook (1969) remetem este conceito para um modelo dependente do tempo, previsível a médio-prazo, baseado numa analogia biológica inepta.

A teoria diz que produtos, como as instâncias vivas, têm um início natural com uma introdução, passando depois para a fase de evolução, alcançando a maturidade, caindo no declínio e por fim, tornando-se obsoletas (Blythe, 2005) (ver ilustração 7[11]).

Na fase de introdução, as vendas do produto aumentam lentamente, e o lucro é pequeno ou negativo porque é muito novo, há também a necessidade de persuadir retalhistas e outros para armazenar o produto (Blythe, 2005).

Na fase de evolução, há um aumento rápido nas vendas devido ao produto começar a ser mais conhecido. Nesta fase os lucros começam a aumentar, mas a competição entra no mercado, então o produtor pode agora necessitar de pensar sobre adaptar o produto ao conhecimento da ameaça competitiva (Blythe, 2005).

Na fase de maturidade, o produto é bem conhecido e está bem estabelecido. Neste ponto, os gastos promocionais são facilmente despendidos e economias de produção de escala tornam-se estabelecidas. Por esta altura, competidores estão perto de entrar no mercado, então a organização precisa de desenvolver uma nova versão do produto (Blythe, 2005).

Na fase de declínio, o produto está a perder mercado e rentabilidade rapidamente. Nesta fase, o comerciante tem de decidir se vale a pena suportar a portabilidade do produto por mais um pouco, ou se ele deve ser autorizado a desaparecer. É possível um produto em declínio reviver e ser relançado em um mercado diferente (Blythe, 2005).

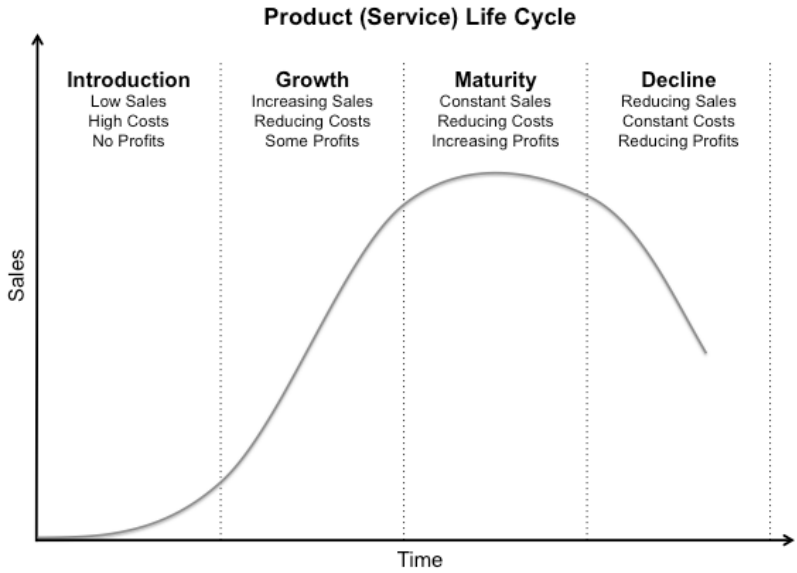


Ilustração 7: Ciclo de vida do produto

11 Ilustração 7 obtida de: https://goo.gl/9sgt9Y

Stark (2011) tem um entendimento diferente de ciclo de vida do produto: (1) Imaginação do produto, (2) Definição do produto, (3) Realização do produto, (4) Uso e apoio ao produto, (5) Aposento do produto. Contudo, nesta dissertação de mestrado, o ciclo de vida do produto proposto por Blythe (2005) é mais adequado.

2.4.2 Ciclo de Vida de Sistemas e Software

A vida de um sistema ou de um software pode ser modelada por um modelo de ciclo de vida por fases (ISO/IEC/IEEE, 2008). Vários tipos ou classes de modelos de ciclo de vida têm vindo a ser descritos. Exemplos deste tipo de modelos são conhecidos por nomes como: (1) cascata, (2) desenvolvimento incremental, (3) desenvolvimento evolucionário, e (4) espiral (ISO/IEC/IEEE, 2008).

Genericamente, a fases de um ciclo de vida de um sistema são: (1) ideia, (2) desenvolvimento, (3) produção, (4) utilização, (5) suporte, e (6) aposento. As fases do ciclo de vida de um software são: (1) desenvolvimento, (2) operação, e (3) manutenção (ISO/IEC/IEEE, 2008).

Todavia, Rajlich e Bennett (2000) argumentam que as fases do ciclo de vida do software incluídas na norma ISO/IEC/IEEE 12207 (2008) são muito alto nível e dividem este nas seguintes fases:

1. Desenvolvimento inicial: Os engenheiros desenvolvem a primeira versão em funcionamento do software;
2. Evolução: Os engenheiros estendem as capacidades e funcionalidades do sistema para conhecer necessidades do utilizador, possivelmente nas direções principais;
3. Manutenção: Os engenheiros fazem reparações de defeitos mínimos e mudanças funcionais simples;
4. Descontinuação: A companhia decide não empreender mais manutenção, procurando a partir deste momento renovar o sistema a maior parte do tempo;
5. Encerramento: A companhia retira o software do mercado e direciona os clientes para um sistema substituível, se existir algum.

Uma variação deste processo, é o modelo faseado por versões evidenciado na ilustração 8, onde a espinha dorsal desta é a evolução (Rajlich & Bennett, 2000). Em certos intervalos, a companhia completa uma versão do seu software e liberta-a para os clientes. A evolução continua, com a companhia eventualmente a lançar o outra versão e apenas dando manutenção à recente.

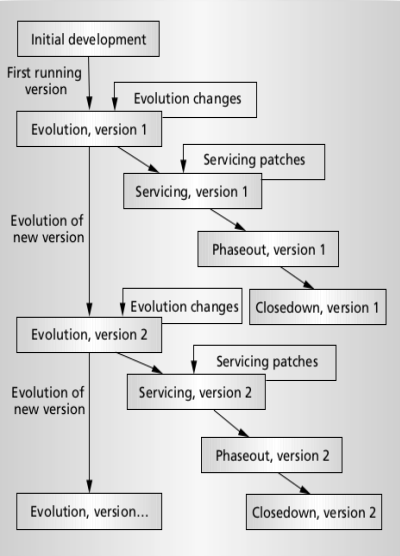


Ilustração 8: Modelo de ciclo de vida de software por versões, adaptado de (Rajlich & Bennett, 2000)

2.4.3 Ciclo de Vida de Requisitos

O ciclo de vida de requisitos ainda não é um conceito muito estudado, sendo assim, é difícil encontrar informação nesta matéria. Contudo, Carlshamre e Regnell (2000) identificam dois modelos para o ciclo de vida dos requisitos. São eles o RDEM (Requirement Driven Evolution Model) e o REPEAT (Requirements Engineering Process At Telelogic).

RDEM

O RDEM divide-se quatro fases (Carlshamre & Regnell, 2000):

1. Capturado: Nada é um requisito até ser capturado;
2. Especificado: Um requisito especificado contém toda a informação necessária para proceder com a implementação e verificação de um constituinte de um sistema. Nesta fase os requisitos estão agrupados em conjuntos denominados de *deltas*. Um delta contém um conjunto específico de atributos;

3. Planeado: Um *delta* para ser implementado ou verificado (realizado no RDEM) necessita de ser planeado;
4. Realizado: Este é o último estado do requisito na terminologia RDEM. Nesta fase o *delta* é transformado numa caraterística de um sistema.

REPEAT

O modelo do ciclo de vida REPEAT para os requisitos está dividido em 6 fases (Carlshamre & Regnell, 2000) (ver ilustração 9):

1. Novo: Este estado representa o estado inicial de um requisito;
2. Atribuído: Um requisito é elevado a atribuído quando um especialista foi atribuído para investigar o requisito;
3. Classificado: Um especialista atribui um valor aos atributos de um requisito;
4. Selecionado: Aqui os requisitos que refletem o produto são selecionados para a implementação;
5. Aplicado: O requisito está implementado;
6. Rejeitado: O requisito é rejeitado, e por isso não está implementado na solução final.

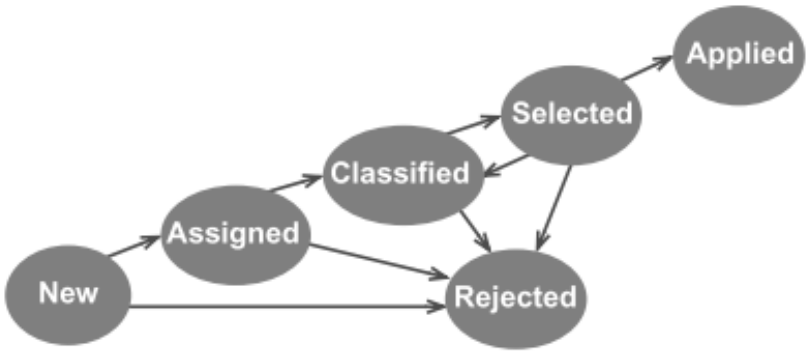


Ilustração 9: Modelo de ciclo de vida REPEAT, adaptado de (Carlshamre & Regnell, 2000)

Tendo em conta que no âmbito desta dissertação de mestrado a gestão de requisitos será feita ao longo de diversos projetos, o modelo REPEAT é o mais adequado.

2.4.4 Sumário

Neste capítulo é referido que existem 3 tipos de ciclos de vida que podem estar relacionados com o conceito de requisito, são eles:

- Ciclo de vida do produto;

- Ciclo de vida do sistema e software;
- Ciclo de vida do requisito.

O ciclo de vida do produto compreende as seguintes fases: (1) introdução, (2) evolução, (3) maturidade e (4) declínio.

Genericamente, o ciclo de vida do sistema é constituído por: (1) ideia, (2) desenvolvimento, (3) produção, (4) utilização, (5) suporte e (6) aposento, já o ciclo de vida do software está organizado em: (1) desenvolvimento inicial, (2) evolução, (3) manutenção, (4) descontinuação e (5) encerramento.

Identificam-se 2 ciclos de vida (RDEM e REPEAT) do requisito e conclui-se que o REPEAT é o mais adequado no âmbito deste trabalho. Este é constituído pelas seguintes fases: (1) novo, (2) atribuído, (3) classificado, (4) selecionado, (5) aplicado e (6) rejeitado.

2.5 Conclusões

Esta revisão de literatura fornece reflexões sobre os elementos que constituem a finalidade desta dissertação de mestrado, dando ao leitor as bases necessárias para interpretar o restante conteúdo deste documento. Em adição, fornece ao autor desta dissertação de mestrado a matéria para a conceção do processo aplicável à Bosch, no capítulo 3, que resolve o problema estabelecido na introdução (capítulo 1).

Desta análise do "estado de arte" conclui-se que, um requisito é uma capacidade que um sistema deve possuir para satisfazer necessidades de utilizadores e pode ser classificado quanto ao nível, como quanto ao seu tipo. Os requisitos podem ser classificados como alto e baixo nível (se a escala for o nível), mas também podem ser classificados como requisitos de utilizador, requisitos de sistema, requisitos funcionais e requisitos não funcionais (se a categoria for o tipo).

A engenharia de requisitos é definida como uma disciplina transversal a diversas áreas, desde que estas tenham o propósito de desenvolver sistemas. Pode também ser considerado um processo que permite levantar, negociar, documentar e gerir requisitos ao longo do ciclo de vida do sistema. As seguintes atividades constituem o processo genérico de engenharia de requisitos: (1) início do processo, (2) levantamento de requisitos, (3) desenvolvimento de requisitos; (4) negociação de requisitos; (5) documentação de requisitos; (6) validação de requisitos; (7) gestão de requisitos.

A gestão de requisitos é frequentemente desprezada na maioria das organizações, porém diversos autores reconhecem grande potencial nesta atividade para aumentar a eficácia e eficiência do processo de engenharia de requisitos. Um grande catalisador para esta atividade ter este potencial é a reutilização de requisitos, devido ao facto de aumentar a produtividade e consistência no desenvolvimento de requisitos.

Dentre os ciclo de vida existentes o ciclo de vida do produto, o ciclo de vida do sistema e software, e o ciclo de vida do requisito são os conciliáveis com a objetividade deste documento. Genericamente, o ciclo de vida do produto compreende as fases de: (1) introdução, (2) evolução, (3) maturidade e (4) declínio. O ciclo de vida do sistema é dividido em: (1) ideia, (2) desenvolvimento, (3) produção, (4) utilização, (5) suporte e (6) aposento, porém o ciclo de vida de um software está organizado em: (1) desenvolvimento inicial, (2) evolução, (3) manutenção, (4) descontinuação e (5) encerramento. O ciclo de vida adequando na abordagem desta dissertação de mestrado é constituído por: (1) novo, (2) atribuído, (3) classificado, (4) selecionado, (5) aplicado e (6) rejeitado.

3 CONCEÇÃO DE UM PROCESSO DE ENGENHARIA DE REQUISITOS

3.1 Introdução

Embora na Bosch *worldwide* já sejam utilizadas algumas ferramentas para o propósito de gestão de requisitos, como é o caso da aplicação DOORS o u Requisite Pro da IBM, no departamento de CI-CWR1 ainda não se faz qualquer esforço para gerir requisitos. Em adição, as ferramentas referenciadas no parágrafo ainda não oferecem funcionalidades que sustentem a gestão do ciclo de vida de requisitos, que é necessária para resolver o problema desta dissertação de mestrado.

Como está esplícito no capítulo 1, a abordagem metodológica utilizada nesta dissertação é a *Design Science Research*. Este método científico sugere a criação de um artefacto, ou seja algum objeto com a contribuição da investigação para a sua conceção (Peffers *et al.*, 2007). Esta contribuição da investigação é frequentemente especificada através de uma revisão de literatura, que nesta dissertação é referente ao anterior capítulo (capítulo 2, os requisitos, engenharia de requisitos e ciclos de vida).

Atendendo que a organização Bosch usa processos para delimitar as funções estabelecidas em cada departamento, o artefacto resultante desta conceção é um processo. Justifica-se esta escolha com a facilidade de alinhar as funcionalidades estipuladas nesta dissertação de mestrado, para resolver o problema, com as atividades já definidas no domínio aplicacional.

Segundo Davenport (1993) um processo pode ser definido como uma ordenação específica de atividades de trabalho ao longo do tempo e espaço, com um início, um fim, e as entradas e saídas claramente identificadas. De modo a complementar esta definição, Møller e Ph (2012) afirmam que um processo pode assumir uma vertente mais administrativa. Por exemplo, certas entidades governamentais podem utilizar um processo para servir as necessidades dos colaboradores de uma determinada organização. As incidências discutidas neste parágrafo sustentam a escolha para o artefacto resultante deste objeto de investigação.

Segundo Browning (2002), uma forma de visualizar processos é através da modelação de processos de negócio. Para modelar processos de negócio, Dufresne e Martin (2003) sugerem diversas metodologias:

1. *Flow Charts*;
2. *Data Flow Diagrams*;
3. *Control Flow Diagrams*;
4. *Functional Flow Block Diagrams*;

5. *Gantt/PERT Diagrams*;
6. *Unified Modeling Language* (UML);
7. *Business Process Modeling Notation* (BPMN).

Visto que no departamento de CI-CWR1, da Bosch Car Multimédia Portugal S.A., constata-se uma grande tendência para uso de UML na atividade de modelação, o autor deste documento opta por utilizar os diagramas de atividade desta notação, numa variante potencializada para os *Work Flow Diagrams* (Chonoles & Schardt, 2003). Esta abordagem deve facilitar a compreensão do processo aplicável à Bosch por parte dos potenciais utilizadores.

Um processo pode ser constituído por (1) atividades, (2) papeis e (3) artefactos (Ramzan & Ikram, 2006). Com base neste facto cria-se a estrutura deste capítulo (os artefactos são remetidos para anexo/apêndice).

No sub-capítulo 3.2, a sequência das atividades, que alinham no processo aplicável à Bosch, é apresentada. Aqui as atividades são divididas em duas componentes do processo: (1) *requirements development* e (2) *requirements management*. Posteriormente, cada atividade é detalhada em tarefas, enquanto se faz uma explicação mais aprofundada. Por fim, a inclusão das atividades (e tarefas adjacentes) é devidamente justificada.

No sub-capítulo 3.3, os papeis necessários à concretização das atividades determinadas no sub-capítulo 3.2 são selecionados. Segue-se uma elucidação destes papeis para se determinem os seus requisitos.

3.2 Descrição do Processo

Este processo é concebido com o propósito de dar as instruções necessárias a atores identificados no seio do departamento CI/CWR1, da organização Bosch Car Multimédia Portugal S.A, para mudar a abordagem atualmente praticada no âmbito de engenharia de requisitos de software.

Estas instruções são dadas sobre a forma de atividades que estão agrupadas em duas componentes: (1) *requirements development* e (2) *requirements management*. Na ilustração 10 pode ter uma visão geral sobre as atividades do processo e as suas interações.

Na componente de *requirements development* alinham as seguintes atividades: (1) *inception*, (2) *elicitation*, (3) *elaboration*, (4) *negotiation*, (5) *prioritization*, (6) *documentation* e (7) *validation*.

A componente de *requirements management* é constituída pelas subsequentes atividades: (1) *lifecycle management*, (2) *traceability*, (3) *allocation* e (4) *change management*.

Este processo é iterativo e a iteração pode mudar a natureza do processo em algumas atividades. Contudo, na primeira iteração o processo começa sempre pela *inception*, de modo de delimitar as variáveis do projeto que se pretende iniciar. Porém, se um projeto não iniciar, sai sempre como produto o *artifact of requirements handling* (ver apêndice I, *artifact of requirements handling*) com descrições relativas ao problema e às possíveis soluções, que devem ser preservadas. Por isso mesmo, este documento deve ser alocado no repositório (ver atividade *allocation*).

Na atividade de *elicitation*, como o próprio nome sugere, os requisitos são levantados das fontes com o auxílio de uma técnica, e escritos no *artifact of requirements handling*, sobre a forma de descrições escritas em linguagem natural.

A *elaboration* é uma atividade que a sua natureza muda com a iteração onde está a ser executada, sendo que na primeira iteração do processo não há hipóteses de reutilizar requisitos do repositório, logo todos os requisitos têm de ser desenvolvidos de raiz, com o auxílio de um padrão para a escrita de requisitos em linguagem natural. Neste atividade detetam-se ainda conflitos peculiares aos requisitos, e resolvem-se aqueles que não têm origem nas vontades dos *stakeholders*.

A atividade *negotiation* é executada se existirem conflitos, relativos aos *stakeholders*, identificados na atividade de *elaboration*. No final desta atividade os conflitos nos requisitos podem estar resolvidos, ou então pode ser necessário levantar novos requisitos, para servir de *input* para a resolução de eventuais conflitos, executando outra vez a atividade.

A componente de *requirements management* inicia com a primeira execução da atividade *life cycle management*. Esta atividade tem a finalidade de atribuir um estado e uma versão aos requisitos, que são manuseados no âmbito do processo aplicável à Bosch, de modo a fazer-se uma gestão do ciclo de vida dos requisitos nestas duas dimensões. Esta funcionalidade é indispensável à reutilização de requisitos na atividade de *elaboration*, para se saber exatamente qual a instância que se está a reutilizar (qual a versão de um determinado requisito).

Na componente de *requirements development*, a atividade *prioritization* é executada. Esta atividade tem o objetivo de identificar o conjunto dos requisitos fundamentais para o sistema de informação, com a ajuda de uma técnica. No final desta atividade, a atividade de *lifecycle management* é executada novamente, uma vez que os estados dos requisitos mudam para selecionado ou rejeitado (ver atividade *life cycle management* de requisitos para visualizar os estados possíveis).

Na atividade *documentation*, os requisitos cujo seu estado é "selecionado", são escritos num documento formal, de obrigatória utilização. No fim o estado destes requisitos é alterado para documentado, então a atividade *lifecycle management* é efetuada mais uma vez.

Na atividade *validation*, atores como o *client*, o *software requirements engineer* e o *project manager* fazem conceções para assinarem o LHPH (ver anexo I, LHPH), validando assim os requisitos presentes neste artefacto. No final desta atividade, como o estado dos requisitos muda para "validado", a atividade *life cycle management* de requisitos é executada novamente.

Na componente de *requirements management*, a atividade *traceability* tem o propósito de determinar relações entre requisitos, de um determinado sistema de informação, com outros requisitos ou instâncias identificáveis de fases avançadas do processo de engenharia de software.

Se o presente projeto é finalizado, a atividade *allocation* é executada e os artefactos desenvolvidos no domínio do projeto concluído são alocados num repositório, de forma a preservar a informação proveniente deste. No final da atividade *allocation*, o processo termina.

Se o projeto não termina, a atividade *change management* é executada. Esta atividade tem o desígnio de prever as peripécias de mudar um requisito num sistema de informação. No entanto, também tem a capacidade de preservar sugestões de mudanças fora do âmbito de um projeto para serem consideradas posteriormente, uma vez que um *change request* pode surgir quando um projeto já não se encontra ativo. Consequentemente, estas sugestões de mudanças devem ser preservadas, alocando o artefacto que preserva a sua essência no repositório.

Uma mudança pode inclusivamente motivar o início de um novo projeto, o que motiva também o aparecimento de um novo sistema de informação e com isto um novo. Se for decidido efetuar uma mudança num sistema de informação, no decorrer de um projeto, é preciso considerar a possibilidade de levantar novos requisitos para completar uma mudança na sua plenitude.

O processo está explanado ao detalhe a baixo, atividade a atividade.

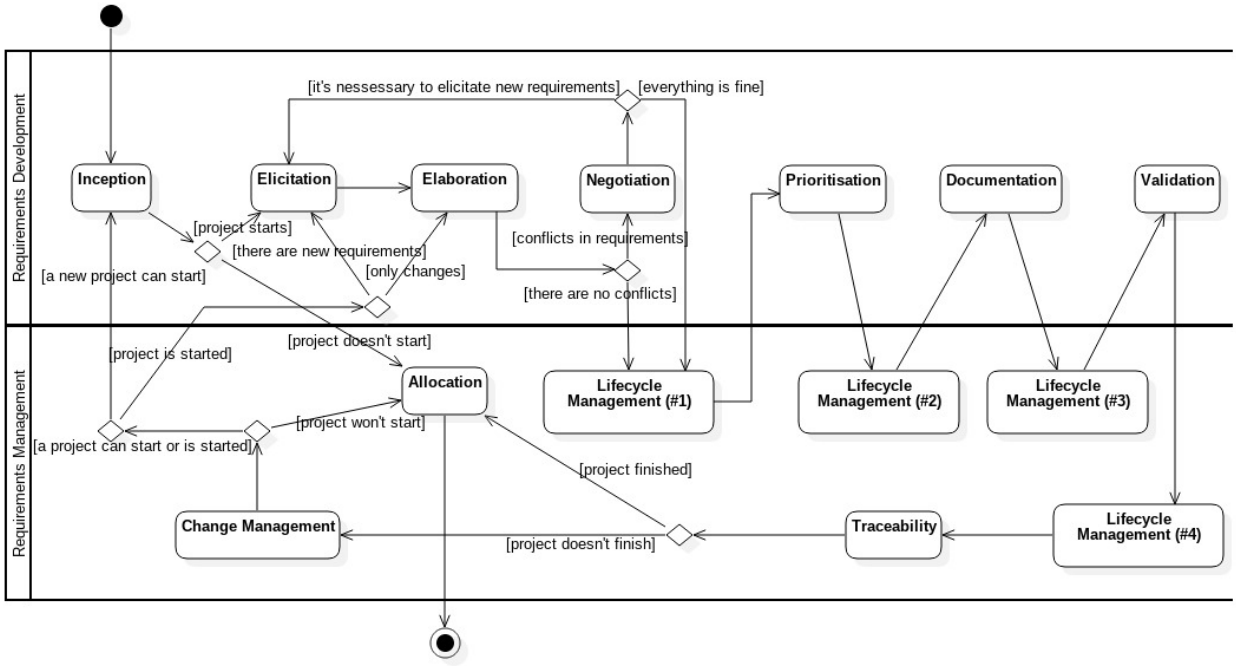


Ilustração 10: Atividades do processo aplicável à Bosch Car Multimédia Portugal S.A.

3.2.1 Descrição de Atividades do Desenvolvimento de Requisitos

a) Inception

Explicação

Esta atividade inicia o processo para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação. Nesta atividade objetiva-se a execução de 5 tarefas (ver ilustração 11):

1. Define the propose of the project: Efetua-se uma reunião com a presença de pelo menos o *client*, o *information systems manager* e um *software requirements engineer*, contudo, outros elementos como o *development team spokesman* ou *users* podem estar presentes (ver capítulo 3.3 para obter mais informação sobre os atores do processo). Nesta reunião o *client* deve expor o que pode motivar um novo projeto;
2. Ask initial questions to the client: Na análise do "estado de arte" estão definidos três grupos de questões com exemplos adequados que devem ser utilizados nesta fase, pelo *software requirements engineer* (ver capítulo 2.3.2, atividade da engenharia de requisitos). Esta tarefa pode ser realizada no seguimento da reunião originada na definição do propósito do projeto, ou

então pode originar uma nova com o mesmo tipo de intervenientes. Porém, o primeiro grupo de questões deve ser utilizado na tarefa anterior.

3. Characterize the problem: Um *software requirements engineer* escreve o problema do *client* em linguagem natural, na área reservada para o efeito no *artifact of requirements handling* (ver apêndice I, *artifact of requirements handling*). As questões para perceber o problema do cliente (ver capítulo 2.3.2, atividades da engenharia de requisitos) tornam-se uma ajuda significativa na realização desta tarefa (Fernandes & Machado, 2015).
4. Set several solutions (with related benefits): No *artifact of requirements handling*, um *software requirements engineer* define diversas ideias de soluções e os benefícios de cada uma. Não se espera que estas soluções sejam infalíveis, mas espera-se que o *client* imagine o seu problema resolvido em todas elas, com maior ou menor eficácia (Fernandes & Machado, 2015).
5. Set viability of the project: O *information system manager* e o *client* percebem se é viável iniciar o projeto considerando variáveis como:
    - Orçamento disponibilizado para projeto;
    - Retorno do investimento feito;
    - Disponibilidade do pessoal;
    - Competências do pessoal;
    - Tecnologias disponibilizadas;
    - Outras restrições impostas pela organização.

Espera-se que esta atividade não seja longa devido à inexistência de um contrato entre o cliente e o fornecedor do sistema (Fernandes & Machado, 2015). Se o projeto iniciar, deve ser destacado um *project manager* (ver capítulo 3.3, atores do processo) e uma equipa de desenvolvimento que o irá acompanhar.

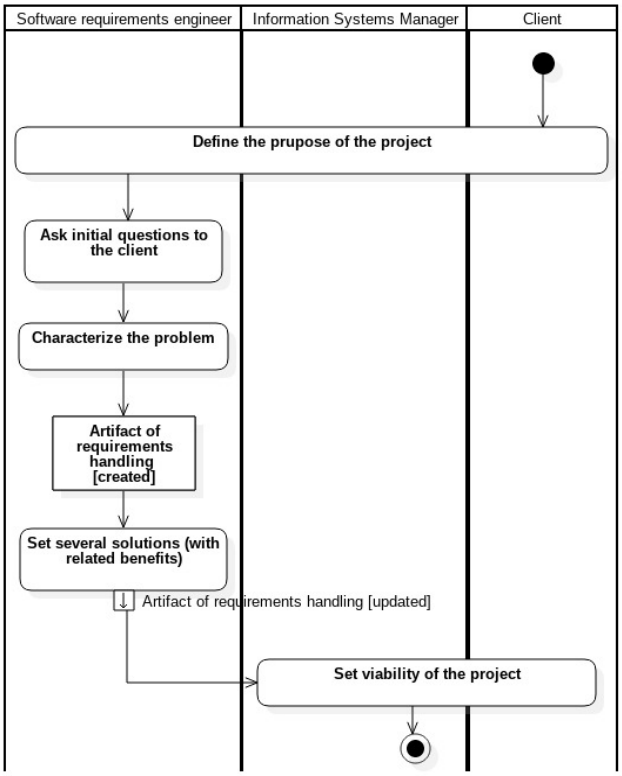


Ilustração 11: Inception

Justificação

Esta atividade dita o início do processo, mas compreende também o início de um projeto para o desenvolvimento de um (ou mais) sistema(s) de informação. Normalmente, um projeto pode nascer com um evento único que se torna num catalisador de um novo sistema, ou então com uma necessidade que evolui com o tempo, o aparecimento do estímulo é imprevisível (Pressman, 2005). É necessário perceber que uma conversa casual pode originar um esforço precipitado na ótica da engenharia de software (Pressman, 2005), portanto a definição do propósito é uma tarefa que não pode ser exequível aceitando imediatamente o possível estímulo. É necessário perceber se o *client* está ciente do foco do estímulo, é necessário entender quem vai utilizar na realidade o(s) sistema(s) que resultará(ão) do projeto, questionando o possível foco do estímulo.

Frequentemente, o *client* concentra-se numa solução desejada sem perceber se esta é a mais eficaz. É por isso que se devem colocar questões ao *client* para que ele se foque no problema, questões estas que ajudarão o *software requirements engineer* a definir o problema que originará um novo projeto. Nestas reuniões iniciais tudo é muito ambíguo, é preciso perceber se a comunicação está a funcionar como é

expectável (Fernandes & Machado, 2015), com isto nasce a necessidade de colocar questões para perceber a efetividade da comunicação dos atores, ou seja, as meta-questões.

O propósito desta atividade é iniciar um projeto. Um projeto inicia quando se assina um contrato que dê garantias tanto para o fornecedor do sistema como para o cliente, então esta atividade constitui um investimento para uma expectativa ser transformada num projeto (Fernandes & Machado, 2015). Para este contrato ser assinado, o problema do *client*, e as soluções propostas pelo fornecedor do sistema têm de ser inferidas por ambos, consequentemente uma descrição do problema e das soluções deve ser documentada no *artifact of requirements handling*. É necessário estar ciente que nesta descrição já devem constar requisitos que não podem ser desprezados e por consequência, levantados na atividade de *elicitation* (ver atividade seguinte), através de uma técnica de levantamento aplicável a artefactos.

Porém, muitas vezes o melhor é que o projeto até nem inicie, é preciso perceber qual é o retorno do investimento feito (Fernandes & Machado, 2015), ou se as restrições impostas pela organização vão permitir a conclusão do projeto. É por isso que o *information systems manager* e o *client* devem perceber se é viável ou não, transformar a expectativa existente em projeto considerando determinadas variáveis.

b) Elicitation

Explicação

O levantamento de requisitos, também designado como descoberta, captura, recolha, aquisição, ou extração de requisitos é uma atividade que deve permitir a um engenheiro de requisitos de software, juntamente das fontes, entender quais são os requisitos de um dado sistema (Fernandes & Machado, 2015).

Levantar requisitos não é uma atividade fácil desde o momento em que envolve o humano, por consequência cada *software requirements engineer* (ver capítulo 3.3, atores do processo) tem a sua forma de levantar requisitos e as suas estratégias para se adaptar a uma determinada situação. Todavia, existem 4 tarefas genéricas que o *software requirements engineer* deve efetuar para levantar requisitos eficazmente (ver ilustração 12):

1. Identify requirements source: Um *software requirements engineer* faz um esforço para perceber onde estão os requisitos que definirão o(s) sistema(s) de informação que resultará(ão) do novo projeto. Na análise do "estado de arte" estão retratadas habituais fontes de requisitos (ver capítulo 2.3.2, atividades da engenharia de requisitos);

2. Choose elicitation techniques: Nesta tarefa, um *software requirements engineer* escolhe as técnicas que vai utilizar consoante as fontes identificadas (existem técnicas para estimular a geração de ideias em pessoas e técnicas aplicáveis em artefactos). Na análise do "estado de arte" estão descritas diversas técnicas para o levantamento de requisitos (ver o capitulo 2.3.2, atividades da engenharia de requisitos);
3. Apply chosen techniques in the requirements sources: Um *software requirements engineer* aplica as técnicas de levantamento escolhidas de forma a levantar os requisitos das fontes selecionadas. No final desta tarefa, os novos requisitos estão escritos em algum apontamento (no âmbito desta dissertação de mestrado, um requisito tem de estar obrigatoriamente escrito, quando este não está escrito é considerado um desejo) e passíveis de serem manuseados;
4. Write requirements or stories resulting from the elicitation: Um *software requirements engineer* escreve os requisitos (descrições em linguagem natural resultantes da aplicação das técnicas de levantamento de requisitos nas fontes de requisitos) no *artifact of requirements handling* (ver apêndice I, *artifact of requirements handling*). Quando se trata de um projeto onde se pretende desenvolver mais do que um sistema de informação, o *software requirements engineer* deve criar um *artifact of requirements handling* para cada sistema de informação.

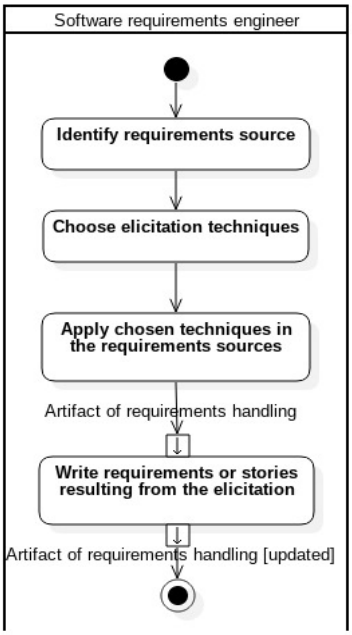


Ilustração 12: Elicitation

Justificação

O levantamento de requisitos é essencial a qualquer projeto de desenvolvimento de sistemas de informação, uma vez que representa a origem dos requisitos, que o cliente quer ver incorporados no sistema desejado (Fernandes & Machado, 2015; IEEE, 2014). Até esta fase, tudo aquilo que o *client* deseja, não pode ser considerado requisitos, pelo que se deve considerar como o nome indica desejos (a menos que estes desejos constem na descrição do problema e soluções proveniente da atividade anterior). Requisitos são capturados com a ajuda de *software requirements engineers*, que conhecem as técnicas apropriadas para transformar adequadamente estes desejos em requisitos. Por outro lado, o requisito é, no âmbito desta dissertação de mestrado, o objeto de estudo, a instância que é submetida às várias atividades das componentes de *requirements development* e *requirements management*. Se os requisitos não são levantados acertadamente, a execução do processo para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação é absolutamente inútil.

O levantamento de requisitos aparenta ser uma atividade simples, contudo levantar requisitos apropriadamente não é assim tão trivial, visto depender muito das competências e experiência dos *software requirements engineers*. Identificar fontes e aplicar técnicas que potenciem a extração de potenciais requisitos são competências que constituem este profissional.

Os requisitos são provenientes de muitas fontes em sistemas de informação típicos, é essencial que todas as possíveis fontes estejam identificadas e avaliadas (IEEE, 2014). Portanto, os *software requirements engineers* devem estar conscientes que se não identificarem corretamente as fontes, muito provavelmente estarão a fazer um esforço para levantar requisitos que não representam o(s) sistema(s) de informação desejado(s).

Os *software requirements engineers*, porém, devem estar sensibilizados para a dificuldade de extrair os requisitos das fontes. Por exemplo, os utilizadores podem ter dificuldade em descrever as suas tarefas, ou podem ocultar informação importante (IEEE, 2014). As técnicas de levantamento de requisitos, quando usadas adequadamente, são uma poderosa ferramenta para a recolha das informações autênticas, logo é importante que os *software requirements engineers* tenham adquirido a essência do maior número de técnicas possível.

Quando as fontes de requisitos estão identificadas e as técnicas para o levantamento de requisitos estão selecionadas, é inevitável os *software requirements engineers* combinarem as duas de forma a levantarem os requisitos, portanto, genericamente a tarefa *apply chosen techniques in the requirements sources* também se podia denominar de *elicitation*, assim como a atividade. Por exemplo, se a fonte

identificada é um *stakeholder* e a técnica selecionada é a entrevista, é necessário agendar a entrevista, estudar o *stakeholder* e preparar a entrevista com questões adequadas para se retirar o máximo desta simbiose.

Do produto desta atividade, resulta um desacordo entre atores, por exemplo o IEEE (2014) afirma que o produto desta atividade são apenas informações que serão depois formalizadas em requisitos pelos *software requirements engineers*, já Fernandes e Machado (2015) argumentam que o produto desta atividade são requisitos ambíguos. Atendendo que há a necessidade de trabalhar o resultado desta atividade, sugere-se a escrita dos requisitos ou informações num documento para posteriormente desenvolver ou cristalizar estes.

c) Elaboration

Explicação

O desenvolvimento de requisitos está relacionado com a organização, cristalização e resolução de conflitos de requisitos. Antes de explicar a atividade, é necessário referir que a entidade "requisito" nesta atividade está envolvida num paradoxo. Um requisito pode ter significados diferentes se estiver a ser tratado em diferentes domínios. Se o requisito estiver a ser manuseado ao nível do requisito, aqui deve ser considerado como uma versão deste, se o requisito estiver a ser tratado no domínio do sistema, então é um requisito, sendo que duas versões diferentes de um requisito ao nível do requisito, são dois requisitos diferentes no domínio do sistema. As seguintes tarefas constituem esta atividade (ver ilustração 13):

1. Identify user requirements: Um *software requirements engineer* identifica os requisitos no(s) *artifact(s) of requirements handling* (ver apêndice I, *artifact of requirements handling*) resultantes da atividade *elicitation* (ver atividade anterior). A identificação de requisitos exige unicidade, logo há a necessidade de colocar atrás de cada requisito de utilizador o seguinte: [Cust.Req.X.VX], onde X $\in$ N* e "VX" representa a versão do requisito que se está a utilizar. O *software requirements engineer* deve também garantir que não existem dois (ou mais) requisitos com o mesmo identificador, ou que o identificador de um determinado requisito não se altera ao longo do tempo. A iteração influencia, porém, a abordagem que o *software requirements engineer* deve adotar. Se este estiver na primeira iteração, todos os requisitos são novos (e a sua versão é igual a 1), logo todos serão identificados sem restrições provenientes da componente de *requirements management* (ver 3.2.2, descrição das atividades da componente de *requirements management*), contudo, se estiver numa iteração posterior, deve primeiro verificar os requisitos que foram

preservados de iterações prévias de forma a dar prioridade primeiro à reutilização de requisitos e segundo à mudança de requisitos. Sempre que ocorrer uma mudança (estado = mudado) num requisito, nasce uma nova versão deste (ver qual foi a última versão na *table of horizontal traceability* (ver apêndice V, *table of horizontal traceability*) para não existirem dois requisitos com o mesmo identificador);

2. Apply sintax patterns in user requirements: Um *software requirements engineer* aplica um padrão de sintaxe, para a escrita dos requisitos de utilizador, em linguagem natural. Fernandes e Machado (2015) sugerem um padrão para a escrita em linguagem natural de requisitos de utilizador: (1) <utilizador>: O tipo de utilizador ou ator associado ao requisito; (2) <funcionalidade>: A funcionalidade que o utilizador desempenha (antes da funcionalidade deve colocar "tem de" quando se trata de uma obrigação, ou "deve" quando é opcional); (3) <objeto/conceito>: A entidade onde a funcionalidade é aplicada; (4) <teste>: Condições impostas ao requisito, facilmente transformadas em não-funcionalidades aplicáveis ao sistema. Exemplo: (1) <O rececionista do hotel> (2) <deve visualizar> (3) <o número do quarto do convidado> (4) <2 minutos após fazer o pedido> (Fernandes & Machado, 2015). Quando um requisito é extenso, o *software requirements engineer* deve dividir este por pontos para eliminar ambiguidades (Fernandes & Machado, 2015);
3. Detect problems in user requirements: Um *software requirements engineer* identifica problemas nos requisitos de utilizador escritos no(s) *artifact(s) of requirements handling*. Estes problemas podem ser relações conflituosas, como: (1) requisitos em sobreposição; (2) requisitos em contradição; ou então falhas em requisitos como: (3) requisitos incoerentes; (4) requisitos incompletos; (5) requisitos vagos (Fernandes & Machado, 2015). No entanto, também podem ser objetivos diferentes de *stakeholders* para um determinado componente, por exemplo uma visão do problema diferente. Este tipo de problemas só se resolve com a negociação de requisitos (ver atividade seguinte). Quando algum destes problemas é detetado, o *software requirements engineer* deve escrever uma descrição deste no *artifact of requirements handling* em comentário (antes da descrição usar o identificador [comment]);
4. Solve problems in user requirements (requirements nature): Um *software requirements engineer* resolve os problemas relacionados com problemas de relações conflituosas e falhas nos requisitos de utilizador;

5. Determine system limits: Um *software requirements engineer* reúne com o *information systems manager* (ver capítulo 3.3, papéis do processo) com o intuito de recolher informações relativas a questões técnicas. Exemplo: Um *user* (ver capítulo 3.3, papéis do processo) pretende a identificação por código de barras de uma determinada instância, o *information systems manager* informa o *software requirements engineer* se a organização possui a tecnologia de leitura de código de barras. No caso de não possuir, este dá um parecer sobre a viabilidade de um investimento na tecnologia, ou então, sugere uma tecnologia idêntica para resolver o problema;
6. Develop system requirements: Um *software requirements engineer* utiliza os requisitos de utilizador (problema) e as informações técnicas provenientes do *information system manager* para construir os requisitos de sistema (solução). O produto desta tarefa deve ser descrições em linguagem natural das funcionalidades e características que a(s) solução(ões) terá(ão) de apresentar para resolver o problema do cliente. Na eventualidade de existirem requisitos de sistema preservados de iterações anteriores, o *software requirements engineer* deve dar prioridade à reutilização ou mudança (nasce nova versão do requisito com a sua mudança) destes, em detrimento do desenvolvimento de novos, de forma a evitar a duplicação de informação. Estas descrições ou requisitos devem estar escritos no(s) *artifact(s) of requirements handling* no final desta tarefa;
7. Identify and classify system requirements: Um *software requirements engineer* identifica os requisitos de sistema desenvolvidos, com o seguinte padrão de identificação: [Prod.Req.X.VX], onde $X \in N^*$ e "VX" corresponde a versão do requisito, de forma a garantir a unicidade destes. Assim como nos requisitos de utilizador, o *software requirements engineer* deve garantir que não existem dois ou mais requisitos com o mesmo identificador, ou que o identificador de um requisito não se altera ao longo do tempo. O *software requirements engineer* deve ainda classificar os requisitos de sistema, ou seja, deve distribuir os requisitos de sistema por 2 classes: requisitos funcionais (ver capítulo 2.2.2, classificação de requisitos) e requisitos não funcionais (ver capítulo 2.2.2, classificação de requisitos). Os requisitos não funcionais podem ainda ser distribuídos por diversas sub-classes (no capítulo 2.2.2, classificação de requisitos, constam 8 sub-classes genéricas de requisitos não funcionais);
8. Apply sintax patterns in system requirements: Um *software requirements engineer* aplica um padrão de sintaxe, para a escrita dos requisitos de sistema, em linguagem natural. Fernandes e Machado (2015) sugerem padrões para a escrita de requisitos de sistema funcionais e não

funcionais. O padrão de escrita para requisitos de sistema funcionais deve ser: (1) <sistema/entidade do sistema>: entidade associada ao requisito; (2) <funcionalidade>: A funcionalidade que o sistema/entidade desempenha (antes da funcionalidade deve colocar "tem de" quando se trata de uma obrigação, ou "deve" quando é opcional); (3) <descrição>: Descrição do evento que motiva a funcionalidade. Exemplo: (1) <O sinal de bateria> (2) <tem de ligar> (3) <quando a bateria está mais fraca do que 20 mAh.> (Fernandes & Machado, 2015). O padrão de escrita para requisitos de sistema não funcionais deve ser: (1) <sistema/entidade do sistema>: entidade que é atribuída uma qualidade; (2) <qualidade>: qualidade do sistema/entidade do sistema. Exemplo: (1) <O produto> (2) <deve ser fácil de utilizar por pessoas resistentes à evolução tecnológica.> (Fernandes & Machado, 2015). Quando um requisito é extenso, o *software requirements engineer* deve dividir por pontos para minimizar ambiguidades (Fernandes & Machado, 2015);

9. Detect problems in system requirements: Igual à tarefa *Detect problems in user requirements*, mas agora aplicado aos requisitos de sistema;
10. Solve problems in system requirements (requirements nature): Igual à tarefa *Solve problems in user requirements (requirements nature)*, mas agora aplicado aos requisitos de sistema.

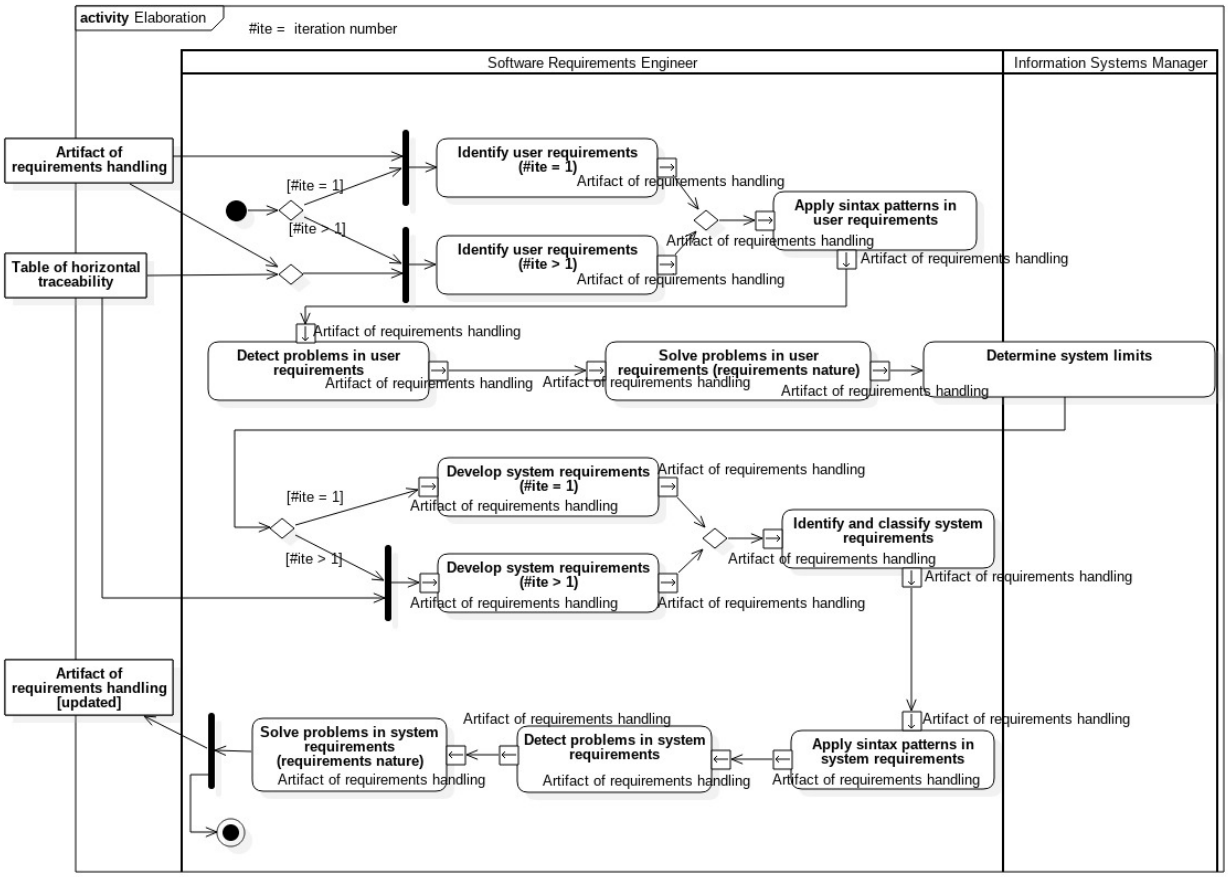


Ilustração 13: Elaboration

Justificação

Como é referido anteriormente, no final do levantamento de requisitos saem descrições que representam uma determinada realidade, no domínio do problema, ou seja, textos extensos que podem conter diversos requisitos. Descrições estas, que são escritas sem grande rigor, no(s) *artifact(s) of requirements handling*. Segundo Fernandes e Machado (2015), esta atividade procura analisar e classificar os requisitos levantados, mas ainda não manuseados. Assim, genericamente, esta atividade está relacionada com manusear os requisitos/descrições escritos no(s) *artifact(s) of requirements handling*, convertendo estes em requisitos cristalizados e livres de inconsistências.

Há uma diferença entre os requisitos que saem da atividade de *elicitation* e os que saem da *elaboration*. Enquanto os requisitos no final da *elicitation* são inutilizáveis no processo aplicável à Bosch Car Multimédia Portugal S.A., os requisitos no final da *elaboration* estão concebidos de forma a serem utilizados na nova abordagem, nomeadamente na componente de *requirements management* (ver Ilustração 10, processo aplicável à bosch car multimédia portugal s.a.).

Com base numa introspeção na organização, estima-se que esta atividade é uma das que gasta mais recursos no processo de engenharia de requisitos, praticado no departamento de CI/CWR1, no entanto, é uma atividade onde se reconhece potencial para melhorar muito a sua eficiência, com a componente de *requirements management*, do processo para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação.

Uma das melhorias que a componente de *requirements management*, do processo para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação oferece é a capacidade de reutilização de requisitos. A reutilização de requisitos eficaz inclui os seguintes benefícios: (1) entregas rápidas; (2) baixos custos de desenvolvimento; (3) consistência entre sistemas de informação; (4) produtividade superior da equipa; (5) menos defeitos; e (6) retrabalho reduzido (Wiegers & Beatty, 2013). Logo, os *software requirements engineers* devem dar prioridade à reutilização de requisitos em detrimento do desenvolvimento de novos.

Segundo Wiegers e Beatty (2013), uma dimensão da reutilização de requisitos é a mudança de requisitos. É mais fácil modificar algo que existe do que criar algo novo (Wiegers & Beatty, 2013), consequentemente, os *software requirements engineers* devem primeiro tentar modificar um requisito alocado no repositório, ou seja, na *table of horizontal traceability* (se não estiver na primeira iteração do processo), de forma a adaptarem este à nova realidade (criando assim uma nova versão deste), ao invés de criarem um novo evitando assim, inclusive, a duplicação de informação.

Internamente à Bosch Car Multimédia Portugal S.A., há um guia para a "otimização da documentação de requisitos" onde é imposto um padrão de identificador para requisitos de utilizador ([Cust.Req.X], onde X $\in$ N*) e um padrão de identificador para requisitos de sistema ([Prod.Req.X], onde X $\in$ N*). Existe a necessidade de adaptar este formato de identificador ao processo aplicável à Bosch, porque o LHPH (ver anexo I, LHPH) é o artefacto para a documentação de requisitos oficial (e de carácter obrigatório) na organização, e este exige que o identificador para os requisitos não se diferencie muito deste modelo.

Esta atividade é essencial para a componente da *requirements management*, nomeadamente na atividade *allocation*, uma vez que requisitos escritos em linguagem natural têm, por norma, muitas ambiguidades, informação perdida e suposições escondidas (Wiegers & Beatty, 2013). Estas questões reduzem o potencial de reutilização de requisitos (Wiegers & Beatty, 2013), assim nasce a necessidade da utilização de um padrão de escrita de requisitos em linguagem natural (Fernandes & Machado, 2015).

É frequente nesta fase os requisitos apresentarem incoerências entre eles, ou até existirem dois ou mais requisitos para o mesmo fim. Por outro lado, os próprios requisitos podem não apresentar toda a

informação que o *client* tentou transpassar, ou podem não fazer sentido no meio onde estão inseridos. É por estas razões que os *software requirements engineers* têm de estar cientes destes problemas e intervirem quando necessário (Fernandes & Machado, 2015).

Como este processo é concebido com o propósito de suportar projetos na área de engenharia de software no seio de uma organização, é necessário perceber que, ora por políticas, ou por falta de orçamento, a organização pode não possuir tecnologia para suportar todos os requisitos de sistema. Com isto surge a necessidade dos *software requirements engineers* adquirirem informação sobre esta componente mais técnica junto do *information systems manager*.

Os requisitos de utilizador frequentemente possuem funcionalidades e não-funcionalidades, que naturalmente são convertidas em requisitos de sistema funcionais e não-funcionais. Os requisitos de sistema devem ser classificados com vista na organização do LHPH.

d) Negotiation

Explicação

Esta atividade tem o propósito de resolver situações onde não há consenso entre os diversos *stakeholders*, relativamente aos requisitos que constituirão o(s) sistema(s) de informação resultante(s) do processo de engenharia de software (Fernandes & Machado, 2015).

Segundo Fernandes e Machado (2015), o processo de negociação está dividido em 3 fases, sendo elas: (1) *pre-negotiation*; (2) *negotiation*; (3) *post-negotiation* (ver Ilustração 14).

As seguintes tarefas constituem a fase de *pre-negotiation*:

1. Elicitate the conflicting objectives: Um *software requirements engineer* (ver capítulo 2.3, atores do processo), recolhe os diferentes objetivos dos *stakeholders,* para o(s) sistema(s) em consideração, que resultam num conflito nos requisitos;
2. Identify and call stakeholders to a meeting: Um *software requirements engineer*, identifica e convoca os *stakeholders* que têm objetivos que resultam em conflitos nos requisitos, para uma reunião.

A seguinte tarefa constitui a fase de *negotiation*:

3. Negotiate requirements: A tarefa *negotiate requirements* reflete-se numa reunião moderada por um *facilitator* (ver capítulo 3.3, atores do processo) onde intervêm os *stakeholders* previamente convocados. Estes *stakeholders* podem ser desde *clients* a *users*, ou a outros interessados que

tenham influência nos requisitos em conflito como o *development team spokesman* (Pressman, 2005) ou até *experts* (ver capítulo 3.3, atores do processo).

Fernandes e Machado (2015) apresentam posturas e estratégias, normalmente adotadas pelos *stakeholders* com objetivos em conflito, na tarefa *negotiate requirements*. Segundo os autores, cinco posturas podem ser adotadas pelos intervenientes, na tarefa *negotiate requirements*:

1. Inatividade: Significa indiferença perante o resultado de uma negociação;
2. Competição: Pressupõe uma ênfase que favorece os interesses de alguém desprezando os dos outros, onde frequentemente se lida com situações ganhar-perder;
3. Acomodação: Implica a satisfação dos outros, sem considerar os próprios interesses. Isto pode significar que um dos participantes está disposto a favorecer os outros, porque ele simplesmente quer a negociação finalizada, ou para assegurar que os outros participantes estão satisfeitos (para reivindicar algo no futuro), ou porque as questões são muito mais relevantes para os outros participantes;
4. Colaboração: Está focada na satisfação dos interesses de todas as partes implicadas, para encontrar alternativas que podem satisfazer todos os *stakeholders*. A ênfase está direcionada para situações ganhar-ganhar;
5. Compromisso: Envolve conceções para encontrar uma solução razoável e equilibrada que deixa todas partes satisfeitas.

Fernandes e Machado (2015), sugerem também 3 estratégias, adotadas pelos participantes, na tarefa de negociar requisitos, de modo a resolver os conflitos (consultar a Tabela 2 para visualizar as características de cada estratégia):

1. Estratégia integrante: Os participantes estão dispostos a cooperar na pesquisa por soluções consensuais;
2. Estratégia distributiva: Os participantes estão apenas preocupados com as suas necessidades e demonstram indiferença para com as dos outros;
3. Estratégia baseada em princípios: Tenta estabelecer um equilíbrio nas vantagens e desvantagens das duas estratégias anteriormente apresentadas.

Tabela 2: Estratégias adotadas numa negociação, adaptado de (Fernandes & Machado, 2015)

Integrante	**Distributiva**	**Baseada em princípios**
Participantes são amigos	Participantes são oponentes	Participantes resolvem problemas
Objetivo é o acordo	Objetivo é a vitória	Objetivo é um resultado racional, alcançado amigavelmente
Faz conceções para promover a relação	Reivindica conceções como uma condição para a relação	Separa as pessoas do problema
Muda a sua posição facilmente	Insiste na sua posição	Foco nos interesses e não nas posições
Encontra uma solução consensual	Força a sua solução	Cria várias soluções possíveis para escolher mais tarde
Insiste no consenso	Insiste na sua opinião	Insiste no uso do objetivo criterioso

Com base nas posturas e estratégias adotadas pelos participantes, a tarefa *negociate requirements* pode ser resolvida de duas formas (Fernandes & Machado, 2015):

1. Consenso ou unanimidade: Requer a aceitação de todas as partes implicadas;
2. Maioria: Prevalece o desejo do grupo mais numeroso.

De forma a apoiar esta tarefa, Aurum e Wohlin (2005) sugerem 4 sistemas:

1. Aspire: Sistema de apoio à negociação pro-ativa que suporta negociações bilaterais;
2. Negoisst: Um sistema de negociação de *business-to-business*[12] eletrónico;
3. EasyWinWin: Um sistema direcionado para negociação de requisitos de software;
4. SmartSettle: Um sistema de apoio à negociação comerciável para negociações complexas;

A fase de post-negotiation reflete-se em um tarefa:

4. Analyse and evaluate the negotiation of requirements: Um *software requirements engineer* analisa e avalia a negociação de requisitos, sugerindo se necessário e possível uma renegociação ou tentando encontrar melhores soluções (Fernandes & Machado, 2015);

Após analisar e avaliar a negociação de requisitos, um *software requirements engineer* deve averiguar se os conflitos nos requisitos, originados nas diferentes visões e objetivos dos *stakeholders* foram resolvidos. Se estes estiverem resolvidos, a atividade termina. Se o *software requirements engineer* constatar o inverso, pode optar por resolver como uma nova iteração da atividade ou então terminar a atividade com o intuito de levantar novos requisitos.

12 Business-to-business é uma expressão utilizada para a denominação do comércio estabelecido entre empresas

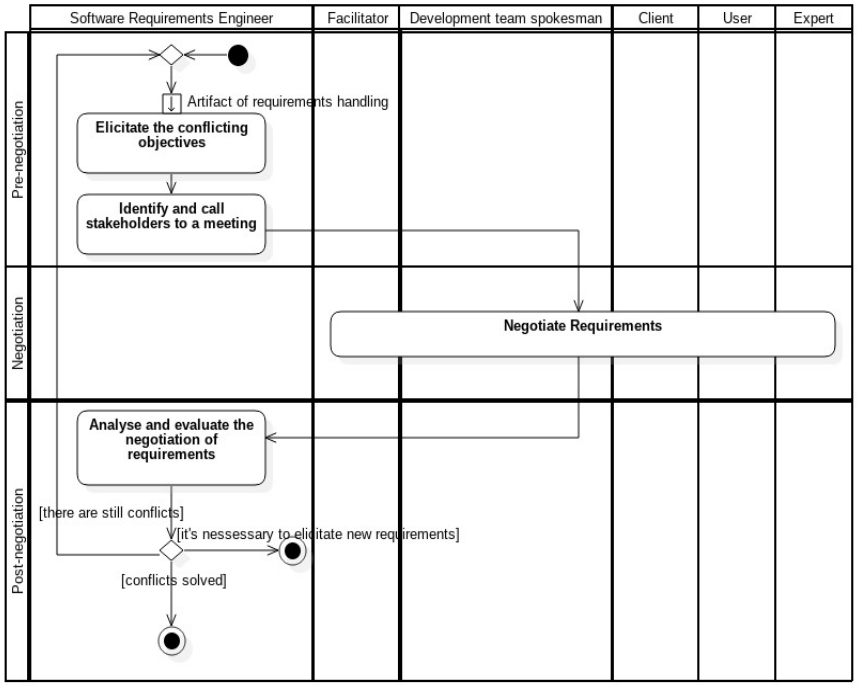


Ilustração 14: Negotiation

Justificação

Os conflitos, surgem na maioria das vezes, no descasamento dos objetivos de *stakeholders* tais como: (1) utilizadores, (2) clientes e (3) desenvolvedores de sistemas de informação (Aurum & Wohlin, 2005). Este descasamento de objetivos dos *stakeholders*, resulta em conflitos nos requisitos, que constituem o(s) sistema(s) de informação, resultante(s) do processo de engenharia de software. Com isto, nasce a necessidade da introdução da atividade de *negotiation*, no processo aplicável à Bosch Car Multimédia Portugal S.A.

Como já foi referido, esta atividade incide na resolução de conflitos nos objetivos, provenientes de *stakeholders*, que resultam em conflitos nos requisitos do(s) sistema(s) de informação resultantes do projeto. A tarefa denominada de *elicitate the conflicting objectives* tem o propósito de recolher estes mesmos objetivos de maneira a obter tópicos passíveis de serem abordados na reunião resultante da tarefa de *negociate requirements*. Fernandes & Machado (2015) e Aurum & Wohlin (2005) sugerem primeiro levantar todos os objetivos de todos os *stakeholders*, de modo a analisar estes, para identificar os conflitos. Porém, no processo aplicável à Bosch Car Multimédia Portugal S.A., é expetável que todos os conflitos nos requisitos sejam identificados na atividade de *elaboration* (ver atividade anterior). Posteriormente, através de uma abordagem *buttom-up* os requisitos conflituosos são transformados em objetivos conflituosos (entenda-se objetivos como os tópicos dos assuntos para discutir na tarefa de *negociate requirements*).

No final da última tarefa, o que é negociável nesta atividade está identificado, contudo falta identificar os responsáveis dos objetivos em conflito, para a tarefa de *negociate requirements*, poder ser efetuada. Segundo Aurum e Wohlin (2005), identificar o pessoal correto pode acelerar a atividade de negociação de requisitos. Após os *stakeholders* estarem corretamente identificados, resta apenas convocar estes para uma reunião agendada numa data viável para todos.

Na tarefa de *negotiate requirements*, uma reunião é instanciada com os *stakeholders* convocados e os objetivos conflituosos levantados. O *facilitator* deve zelar pela resolução dos conflitos por consenso ou maioria, para isso deve perceber que posturas e estratégias são adotadas pelos participantes (Aurum & Wohlin, 2005; Fernandes & Machado, 2015). Esta tarefa, por vezes pode tornar-se complexa, logo, o *falicitator* deve estar consciente da existência de sistemas concebidos para o propósito (Aurum & Wohlin, 2005).

No fim da reunião para a negociação dos requisitos, é necessário perceber se esta foi eficaz, ou assegurar que o compromisso dos *stakeholders* a respeito do acordo alcançado será mantido ao longo do tempo (Fernandes & Machado, 2015), é por isso que o *software requirements engineer* deve analisar e avaliar a negociação de requisitos.

Com base na análise da negociação de requisitos, os *software requirements engineers* devem questionarem-se se os conflitos foram resolvidos. Frequentemente, os conflitos não são resolvidos numa negociação, daí a natureza iterativa desta atividade.

e) Prioritisation

Explicação

A *prioritisation* ajuda os *stakeholders* a identificarem os requisitos fundamentais para o(s) sistema(s) de informação resultante(s) do projeto em execução (Fernandes & Machado, 2015). As seguintes tarefas constituem esta atividade (ver ilustração 15):

1. Select requirements for prioritisation: Um *software requirements engineer* (ver capítulo 3.3, atores do processo) recolhe os requisitos que pretende prioritizar na atual iteração da atividade. Inicialmente, todos os requisitos devem ser prioritizados, devido à necessidade de selecionar os requisitos fundamentais para o(s) sistema(s), entre os existentes. Situações de indefinição podem surgir, onde poderão ser só resolvidas com uma técnica de prioritização diferente. Em iterações avançadas, a tendência é prioritizar um menor conjunto de requisitos para resolver situações de indefinição;

2. Call stakeholders to prioritisation: Um *software requirements engineer* convoca os *stakeholders* para uma reunião. *Stakeholders*, cujo seu papel é neutro na evolução dos requisitos (por exemplo: o *stakeholder* negativo), não devem ser considerados;
3. Select prioritisation technique: Um *software requirements engineer* seleciona uma técnica de prioritização que entende ser ajustável à iteração atual da atividade. O conhecimento do maior número de técnicas de prioritização possível, é uma das competências que os *software requirements engineers* devem ter adquirido. Porém, na análise do "estado de arte" constam as técnicas de prioritização mais populares (ver capítulo 2.3.2, atividades da engenharia de requisitos). Uma técnica pode ser simples para situações onde não há um grande número de indefinições nos requisitos candidatos, ou complexas para contrariar indefinições (Fernandes & Machado, 2015);
4. Organize requirements according to the selected technique: Dependendo do princípio da técnica de prioritização, um *software requirements engineer* deve estruturar os requisitos de maneira que os *stakeholders* apliquem a técnica selecionada. Esta tarefa deve ser executada antes da reunião agendada;
5. Transmit information about selected technique and requirements to prioritise: Na reunião agendada, antes de iniciar a prioritização propriamente dita, um *software requirements engineer*, deve transmitir aos *stakeholders*, informações relativas à técnica de prioritização escolhida, e aos requisitos candidatos, que serão úteis para estes prioritizarem os requisitos adequadamente;
6. Prioritise requirements: A reunião agendada prossegue com os diversos *stakeholders* (ver atores do processo aplicável à Bosch que podem participar nesta tarefa, na ilustração 15) a conduzirem a prioritização de requisitos baseada na informação que foi previamente fornecida (Fernandes & Machado, 2015). Segundo Fernandes e Machado (2015), a tarefa de prioritizar os requisitos precede de um critério e escalas pré-estabelecidos. Vários critérios podem ser considerados na prioritização tendo em vista uma situação específica (Fernandes & Machado, 2015): importância, urgência, utilidade, penalidade, (in)satisfação, tempo de desenvolvimento, custo, volatilidade, e risco. Os critérios podem ser atribuídos com diferentes medidas e escalas. Há quatro tipos de escalas de medidas principais (Fernandes & Machado, 2015):
    a) Escala nominal: Esta escala serve apenas para identificar itens pertencentes ou não a uma categoria. Exemplo: Uma placa-mãe pertence à categoria de informática;

b) Escala ordinal: Permite ver se um objeto é mais importante que um outro, mas sem considerar o quanto. Exemplo: Neste momento a Internet é mais importante que há vinte anos (é um facto que é mais importante, mas é subjetivo o quanto);

c) Escala intervalar: Permite a alguém quantificar as distâncias entre as medidas. Contudo, apenas permite atribuir um significado à diferença entre os números sem os relacionar Exemplo: uma escala onde se inclui todos os critérios referidos;

d) Escala proporcional (ou escala relacional): O quociente entre duas medidas é significativo, independentemente da unidade de medição. Exemplo: Uma escala onde se inclui apenas um critério, para haver uma relação entre as medidas, de modo a haver significado na medição;

7. Apresentar resultados: Por fim, um *software requirements engineer* apresenta os resultados da prioritização. Nestes resultados podem constar os requisitos fundamentais, se a prioritização for bem-sucedida. No entanto, se a prioritização for mal-sucedida, estes resultados devem ser constituídos por uma parte dos requisitos fundamentais e algumas indefinições no requisitos candidatos. Algumas técnicas de prioritização requerem cálculos que devem ser executados antes de apresentar os resultados (Fernandes & Machado, 2015).

Se o produto da atividade de prioritização é os requisitos fundamentais do sistema, então esta termina. Porém, se no final da atividade ainda existem algumas indefinições, relativamente aos requisitos que constituem o(s) sistema(s), uma nova iteração desta atividade deve ser iniciada de forma a atacar estas indefinições com uma técnica de prioritização diferente.

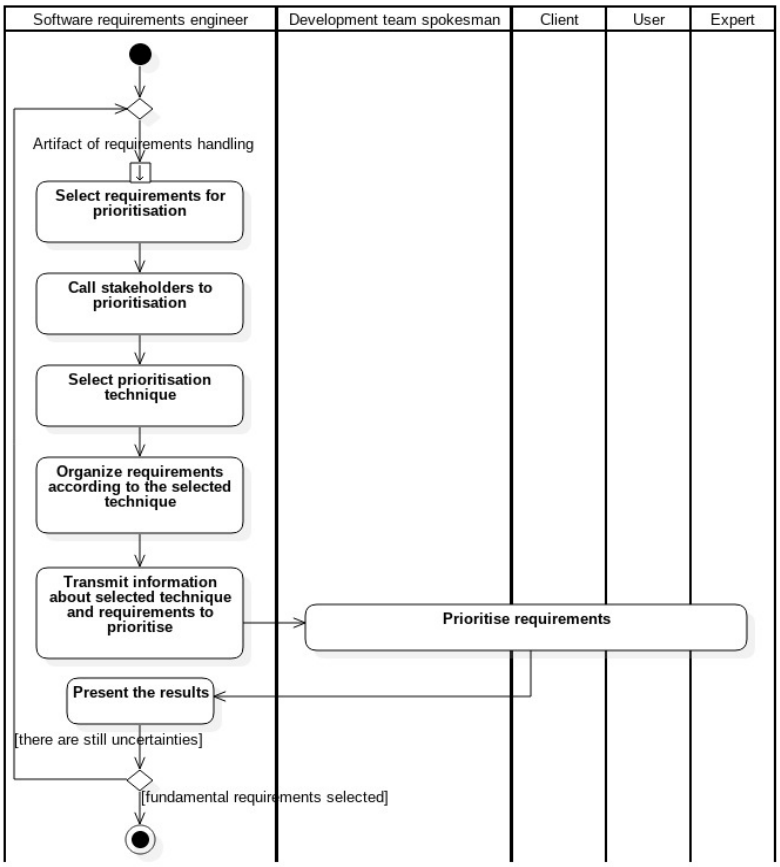


Ilustração 15: Prioritisation

Justificação

Num projeto de um (ou mais) sistema(s), é necessário estabelecer o conjunto dos requisitos candidatos, os requisitos que são passíveis de serem incorporados no(s) sistema(s), devido à sua relevância desde o início (Fernandes & Machado, 2015). Baseado neste conjunto, um subconjunto que inclui os requisitos mais importantes tem de ser selecionado (Fernandes & Machado, 2015). Este subconjunto de requisitos irá constituir o produto(s) mínimo(s) viável(is) (Ries, 2011), ou seja, o(s) produto(s) que irá(ão), pelo menos, satisfazer minimamente todos os *stakeholders*.

Desde a atividade de levantamento, requisitos vão surgindo com a possibilidade de integrar o(s) sistema(s) de informação, resultante(s) do projeto iniciado. Todos os requisitos que foram levantados, e posteriormente desenvolvidos, no âmbito do projeto corrente, devem ser considerados candidatos a constituir o subconjunto de requisitos fundamentais (Fernandes & Machado, 2015). Logo, estes devem ser recolhidos para a prioritização de modo que os *stakeholders* determinem aqueles que constituirão este subconjunto.

A atividade de prioritização de requisitos tem o propósito de ajudar os *stakeholders* a definir a importância dos requisitos candidatos (Aurum & Wohlin, 2005), portanto é necessário juntar estes indivíduos de forma a ser possível prioritizar requisitos com as suas considerações.

Há três instâncias indispensáveis numa sessão de prioritização. Duas delas já foram referidas, sendo elas os requisitos candidatos e os *stakeholders*. Porém, falta uma última de importância crítica, é ela a técnica de prioritização. Uma técnica de prioritização é uma ferramenta para os *stakeholders*, que os ajuda a atribuir um valor aos requisitos candidatos (Fernandes & Machado, 2015). Sem uma técnica de prioritização, os *stakeholders* não saberiam decidir quais os requisitos mais importantes eficazmente.

Como já foi referido, uma técnica de prioritização pode ser simples ou complexa, cada uma com os seus princípios. A técnica pode exigir que o *software requirements engineer* estruture os requisitos adequadamente antes da prioritização. Por exemplo, a técnica AHP (*analytical hierarchy process*) exige que os requisitos sejam colocados numa matriz (Aybuke Aurum & Wohlin, 2005; Fernandes & Machado, 2015; Karlsson *et al.*, 1998).

Num cenário comum, a maioria dos *stakeholders* peculiares ao projeto é leiga na matéria providenciada pela disciplina de engenharia de requisitos. Por consequência, estes também não estão consciencializados das técnicas de prioritização de requisitos, ou provavelmente nunca ouviram falar da atividade de prioritização de requisitos. Desse modo, o *software requirements engineer*, como tem formação suficiente para isso, deve transmitir aos *stakeholders*, informações necessárias, para estes poderem aplicar uma técnica de prioritização nos requisitos candidatos (Fernandes & Machado, 2015). É habitual porém, que os *stakeholders* tenham só conhecimento dos requisitos por si providenciados. Desse modo consequente, o *software requirements engineer*, deve dar uma visão geral dos requisitos candidatos para o(s) sistema(s), antes de começar a tarefa de prioritizar, para os *stakeholders* poderem avaliar o nível de importância de cada requisito.

A palavra critério, na sua natureza, remete para uma regra ou um princípio para avaliar algo. A atividade de prioritização precede de um critério devido a este facto. É necessário um critério para poder comparar a importância dos requisitos candidatos. Para equiparar requisitos, um critério não é suficiente, também é necessário colocar este critério, numa escala, para este ser atribuído com diferentes medidas (Fernandes & Machado, 2015).

Uma sessão de prioritização pode ser bem-sucedida ou mal-sucedida. Isso depende muito da aplicação da técnica, critério e escala na prioritização em questão. Contudo, os resultados provenientes da

iteração, devem ser comunicados aos *stakeholders*, para eles perceberem quais são os requisitos selecionados para o(s) sistema(s), ou as indefinições ainda existentes.

Como a sessão pode ser mal-sucedida, esta atividade possui uma abordagem iterativa, porque esta só termina, quando for encontrado o subconjunto de requisitos fundamentais.

f) Documentation

Explicação

Da *documentation*, deve sair como produto, um documento formal que contém os requisitos, que irão constituir o(s) sistema(s) de informação resultante(s) do projeto. Este documento, é o produto final do trabalho realizado pelos *software requirements engineers*, na componente de *requirements development* (Pressman, 2005). Para esta atividade ser efetuada, duas tarefas devem ser realizadas (ver ilustração 16):

1. Elicitate requirements and descriptions (state = selected): Um *software requirements engineer* (ver capítulo 3.3, atores do processo) recolhe o subconjunto dos requisitos (e suas descrições prévias) fundamentais ao(s) sistema(s) de informação (produto da atividade de *prioritisation*). Na tarefa *create/update state table*, da atividade de *lifecycle management* (ver com maior detalhe no capítulo 3.2.2, descrição das atividades da componente de *requirements management*), deve constatar que o estado do subconjunto dos requisitos fundamentais é "selecionado";
2. Desenvolver LHPH: Um *software requirements engineer* desenvolve o LHPH (ver anexo I, LHPH), documento oficial e de carácter obrigatório, para a especificação de requisitos de software na Bosch (quando se trata de um projeto onde se pretende desenvolver mais do que um sistema de informação, o *software requirements engineer* de criar um LHPH para cada sistema de informação). O LHPH tem uma estrutura que não obriga o *software requirements engineer* a ser estritamente fiel a esta. Há pontos no documento que podem não corresponder às necessidades da documentação de requisitos para o projeto em questão, sendo assim, o *software requirements engineer* deve apenas mencionar, que o ponto em questão, não se aplica nestas circunstâncias. Contudo, este artefacto apresenta uma estrutura projetada por pessoas especializadas na área, não são aconselhadas mudanças de grandes proporções.

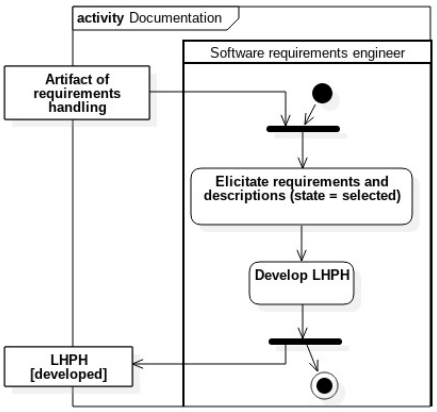


Ilustração 16: Documentation

Justificação

Como está exposto anteriormente, o *artifact of requirements handling* (ver apêndice I, *artifact of requirements handling*), fortemente utilizado nas atividades de levantamento e desenvolvimento de requisitos, contém os requisitos levantados e desenvolvidos, no âmbito da atual iteração do processo aplicável à bosch car multimédia portugal s.a. No entanto, este artefacto para além de ser concebido com o propósito de suportar exclusivamente o trabalho do *software requirements engineer*, contém os requisitos que são selecionados e os que são rejeitados na atividade de prioritização de requisitos. Qualquer processo de engenharia de requisitos contém a atividade de documentação de requisitos, devido à necessidade de formalizar os requisitos que constituem o(s) sistema(s) de informação, documentando-os (Pandey *et al.*, 2010). Ao documentar estes requisitos no documento formal, o *software requirements engineer* está a abrir uma janela para comunicar o seu trabalho a outras entidades.

Os requisitos que possuem o estado "selecionado", são aqueles que pertencem ao subconjunto dos requisitos fundamentais para o(s) sistema(s) de informação. Naturalmente, os *software requirements engineers* necessitam de recolher estes requisitos de forma a poderem formalizar-los.

Pressman (2005) sugere que os *software requirements engineers* devem seguir um padrão, no que toca à documentação de requisitos. Internamente à Bosch, há um artefacto, de carácter obrigatório, e por consequência já utilizado em projetos de engenharia de software da organização, ao qual se deu o nome de LHPH. Como este artefacto é obrigatório para a especificação de requisitos, o *software requirements engineer* tem a obrigação de o desenvolver.

De modo a desambiguar algumas questões, o LHPH compreende pontos onde sugere descrições, do problema, ou de alguma componente do problema. Com isto, recolher as descrições de onde são originados os requisitos torna-se útil.

g) Validation

Explicação

O objetivo desta atividade é assegurar que os requisitos definem o(s) sistema(s) desejado(s) pelo *client* (Fernandes & Machado, 2015). Para isto alguma tarefas devem ser efetuadas por diferentes atores (ver ilustração 17):

1. Verify if requirements are according to the required solution(s): O *client* (ver capítulo 3.3, atores do processo) analisa o(s) LHPH(s) (ver anexo I, LHPH) resultante(s) da atividade de *documentation* com detalhe, e verifica se os requisitos que estão neste especificados, correspondem as características do(s) sistema(s) de informação desejado(s);

Se os requisitos corresponderem à(s) solução(ões) pretendida(s) pelo *client*, a tarefa *sign the LHPH* é executada. Todavia, se o *client* sentir que os requisitos especificados não correspondem à(s) solução(ões) desejada(s), a tarefa *adjust requirements or cut them from the LHPH* é desempenhada.

2. Adjust requirements or cut them from the LHPH: Um *software requirements engineer* (ver capítulo 3.3, atores do processo) verifica que requisitos o *client* considera desajustados. De seguida, deve perceber com este, como resolver este desajuste. Se o *client* entender que algum requisito está completamente desenquadrado daquilo que pretende, retirar o requisito do LHPH resolve o desajuste. Porém, se o *client* entender que o requisito foi mal percebido, ou está incompleto, o *software requirements engineer* tem de alterar este (a versão deste, ao nível do requisito) no *artifact of requirements handling* (ver apêndice I, *artifact of requirements handling*) e no LHPH. No fim desta tarefa, o *client* volta a verificar se os requisitos que constituem o(s) LHPH(s) correspondem à(s) solução(ões) pretendida(s);
3. Sign the LHPH: O *project manager* (ver capítulo 3.3, atores do processo) e o *client* assinam o LHPH e assinam também um acordo que dá garantias ao fornecedor do(s) sistema(s) de informação que a partir deste momento, os requisitos que constituem o(s) LHPH(s) serão os requisitos que constituirão o(s) sistema(s) de informação resultante(s) do projeto. Mudar os requisitos a partir desta fase, resulta numa ação mais ponderada e custosa, já que pode resultar

em mudanças em SLOs (*Software Lifecycle Objects*) (Aurum & Wohlin, 2005), de fases avançadas, do processo de engenharia de software. Esta tarefa dita o fim da iteração atual da componente de *requirements development*.

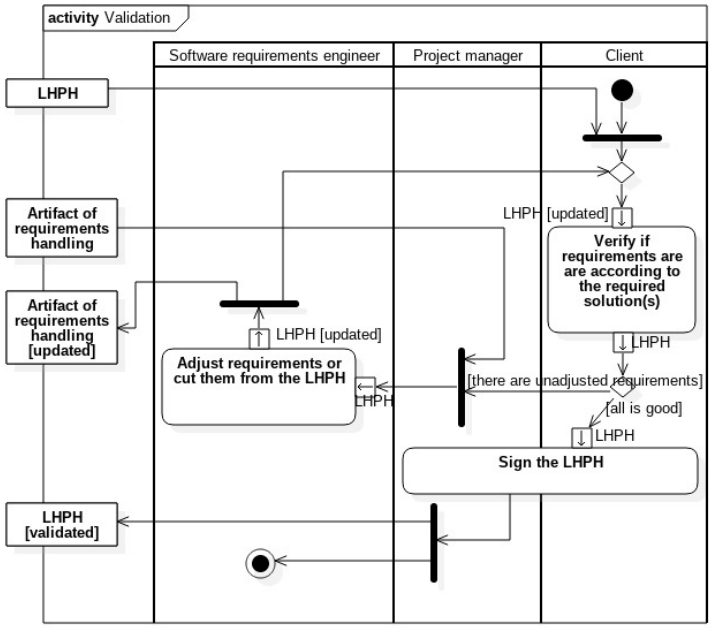


Ilustração 17: Validation

Justificação

A satisfação do *client*, é um aspeto que tem de ser considerado crítico, pelo *project manager*. O(s) sistema(s) de informação até pode(m) estar bem desenvolvido(s), no entanto, se o *client* quando recebe o(s) sistema(s) sente que este(s) não resolve(m) o seu problema, o seu nível de insatisfação será elevado.

Para dar garantias ao *project manager* de que do projeto resultará(ão) o(s) sistema(s) desejado(s), o *client* deve analisar o(s) LHPH(s), documento(s) onde constam os requisitos que irão definir exatamente o(s) sistema(s) que será(ão) posteriormente desenvolvido(s), de forma a comparar o seu desejo com o que está especificado no artefacto.

Da análise feita pelo *client* a um LHPH, podem ocorrer dois desfechos. O *client* concorda com a especificação feita pelo *software requirements engineer* e as duas partes assinam um acordo, ou discorda e obriga o *software requirements engineer* a proceder a algumas alterações, motivando assim nova iteração da atividade. Isto acontece pois é menos custoso mudar apenas requisitos do que mudar estes e as suas derivações em fases posteriores do processo de engenharia de software (Aurum & Wohlin, 2005).

Os requisitos especificados no(s) LHPH(s), não podem ser diferentes dos requisitos incluídos no(s) *artifact(s) of requirements handling*, logo as alterações sugeridas pelo *client*, devem resultar em mudanças nos artefactos criados até então (na componente de *requirements development*).

Sign the LHPH, é uma tarefa que distribui responsabilidades pelo *client* e pelo *project manager*, relativamente ao que está escrito no documento, pois há entendimento comum em relação ao seu conteúdo. Isto dá segurança ao *project manager*, na medida em que o *client* quando recebe o(s) sistema(s) de informação, perde poder de argumentação, se este(s) não corresponder(em) às suas expetativas.

3.2.2 Descrição de Atividades da Gestão de Requisitos

a) Lifecycle Management

Explicação

Esta atividade tem o propósito de auxiliar o *software requirements engineer* (ver capítulo 3.3, atores do processo) a gerir o ciclo de vida dos requisitos no processo aplicável à Bosch.

Três tarefas compõe esta atividade (ver ilustração 18):

1. Elicitate all requirements: Um *software requirements engineer* recolhe todos os requisitos, nas diversas versões (e informação adjacente), do(s) *artifact(s) of requirements handling* (ver apêndice I, *artifact of requirements handling*) desenvolvido(s), no âmbito do processo aplicável à Bosch, com vista a preencher os artefactos *state table* (ver apêndice II, *state table*) e table of horizontal traceability (ver apêndice V, table of horizontal traceability);
2. Create/update state table: Um *software requirements engineer* deve colocar o ID do requisito (sem a versão, usar o seguinte padrão: [Cust.Req.X] ou [Prod.Req.X]) na coluna adequada (sempre que o estado do requisito é novo, surge um novo ID de Requisito). Quando as diversas versões dos requisitos são manuseadas, em atividades do processo para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação, o seu estado muda, o que resulta na execução da atividade de *lifecycle management* (ver ilustração 10). Isto implica que um requisito (ou uma versão deste) pode ter mais do que um estado em diferentes sistemas de informação. Para garantir esta funcionalidade, o *software requirements engineer* deve usar o padrão [nome_do_sistema_de_informação-VX] (onde VX é a versão do requisito e X $\in$ N*), na coluna da *state table* que corresponde ao estado da versão do requisito no sistema de informação coincidente, para identificar o estado das diferentes instâncias. Isto resulta também na

possibilidade de uma coluna correspondente a um estado, ter associado o mesmo requisito em diversas versões, ou até a mesma versão do requisito em diferentes sistemas de informação. O nome do sistema de informação deve ser retirado do *artifact of requirements handling*, onde estão incluídas as versões dos requisitos. Wiegers e Beatty (2013) afirmam que é necessário definir os estados possíveis dos requisitos no rastreio de estados de requisitos. Estes são os estados que um requisito pode possuir:

- Novo: O requisito é recente no sistema (novo ID);
- Reutilizado: O requisito é reaproveitado do repositório sem sofrer qualquer alteração;
- Mudado: O requisito está associado a uma decisão de mudança na atividade de *change management* (implica nova versão do requisito);
- Selecionado: O requisito pertence ao sub-grupo dos requisitos fundamentais de um sistema de informação resultante da atual iteração, estabelecido na atividade de *prioritisation*;
- Rejeitado: O requisito não pertence ao sub-grupo dos requisitos fundamentais de um sistema de informação resultante da atual iteração, estabelecido na atividade de *prioritisation*;
- Documentado: O requisito encontra-se especificado em um LHPH (ver anexo I, LHPH);
- Validado: O *project manager* e o *client* (ver capítulo 3.3, atores do processo) estão de acordo relativamente à validade do requisito para um sistema de informação resultante da atual iteração;
- Implementado: O requisito é rastreável até um SLO proveniente da fase de implementação do processo de engenharia de software;
- Para mudança: Associado a uma futura mudança num sistema de informação, presente na *wainting list of changes* (ver apêndice I, *wainting list of changes*).

3. Create/update table of horizontal traceability: Um *software requirements engineer* deve colocar o ID do requisito (sem a versão, usar o seguinte padrão: [Cust.Req.X] ou [Prod.Req.X]) na coluna adequada (sempre que o estado do requisito é novo, surge um novo ID de Requisito) da *table of horizontal traceability*. Isto resulta na criação ou atualização da *table of horizontal traceability*. Uma atualização na *table of horizontal traceability* é efetuada através de:
   - O aparecimento de um novo requisito no sistema (exige um novo ID de requisito, uma nova descrição e eventualmente uma observação);

- A mudança de um requisito (estado = mudado), resultando no aparecimento de uma nova versão de um requisito (exige uma nova descrição e eventual observação);
- Uma alteração na descrição e/ou observação de uma versão de um requisito.

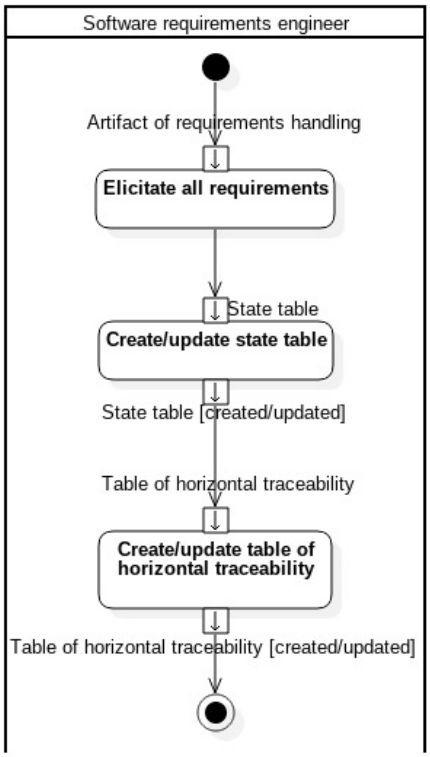


Ilustração 18: Lifecycle management

Justificação

A necessidade de gerir os requisitos independentemente de um projeto ou de um sistema de informação, para potenciar a reutilização de requisitos, resulta na necessidade da inclusão desta atividade no processo para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação.

O ID do requisito, nesta atividade tem especial relevância, porque é a única maneira de identificar um requisito, neste nível holístico, onde os sistemas de informação são só referenciados, e onde se trabalha ao nível dos requisitos ao invés do nível da necessidade do sistema de informação, daí o ID do requisito ser independente da versão, ao contrário do que se sucede em todas as outras atividade do processo aplicável à Bosch.

Segundo Carlshamre e Regnell (2000), o modelo de ciclo de vida de requisitos REPEAT é baseado em transições de estados, resultantes do produto das atividades da engenharia de requisitos. Wiegers & Beatty (2013) também alertam para a necessidade de gerir o ciclo vida dos requisitos por estados. Com base nestes

factos, a tarefa *create/update state table* constitui a atividade com o propósito de ajudar o *software requirements engineer* a gerir o ciclo de vida dos requisitos, ou seja, a identificar qual foi a última atividade do processo aplicável à Bosch (com o apoio do processo de engenharia de software, devido ao estado "implementado"), onde o requisito foi manuseado.

Os estados que um requisito pode ter atribuído, nascem das atividades incluídas (e como estas influenciam cada instância) no processo para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação e no processo de engenharia de software. Na seguinte tabela pode consultar as atividades que originaram os estados no âmbito desta dissertação de mestrado (ver tabela 3).

Tabela 3: Origem dos estados de requisitos

Atividade	**Estado**
Elaboration	Novo, Reutilizado e Mudado
Prioritisation	Selecionado e Rejeitado
Documentation	Documentado
Validation	Validado
Implementation (processo de engenharia de software)	Implementado
Change Management	Para mudança

Wiegers & Beatty (2013) afirmam que controlar as versões *draft* de um requisito ou de um documento ajuda o *software requirements engineer* a reter um histórico de mudanças. Por outro lado, a rastreabilidade horizontal relaciona versões ou variantes do mesmo tipo de informação (Campos, 2013), ou seja, a rastreabilidade horizontal de requisitos precede de um mecanismo que permita a retenção do histórico de mudanças em requisitos. Assim nasce a gestão do ciclo de vida dos requisitos por versões.

Um documento que permita guardar todas as variantes ou versões dos requisitos, torna-se numa tecnologia viável para alocar os requisitos num repositório para reutilização ou mudança destes (Wiegers & Beatty, 2013).

Wiegers & Beatty (2013) referem ainda que para gerir o ciclo de vida dos requisitos por versões é fundamental definir um esquema de identificação de versões. No âmbito desta dissertação entende-se que a geração de uma versão de um requisito, quando este está associado a uma decisão de mudança num sistema de informação (estado = mudado), resulta num esquema de identificação de versões suficiente.

A tarefa de *create/update table of horizontal traceability* envolve características que poderiam ser adaptáveis na próxima atividade, *traceability*. Porém, como aborda o requisitos holisticamente foi inserida nesta atividade.

b) Traceability

Explicação

Como já está referenciado na análise do "estado de arte" a definição mais usada para rastreabilidade de requisitos é "habilidade de descrever e seguir o ciclo de vida de um requisito, para à frente e para trás, idealmente ao longo do todo o ciclo de vida do sistema" (Aurum & Wohlin, 2005; Fernandes & Machado, 2015; Gotel & Finkelstein, 1993). A atividade de rastreabilidade de requisitos no âmbito desta dissertação de mestrado pode ser dividida em (1) *requirements relations* (Aurum & Wohlin, 2005) e (2) *vertical traceability* (Campos, 2013). As seguintes atividades constituem a atividade *traceability* (ver ilustração 20):

1. Elicitate requirements (and additional information) inherent to the information system: Um *software requirements engineer* (ver capítulo 3.3, atores do processo), deve levantar os requisitos peculiares ao sistema de informação ao qual se aplica esta gestão de requisitos. A informações relativas aos requisitos podem ser retiradas do *artifact of requirements handling* (ver apêndice I, *artifact of requirements handling*), para garantir que os artefactos provenientes desta atividade, possuem as informações dos requisitos que foram selecionados e rejeitados, para o sistema de informação e da *state table* (ver apêndice II, *state table*) para se levantarem os estados e eventuais observações dos requisitos;

A componente da identificação de relações de requisitos da atividade de *traceability* divide-se nas seguintes tarefas:

2. Create/update matrix of requirements relations (user - user): Um *software requirements engineer* procede ao preenchimento da *matrix of requirements relations* (ver apêndice IV, *matrix of requirements relations*), com o ID do requisito e a sua descrição escrita em linguagem natural, dos requisitos de utilizador, nas colunas (referentes aos requisitos afetantes) e nas linhas (referentes ao requisitos afetados) desta (pode ser a origem ou uma atualização do artefacto). A célula que corresponde a um "*match*", entre o requisito afetante e o requisito afetado, reflete uma relação entre os requisitos. Numa célula que se denote uma relação, deve ser colocado o tipo de relação correspondente com o seguinte formato "<<tipo_de_relação>>". Os tipo de relações utilizador – utilizador, podem ser representados nas seguintes interdependências (ver tabela 4): (1) <<e>>; (2) <<é_pré-requisito_de>>; (Carlshamre *et al.*, 2001) (3) <<aumenta_valor_de>>; (4) <<diminui_valor_de>>; (5) <<aumenta_custo_de>> e (6) <<diminui_custo_de>> (Aurum &

Wohlin, 2005). Estas relações são impostas por *stakeholders* na atividade de *elicitation*, logo o *software requirements engineer* não pode criar relações nos requisitos (utilizador – utilizador);

3. Create/update matrix of requirements relations (user - system): Um *software requirements engineer* preenche a *matrix of requirements relations*, com o identificador e descrição dos requisitos de utilizador, nas colunas reservadas para as informações dos requisitos afetantes, e com o identificador e descrição dos requisitos de sistema, nas linhas reservadas para as informações relativas aos requisitos afetados (resulta na geração do documento ou na sua atualização). Na célula que corresponde ao "*match*", o *software requirements engineer* deve colocar simplesmente um "x";
4. Create/update matrix of requirements relations (system - system): Um *software requirements engineer* preenche a *matrix of requirements relations*, com o identificador e descrição escrita em linguagem natural, dos requisitos de sistema, nas colunas (referentes aos requisitos afetantes) e linhas (referentes aos requisitos afetados) para o efeito. Assim como nas matrizes previamente desenvolvidas, a célula que corresponde a um "*match*" entre o requisito afetante e o requisito afetado, reflete uma relação entre os requisitos. Numa célula onde se observe uma relação, deve ser colocado o tipo de relação correspondente com o seguinte formato "<<tipo_de_relação>>". As relações sistema – sistema podem ser representadas nas seguintes interdependências (ver tabela 4): (1) <<e>>; (2) <<é_pré-requisito_de>>; (Carlshamre *et al.*, 2001) (3) <<aumenta_valor_de>>; (4) <<diminui_valor_de>>; (5) <<aumenta_custo_de>>, (6) <<diminui_custo_de>> e (7) <<mudou_para>> (Aurum & Wohlin, 2005). Estas relações são identificadas por um *software requirements engineer*, ou são derivadas das relações de requisitos utilizador – utilizador. São estas relações que fornecem a informação necessária aos *software requirements engineers* que medem o impacto de mudança de requisitos na atividade de gestão de mudança de requisitos (Aurum & Wohlin, 2005);
5. Create/update graph (system - system): Um *software requirements engineer* cria ou atualiza um grafo, que represente visualmente as relações de requisitos sistema – sistema, com o auxílio da informação resultante da *matrix of requirements relations (system – system)*, e de uma ferramenta que permita criar diagramas. Sugere-se o "yEd Graph Editor", ferramenta multi-plataforma e de código aberto (consultar a hiperligação https://www.yworks.com/products/yed para descarregar esta). Como pode ver na ilustração 19, os nodos representam os requisitos, as

setas representam as relações (nas setas tem de adicionar uma descrição com o tipo de interdependência, com o formato exposto na tarefa anterior) (Carlshamre *et al.*, 2001);

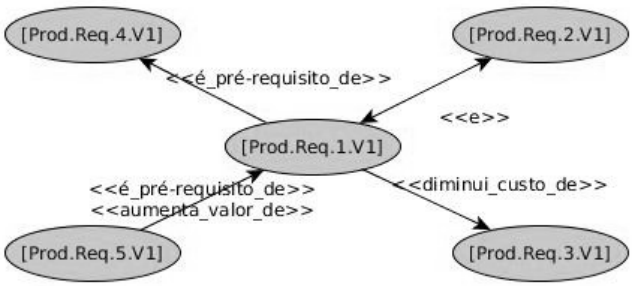


Ilustração 19: Exemplo de grafo

6. Create/update tables of requirements relations: Um *software requirements engineer* preenche as *tables of requirements relations* (ver apêndice III, *tables of requirements relations*) com as informações dos requisitos, referentes às colunas coincidentes (ID requisito, descrição, observações e estado). Isto pode ser o início do documento, ou uma atualização deste. As relações de requisitos devem ser extraídas das matrizes de relações de requisitos, com a intenção de preencher as colunas para esse propósito. O *software requirements engineer*, deve utilizar o seguinte formato "<<tipo_de_relação>>_[Cust.Req.X.VX]" se estiver a tratar relações utilizador – utilizador, ou "<<tipo_de_relação>>_[Prod.Req.X.VX]" se estiver a abordar relações sistema – sistema. Se estiver a referir-se a relações utilizador – sistema, o identificador do requisito serve para o efeito;

Tabela 4: Inferência de interdependências

Interdependência	Descrição	Em conflito com
<<e>>	R1[13] precisa de R2[14] e R2 precisa de R1 para ambos funcionarem	<<é_pré_requisito_de>>
<<é_pré_requisito_de>>	R2 precisa de R1 para funcionar, mas não se verifica o vice-versa	<<e>>
<<aumenta_valor_de>>	R1 afeta o valor de R2 para o cliente positivamente	<<diminui_valor_de>>
<<diminui_valor_de>>	R1 afeta o valor de R2 para o cliente negativamente	<<aumenta_valor_de>>
<<aumenta_custo_de>>	R1 afeta o custo de implementação de R2 positivamente	<<diminui_custo_de>>
<<diminui_custo_de>>	R1 afeta o custo de implementação de R2 negativamente	<<aumenta_custo_de>>
<<mudou_para>>	R2 é uma nova versão de R1	-

A componente de *vertical treceability*, nesta dissertação de mestrado, tem o desígnio de descobrir que instâncias são derivações dos requisitos associados ao sistema de informação. Esta começa com uma verificação, por parte do *software requirements engineer*, na busca da realização de fases posteriores à engenharia de requisitos, no processo de engenharia de software.

Se já foram realizadas fases avançadas do processo de engenharia de requisitos, a seguinte tarefa é efetuada, se não, a atividade de *traceability* é terminada:

7. Elicitate SLOs of advanced phases matching the requirements: os *software requirements engineers* reúnem com o *development team spokesman* (ver capítulo 3.3, atores do processo), de modo a levantar SLOs provenientes das fases de análise, conceção, desenvolvimento e fase de teste do processo de engenharia de software. Todos os SLOs levantados, no decorrer desta atividade, devem ser derivações de requisitos intrínsecos ao sistema de informação. Uma abordagem alternativa pode ser a rastreabilidade baseada em eventos (Campos, 2013; Cleland-Huang, Chang, & Christensen, 2003), onde o *development team spokesman* notifica o *software requirements engineer* quando terminar a sua contribuição para o projeto;

Há a hipótese, de mesmo sendo realizadas fases avançadas do processo de engenharia de software, não serem disponibilizados SLOs, por restrições impostas pela organização, ou por outras questões, então, se

13 R1 corresponde ao requisito afetante

14 R2 corresponde ao requisito afetado

o *software requirements engineer* encontrou SLOs, na tarefa anterior, a seguinte tarefa é efetuada, se não, a atividade de *traceability* termina:

8. Create/update table of vertical traceability: Um *software requirements engineer* deve primeiramente preencher o artefacto *table of vertical traceability* (ver apêndice VI, *table of vertical traceability*) com a informação dos requisitos de sistema recolhida na tarefa de *elicitate requirements (and additional information) ihnerent to the information system* desta atividade (isto pode consistir em criar ou atualizar o artefacto). De seguida, o *software requirements engineer* deve preencher as colunas referentes aos SLOs correspondentes aos requisitos expressos com as instâncias levantadas (ou referência destas) na tarefa de *elicitate SLOs of advanced phases matching the requirements*.

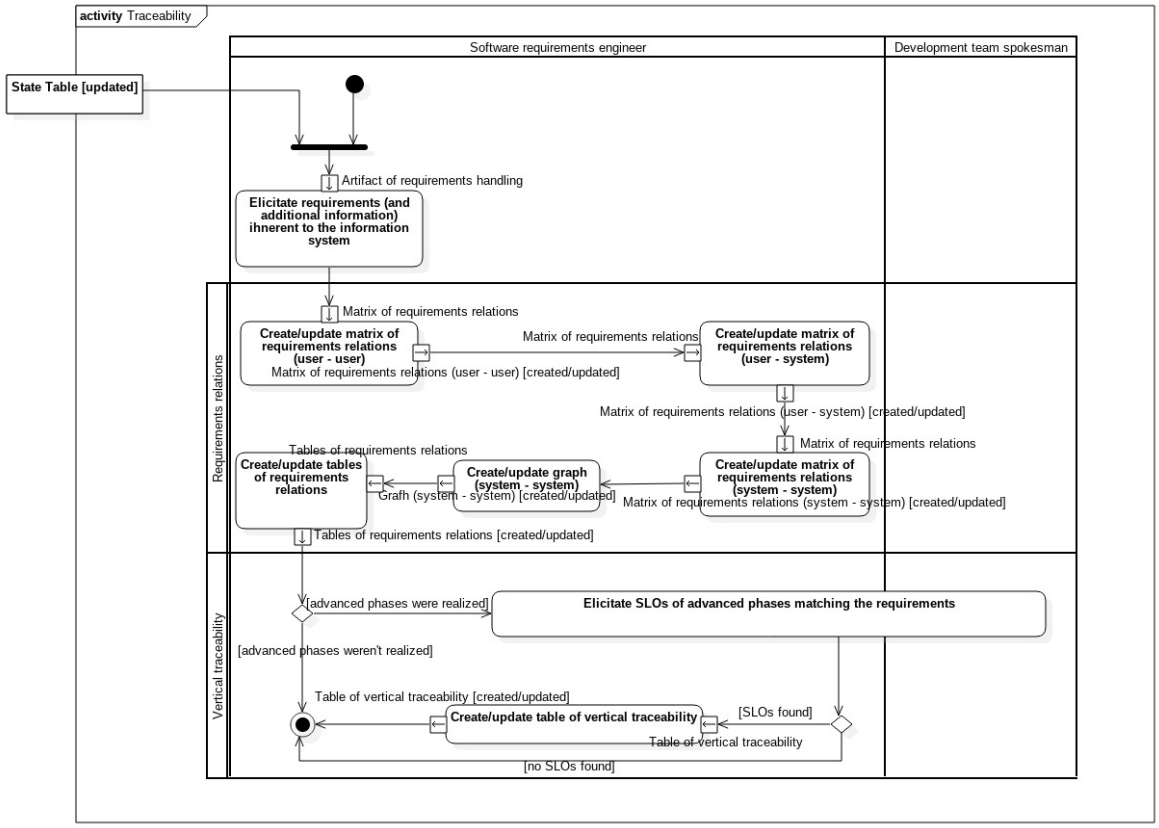


Ilustração 20: Traceability

Justificação

A gestão de requisitos precede claramente da rastreabilidade de requisitos, uma vez que os requisitos estão sempre em constantes mudanças, e são precisos mecanismos para gerir estes contextos de

instabilidade, a fim de avaliar o impacto que estas mudanças nos requisitos podem ter no projeto (Fernandes & Machado, 2015).

A necessidade de rastrear sempre os requisitos que representem o sistema de informação, exige a recolha de todos os requisitos que foram levantados no âmbito deste, antes de se proceder à execução de qualquer componente específica da atividade de *traceability*.

O tópico de relações de requisitos é visto como um aspeto específico de rastreabilidade, desde o momento em que este consiste em associar informação relacionada, de um tipo específico, denominado requisitos (Aurum & Wohlin, 2005).

As relações de requisitos são divididas em três categorias, no âmbito desta dissertação de mestrado, devido à diferença da natureza da origem e área de atuação destas.

As relações de requisitos utilizador – utilizador surgem com a necessidade de guardar imposições, expressas pelo *client*, na atividade de *elicitation*, praticada na Bosch Car Multimédia Portugal S.A. Estas relações ajudam o *software requirements engineer* na atividade de *elaboration*, mais especificamente, na reutilização de requisitos (Aurum & Wohlin, 2005). Quando um *software requirements engineer* deseja reutilizar um requisito, tem logo acesso às precedências deste ou às consequências da sua utilização.

As relações de requisitos utilizador – sistema aparecem apenas com o propósito de representar a derivação dos requisitos de utilizador para os requisitos de sistema. Estas relações são detetadas pelo *software requirements engineer*, e auxiliam este a perceber quais são os requisitos pai e filho nestas transições, posteriormente. Também têm um papel determinante na atividade de *elaboration*, precisamente na reutilização de requisitos (Aurum & Wohlin, 2005), pelos motivos já expostos.

As relações de requisitos sistema – sistema nascem da conveniência em detetar interdependências nos requisitos, que estão mais próximos do(s) produto(s) resultante(s) do projeto em execução. O *software requirements engineer* deteta estas relações, e é importante que detete todas as relações deste tipo, pois um bom trabalho nesta fase, resultará numa gestão de mudança nos requisitos e análise de impacto desempenhadas com maior qualidade (Aurum & Wohlin, 2005). Assim como nas duas categorias de relações de requisitos já discutidas, estas relações também são essenciais na reutilização de requisitos (Aurum & Wohlin, 2005). Como o processo aplicável à Bosch suporta rastreabilidade vertical de requisitos, esta categoria de relações também é determinante na reutilização de SLOs (Aurum & Wohlin, 2005), em fases avançadas do processo de engenharia de software, quando se procede à reutilização de requisitos.

As interdependências de requisitos propostas, para as relações de requisitos utilizador – utilizador e sistema – sistema, nascem de uma síntese das interdependências apresentadas por Carlshamre *et al*. (2001)

e Aurum & Wohlin (2005), adequando estas às necessidades do processo adaptável à Bosch. Todas as interdependências (sugeridas pelos autores) rejeitadas, são desprezáveis, porque não têm qualquer utilidade no contexto onde seriam inseridas. As relações de requisitos utilizador – sistema, representam derivações, logo a interdependência "*Refined_to*" (Aurum & Wohlin, 2005) é considerada apropriada. Na identificação de relações de requisitos utilizador – sistema, na matriz para o efeito, utiliza-se um "x" devido à existência de apenas uma interdependência nesta categoria.

A identificação de relações de requisitos não é, por norma, uma atividade trivial. Para auxiliar esta, o *software requirements engineer* utiliza as matrizes, porque estas permitem uma visão minuciosa e comparativa dos requisitos através de linhas e colunas, facilitando assim a identificação de relações. Como as relações de requisitos sistema – sistema têm um papel na medição do impacto de mudança, a quantidade e variedade de relações desta categoria deve aumentar, logo um grafo, que represente exatamente a matriz de relações sistema – sistema, torna-se uma ajuda significativa na análise do impacto de mudança, uma vez que dá uma perspetiva visual sobre as relações muito mais atrativa. As tabelas permitem identificar com maior facilidade as relações de um requisito, quando se pretende por exemplo reutilizar este. Pode ser considerado um documento de oficialização das relações de requisitos, um documento que não auxilia o *software requirements engineer* na identificação de relações de requisitos, mas que lhe facilita a vida em atividades dependentes destas.

A imprescindibilidade de relacionar os requisitos implementados, do sistema de informação com SLOs que são o reflexo destes, fundamenta a incorporação da componente de *vertical traceability* nesta atividade. É preciso estar consciente que o ideal é um requisito ser sempre transformado em algo "palpável", e os requisitos mudam normalmente por deteções de limitações no produto (Aurum & Wohlin, 2005). O relacionamento dos SLOs, nas diversas fases facilita a tarefa de medição do impacto de mudanças nos requisitos, porque mudar um requisito, implica mudar todas as instâncias provenientes deste.

A rastreabilidade vertical não é restrita só à engenharia de requisitos, e requer que fases do processo de engenharia de software sejam efetuadas, como por exemplo a conceção, o desenvolvimento ou fase de testes. Então é lógico que a tarefa *create/update table of vertical traceability*, precede da execução destas fases.

No departamento de CI/CWR1 da Bosch Car Multimédia Portugal S.A., constata-se que abordagens orientadas para o desenvolvimento ágil são desprezadas, isto resulta num artefacto para a rastreabilidade vertical estruturado e preparado exclusivamente para o desenvolvimento em cascata.

c) Allocation

Explicação

A *allocation*, permite a um *software requirements engineer*, guardar todos requisitos que levantou e manuseou num projeto, para poder reutilizar estes num sistema de informação diferente, com características idênticas. As seguintes tarefas compõe a atividade de *allocation* (ver ilustração 21):

1. Elicitate the artifacts resulting of project: Um *software requirements engineer* recolhe todos os artefactos desenvolvidos no âmbito do projeto finalizado;
2. Allocate artifacts in the O:/: Um *software requirements engineer* guarda os artefactos resultantes do projeto finalizado na diretoria O:/. Estes artefactos, podem ser distribuídos por dois domínios, associados ao nível a que são tratados os requisitos constituintes. No domínio do sistema de informação inserem-se os seguintes artefactos:
   a) *Artifact of requirements handling*;
   b) LHPH;
   c) *Table of requirements relations*;
   d) *Matrices of requirements relations* ;
   e) *Graph (system - system)*;
   f) *Table of vertical traceability*.

   Já no domínio dos requisitos atuam os seguintes artefactos:
   a) *State table*;
   b) *Table of horizontal traceability*.

   Os artefactos que estão associados ao domínio dos requisitos são tratados de maneira diferente, uma vez que não estão inseridos no âmbito de nenhum sistema de informação. Logo só existirá apenas um exemplar que será alocado na raiz de uma pasta criada pelo *software requirements engineer* na diretoria O:/. Os restantes artefactos, como estão diretamente associados a um sistema de informação devem ser incluídos numa pasta com o nome do sistema de informação na pasta criada na diretoria O:/.

A diretoria O:/ é acedida em qualquer computador interior à rede da Bosch, contudo os seus acessos são limitados. Só atores com acesso a uma palavras-chave, (fornecida pelo *information systems manager* (ver capítulo 3.3, atores do processo)) podem aceder a estes artefactos.

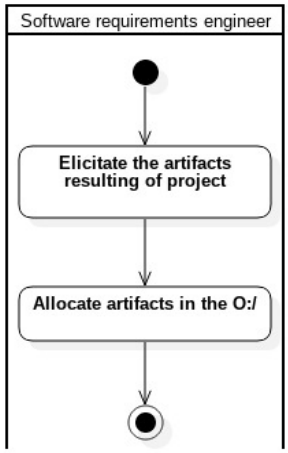


Ilustração 21: Allocation

Justificação

Com vista na atividade de *elaboration*, do processo aplicável à Bosch, para tornar esta eficiente, com a reutilização e mudança de requisitos (Wiegers & Beatty, 2013), associados a diferentes sistemas de informação, é necessário preservar o produto do projeto finalizado, alocando este numa diretoria fidedigna (que garanta que este não se perde).

Todos os requisitos levantados e desenvolvidos no âmbito do processo aplicável à Bosch estão incluídos na *table of horizontal traceability*, associada ao domínio dos requisitos, porém, também é relevante preservar toda a gestão de requisitos feita ao nível dos requisitos e do sistema de informação para perceber por exemplo que relações tem uma determinada versão de um requisito num sistema de informação, ou que estados estão associados a esta. Toda a informação é relevante.

Wiegers & Beatty (2013) afirmam que um repositório para alocar os requisitos pode ser uma pasta em rede, daí ser utilizada a diretoria O:/ para o efeito. A diretoria O:/ dá garantias na segurança dos documentos com a tolerância a falhas, e controlo de acessos. Isto dá mais garantias que os documentos não serão perdidos ou adulterados.

d) Change management

Explicação

A *change management* é uma atividade que inicia sempre com o aparecimento de um pedido de mudança, efetuado pelo *client*, e tem o propósito de planear e/ou preservar mudanças pretendidas num determinado sistema de informação, assim como minimizar o impacto e risco da mudança. Quatro tarefas constituem esta atividade do processo aplicável à Bosch (ver ilustração 22):

1. Evaluate change request: Um *project manager*, em conjunto com o *information systems manager* (ver capítulo 3.3, atores do processo), analisa o pedido de mudança, aplicável num sistema de informação concebido no âmbito de um dos seus projetos, de forma a estabelecer (Wiegers & Beatty, 2013):
   a) A sua viabilidade técnica;
   b) O seu custo;
   c) O alinhamento com os requisitos do sistema de informação;
   d) O alinhamento com restrições de recursos.

   Para dar mais significado a isto, um *software requirements engineer*, deve fazer uma análise do impacto da possível mudança. De modo a fazer uma análise do impacto da mudança, o *software requirements engineer* deve (Wiegers & Beatty, 2013):
   a) Identificar que requisitos de utilizador são afetados diretamente pela mudança;
   b) Para cada requisito afetado diretamente pela mudança, verificar o efeito cascata[15] da sua mudança, ou seja:
      i. Com recurso às *tables of requirements relations* (ver apêndice III, *table of requirements relations*), desenvolvidas na atividade de *traceability*, verificar as relações utilizador – sistema para identificar os requisitos de sistema derivados deste;
      ii. Verificar, com o auxílio das *tables of requirements relations* e do *graph (system - system)*, desenvolvidos na atividade de *traceability*, os requisitos de sistema que se relacionam com os requisitos de sistema identificados no ponto anterior (devido à existência de impacto direto e indireto nas relações, há a necessidade de definir um limite do impacto

15 Efeito cascata sugere a ideia de um efeito ser a causa de outro efeito gerando uma série de acontecimentos semelhantes de média, longa ou infinita duração

considerável, definido com a consciência do *software requirements engineer* e aplicável a cada caso);

iii. Para todos os requisitos de sistema relacionáveis (direta ou indiretamente) ao requisito afetado diretamente pela mudança, identificar SLOs derivados, com o auxílio da *table of vertical traceability* (ver apêndice VI, *table of vertical traceability*).

c) Identificar as tarefas necessárias para implementar a mudança (por exemplo levantar novos requisitos), e estimar o esforço necessário para completar estas tarefas, medido em horas-homem.

No final desta tarefa é assegurado que as consequências de aceitar a mudança estão percebidas (Wiegers & Beatty, 2013);

De seguida, uma verificação é feita para determinar se este pedido de mudança é feito no domínio de um projeto ou se motiva o início de um novo. Se este pedido de mudança não foi executado no contexto de um projeto ou não motivar o início de um novo, estas tarefas são executadas:

2. Add the change on waiting list: Um *software requirements engineer* coloca na *wainting list of changes* (ver apêndice VII, *waiting list of changes*) a mudança sugerida no pedido de mudança para ser considerada futuramente. Na lista de espera de mudança o deve colocar:
   a) A mudança escrita em linguagem natural exatamente como foi apresentada no pedido de mudança;
   b) O motivo apresentado pelo *client* para a mudança (ex: sistema obsoleto);
   c) Eventuais observações para desambiguar a mudança;
   d) O ID dos requisitos de utilizador afetados diretamente pela mudança;
   e) O impacto da mudança avaliado de 1 a 5 (ver *waiting list of changes* para adquirir informação de como classificar);
   f) O esforço estimado da mudança em horas-homem efetuado na tarefa de *evaluate change request*;
3. Uptade requirements state: Um *software requirements engineer* atualiza o estado das versões dos requisitos, incluídas na coluna dos requisitos afetados, da *waiting list of changes*, para "para mudança" na *state table* (ver apêndice II, *state table*), com a indicações expressadas na atividade de *lifecycle management* do processo aplicável à Bosch;

Se este pedido é feito no âmbito de um projeto ou motiva um novo, a seguinte tarefa é efetuada:

4. Make change decision: O *project manager* e o *client* (ver capítulo 3.3, atores do processo) decidem se as mudanças (a mudança atual e as em lista de espera) são aplicadas ou não, com base na análise feita ao pedido de mudança (Wiegers & Beatty, 2013).

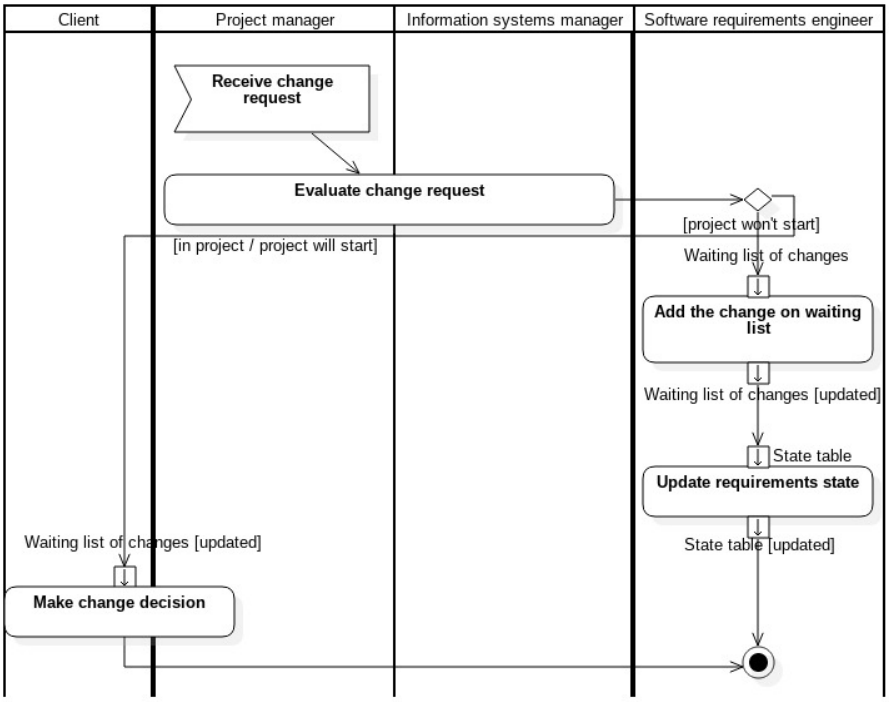


Ilustração 22: Change management

Justificação

Mudar os requisitos dos sistemas de informação é uma necessidade na Bosch, devido ao mundo e a organização estarem em constante mudança (Leffingwell & Widrig, 2003; Wiegers & Beatty, 2013). Novas oportunidades no mercado surgem, as políticas e regulamentações da organização mudam e desenvolvem-se novas necessidades no negócio, o que torna os sistemas de informação obsoletos e resulta na necessidade de os atualizar periodicamente (Wiegers & Beatty, 2013).

Se a gestão de uma mudança não é efetuada, então a equipa de desenvolvimento não saberá ao certo o que será entregue, nem que implicações a mudança terá num determinado sistema de informação, o que acaba por resultar numa lacuna de expectativas (Wiegers & Beatty, 2013).

Sempre que surge um pedido de mudança, é preciso analisar este para tirar ilações referentes ao impacto que a mudança terá no sistema de informação a aplicar, de forma a minimizar o impacto e risco de implementar esta.

Um pedido de mudança efetuado pelo *client* corresponde frequentemente à mudança de diversos requisitos de utilizador que são derivados em requisitos de sistema. O requisito de sistema é o primeiro SLO a

aparecer no processo de engenharia de software, que define que características o sistema de informação terá, então todos os SLOs de fases posteriores são derivações desta instância. Outro aspeto considerável é a consciencialização do *software requirements engineer* para a existência de interdependências nos requisitos de sistema. Devido a estas incidências, alterar um requisito, implica por norma muito mais do que isso, criando um efeito cascata para outros SLOs do mesmo tipo ou de tipos diferentes (Wiegers & Beatty, 2013).

O processo aplicável à Bosch tem de garantir que nenhum requisito se perde no final de uma iteração. Devido a este facto, quando surge um *client* com uma mudança nos requisitos, fora de um projeto de desenvolvimento de sistemas de informação, o processo tem de ter um mecanismo onde permita guardar a ideia para ser considerada futuramente.

Como na *state table* há o estado "para mudança", é necessário proceder à mudança do estado no artefacto.

Por fim é preciso tomar uma decisão, relativamente à aceitação ou rejeição da mudança proposta. A análise do pedido de mudança tem um papel importante nesta decisão (Wiegers & Beatty, 2013).

3.3 Papéis do Processo

A engenharia de requisitos é frequentemente colocada sobre a forma de um processo, porque é uma disciplina um pouco mais exigente que outras sub-disciplinas da engenharia de software, no que toca à sensibilidade do humano. Ao longo do documento é referido que no mundo dos requisitos muitas vezes os profissionais desta área desenvolvem o método com que se sentem mais confortáveis, porque neste domínio cada caso é um caso. Não há um método infalível que garanta, por exemplo a atividade de *elicitation* executada na plenitude.

Por outro lado, um processo é considerado muitas vezes um algoritmo para pessoas, de forma a criar uma sequência de atividades com um determinado fim, com o auxílio de artefactos. A estas pessoas são atribuídos um ou mais papeis, o que faz destas atores.

Para um ator assumir um determinado papel, há a necessidade de conjugar competências ou habilitações que são indispensáveis para as atividades ser executadas adequadamente. Com base nesta abordagem, propõe-se os seguintes papeis:

- *Information systems manager*;
- *Project manager*;
- *Software requirements engineer*;

- *Development team spokesman*;
- *Facilitator*;
- *Client*;
- *User*;
- *Expert*.

Information systems manager

O *information systems manager* deve conhecer a visão global dos sistemas de informação e garantir que os sistemas instalados na organização estão alinhados com os objetivos da empresa.

Este profissional é responsável por ser o "guardião" do serviço providenciado por um sistema de informação aos processos da organização, ou seja, deve garantir que o sistema de informação é útil no domínio aplicacional (Amaral, 2005).

É histórica a dificuldade de entendimento entre os gestores de uma organização e os seus informáticos (Amaral, 2005). O *information systems manager*, deve então ter o papel de interlocutor entre as pessoas que trabalham a um nível mais técnico (informáticos) e as pessoas que trabalham um nível mais holístico sobre a organização (gestores), porque tem a capacidade de comunicar na linguagem de ambos (Amaral, 2005).

Segundo Amaral (2005), um informático só tem a capacidade de desenvolver sistemas de informação, contudo, a visão geral de um sistema de informação, como é concebido para estar alinhado com os processos da organização, precede também de uma componente de planeamento. É precisamente esta componente de planeamento de sistemas de informação que cria estes atritos entre informáticos e gestores. Então, o *information systems manager* tem também este papel de assumir a paternidade da visão geral do sistema de informação, garantindo que a construção de um sistema de informação é verdadeiramente participada por gestores e informáticos (Amaral, 2005).

Competências-chave do *information systems manager*:

- Grande capacidade de comunicação oral e escrita;
- Conhecimentos no desenvolvimento de sistemas de informação;
- Conhecimentos de gestão e administração.

Project manager

O *project manager* é a pessoa destacada pela organização para liderar a equipa responsável por alcançar os objetivos de um projeto (Institute, 2013).

Como a gestão de projetos é uma disciplina estratégica, o *project manager* torna-se o elo de ligação entre a estratégia e a equipa (Institute, 2013). Os projetos são essenciais para a sobrevivência e crescimento da organização, uma vez que projetos criam valor na forma de processos de negócio melhorados, que são indispensáveis no desenvolvimento de novos produtos e serviços, que tornam mais fácil à organização responder às mudanças relativas ao ambiente, concorrência e mercado (Institute, 2013). Assim sendo, o papel do *project manager* é maioritariamente estratégico (Institute, 2013).

A gestão de projetos exige o conhecimento da área de incidência do projeto, recolhida de modelos de referência, mas também competências gerais, aplicáveis a todos os projetos.

Competências-chave do *project manager*:

- Conhecimentos sobre gestão de projetos (Institute, 2013);
- Aplicação do conhecimento sobre gestão de projetos (Institute, 2013);
- Habilidades interpessoais (Institute, 2013): (1) liderança, (2) construção de equipas, (3) motivação, (4) comunicação, (5) influência, (6) tomada de decisões, (7) consciência política e cultural, (8) negociação, (9) ganho de confiança, (10) gestão de conflitos e (11) *coaching*.

Software requirements engineer

O *software requirements engineer*, frequentemente citado como analista de negócio, tem a responsabilidade de levantar, analisar, documentar e validar as necessidades dos *stakeholders* do projeto (Wiegers & Beatty, 2013).

Este profissional serve de ponte entre a comunidade do cliente e a equipa de desenvolvimento de software, logo é o principal interpretador dos requisitos que fluem entre estas duas entidades, usando a sua poderosa capacidade comunicativa para criar assim um canal comunicativo entre ambos (Wiegers & Beatty, 2013).

O *software requirements engineer* está envolvido ao longo de todo o ciclo de vida do desenvolvimento de software para estabelecer uma conexão entre os requisitos e outros elementos do sistema de informação (Wiegers & Beatty, 2013).

Competências-chave do *software requirements engineer*:

- Capacidade de ouvir para não perder detalhes no levantamento de requisitos;
- Entrevistar e questionar;
- Capacidade de aprender para se introduzir facilmente num novo domínio;
- Grande capacidade de comunicação oral e escrita;
- Capacidade de ajustar a sua linguagem consoante o *stakeholder*;
- Conhecer técnicas que facilitem as atividades de levantamento e prioritização de requisitos;
- Não tendo de ser um especialista, deve ter bases na programação de software;
- Capacidades de modelação;
- Experiência com as ferramentas Microsoft Word e Microsoft Excel;
- Ter noções de gestão.

Development team spokesman

A equipa de desenvolvimento é constituída por arquitetos de software, programadores e testadores de software, então o *development team spokesman* deve ser um membro da equipa, podendo ter qualquer um dos papeis apresentados neste parágrafo. Porém, o *development team spokesman* deve estar consciente do trabalho desenvolvido em cada fase posterior à análise e deve entender este, logo deve ter noções de arquitetura, desenvolvimento e teste de sistemas de informação. O representante também deve ser reconhecido como tal pela equipa, por isso sugere-se a eleição deste no início do projeto através de uma votação.

Competências-chave do *development team spokesman*:

- Conhecimentos no domínio da conceção, desenvolvimento e teste de sistemas de informação;
- Capacidade de comunicação escrita e oral (tem de falar a linguagem de toda a equipa de desenvolvimento e dos *software requirements engineers*).

Facilitator

O *facilitator* tem a responsabilidade de zelar pela resolução dos conflitos que originam uma negociação de requisitos e facilitar o levantamento de requisitos em técnicas de grupo (Pressman, 2005). Isto implica que o *facilitator* tenha conhecimento do domínio destino do sistema de informação em desenvolvimento, experiência na resolução de conflitos e experiência na condução de sessões de levantamento de requisitos baseado em grupos de pessoas. Normalmente este papel é desempenhado por

um indivíduo que concilia outra atividade na equipa de desenvolvimento de software (incluindo os *software requirements engineers*) com a de *facilitator*.

Competências-chave do *facilitator*:

- Grandes capacidades de comunicação oral;
- Reconhecer as posturas e estratégias adotadas pelos diversos *stakeholders* para determinar como resolver a negociação;
- Experiência em levantamento de requisitos com base em técnicas de grupos de pessoas.

Client

O *client* é a entidade que encomenda e paga o desenvolvimento de um sistema, normalmente após negociar o preço com o produtor ou fornecedor (Fernandes & Machado, 2015).

Como o desenvolvimento é pedido pelo *client*, é legítimo que este tenha o poder de decidir ou ter influência em diversas questões, nomeadamente o âmbito, funcionalidades e custo (Fernandes & Machado, 2015).

Os *clients* podem ser, por exemplo, os gestores, proprietários de empresas ou pessoas responsáveis por os departamentos onde o sistema será colocado em funcionamento, sendo que há a possibilidade destes não serem os utilizadores finais do sistema (Fernandes & Machado, 2015). Por exemplo, aqui na Bosch, o gestor do departamento de QMM7 (departamento associado à qualidade do produto) é o *client* do GEIME (sistema de informação fortemente referenciado no capítulo 4), porém não interage diretamente com este.

Para além da necessidade de assegurar o suporte contínuo e o compromisso do *client* durante o desenvolvimento de um sistema, é também extremamente aconselhável incluir eles na atividade de *elicitation* (Fernandes & Machado, 2015).

User

O *user* é uma pessoa que opera e interage diretamente com o sistema, quando este está numa operação eficaz no seu ambiente (Fernandes & Machado, 2015). Como o processo trata sistemas de informação, os *users* são as pessoas que estão em frente ao ecrã do computador a introduzir dados e a observar os resultados (Fernandes & Machado, 2015).

Numa primeira fase, antes de o sistema ser colocado em exploração, as pessoas que são designadas como *users* são apenas potenciais utilizadores (Fernandes & Machado, 2015). Estas pessoas só são utilizadores efetivos quando o sistema é implementado e colocado em funções (Fernandes & Machado, 2015).

Em novos ciclos de desenvolvimento, para fins de manutenção, os *users* com grande experiência na interação com o sistema, devem ser identificados de modo a obter informação relevante para melhorar ou corrigir os sistema (Fernandes & Machado, 2015).

Expert

Segundo Fernandes e Machado (2015), o *expert* pode estar inerente aos domínios do problema e da solução, porém, aqui nesta dissertação de mestrado apenas o *expert* do problema é considerado como um papel do processo.

O *expert* fornece um profundo conhecimento, habilidades de alto nível e uma extensa experiência prática do domínio do problema onde o sistema de informação irá atuar (Fernandes & Machado, 2015). Este papel é normalmente assumido por engenheiros ou proprietários de processos.

3.4 Conclusões

O processo aplicável à Bosch é constituído por onze atividades, que são distribuídas pelas componentes de *development* e *management*. Estas atividades têm o propósito de estabelecer um mecanismo, para a gestão de requisitos ao longo do ciclo de vida de sistemas de informação, desenvolvidos no departamento de CI-CWR1, da organização Bosch Car Multimédia Portugal S.A.

A componente de *development* é constituído pelas atividades de (1) *Inception*, (2) *Elicitation*, (3) *Elaboration*, (4) *Negotiation*, (5) *Prioritisation*, (6) *Documentation* e (7) *Validation*. Esta componente é inevitável no processo aplicável à Bosch, devido ao facto de ser necessário levantar e manusear os requisitos que representam com exatidão um determinado sistema de informação, para se averiguar utilidade na componente de *management*.

A componente de *management* é constituído pelas atividades de (1) *Lifecycle Management*, (2) *Traceability*, (3) *Allocation* e (4) *Change Management*. Com estas quatro atividades, é expectável que o processo de engenharia de requisitos do departamento CI-CWR1 se torne mais eficaz e eficiente, resultando numa minimização da utilização de recursos, e no desenvolvimento metódico de requisitos, tornando estes menos ambíguos e mais estáveis.

O processo aplicável à Bosch tem precedências na sua utilização. Para as atividades delimitadas neste capítulo serem produtivas, é necessário a atribuição de papeis, de modo a garantir determinadas características, que são necessárias nos potenciais utilizadores. São identificados os seguintes papeis:

1. *Information systems manager*;

2. *Project manager*;
3. *Software requirements engineer*;
4. *Development team spokesman*;
5. *Facilitator*;
6. *Client*;
7. *User*;
8. *Expert*.

Com este capítulo finalizado, resta apenas uma (ou mais) experimentação(ões) ao processo, no domínio aplicacional, tirando assim algumas ilações sobre os esforços efetuados até ao momento.

4 EXPERIMENTAÇÃO DO PROCESSO DE ENGENHARIA DE REQUISITOS

4.1 Introdução

No capítulo anterior, concebeu-se um processo com a finalidade de resolver o problema, de gerir requisitos ao longo do ciclo de vida de sistemas de informação, manifestado na organização Bosch Car Multimédia Portugal S.A. (secção de CI-CWR12 do departamento CI-CWR1, para ser mais específico).

Segundo Peffers *et al* (2007), o método de investigação DSR impõe a necessidade de demonstrar, avaliar e comunicar o artefacto concebido no âmbito do trabalho de investigação, daí a inserção deste capítulo no documento. De modo a satisfazer estas imposições do método de investigação escolhido, duas experimentações são efetuadas ao processo aplicável à Bosch (em projetos que surgiram na organização), intercaladas por uma divulgação do trabalho efetuado na Bosch Car Multimédia Portugal S.A.

No sub-capítulo 4.2, surge a primeira experimentação do processo aplicável à Bosch. O projeto utilizado para este caso chama-se "GEIME" e está associado a uma aplicação, que tem o propósito de gerir dispositivos para calibração, dentro da organização. Este sub-capítulo divide-se numa apresentação do problema, numa descrição do processo aplicável à Bosch em prática e numa discussão de resultados provenientes do processo em prática.

No sub-capítulo 4.3, é divulgado o trabalho adjacente à conceção do processo e à experimentação 1 do projeto "GEIME". Esta divulgação é efetuada através de 3 reuniões, de onde saem sugestões importantes, para a alteração de elementos, no processo à Bosch.

No sub-capítulo 4.4, aparece a segunda experimentação do processo aplicável à Bosch. O projeto utilizado para esta situação denomina-se "ALR", e compromete-se em desenvolver uma aplicação, que substituirá uma folha de cálculo, no desígnio de analisar lotes decorrentes do processo produtivo da Bosch Car Multimédia Portugal S.A. Do mesmo modo que na primeira experimentação, este sub-capítulo divide-se numa apresentação do problema, numa descrição do processo aplicável à Bosch em prática e numa discussão de resultados oriundos do processo em prática.

4.2 Experimentação 1 - “GEIME”

O GEIME é uma aplicação desenvolvida no departamento de CI-CWR1, há sensivelmente vinte anos, que tem o propósito de auxiliar o departamento de QMM7 a fazer uma gestão da calibração de equipamentos da organização, para garantir qualidade no funcionamento destes.

Em todos os departamentos da Bosch há um gestor de equipamentos, que é notificado pela aplicação quando há a necessidade de se realizar a calibração de um dos equipamentos do seu departamento. No departamento de QMM7 há um gestor da aplicação que garante que os equipamentos são calibrados.

Para esta aplicação funcionar, há uma enorme base de dados onde estão incluídas as referências de todos os equipamentos da empresa, assim como informação sobre o seu estado e datas agendadas para a calibração.

É importante realçar que esta experimentação está associada a uma versão ainda primordial do processo criado no âmbito desta dissertação, então devem-se constatar algumas divergências entre o que está especificado no capítulo 3, e o que está expresso neste subcapítulo.

4.2.1 Descrição do Problema

Como está mencionado neste documento, a aplicação de seu nome GEIME foi desenvolvida sensivelmente há vinte anos. Este facto reflete-se numa utilização cada vez mais dificultada pelo obsoletismo da aplicação, o que resulta numa insatisfação natural por parte dos utilizadores desta.

Com este descontentamento, dentro do departamento de QMM7, nasce o desejo de reabilitar o GEIME, a médio prazo, criando assim uma nova versão da aplicação.

Atendendo que o departamento de CI-CWR1 tem demonstrado indisponibilidade devido ao elevado número de projetos em execução, e há a necessidade de perceber quais são as necessidades dos atuais utilizadores da aplicação, contratou-se um estagiário com o propósito de fazer um levantamento dos requisitos.

Uma vez que este estagiário não tem qualquer experiência com a engenharia de requisitos, nasce uma oportunidade de experimentar o processo concebido no âmbito desta dissertação de mestrado.

4.2.2 Processo Aplicável à Bosch em Prática

Visto que se pretende fazer uma evolução do sistema de informação em funcionamento na empresa, duas iterações do processo aplicável à Bosch são executadas.

A iteração I está relacionada com a recolha, desenvolvimento e gestão do maior número de requisitos da aplicação considerada obsoleta. A iteração II está associada à especificação dos requisitos que constituem a nova versão do GEIME.

Iteração I

Inception

Nesta atividade uma reunião é efetuada, com o propósito do *client* transmitir ao *software requirements engineer* o problema que motiva a reabilitação do GEIME. O *software requirements engineer* coloca as questões iniciais de maneira a obter mais informação.

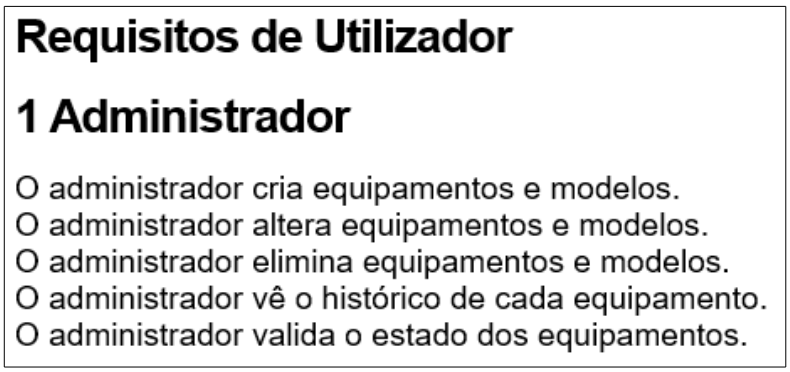
Atendendo que nesta fase a atividade de *inception* ainda não exige a necessidade de gravar as descrições do problema e das soluções, o *artifact of requirements handling* não é criado.

Elicitation

Visto que o GEIME foi desenvolvido numa época em que a engenharia de requisitos ainda era desprezada, não são detetáveis quaisquer documentos cuja a sua finalidade seja de alguma forma especificar requisitos da aplicação.

Então, de modo a recolher o maior número de requisitos possível, aplica-se a técnica de *reverse engineering*, já referida neste documento (ver capítulo 2.3.2, atividades da engenharia de requisitos), na aplicação e no seu manual de instruções.

Posteriormente, estes requisitos, expressos em descrições são documentados no *artifact of requirements handling*: Na ilustração 23, pode ver um exemplo, de uma descrição documentada, no *artifact of requirements handling*.


Requisitos de Utilizador

1 Administrador

O administrador cria equipamentos e modelos.
O administrador altera equipamentos e modelos.
O administrador elimina equipamentos e modelos.
O administrador vê o histórico de cada equipamento.
O administrador valida o estado dos equipamentos.

Ilustração 23: Descrição do artifact of requirements handling (1ª iteração) do projeto "GEIME"

Elaboration

O *artifact of requirements handling* resultante da atividade de *elicitation* é utilizado em todo o período desta atividade. Inicialmente, é feita uma análise a todas as descrições concebidas na atividade anterior de modo a serem identificados os requisitos de utilizador. Para identificar um requisito, o *software requirements engineer* copia e cola, no parágrafo seguinte uma descrição, antecedida do padrão de identificador já mencionado no capítulo 3. Atente que nesta fase do processo ainda não existem versões de requisitos, logo o identificador ainda apresenta-se sem o "VX" do padrão estabelecido na conceção do processo.

De seguida, são aplicados padrões de sintaxe em todos os requisitos de utilizador identificados. Com isto podem nascer mais requisitos porque um requisito pode ser ambíguo devido ao seu tamanho, e ser necessária uma fragmentação deste, para resolver esta questão. Na ilustração 24, pode visualizar um exemplo, do resultado desta atividade, até ao momento.

[Cust.Req.1] O administrador deve criar uma ficha do equipamento.
[Cust.Req.2] O administrador deve criar um modelo.
[Cust.Req.3] O administrador deve alterar uma ficha de equipamento.
[Cust.Req.4] O administrador deve alterar modelos.
[Cust.Req.5] O administrador deve eliminar uma ficha de equipamento.
[Cust.Req.6] O administrador deve eliminar modelos.
[Cust.Req.7] O administrador deve ter acesso ao histórico dos equipamentos.

Ilustração 24: Requisitos (user) do artifact of requirements handling (1ª ite.) do projeto "GEIME"

Quando os requisitos estão escritos com o padrão de sintaxe sugerido, é feita uma análise geral ao documento com o propósito de identificar conflitos na natureza dos requisitos. Como os requisitos são levantados por *reverse engineering*, não há nesta fase requisitos levantados de *stakeholders*, logo não há conflitos que podem surgir devido à incompatibilidade de visões de *stakeholders*. Qualquer conflito identificado é solucionado de seguida.

No momento em que os requisitos de utilizador já não apresentam quaisquer problemas, os requisitos de sistemas são desenvolvidos, utilizando os requisitos de utilizador como referência e aplicando os padrões de sintaxe e estratégias para resolução de conflitos utilizados nos requisitos de utilizador. Aqui, separar os requisitos funcionais dos não funcionais é também uma ajuda na organização do documento. A ilustração 25 apresenta uma derivação de um requisito de utilizador em um requisito de sistema.

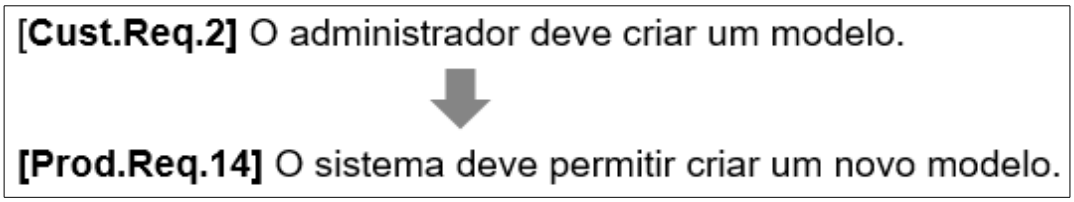


Ilustração 25: Derivação de requisito de utilizador em requisito de sistema do projeto "GEIME"

O processo indica que o *software requirements engineer* deve se aconselhar com o *information systems manager* para definir os limites do sistema, mas uma vez que se levantaram requisitos por *reverse engineering* de uma aplicação já desenvolvida, neste caso não é aplicável. Assim que os requisitos de utilizador e de sistema, já estiverem desenvolvidos como indica o processo aplicável à Bosch, a atividade termina.

Negotiation

Esta atividade não se aplica nesta fase.

Prioritisation

Esta atividade não se aplica nesta fase.

Documentation

Nesta iteração o objetivo é levantar o maior número de requisitos para reutilizar na iteração seguinte. De modo a poupar recursos, esta atividade é desconsiderada nesta fase.

Validation

Esta atividade não se aplica nesta fase.

Lifecycle Management

Neste caso, a atividade de *lifecycle management* ainda tem um funcionamento diferente do apresentado no processo aplicável à Bosch, no capítulo 3. Enquanto aqui a gestão do ciclo de vida por versões é orientada para o sistema de informação, no capítulo 3 a gestão do ciclo de vida por versões é orientada para os requisitos.

Como resultado desta atividade dois artefactos são obrigatoriamente desenvolvidos. Num destes artefactos, ao qual se deu o nome de *state table*, o *software requirements engineer* atribui o estado "novo" a

todos os requisitos, uma vez que todos os requisitos até esta fase foram desenvolvidos de raiz. Na ilustração 26 pode visualizar parte da *state table*, desenvolvida até ao momento.

ID Requisito	Descrição	Versão do Sistema de Informação	Estado							
			Novo	Alterado	Selecionado	Rejeitado	Documentado	Validado	Implementado	Para Mudança
Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6	Coluna7	Coluna8	Coluna9	Coluna10	Coluna11
[Cust.Req.1]	O administrador deve criar uma ficha do equipamento.	[V.1]	X							
[Cust.Req.2]	O administrador deve criar um modelo.	[V.1]	X							
[Cust.Req.3]	O administrador deve alterar uma ficha de equipamento.	[V.1]	X							
[Cust.Req.4]	O administrador deve alterar modelos.	[V.1]	X							
[Cust.Req.5]	O administrador deve eliminar uma ficha de equipamento.	[V.1]	X							
[Cust.Req.6]	O administrador deve eliminar modelos	[V.1]	X							
[Cust.Req.7]	O administrador deve ter acesso ao histórico dos equipamentos.	[V.1]	X							
[Cust.Req.8]	O gestor da aplicação deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição;	[V.1]	X							
[Cust.Req.9]	O gestor da aplicação deve alterar um modelo.	[V.1]	X							
[Cust.Req.10]	O gestor da aplicação deve eliminar um modelo.	[V.1]	X							

Ilustração 26: State table do projeto "GEIME"

Como pode constatar, nesta fase o artefacto ainda não apresenta todos os estados definidos no capítulo 3. Em adição, apresenta também uma coluna que representa a versão do sistema de informação onde o requisito é manuseado pela última vez, porém na versão final do processo aplicável à Bosch este dado deixa de fazer sentido. Este dado é fornecido pelo artefacto que é apresentado no parágrafo seguinte.

O outro artefacto que resulta desta atividade tem o nome de "tabela de identificação da versão do sistema de informação dos requisitos" e tem o desígnio de auxiliar o *software requirements engineer* a identifica a versão do sistema de informação onde um requisito foi manuseado pela última vez. Na ilustração 27 pode visualizar a "tabela de identificação da versão do sistema de informação".

ID Requisito	Descrição	Estado	Ultima Mudança de Requisito											
			[V1]	[V2]	[V3]	[V4]	[V5]	[V6]	[V7]	[V8]	[V9]	[V10]	[V11]	[V12]
Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6	Coluna7	Coluna8	Coluna9	Coluna10	Coluna11	Coluna12	Coluna13	Coluna14	Coluna15
[Cust.Req.1]	O administrador deve criar uma ficha do equipamento.	Novo	X											
[Cust.Req.2]	O administrador deve criar um modelo.	Novo	X											
[Cust.Req.3]	O administrador deve alterar uma ficha de equipamento.	Novo	X											
[Cust.Req.4]	O administrador deve alterar modelos.	Novo	X											
[Cust.Req.5]	O administrador deve eliminar uma ficha de equipamento.	Novo	X											
[Cust.Req.6]	O administrador deve eliminar modelos	Novo	X											
[Cust.Req.7]	O administrador deve ter acesso ao histórico dos equipamentos.	Novo	X											
[Cust.Req.8]	O gestor da aplicação deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição;	Novo	X											
[Cust.Req.9]	O gestor da aplicação deve alterar um modelo.	Novo	X											
[Cust.Req.10]	O gestor da aplicação deve eliminar um modelo.	Novo	X											
[Cust.Req.11]	O gestor da aplicação deve criar um tipo de equipamento com as seguintes informações: 1. Código; 2. Descrição.	Novo	X											
[Cust.Req.12]	O gestor da aplicação deve alterar um tipo de equipamento.	Novo	X											
[Cust.Req.13]	O gestor da aplicação deve eliminar um tipo de equipamento.	Novo	X											
[Cust.Req.14]	O gestor da aplicação deve criar uma secção	Novo	X											
[Cust.Req.15]	O gestor da aplicação deve atribuir um equipamento a	Novo	X											

Ilustração 27: Tabela de identificação da versão do sistema de informação do projeto "GEIME"

Na seguinte hiperligação pode descarregar este artefacto: https://goo.gl/SMvWAv.

Este artefacto tornou-se obsoleto na versão final do processo concebido, sendo a sua finalidade assumida pelo artefacto *table of horizontal traceability*, e aplicada aos requisitos ao invés de aos sistemas de informação.

Traceability

Nesta fase, a atividade de *traceability* é constituída pela identificação das relações nos requisitos, rastreabilidade horizontal e rastreabilidade vertical.

Relativamente à identificação de relações de requisitos, o artefacto *table of requirements relations* é desenvolvido, ainda que de uma forma primordial. Atendendo que não há ainda tipos de relações, apenas são identificadas as relações com um "x", desprezando o tipo de relação em que se enquadra a relação. Estas relações são identificadas pelo *software requirements engineer*, e estão ao encargo da deliberação deste. O artefacto *matrix of requirements relations* ainda não existe nesta fase, portanto é desconsiderado.

Na ilustração 28 pode visualizar um exemplo de identificação de relações dos requisitos de utilizador na *table of requirements relations*.

Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6
ID Requisito	Descrição	Estado	Versão de Sistema de Informação	Relações com Requisitos de Utilizador	Relações com Requisitos de Sistema
[Cust.Req.1]	O administrador deve criar uma ficha do equipamento com as seguintes informações: 1. NICE; 2. Nº de inventário; 3. Nº FVB; 4. Nº de série; 5. Tipo de equipamento; 6. Modelo do equipamento; 7. Descrição; 8. Estado; 9. Lugar;	Novo	[V.1]		[Prod.Req.66]
[Cust.Req.2]	O administrador deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição.	Novo	[V.1]		[Prod.Req.14]
[Cust.Req.3]	O administrador deve alterar uma ficha de equipamento.	Novo	[V.1]		[Prod.Req.67]
[Cust.Req.4]	O administrador deve alterar modelos.	Novo	[V.1]		[Prod.Req.15]
[Cust.Req.5]	O administrador deve eliminar uma ficha de equipamento.	Novo	[V.1]		[Prod.Req.68]
[Cust.Req.6]	O administrador deve eliminar modelos	Novo	[V.1]		[Prod.Req.16]
[Cust.Req.7]	O administrador deve ter acesso ao histórico dos equipamentos.	Novo	[V.1]		[Prod.Req.69]
[Cust.Req.8]	O gestor da aplicação deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição;	Novo	[V.1]		[Prod.Req.14]
[Cust.Req.9]	O gestor da aplicação deve alterar um modelo.	Novo	[V.1]		[Prod.Req.14] [Prod.Req.15]
[Cust.Req.10]	O gestor da aplicação deve eliminar um modelo.	Novo	[V.1]		[Prod.Req.16] [Prod.Req.14]
[Cust.Req.11]	O gestor da aplicação deve criar um tipo de equipamento com as seguintes informações: 1. Código; 2. Descrição.	Novo	[V.1]		[Prod.Req.18]

Ilustração 28: Table of requirements relations (requisitos de utilizador) do projeto "GEIME"

Na ilustração 29 pode visualizar um exemplo de identificação de relações dos requisitos de sistema na *table of requirements relations*.

Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6
ID Requisito	Descrição	Estado	Versão de Sistema de Informação	Relações com Requisitos de Utilizador	Relações com Requisitos de Sistema
[Prod.Req.1]	O sistema deve permitir autenticação.	Novo	[V.1]	[Cust.Req.60]	TODOS
[Prod.Req.2]	O sistema deve diferenciar tipos de acesso.	Novo	[V.1]	[Cust.Req.60] [Cust.Req.19]	TODOS
[Prod.Req.3]	O sistema deve permitir acesso a diferentes módulos.	Novo	[V.1]	[Cust.Req.60] [Cust.Req.19]	[Prod.Req.2]
[Prod.Req.4]	Quando um utilizador está autenticado o sistema deve exibir uma lista dos equipamentos que lhe estão atribuídos.	Novo	[V.1]	[Cust.Req.45] [Cust.Req.46]	[Prod.Req.2] [Prod.Req.14] [Prod.Req.18] [Prod.Req.28] [Prod.Req.29] [Prod.Req.30]
[Prod.Req.5]	O sistema deve exibir os equipamentos que serão enviados para calibração, num período de 15 dias.	Novo	[V.1]	[Cust.Req.27] [Cust.Req.41]	[Prod.Req.14] [Prod.Req.18]
[Prod.Req.6]	O sistema deve exibir prioritariamente os equipamentos cuja data de envio para calibração já tenha sido ultrapassada.	Novo	[V.1]	[Cust.Req.27] [Cust.Req.41]	[Prod.Req.4] [Prod.Req.14]
[Prod.Req.7]	O sistema deve limitar a visualização dos equipamentos consoante o perfil do utilizador.	Novo	[V.1]	[Cust.Req.46]	[Prod.Req.4] [Prod.Req.14]
[Prod.Req.8]	O sistema deve exibir os seguintes dados de um equipamento: 1. NICE; 2. Nº de inventário; 3. Nº FVB; 4. Nº de série; 5. Tipo de equipamento; 6. Modelo do equipamento; 7. Descrição; 8. Estado; 9. Lugar;	Novo	[V.1]	[Cust.Req.1]	[Prod.Req.4] [Prod.Req.14]
[Prod.Req.9]	O sistema deve exibir os seguintes dados da última reparação de um equipamento: 1. Data de envio; 2. Responsável; 3. Nº de EBS; 4. Descrição da avaria; 5. Lugar da reparação;	Novo	[V.1]	[Cust.Req.56]	[Prod.Req.4] [Prod.Req.14]

Ilustração 29: Table of requirements relations (requisitos de sistema) do projeto "GEIME"

Quanto à rastreabilidade horizontal, a coluna do artefacto *table of horizontal traceability*, referente à primeira versão do sistema de informação, é preenchida com a informação dos requisitos desenvolvidos na atividade de *elaboration*. Como é mencionado neste capítulo, neste estado de evolução do processo aplicável à Bosch, ainda não se atribuem versões aos requisitos ao invés de atribuir ao sistema de informação. Na ilustração 30 deve visualizar parte do artefacto *table of horizontal traceability* já preenchido.

ID Requisito	Versão 1	
	Estado	Descrição
Coluna1	Coluna2	Coluna3
[Cust.Req.1]	Novo	O administrador deve criar uma ficha do equipamento com as seguintes informações: 1. NICE; 2. Nº de inventário; 3. Nº FVB; 4. Nº de série; 5. Tipo de equipamento; 6. Modelo do equipamento; 7. Descrição; 8. Estado; 9. Lugar;
[Cust.Req.2]	Novo	O administrador deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição.
[Cust.Req.3]	Novo	O administrador deve alterar uma ficha de equipamento.
[Cust.Req.4]	Novo	O administrador deve alterar modelos.
[Cust.Req.5]	Novo	O administrador deve eliminar uma ficha de equipamento.
[Cust.Req.6]	Novo	O administrador deve eliminar modelos

Ilustração 30: Table of horizontal traceability do projeto "GEIME"

Apesar do sistema de informação GEIME já se encontrar desenvolvido, não foram disponibilizados SLOs provenientes da fase de *implementation* do processo de engenharia de software. Com este facto justifica-se a ausência do desenvolvimento do artefacto *table of vertical traceability* nesta experimentação.

Change Management

Neste momento ainda não há qualquer desenvolvimento no processo aplicável à Bosch no sentido de *change management*, logo nada é realizado nesta atividade.

Allocation

Nesta atividade, todos os artefactos desenvolvidos foram alocados na unidade O:/. O processo aqui, ainda exige a realização de um artefacto, que contém a informação geral dos requisitos desenvolvidos na atual iteração. Esse artefacto tem o nome de "informação geral". Na ilustração 31 pode visualizar um excerto do artefacto desenvolvido nesta atividade.

ID Requisito	Descrição	Versão do Sistema de Informação	Estado	Relações com Requisitos de Utilizador	Relações com Requisitos de Sistema
Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6
[Cust.Req.1]	O administrador deve criar uma ficha do equipamento com as seguintes informações: 1. NICE; 2. Nº de inventário; 3. Nº FVB; 4. Nº de série; 5. Tipo de equipamento; 6. Modelo do equipamento; 7. Descrição; 8. Estado; 9. Lugar;	[V1]	Novo		[Prod.Req.66]
[Cust.Req.2]	O administrador deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição.	[V1]	Novo		[Prod.Req.14]
[Cust.Req.3]	O administrador deve alterar uma ficha de equipamento.	[V1]	Novo		[Prod.Req.67]
[Cust.Req.4]	O administrador deve alterar modelos.	[V1]	Novo		[Prod.Req.15]
[Cust.Req.5]	O administrador deve eliminar uma ficha de equipamento.	[V1]	Novo		[Prod.Req.68]
[Cust.Req.6]	O administrador deve eliminar modelos	[V1]	Novo		[Prod.Req.16]
[Cust.Req.7]	O administrador deve ter acesso ao histórico dos equipamentos.	[V1]	Novo		[Prod.Req.69]
[Cust.Req.8]	O gestor da aplicação deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição;	[V1]	Novo		[Prod.Req.14]
[Cust.Req.9]	O gestor da aplicação deve alterar um modelo.	[V1]	Novo		[Prod.Req.14] [Prod.Req.15]
[Cust.Req.10]	O gestor da aplicação deve eliminar um modelo.	[V1]	Novo		[Prod.Req.16] [Prod.Req.14]
[Cust.Req.11]	O gestor da aplicação deve criar um tipo de equipamento com as seguintes informações: 1. Código; 2. Descrição.	[V1]	Novo		[Prod.Req.18]
[Cust.Req.12]	O gestor da aplicação deve alterar um tipo de equipamento.	[V1]	Novo		[Prod.Req.18] [Prod.Req.19]
[Cust.Req.13]	O gestor da aplicação deve eliminar um tipo de equipamento.	[V1]	Novo		[Prod.Req.20] [Prod.Req.18]

Ilustração 31: Informação geral do projeto "GEIME"

Na seguinte hiperligação pode descarregar este artefacto: https://goo.gl/20Mhkp.

Iteração II

Inception

Uma vez que esta experimentação corresponde a um projeto com duas iterações do processo aplicável à Bosch, esta atividade não é efetuada nesta fase.

Elicitation

Dado esta iteração estar associada ao desenvolvimento de uma nova versão do sistema de informação "GEIME", nesta atividade de *elicitation* é utilizada uma diferente abordagem para o levantamento de requisitos.

Neste levantamento de requisitos, a técnica utilizada é a entrevista e fazem-se conceções para os entrevistados esquecerem a aplicação com que já trabalham, tentando acima de tudo procurar o foco no seu problema, e não como ele é resolvido.

No total, três entrevistas são realizadas a *stakeholders* cujo seu papel na aplicação não é correlativo. Dentre estes futuros utilizadores, esperam-se necessidades diferentes, que são descritas primeiramente num bloco de notas, e posteriormente copiadas para o *artifact of requirements handling*.

Na ilustração 32 deve visualizar um exemplo de uma destas descrições levantadas nesta atividade.

1. Levantamento de requisitos

1.1 Descrição do problema do cliente

O gestor da aplicação cria as fichas dos equipamentos com as seguintes informações:

1. NICE;
2. Nº de inventário;
3. Nº FVB;
4. Nº de série;
5. Tipo de equipamento;
6. Modelo do equipamento;
7. Descrição;
8. Estado;
9. Lugar;

O gestor da aplicação cria os modelos.

O gestor da aplicação altera os modelos.

O gestor da aplicação elimina os modelos.

O gestor da aplicação cria os tipos de equipamentos.

O gestor da aplicação altera os tipos de equipamentos.

O gestor da aplicação elimina os tipos de equipamentos.

O gestor da aplicação vê o histórico de cada equipamento.

O gestor da aplicação valida o estado dos equipamentos.

O gestor da aplicação cria os lugares.

O gestor da aplicação altera os lugares.

O gestor da aplicação elimina os lugares.

Ilustração 32: Descrição do artifact of requirements handling (2ª iteração) do projeto "GEIME"

Elaboration

Nesta atividade de *elaboration*, atendendo que o *software requirements engineer* já se encontra numa segunda iteração do processo aplicável à Bosch, já pode utilizar requisitos alocados no repositório de modo a tornar esta atividade mais eficaz e eficiente com a reutilização (ou mudança) de requisitos.

Deste modo consequente, o *software requirements engineer* tem de desenvolver os requisitos tendo sempre "ao alcance" dois documentos. Um documento é o *artifact of requirements handling*, onde foi escrita a descrição do problema, resultante da atividade de *elicitation*. O outro identifica-se como *table of horizontal traceability*, que é previamente alocado na primeira iteração. Este artefacto deve permitir ao *software*

requirements engineer, pesquisar por requisitos que já foram desenvolvidos, que podem facilmente ser reutilizados ou mudados, visto ser mais eficiente e eficaz utilizar ou mudar algo já desenvolvido.

O *software requirements engineer*, uma vez que tem liberdade para organizar o *artifact of requirements handling* para a sua conveniência, opta por deixar os requisitos reutilizados sem qualquer formatação, os requisitos mudados são realçados com um fundo amarelo e a cor da letra dos novos é mudada para vermelho. Atente que todas as regras estabelecidas para esta atividade no ponto 3 têm também de ser obrigatoriamente aplicáveis nesta iteração, sendo que agora já há a necessidade de consultar o information systems manager para determinar os limites do sistema.

Na ilustração 33 deve visualizar um exemplo de um requisito reutilizado da iteração anterior.

[Cust.Req.8] O gestor da aplicação deve criar um modelo com as seguintes informações:

1. Código;
2. Descrição.

Ilustração 33: Requisito reutilizado do projeto "GEIME"

Na ilustração 34 deve visualizar um exemplo de um requisito que o seu estado é "mudado".

[Cust.Req.1] O gestor da aplicação deve criar uma ficha do equipamento com as seguintes informações:

1. NICE;
2. Nº de inventário;
3. Nº FVB;
4. Nº de série;
5. Tipo de equipamento;
6. Modelo do equipamento;
7. Descrição;
8. Estado;
9. Lugar;

Ilustração 34: Requisito mudado do projeto "GEIME"

Na ilustração 35 deve visualizar um exemplo de um requisitos que é novo.

[Cust.Req.71] O gestor da aplicação deve ter acesso a estatísticas mensais como:

1. Nº de calibrações internas;
2. Nº de verificações;
3. Nº de calibrações que falham;
4. Nº de reparações;
5. Nº de equipamentos suspensos que são reativados;
6. Custos das calibrações internas;
7. Nº de calibrações externas;
8. Custos das calibrações externas;

Ilustração 35: Requisito novo do projeto "GEIME"

Na atual versão do processo aplicável à Bosch, o estado "mudado" é atribuído a um requisito que foi associado a uma mudança estabelecida na atividade de *change management*. Porém, como nesta fase ainda não há tecnologia que sustente a atividade de *change management*, atribui-se este estado aos requisitos que foram mudados da iteração anterior. Os requisitos que são alterados na presente iteração não têm este estado atribuído, sendo o seu estado "novo".

Negotiation

Devido à insuficiência de recursos e à indisponibilidade dos *stakeholders*, esta atividade é ignorada.

Prioritisation

Devido à insuficiência de recursos e à indisponibilidade dos *stakeholders*, esta atividade é ignorada.

Documentation

Esta atividade está associada à realização de um documento de carácter oficial, que tem o propósito de comunicar o trabalho efetuado no âmbito da engenharia de requisitos. Este documento já existe na Bosch, como pode observar no capítulo anterior, e tem o nome de LHPH.

O LHPH é resultado de todo o trabalho realizado até ao momento pelo *software requirements engineer*, ou seja, o produto que resolve o problema desta experimentação.

Na ilustração 36 deve visualizar a estrutura do documento.

Overview

Chapter	relevant	Not relevant because… (e.g. included in chapter …)
1 History with review and acceptance	☑	
2 Project charter (all subchapters have to be filled out)	☑	
3 Essential project notes (all subchapters have to be filled out)	☐	
Business requirements specifications		
4 Process requirements (current/target)	☑	
5 Functional requirements /data requirements (current/target)	☑	
6 Report requirements	☑	
7 Interfaces (current/target)	☐	
8 User interface	☐	
9 Roles and authorizations	☑	
10 Migration	☑	
11 Archiving	☐	
12 Documentation	☑	
13 Non-functional Requirements	☑	
14 Testing (test plan)	☐	
15 User training	☐	
16 Organizational change management	☐	
17 Installation and operation of the systems	☐	
18 Application security	☑	
19 Additional acceptance criteria	☐	
20 Open points business requirements specifications	☐	
System/performance specifications		
21 Architecture overview (only performance specification))	☐	
22 Process requirements (target)	☑	
23 Functional requirements /data requirements (target)	☑	
24 Report requirements	☑	
25 Interfaces (target)	☐	
26 User interface	☐	
27 Roles and authorizations	☑	
28 Technical specifications	☐	
29 Migration	☑	
30 Archiving	☐	
31 Documentation	☑	
32 Non-functional Requirements	☑	
33 Testing (test plan)	☐	
34 User training	☐	
35 Organizational change management	☐	
36 Installation and operation of the systems	☐	
37 Application security	☑	
38 Additional acceptance criteria	☐	
39 Open points system/performance specifications	☐	
Appendix		
40 Glossary	☐	

Ilustração 36: Estrutura do LHPH do projeto "GEIME"

Este documento foi sujeito a diversas atualizações, que foram colmatando falhas que se evidenciaram. Por consequência, este documento tem pontos que não se aplicam nestas circunstâncias.

Salienta-se neste documento, a descrição do problema do cliente, que é visualizada em parte na ilustração 37.

4.3 Description of problem to be solved from the users' perspective

O gestor da aplicação cria as fichas dos equipamentos com as seguintes informações:

1. NICE;
2. Nº de inventário;
3. Nº FVB;
4. Nº de série;
5. Tipo de equipamento;
6. Modelo do equipamento;
7. Descrição;
8. Estado;
9. Lugar;

O gestor da aplicação cria os modelos.

O gestor da aplicação altera os modelos.

O gestor da aplicação elimina os modelos.

O gestor da aplicação cria os tipos de equipamentos.

O gestor da aplicação altera os tipos de equipamentos.

O gestor da aplicação elimina os tipos de equipamentos.

O gestor da aplicação vê o histórico de cada equipamento.

O gestor da aplicação valida o estado dos equipamentos.

Ilustração 37: Descrição do problema do LHPH do projeto "GEIME"

Já na ilustração 38, deve visualizar alguns requisitos de utilizador especificados.

5.2 Description of functional requirements from the users' perspective

Gestor da aplicação

[Cust.Req.1] O gestor da aplicação deve criar uma ficha do equipamento com as seguintes informações:

1. NICE;
2. Nº de inventário;
3. Nº FVB;
4. Nº de série;
5. Tipo de equipamento;
6. Modelo do equipamento;
7. Descrição;
8. Estado;
9. Lugar;

[Cust.Req.3] O gestor da aplicação deve alterar uma ficha de equipamento.

[Cust.Req.5] O gestor da aplicação deve eliminar uma ficha de equipamento.

[Cust.Req.7] O gestor da aplicação deve ter acesso ao histórico das:

1. Calibrações que um equipamento foi submetido;
2. Verificações que um equipamento foi submetido;
3. Reparações que um equipamento foi submetido;
4. Localizações;

[Cust.Req.8] O gestor da aplicação deve criar um modelo com as seguintes informações:

1. Código;
2. Descrição.

[Cust.Req.9] O gestor da aplicação deve alterar um modelo.

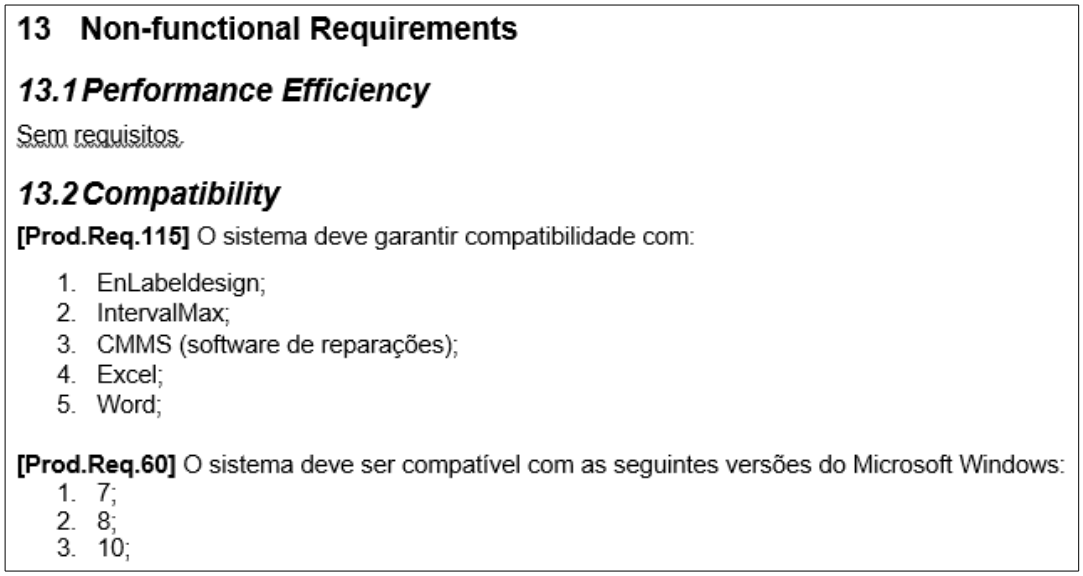
[Cust.Req.10] O gestor da aplicação deve eliminar um modelo.

[Cust.Req.14] O gestor da aplicação deve criar uma secção.

[Cust.Req.15] O gestor da aplicação deve associar um equipamento a uma linha de produção.

Ilustração 38: Especificação de requisitos de utilizador no LHPH do projeto "GEIME"

Na ilustração 39, pode visualizar um número reduzido de requisitos não funcionais, divididos por algumas das categorias disponibilizadas pelo documento.



13 Non-functional Requirements

13.1 Performance Efficiency

Sem requisitos

13.2 Compatibility

[Prod.Req.115] O sistema deve garantir compatibilidade com:

1. EnLabeldesign;
2. IntervalMax;
3. CMMS (software de reparações);
4. Excel;
5. Word;

[Prod.Req.60] O sistema deve ser compatível com as seguintes versões do Microsoft Windows:

1. 7;
2. 8;
3. 10;

Ilustração 39: Especificação de requisitos de sistema não funcionais no LHPH do projeto "GEIME"

Validation

Nesta atividade, o LHPH é analisado e validado pelo responsável do *software requirements engineer*, que assume os papeis de *project manager* e de *client* simultaneamente.

Lifecycle Management

Esta atividade não é desempenhada nesta fase.

Traceability

Nesta atividade, a *table of horizontal traceability* é atualizada com as mudanças efetuadas e com os novos requisitos desenvolvidos na atual iteração. Na ilustração 40, deve visualizar a *table of horizontal traceability*, com os requisitos da nova versão do sistema de informação.

ID Requisito	Versão 1		Versão 2	
	Estado	Descrição	Estado	Descrição
Coluna1	Coluna2	Coluna3	Coluna4	Coluna5
[Cust.Req.1]	Novo	O administrador deve criar uma ficha do equipamento com as seguintes informações: 1. NICE; 2. Nº de inventário; 3. Nº FVB; 4. Nº de série; 5. Tipo de equipamento; 6. Modelo do equipamento; 7. Descrição; 8. Estado; 9. Lugar;	Alterado	O gestor da aplicação deve criar uma ficha do equipamento com as seguintes informações: 1. NICE; 2. Nº de inventário; 3. Nº FVB; 4. Nº de série; 5. Tipo de equipamento; 6. Modelo do equipamento; 7. Descrição; 8. Estado; 9. Lugar;
[Cust.Req.2]	Novo	O administrador deve criar um modelo com as seguintes informações: 1. Código; 2. Descrição.		
[Cust.Req.3]	Novo	O administrador deve alterar uma ficha de equipamento.	Alterado	O gestor da aplicação deve alterar uma ficha de equipamento.
[Cust.Req.4]	Novo	O administrador deve alterar modelos.		
[Cust.Req.5]	Novo	O administrador deve eliminar uma ficha de equipamento.	Alterado	O gestor da aplicação deve eliminar uma ficha de equipamento.
[Cust.Req.6]	Novo	O administrador deve eliminar modelos		

Ilustração 40: Table of horizontal traceability do projeto "GEIME" (2ª iteração)

Change Management

Esta atividade não é desempenhada nesta fase.

Allocation

Todos artefactos desenvolvidos ou atualizados nesta iteração são alocados na diretoria O:/.

4.2.3 Discussão de Resultados

Antes de qualquer averiguação, deve ser referido que o problema associado a esta primeira experimentação desta dissertação não tem a complexidade necessária para pôr completamente à prova o processo aplicável à Bosch. Em adição, para se tirar todo o partido dos benefícios do processo concebido no capítulo 3, é necessário dar tempo à experimentação do processo, porque há a tendência do processo começar a ser mais eficaz e eficiente à medida que o número de iterações do processo vai aumentando. Como começam a haver mais requisitos, existe maior potencial de reutilização. Já que a unidade curricular de

dissertação é referente a 40 ects[16], e ao período de um ano letivo, esta experimentação pode ser considerada ligeira.

Nesta experimentação é difícil avaliar se a abordagem utilizada é mais eficaz e eficiente do que um levantamento e desenvolvimento dos requisitos de raiz para a nova versão do sistema de informação GEIME. É fácil argumentar que o facto de se fazer apenas uma iteração torna-se, por consequência, menos árduo do que fazer duas iterações. Contudo, é necessário estar consciente que a abordagem utilizada também dá um conhecimento sobre o domínio aplicacional ao *software requirements engineer*, que facilita a comunicação com futuros utilizadores. Consequentemente, isto resulta em mais assertividade na recolha e desenvolvimento dos requisitos.

Como só se levantaram os requisitos para futuramente se desenvolver a nova versão do GEIME, é difícil avaliar se a atividade de *elicitation* foi efetuada com qualidade, uma vez que muitas vezes os futuros utilizadores do sistema só detetam algumas necessidades que possuem, depois do sistema estar já no ativo.

Relativamente à atividade de *elaboration*, é seguro afirmar-se que foram desenvolvidos requisitos escritos em linguagem natural, com maior qualidade e menos ambíguos do que se não tivesse sido utilizado o processo aplicável à Bosch. A atividade de *traceability* também dá mais segurança ao *software requirements engineer*, na medida que este tem mais consciência que mudar um requisito, resulta em consequentes mudanças em outros requisitos, que são facilmente reconhecidos com as relações identificadas. Isto reduz a probabilidade de se preservarem inconsistências nos requisitos.

A atribuição de um id e de um estado a cada requisito também dá uma grande sobriedade ao *software requirements engineer*, na medida que se reduz o risco de este começar a confundir certos componentes do sistema, visto estar a tratar instâncias abstratas.

Apesar de todas as incertezas, o processo aplicável à Bosch foi útil na resolução do problema proposto. Os resultados do processo aplicável refletem-se numa satisfação do estagiário, uma vez que é sempre bom ter alguma estrutura que suporte o seu trabalho. Por outro lado, o seu responsável demonstrou satisfação com uma especificação de requisitos "organizada" e "uniformizada".

Focando agora a eficiência, o *software requirements engineer* denotou um decréscimo acentuado da dificuldade do levantamento e desenvolvimento de requisitos na segunda iteração. Isto resultou também numa diminuição dos recursos utilizados, para a sua realização em horas-homem, o que prova a teoria do autor desta dissertação que é mais fácil mudar algo do que fazer-lo de raiz.

---

16 Ects é a unidade de medida do trabalho do estudante na Universidade do Minho (1 ano letivo = 60 ects)

4.3 Divulgação do Trabalho na Bosch

Finalizada a primeira experimentação, surge a necessidade de apresentar o processo aplicável à Bosch (e como ele se comportou) aos interessados. Como se trata de um processo, que é concebido com o propósito de facilitar uma atividade no seio do departamento de CI-CWR1, é útil existir uma discussão entre o autor desta dissertação de mestrado e os colaboradores interessados no potencial que esta promete oferecer, nomeadamente a secção de CI-CWR12 (também conhecida como a secção responsável pelo apoio à organização com o desenvolvimento de sistemas de informação locais).

Para a divulgação do trabalho, três reuniões são agendadas e realizadas na sala de reuniões do departamento. A primeira reunião corresponde à apresentação do processo aplicável à Bosch. A segunda reflete-se na continuação da apresentação da reunião anterior e numa discussão sobre determinadas atividade ou tarefas do processo. A terceira é referente à divulgação da experimentação efetuada, onde participa também o *software requirements engineer* destacado, para dar também o seu parecer sobre algumas questões referentes ao processo.

Na seguinte hiperligação pode descarregar a apresentação elaborada no âmbito deste capítulo: https://goo.gl/xGGe2p.

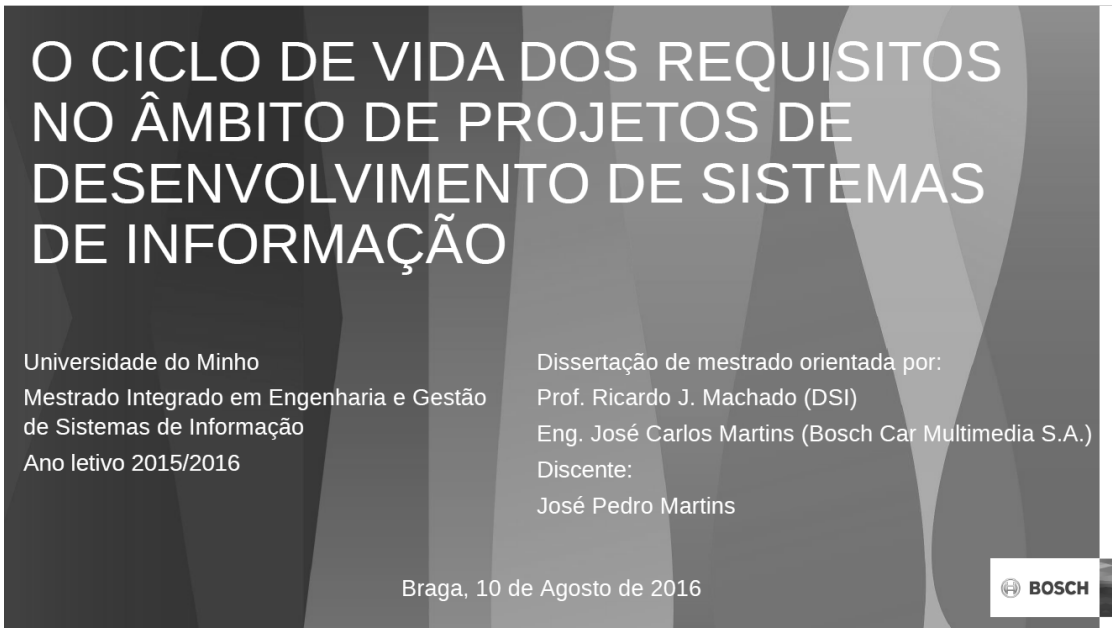


Ilustração 41: Primeiro diapositivo da apresentação realizada na Bosch

Nestas três reuniões, algumas críticas construtivas e sugestões são proferidas pelos colaboradores associados à secção CI-CWR12, e pelo estagiário que assumiu o papel de *software requirements engineer* na primeira experimentação do processo aplicável à Bosch.

Estas incidências resultam em mudanças, de pequena ou grande dimensão nas atividades e artefactos do processo, de forma a estes constituintes do processo ficarem mais potenciados para a sua finalidade.

É nestas reuniões que se constata que o processo, como ele está no momento, não suporta paridade de projetos, nem projetos designados para conceber mais do que um sistema de informação. Este problema resulta numa revolução no processo concebido até este momento, incidente mais na componente de *requirements management*.

Esta mudança extingue os artefactos "tabela de identificação da versão do sistema de informação de requisitos" e "informação geral". Além disso, muda o propósito do artefacto *table of horizontal traceability*, tornando este bem mais relevante, devido à necessidade de existir uma gestão do ciclo de vida dos requisitos por versões. Isto faz a tarefa privada deste artefacto, ser transferida para a atividade de *lifecycle management*.

Por outro lado, assume também a peculiaridade do extinto "informação geral", como artefacto de apoio à atividade de *elaboration*, para a reutilização de requisitos. Em adição, nasce o paradoxo de um requisito assumir o papel de versão de requisito nos artefactos de *state table* e *table of horizontal traceability*.

As Atividades, artefactos e atores do processos foram entretanto traduzidos para inglês por indicação do orientador desta dissertação de mestrado. Houve também uma mudança da notação utilizada para a modelação do processo, sendo agora utilizados os diagramas de atividade da linguagem de modelação UML, potencializada para *workflow diagrams*.

4.4 Experimentação 2 - "ALR"

O ALR, acrónimo para Active Lot Release, será uma aplicação que substituirá uma rudimentar folha de cálculo (no ativo), que tem o propósito de avaliar os componentes de um lote no final do processo produtivo, antes de este ser enviado para o cliente. Com isto, o objetivo deste projeto é converter uma folha de cálculo (que é manual), em algo totalmente automatizado e em *real time*, de forma a minimizar gastos para esta atividade, conseguindo os resultados expectáveis em menos tempo (o cenário desejável é coincidir com o final do processo produtivo).

4.4.1 Descrição do Problema

Sendo este um projeto que começou há alguns meses, reuniões semanais foram efetuadas, entre pessoas do domínio do problema, e pessoas destacadas para o desenvolvimento da solução, com a finalidade de se levantarem os requisitos para o sistema de informação em equação. No entanto, constata-se a ausência

da utilização de uma metodologia para o efeito, resultando numa dificuldade em determinar exatamente, quais serão as características do sistema de informação objetivado no âmbito deste projeto, dando até a sensação que discute sempre as mesmas temáticas, que se julgaram estipuladas.

Com as ocorrências referidas no parágrafo acima, reconhece-se utilidade nesta dissertação para mitigar este levantamento de requisitos *ad hoc*, com as conceções efetuadas nas atividades de *elicitation* e *elaboration* do processo aplicável à Bosch.

Uma vez que foram efetuadas modificações no processo aplicável à bosch, após a primeira experimentação deste, nasce também a necessidade de experimentar, num contexto mais aplicacional, as mudanças efetuadas, especialmente na componente de *management*.

4.4.2 Processo Aplicável à Bosch em Prática

Inception

Uma vez que o que as potencialidades desta dissertação de mestrado só são sugeridas quando o projeto já está iniciado, não são efetuados esforços nesta atividade.

Elicitation

Inicialmente, o desenvolvedor desta dissertação de mestrado, é inserido nas reuniões que são realizadas todas as semanas, com o propósito perceber quais são os requisitos para o sistema de informação "ALR". Consequentemente, todos os "artefactos" desenvolvidos nestas reuniões são fornecidos ao *software requirements engineer* para facilitar a recolha dos requisitos. Na ilustração 42 deve visualizar um excerto de um destes "artefactos" fornecidos.

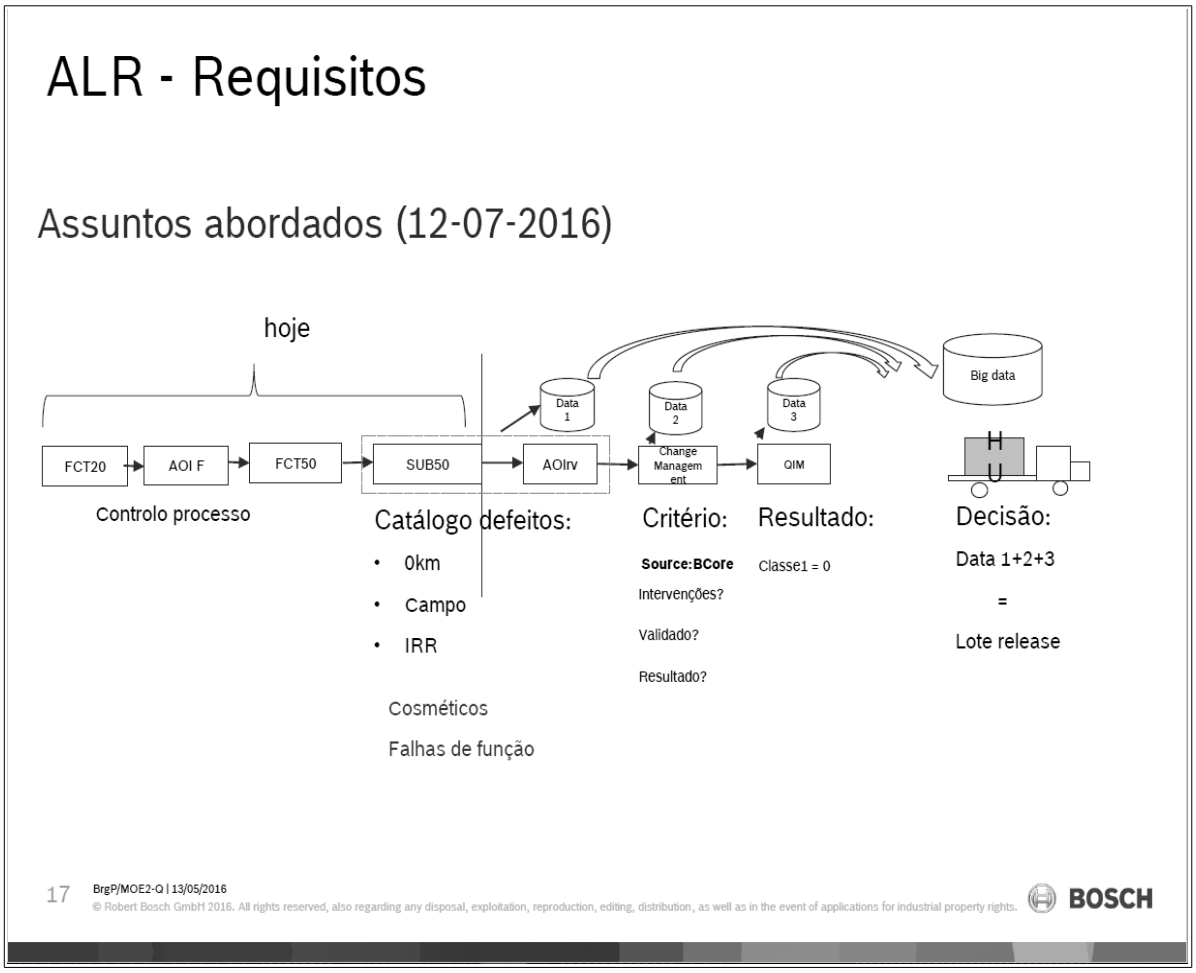


Ilustração 42: Documentação referente ao ALR fornecido para a atividade de elicitation

Posteriormente, duas entrevistas de levantamento de requisitos são efetuadas a pessoas do domínio do problema, de modo a esclarecer certas ambiguidades.

A informação recolhida no âmbito desta atividade é preservada num bloco de notas, e é posteriormente transcrita sobre a forma de descrições no *artifact of requirements handling*. Na ilustração 43 deve visualizar um exemplo de descrição do *artifact of requirements handling*.

2. Analisar Lotes

O A.L.R. será um sistema que irá incidir no final de um processo concebido para produzir um lote (palete no final do processo). Por consequência, este componente terá como resultado, uma decisão referente à libertação do lote para o cliente, tomada com uma inferência dos dados relativos ao processo com um padrão de qualidade pré-estabelecido. Um lote nasce bloqueado, o que indica que ainda não está preparado para ser libertado. Consequentemente, este facto indica que todos os lotes têm ser sujeitos a um mecanismo de avaliação da qualidade do lote. No final desta avaliação, o lote está libertado ou bloqueado.

O A.L.R. deverá fazer a avaliação já a cima mencionada, salvo algumas exceções onde o sistema deverá permitir a intervenção humana para libertar ou bloquear deliberadamente um lote.

Ilustração 43: Descrição resultante da atividade de elicitation do projeto "ALR"

Como se identificam diferentes componentes do ALR, há a preocupação de dividir estas descrições pelos componentes identificados.

Elaboration

Inicialmente, as descrições transcritas para o *artifact of requirements handling* são avaliadas, para serem identificados os requisitos com um padrão ([Cust.Req.X.VX] para requisitos de utilizador e [Prod.Req.X.VX] para requisitos de sistema). Atente que nesta fase já é utilizada a gestão do ciclo de vida dos requisitos por versões, então o "VX" já se aplica aqui contrariamente ao observado na primeira experimentação.

De seguida, são aplicados os padrões de sintaxe, como é indicado no capítulo 3, conceção de um processo aplicável à Bosch. Na ilustração 44 deve visualizar um fragmento do trabalho desenvolvido no *artifact of requirements handling* até ao momento.

Requisitos de Utilizador

[Cust.Req.4.V1] O PSP tem de fornecer dados da formação de uma palete para a libertação automática de um lote.

[Cust.Req.5.V1] O OracleCMDB tem de fornecer dados do resultado da passagem dos produtos nos postos da produção para a libertação automática de um lote.

[Cust.Req.6.V1] O OracleCMDB (TAS) tem de fornecer dados dos testes a fazer em cada posto para a libertação automática de um lote.

[Cust.Req.7.V1] O B-Core tem de fornecer dados das intervenções feitas ao processo para a libertação automática de um lote.

[Cust.Req.8.V1] O QIM tem de fornecer dados dos testes ao produto em modo cliente para a libertação automática de um lote.

Ilustração 44: Requisitos de utilizador com padrão de sintaxe do projeto "ALR"

No final da tarefa de escrita de requisitos de utilizador com os padrões de sintaxe, é necessário analisar o documento com o intuito de identificar conflitos adjacentes à natureza dos próprios requisitos, assim como originários de diferentes visões dos utilizadores. Os conflitos identificados, associados à natureza dos requisitos são posteriormente resolvidos. Como não há tempo nem disponibilidade dos *stakeholders* para se realizar a atividade de *negotiation*, os conflitos associados a diferentes visões de *stakeholders* são resolvidos com o aconselhamento de um *stakeholder*, que tem as caraterísticas necessárias para exercer o papel de *expert* do processo aplicável à Bosch.

No momento em que os requisitos já não apresentam quaisquer conflitos, o *software requirements engineer* reúne-se com o *information systems manager* para delimitar os limites que o sistema poderá

possuir. De seguida, os requisitos de sistemas são desenvolvidos, utilizando os requisitos de utilizador e as limitações impostas pelo *information systems manager* como referência, e aplicando os padrões de sintaxe e estratégias para resolução de conflitos utilizados nos requisitos de utilizador. Aqui, separar os requisitos funcionais dos não funcionais é também uma ajuda na organização do documento. Na ilustração 45 pode visualizar um exemplo de derivação de um requisito de utilizador em um requisito de sistema para esta experimentação.

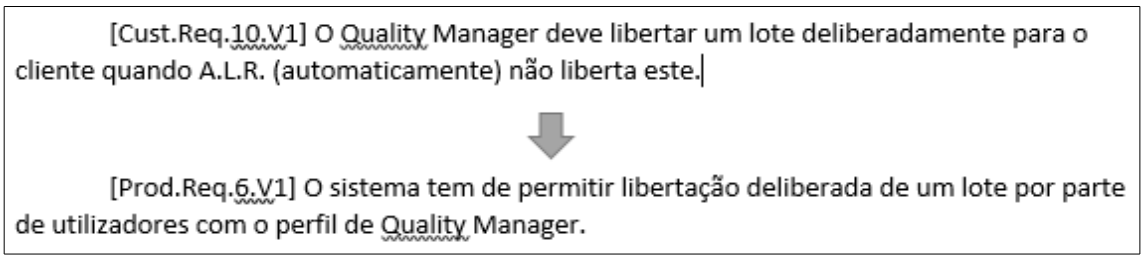


Ilustração 45: Derivação de requisito de utilizador em requisito de sistema do projeto "ALR"

Para finalizar a atividade de *elaboration* resta analisar novamente o *artifact of requirements handling*, para se identificarem conflitos nos requisitos de sistema. Como não são identificados quaisquer conflitos, esta atividade termina.

Negotiation

Esta atividade não é experimentada no âmbito do projeto "ALR".

Prioritization

Esta atividade não é experimentada no âmbito do projeto "ALR".

Documentation

Esta atividade não é experimentada no âmbito do projeto "ALR".

Validation

Esta atividade não é experimentada no âmbito do projeto "ALR".

Lifecycle Management

No início desta atividade, os requisitos levantados e desenvolvidos no âmbito do projeto associado a esta experimentação, são recolhidos do *artifact of requirement handling*, para preencher os dois artefactos que são utilizados nesta atividade.

De seguida, o artefacto *state table* é preenchido pelo *software requirements engineer*, com os IDs dos requisitos retirados dos *artifact of requirements handling* (sem a versão). Após isto, o *software requirements engineer* preenche a coluna referente ao estado "novo" com um "ALR-V1" para se identificar a versão do requisito e em que projeto está presente. Na ilustração 46 pode visualizar parte da *state table* realizada.

ID Requisito	Estado								
	Novo	Reutilizado	Mudado	Selecionado	Rejeitado	Documentado	Validado	Implementado	Para Mudança
Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6	Coluna7	Coluna8	Coluna9	Coluna10
[Cust.Req.1]	[ALR-V1]								
[Cust.Req.2]	[ALR-V1]								
[Cust.Req.3]	[ALR-V1]								
[Cust.Req.4]	[ALR-V1]								
[Cust.Req.5]	[ALR-V1]								
[Cust.Req.6]	[ALR-V1]								
[Cust.Req.7]	[ALR-V1]								
[Cust.Req.8]	[ALR-V1]								
[Cust.Req.9]	[ALR-V1]								
[Cust.Req.10]	[ALR-V1]								
[Cust.Req.11]	[ALR-V1]								
[Cust.Req.12]	[ALR-V1]								
[Cust.Req.13]	[ALR-V1]								
[Cust.Req.14]	[ALR-V1]								
[Cust.Req.15]	[ALR-V1]								
[Cust.Req.16]	[ALR-V1]								
[Cust.Req.17]	[ALR-V1]								
[Cust.Req.18]	[ALR-V1]								
[Cust.Req.19]	[ALR-V1]								
[Cust.Req.20]	[ALR-V1]								

Ilustração 46: State table do projeto "ALR"

Para finalizar esta atividade resta apenas desenvolver o artefacto *table of horizontal traceability*. Primeiro é necessário que o *software requirements engineer* preencha a coluna alusiva aos id dos requisitos. A seguir, este individuo preenche a coluna referente à primeira versão dos requisitos com a descrição proveniente do *artifact of requirements handling*. Na ilustração 47 deve visualizar parte de um excerto da *table of horizontal traceability* efetuada nesta atividade.

ID Requisito	Versão 1		Versão 2		Versão 3	
	Descrição	Observações	Descrição	Observações	Descrição	Observações
Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6	Coluna7
[Cust.Req.1]	Um utilizador humano tem de iniciar sessão no sistema.					
[Cust.Req.2]	Um utilizador humano deve terminar sessão no sistema.					
[Cust.Req.3]	Um utilizador humano tem de ter um perfil associado.					
[Cust.Req.4]	O PSP tem de fornecer dados da formação de uma palete para a libertação automática de um lote.					
[Cust.Req.5]	O OracleCMDB tem de fornecer dados do resultado da passagem dos produtos nos postos da produção para a libertação automática de um lote.					
[Cust.Req.6]	O OracleCMDB (TAS) tem de fornecer dados dos testes a fazer em cada posto para a libertação automática de um lote.					
[Cust.Req.7]	O B-Core tem de fornecer dados das intervenções feitas ao processo para a libertação automática de um lote.					

Ilustração 47: Table of horizontal traceability do projeto "ALR"

Traceability

Assim como na atividade anterior, o *software requirements engineer* levanta todos os requisitos desenvolvidos nesta experimentação do *artifact of requirements handling* de modo a possibilitar a realização dos artefactos utilizados nesta atividade.

Como pode reparar no capítulo 3, conceção de um processo aplicável à Bosch, esta atividade é constituída pelas componentes de *requirements relations* e *vertical treceability*.

Relativamente à componente de *requirements relations*, o *software requirements engineer*, desenvolve uma *matrix of requirements relations* para relações entre requisitos de utilizador e requisitos de sistema, e outra para relações entre requisitos de sistema e requisitos de sistema. Desenvolve também um grafo para identificação de relações entre requisitos (sistema – sistema). De modo a concluir esta componente resta apenas elaborar o artefacto *tables of requirements relations*.

Como pode visualizar na ilustração 48, na *matrix of requirements relations* das relações utilizador - sistema, o *software requirements engineer* preenche a coluna, respeitante aos requisitos afetantes com o ID e descrição dos requisitos de utilizador e a linha relativa ao requisitos afetados com o ID e descrição dos requisitos de sistema utilizando a informação do *artifact of requirements handling*. Depois identifica as relações entre os requisitos com o "x" na célula que corresponde a um *match*.

	ID Requisito (afetado) →	[Prod.Req.1.V1]	[Prod.Req.2.V1]	[Prod.Req.3.V1]	[Prod.Req.4.V1]	[Prod.Req.5.V1]
ID Requisito (afetante) ↓	Descrição	O sistema tem de garantir a autenticação de um utilizador, através da introdução de um nome de utilizador e uma palavra-chave.	O sistema tem de permitir a um utilizador terminar a sua sessão.	O sistema tem de diferenciar o perfil do utilizador na autenticação deste.	A palavra-chave de um utilizador tem de ter pelo menos uma letra maiúscula e um caractere especial.	O sistema tem de libertar um lote automaticamente, com base numa inferência dos padrões de qualidade expressos em regras do sistema com os dados provenientes dos: • PSP; • OracleCMDB; • B-Core; • QIM; • USP.
[Cust.Req.1]	Um utilizador humano tem de iniciar sessão no sistema.	x				
[Cust.Req.2]	Um utilizador humano deve terminar sessão no sistema.		x			
[Cust.Req.3]	Um utilizador humano tem de ter um perfil associado.			x		
[Cust.Req.4]	O PSP tem de fornecer dados da formação de uma palete para a libertação automática de um lote.					x
[Cust.Req.5]	O OracleCMDB tem de fornecer dados do resultado da passagem dos produtos nos postos da produção para a libertação automática de um lote.					x
[Cust.Req.6]	O OracleCMDB (TAS) tem de fornecer dados dos testes a fazer em cada posto para a libertação automática de um lote.					x
[Cust.Req.7]	O B-Core tem de fornecer dados das intervenções feitas ao processo para a libertação automática de um lote.					x

Ilustração 48: Matrix of requirements relations (utilizador - sistema) do projeto "ALR"

O mesmo procedimento é efetuado na *matrix of requirements relations* das relações sistema – sistema, com a diferença de agora a coluna referente aos requisitos afetantes ser preenchida com requisitos

de sistema e o "x" ser substituído pelo tipo de relação na célula correspondente ao match. Na ilustração 49 deve visualizar uma passagem deste artefacto.

	ID Requisito (afetado) →	[Prod.Req.1.V1]	[Prod.Req.2.V1]	[Prod.Req.3.V1]	[Prod.Req.4.V1]	[Prod.Req.5.V1]
ID Requisito (afetante) ↓	Descrição	O sistema tem de garantir a autenticação de um utilizador, através da introdução de um nome de utilizador e uma palavra-chave.	O sistema tem de permitir a um utilizador terminar a sua sessão.	O sistema tem de diferenciar o perfil do utilizador na autenticação deste.	A palavra-chave de um utilizador tem de ter pelo menos uma letra maiúscula e um caractere especial.	O sistema tem de libertar um lote automaticamente, com base numa inferência dos padrões de qualidade expressos em regras do sistema com os dados provenientes dos: • PSP; • OracleCMDB; • B-Core; • QIM; • USP.
[Prod.Req.1.V1]	O sistema tem de garantir a autenticação de um utilizador, através da introdução de um nome de utilizador e uma palavra-chave.	x	<<é_pré_requisito_de>>	<<e>>	<<e>>	
[Prod.Req.2.V1]	O sistema tem de permitir a um utilizador terminar a sua sessão.		x			
[Prod.Req.3.V1]	O sistema tem de diferenciar o perfil do utilizador na autenticação deste.	<<e>> / <<aumenta_custo_de>>		x		
[Prod.Req.4.V1]	A palavra-chave de um utilizador tem de ter pelo menos uma letra maiúscula e um caractere especial.	<<e>> / <<aumenta_custo_de>>			x	
[Prod.Req.5.V1]	O sistema tem de libertar um lote automaticamente, com base numa inferência dos padrões de qualidade expressos em regras do sistema com os dados provenientes dos: • PSP; • OracleCMDB; • B-Core; • QIM; • USP.					x

Ilustração 49: Matrix of requirements relations (sistema - sistema) do projeto "ALR"

Na *matrix of requirements relations* (sistema – sistema) já estão identificadas as relações sistema – sistema, contudo é valorizável que o *software requirements engineer* represente estas graficamente. Para isto, um grafo é feito com recurso ao software yEd graph editor. A ilustração 50 corresponde a uma demonstração deste grafo.

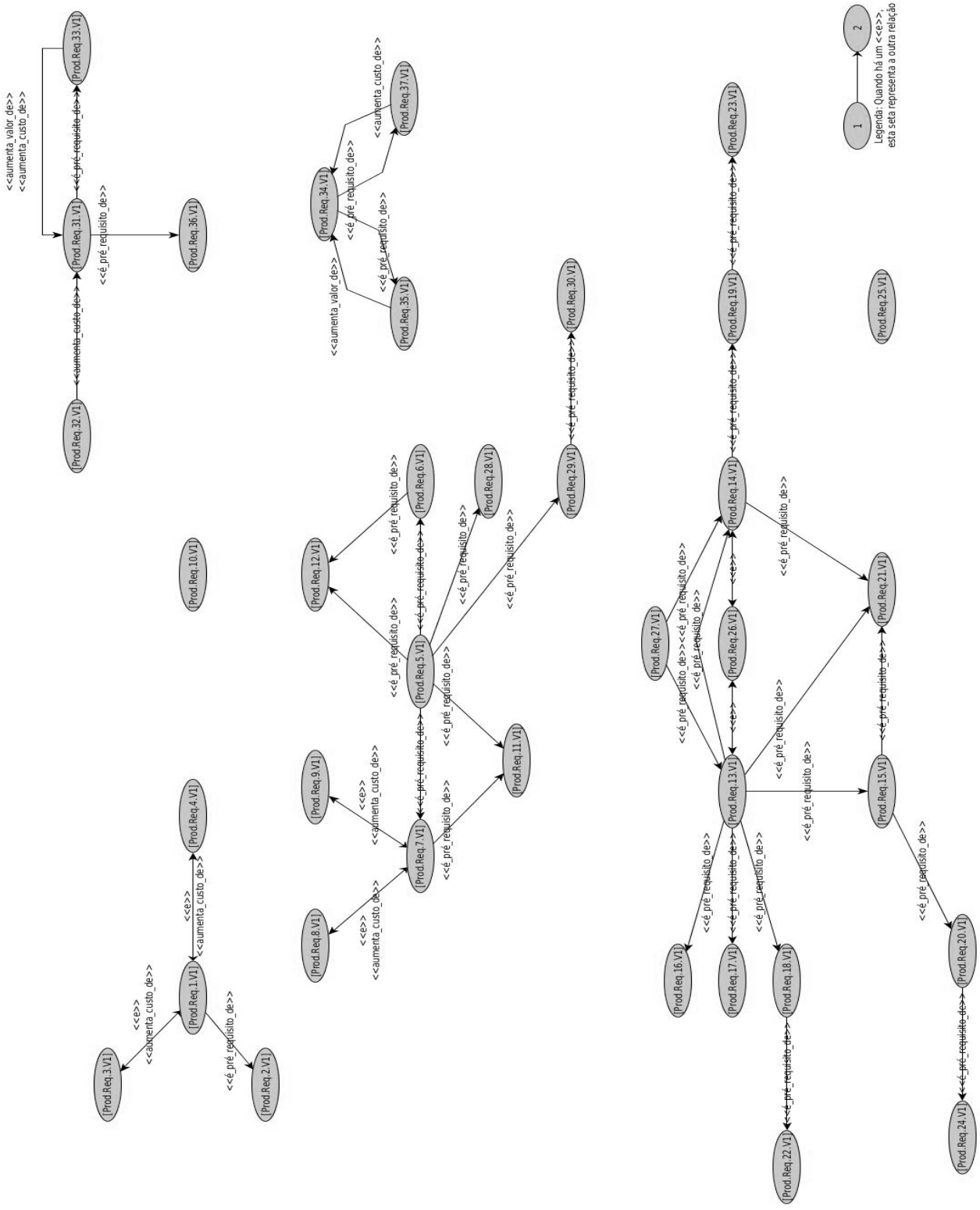


Ilustração 50: Grafo alusivo às relações entre requisitos de sistema do projeto "ALR"

Com a informação recolhida do *artifact of requirements handling* e consultando a *state table* desenvolvida na atividade anterior, o *software requirements engineer* preenche as colunas com a informação

ajustável de ambas as folhas de cálculo, do artefacto *tables of requirements relations*.

Agora, utilizando a informação produzida nos artefactos desenvolvidos até ao momento no decorrer desta atividade, faz-se a atribuição das relações na folha de cálculo respeitante aos requisitos de utilizador, com o ID do requisito de sistema relacionável, como pode visualizar na ilustração 51.

Relações de Requisitos de Utilizador

Coluna1	Coluna2	Coluna3	Coluna4	Coluna6	Coluna7
ID Requisito	Descrição	Observações	Estado	Relações com Requisitos de Utilizador	Relações com Requisitos de Sistema
[Cust.Req.1.V1]	Um utilizador humano tem de iniciar sessão no sistema.		Novo		[Prod.Req.1.V1]
[Cust.Req.2.V1]	Um utilizador humano deve terminar sessão no sistema.		Novo		[Prod.Req.2.V1]
[Cust.Req.3.V1]	Um utilizador humano tem de ter um perfil associado.		Novo		[Prod.Req.3.V1]
[Cust.Req.4.V1]	O PSP tem de fornecer dados da formação de uma palete para a libertação automática de um lote.		Novo		[Prod.Req.5.V1]
[Cust.Req.5.V1]	O OracleCMDB tem de fornecer dados do resultado da passagem dos produtos nos postos da produção para a libertação automática de um lote.		Novo		[Prod.Req.5.V1]
[Cust.Req.6.V1]	O OracleCMDB (TAS) tem de fornecer dados dos testes a fazer em cada posto para a libertação automática de um lote.		Novo		[Prod.Req.5.V1]
[Cust.Req.7.V1]	O B-Core tem de fornecer dados das intervenções feitas ao processo para a libertação automática de um lote.		Novo		[Prod.Req.5.V1]
[Cust.Req.8.V1]	O QIM tem de fornecer dados dos testes ao produto em modo cliente para a libertação automática de um lote.		Novo		[Prod.Req.5.V1]
[Cust.Req.9.V1]	O USP tem de fornecer dados dos testes ao produto em modo cliente para a libertação automática de um lote.		Novo		[Prod.Req.5.V1]
[Cust.Req.10.V1]	O Quality Manager deve libertar um lote deliberadamente para o cliente quando A.L.R. (automaticamente) não liberta este.		Novo		[Prod.Req.6.V1]
[Cust.Req.11.V1]	O Quality Manager deve bloquear um lote deliberadamente quando o ALR (automaticamente) liberta este.		Novo		[Prod.Req.7.V1]
[Cust.Req.12.V1]	O Quality Analyst deve bloquear um lote deliberadamente quando o ALR (automaticamente) liberta este.		Novo		[Prod.Req.7.V1]

Ilustração 51: Tables of requirements relations (requisitos de utilizador) do projeto "ALR"

O mesmo é feito de seguida na folha de cálculo referente aos requisitos de sistema, com a variante de as relações serem identificadas com o tipo de relação associado ao ID do requisito de sistema relacionável, como pode verificar na ilustração 52.

Relações de Requisitos de Sistema

Coluna1	Coluna2	Coluna3	Coluna4	Coluna6	Coluna7
ID Requisito	Descrição	Observações	Estado	Relações com Requisitos de Utilizador	Relações com Requisitos de Sistema
[Prod.Req.1.V1]	O sistema tem de garantir a autenticação de um utilizador, através da introdução de um nome de utilizador e uma palavra-chave.		Novo	[Cust.Req.1.V1]	[<<é_pré_requisito_de>>_Prod.Req.2.V1], [<<e>>_Prod.Req.3.V1], [<<e>>_Prod.Req.4.V1]
[Prod.Req.2.V1]	O sistema tem de permitir a um utilizador terminar a sua sessão.		Novo	[Cust.Req.2.V1]	
[Prod.Req.3.V1]	O sistema tem de diferenciar o perfil do utilizador na autenticação deste.		Novo	[Cust.Req.3.V1]	[<<e>>_Prod.Req.1.V1], [<<aumenta_custo_de>>_Prod.Req.1.V1]
[Prod.Req.4.V1]	A palavra-chave de um utilizador tem de ter pelo menos uma letra maiúscula e um caractere especial.		Novo		[<<e>>_Prod.Req.1.V1], [<<aumenta_custo_de>>_Prod.Req.1.V1]
[Prod.Req.5.V1]	O sistema tem de libertar um lote automaticamente, com base numa inferência dos padrões de qualidade expressos em regras do sistema com os dados provenientes dos: • PSP; • OracleCMDB; • B-Core; • QIM; • USP.		Novo	[Cust.Req.4.V1], [Cust.Req.5.V1], [Cust.Req.6.V1], [Cust.Req.7.V1], [Cust.Req.8.V1], [Cust.Req.9.V1]	[<<é_pré_requisito_de>>_Prod.Req.6.V1], [<<é_pré_requisito_de>>_Prod.Req.7.V1], [<<é_pré_requisito_de>>_Prod.Req.11.V1], [<<é_pré_requisito_de>>_Prod.Req.12.V1], [<<é_pré_requisito_de>>_Prod.Req.28.V1], [<<é_pré_requisito_de>>_Prod.Req.29.V1]
[Prod.Req.6.V1]	O sistema tem de permitir libertação deliberada de um lote por parte de utilizadores com o perfil de Quality Manager.		Novo	[Cust.Req.10.V1]	[<<é_pré_requisito_de>>_Prod.Req.12.V1]
[Prod.Req.7.V1]	O sistema tem de permitir bloqueio deliberado de um lote por parte de utilizadores com os seguintes perfis: • Quality Manager; • Quality Analyst.		Novo	[Cust.Req.11.V1], [Cust.Req.12.V1]	[<<e>>_Prod.Req.8.V1], [<<e>>_Prod.Req.9.V1], [<<é_pré_requisito_de>>_Prod.Req.11.V1]
[Prod.Req.8.V1]	O sistema tem de exigir a inserção de um comentário quando um lote é libertado deliberadamente.		Novo	[Cust.Req.13.V1]	
[Prod.Req.9.V1]	O sistema tem de exigir o anexo de um ficheiro com uma análise do risco quando um lote é libertado deliberadamente.		Novo	[Cust.Req.13.V1]	
[Prod.Req.10.V1]	O sistema tem de enviar um aviso para os utilizadores com o perfil de "Team Leader" sempre que um lote está completo.		Novo	[Cust.Req.14.V1]	

Ilustração 52: Tables of requirements relations (requisitos de sistema) do projeto "ALR"

Focando agora atenções na componente de *vertical traceability*, como foram desenvolvidos diagramas de caso de uso neste projeto, é possível experimentar o mecanismo de rastreabilidade vertical do processo aplicável à Bosch. Para esse fim, o artefacto de nome *table of vertical traceability* é primeiramente preenchido com os IDs, descrições e estado dos requisitos de sistema, retirados dos *artifact of requirements handling* e *state table*. Posteriormente, olhando para os diagramas de caso de uso incluídos no *artifact of requirements handling*, o *software requirements engineer* faz uma associação entre os requisitos de sistema e os casos de uso incluídos nos diagramas, utilizando o ID do caso de uso relacionável. Na ilustração 53 deve visualizar um excerto da *table of vertical traceability* desenvolvida.

ID Requisito	Descrição	Observações	Estado	Instâncias de Fases Avançadas do Processo de Engenharia de Software			
				Análise	Conceção	Implementação	Teste
Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6	Coluna7	Coluna8
[Prod.Req.1.V1]	O sistema tem de garantir a autenticação de um utilizador, através da introdução de um nome de utilizador e uma palavra-chave.		Novo	[UC 1.1]			
[Prod.Req.2.V1]	O sistema tem de permitir a um utilizador terminar a sua sessão.		Novo	[UC 1.2]			
[Prod.Req.3.V1]	O sistema tem de diferenciar o perfil do utilizador na autenticação deste.		Novo				
[Prod.Req.4.V1]	A palavra-chave de um utilizador tem de ter pelo menos uma letra maiúscula e um caractere especial.		Novo				
[Prod.Req.5.V1]	O sistema tem de libertar um lote automaticamente, com base numa inferência dos padrões de qualidade expressos em regras do sistema com os dados provenientes dos: • PSP; • OracleCMDB; • B-Core; • QIM; • USP.		Novo	[UC 2.2.1]			
[Prod.Req.6.V1]	O sistema tem de permitir libertação deliberada de um lote por parte de utilizadores com o perfil de Quality Manager.		Novo	[UC 2.2]			
[Prod.Req.7.V1]	O sistema tem de permitir bloqueio deliberado de um lote por parte de utilizadores com os seguintes perfis: • Quality Manager; • Quality Analyst.		Novo	[UC 2.3]			
[Prod.Req.8.V1]	O sistema tem de exigir a inserção de um comentário quando um lote é libertado deliberadamente.		Novo	[UC 2.2.2]			
[Prod.Req.9.V1]	O sistema tem de exigir o anexo de um ficheiro com uma análise do risco quando um lote é libertado deliberadamente.		Novo	[UC 2.2.3]			

Ilustração 53: Table of vertical traceability do projeto "ALR"

Allocation

Todos os artefactos desenvolvidos no âmbito deste projeto são alocados no O:/. O *artefactos table of horizontal traceability* e *state table* ficam na raiz da pasta criada para este processo. Já os restantes são incluídos numa pasta denominada ALR.

Change Management

Esta atividade não é experimentada no âmbito do projeto “ALR”.

4.4.3 Discussão de Resultados

Se o problema inerente ao projeto GEIME já não permitia experimentar o processo aplicável à Bosch como um todo, o problema que surgiu na organização para o projeto ALR é ainda mais elementar, uma vez que só permitiu experimentar as atividades de *elicitation* e *elaboration*. Então, assim como no projeto GEIME,

esta experimentação pode ser também considerada ligeira, e objetivada para experienciar certos componentes do processo aplicável à Bosch que não foram testados na primeira experimentação.

Como já foi referido, o problema que surgiu na organização Bosch Car Multimédia Portugal S.A. é resolvido no final da atividade de *elaboration*. O problema não exige uma especificação oficial dos requisitos, mas sim um documento com alguma organização que inclui os requisitos para o sistema de informação "ALR". Como não houve tempo para se efetuar a atividade de *documentation*, o *artifact of requirements handling*, apesar de ser um documento otimizado para a utilização do *software requirements engineer*, resolve o problema que surgiu na organização. A prova disto é o *feedback* positivo dos *stakeholders* do projeto.

O leitor deve questionar o facto de não se usar uma abordagem idêntica à primeira experimentação, mas como se trata da evolução de uma folha de cálculo para um sistema de informação, é contraproducente efetuar-se uma iteração para se levantarem requisitos de uma folha de cálculo. Por outro lado, é necessário perceber que nesta experimentação, como se muda a abordagem de gestão do ciclo de vida na divulgação de resultados (ver capítulo 4.3), há a obrigatoriedade de se efetuar novamente uma primeira iteração, desprezando os requisitos provenientes da primeira experimentação.

A mudança na abordagem da gestão do ciclo de vida motiva também uma nova experimentação do processo aplicável à Bosch, com os ajustes provenientes da divulgação de resultados, e o projeto ALR é uma boa janela para experienciar estas nuances, assim como questões que ficaram por testar da primeira experimentação.

A atividade de *lifecycle management* é uma das que sofreu mais modificações após a divulgação de resultados, logo tem potencial para ser experimentada. Tanto a gestão do ciclo de vida dos requisitos por estados através da *state table*, como a gestão do ciclo de vida dos requisitos por versões através da *table of horizontal traceability* revelaram-se aplicáveis num contexto mais prático. Porém, esta experimentação não permitiu testar o processo com paridade de projetos, potencialidade que foi o maior impulsionador para a mudança efetuada na atividade de *lifecycle management*.

Na atividade de *traceability*, o componentes de *vertical traceability* e certos artefactos ficaram por ser testados na experimentação alusiva ao projeto GEIME, logo foi interessante utilizar este projeto para verificar como estes se comportam em ambiente prático. Artefactos como a *matrix of requirements relations* e o grafo para relações sistema – sistema revelaram-se muito interessantes na medida que se consegue ter uma visão mais facilitada sobre as relações dos requisitos. Já a componente de *vertical traceability* foi finalmente experimentada, identificando-se relações entre os requisitos de sistema e casos de uso com recurso à *table of*

vertical traceability. Esta funcionalidade é de facto útil na medida que permite relacionar os requisitos com outros SLOs de fases mais avançadas do projeto.

Esta caso também não permite testar o potencial de reutilização de requisitos, devido ao facto de serem necessárias diversas iterações para ser ter uma abstração do potencial deste método. No entanto, revelou-se útil devido à possibilidade de experimentar métodos e artefactos que ainda não tinham sido submetidos a qualquer manipulação.

4.5 Conclusões

Este capítulo surge devido à necessidade de testar o processo concebido no terceiro capítulo no ambiente aplicacional, onde se observou o problema que se revelou no estímulo para o início desta dissertação de mestrado.

O projeto GEIME está associado a uma reabilitação de um sistema de informação, que tem o propósito de gerir dispositivos para calibração, no seio da Bosch Car Multimédia Portugal S.A. Duas iterações são efetuadas, de modo a se levantarem os requisitos para a nova versão do GEIME. Desta experimentação conclui-se que, para o processo aplicável à Bosch ser completamente eficaz é necessário mais do que duas iterações. Contudo, pode-se afirmar que o processo aplicável à Bosch resolveu o problema que despontou na organização, através de uma especificação de requisitos qualificada.

A divulgação do trabalho na Bosch permite que os colaboradores da organização, associados à secção de CI-CWR12, do departamento CI-CWR1, tenham uma abstração das funcionalidades oferecidas pelo processo aplicável da Bosch. Como resultado desta atividade, alguns ajustes são feitos, fruto de sugestões apresentadas.

O projeto ALR está relacionado com conversão de uma rudimentar folha de cálculo numa aplicação informática, que tenha a finalidade de avaliar os componentes de um lote, no final do processo produtivo da Bosch Car Multimédia Portugal S.A. Apenas uma iteração do processo aplicável à Bosch é feita nesta experimentação, e resolve o problema deste caso no final da atividade de *elaboration*. Porém, este projeto também é utilizado para experimentar nuances, ou funcionalidades que ficaram por testar, da primeira experimentação, na componente de *requirements management*.

5 CONCLUSÃO

5.1 Síntese do Trabalho

Este trabalho pretendeu estabelecer um mecanismo no departamento de CI-CWR1 (secção CI-CWR12), da organização Bosch Car Multimédia Portugal, S.A., que possibilitasse a gestão do ciclo de vida de requisitos, de sistemas de informação que são desenvolvidos nesse ambiente industrial.

Desse modo, atribuiu-se especial atenção à disciplina de engenharia de requisitos, frequentemente citada como uma sub-disciplina da disciplina de engenharia de software, tentando introduzir esta num ciclo de vida, associado ao objeto "requisito", que não termine juntamente com um projeto de desenvolvimento de um sistema de informação.

De entre as atividades do processo de engenharia de requisitos, apesar de todas terem relevância, destaca-se a gestão de requisitos, devido a esta ser uma atividade que tem a particularidade de ter uma funcionalidade mais direcionada para o auxílio às demais atividades. Em adição, uma gestão de requisitos assente em um ciclo de vida dos requisitos resulta na solução para o problema inerente a esta dissertação de mestrado.

Da atividade de gestão de requisitos atribui-se relevo as funcionalidades de (1) rastreio de estados de requisitos, (2) rastreio de versões de requisitos, (3) gestão de mudança de requisitos e (4) rastreabilidade de requisitos, para incluir no artefacto, que é no fundo a solução que advém deste trabalho.

O artefacto, referido no parágrafo anterior, é um processo devido à necessidade deste ser moldável a um contexto onde são definidas as tarefas, com uma abordagem orientada para processos. A conceção do processo aplicável à Bosch resultou na identificação de onze atividades, necessárias para resolver o problema desta dissertação, que são divididas por duas componentes, com propósitos diferentes. Também se identificaram oito papeis que utilizam oito artefactos (apresentados em anexo), para realizarem as atividades identificadas.

As duas componentes do processo aplicável à Bosch são: (1) *development* e (2) *management*. Associadas à componente de *devolopment* estão as atividades: (1) *inception*, (2) *elicitation*, (3) *elaboration*, (4) *negotiation*, (5) *prioritization*, (6) *documentation* e (7) *validation*. As seguintes atividades constituem a componente de *management*: (1) *lifecycle management*, (2) *traceability*, (3) *allocation* e (4) *change management*. A componente de *development* tem a finalidade de levantar e desenvolver os requisitos de um determinado cliente/utilizador, para resolver um problema no seio da organização.

5 Conclusão

A componente de *management* está relacionada com dois tipos de gestão. A gestão do ciclo de vida dos requisitos (atividade de *lifecycle management*) é convencionada segundo duas dimensões: (1) o seu estado e (2) a sua versão. Esta gestão não está associada a um projeto ou sistema de informação independente, mas para esta funcionar é necessário localizar os requisitos nos diferentes sistemas de informação. A gestão de requisitos no ciclo de vida de sistemas de informação suporta funcionalidades como rastreabilidade e gestão de mudança, dos requisitos associados a um sistema de informação.

Na conceção do processo foram identificados os seguintes papeis para a execução das atividades sugeridas: (1) *information systems manager*, (2) *project manager*, (3) *software requirements engineer*, (4) *development team spokesman*, (5) *facilitator*, (6) *client*, (7) *user* e (8) *expert*.

Também são sugeridos, os seguintes artefactos, para realizar as diversas atividades do processo: (1) *artifact of requirements handling*, (2) *state table*, (3) *tables of requirements relations*, (4) *matrix of requirements relations*, (5) *table of horizontal traceability*, (6) *table of vertical traceability*, (7) *waiting list of changes* e (8) LHPH. A experimentação do processo aplicável à Bosch foi efetuada em dois projetos: (1) "GEIME" e (2) "ALR".

Na experimentação referente ao projeto "GEIME" foram realizadas duas iterações, sendo a primeira para levantar (por *reverse engeneering*) e desenvolver os requisitos referentes à versão obsoleta deste sistema de informação. Já a segunda, tem o intuito de levantar e desenvolver os requisitos relativos à nova versão do "GEIME".

Na experimentação associada ao projeto "ALR", apenas se efetuou uma iteração do processo. As atividades de *elicitation* e *elaboration* resolveram o problema inerente a este experimentação. Porém, o autor optou por fazer algumas experiências em atividades da componente de *management*, que foram alteradas na divulgação da primeira experimentação do processo na Bosch, ou não foram experimentadas com o projeto "GEIME".

No período entre a experimentação 1 (projeto "GEIME") e a experimentação 2 (projeto "ALR"), o autor apresentou os resultados do processo aplicável à Bosch, no departamento CI-CWR1 da organização. Desta divulgação resultaram algumas mudanças, no processo concebido no capítulo 3, com maior incidência na gestão do ciclo de vida dos requisitos.

Em suma, deste trabalho de investigação resulta um processo de engenharia de requisitos concebido e experimentado, para mitigar o problema da ausência de um mecanismo, para a gestão de requisitos, nos ciclos de vida dos sistemas de informação existentes na Bosch Car Multimedia Portugal, S.A.

5.2 Limitações

Este trabalho apresenta algumas limitações, fruto das restrições inerentes à realização de um trabalho académico em ambiente industrial.

A funcionalidade de reutilização de requisitos do processo aplicável à Bosch necessita de mais do que um ano letivo (coincidente com o período de realização desta dissertação), para se retirar maior benefício desta. Esta restrição temporal, também se aplica na indefinição de como se comportará o processo, à medida que o período de atividade vai aumentando e consequentemente o número de requisitos (e versões de requisitos) também.

Em nenhum dos projetos foi experimentada a atividade de *change management*. O projeto "GEIME" tinha potencial para testar esta atividade, contudo quando o projeto surgiu ainda não havia grande desenvolvimento nesse sentido. O projeto "ALR" só exigiu uma iteração, logo essa atividade não foi necessária.

Os projetos "GEIME" e "ALR" não tiveram a complexidade para colocar completamente à prova o processo aplicável à Bosch. Em nenhum dos projetos foi necessário utilizar todas as atividades e artefactos fornecidos pelo processo. Em adição, nenhum dos projetos exigiu rastreabilidade vertical até SLOs da fase de desenvolvimento, do processo de engenharia de software.

No departamento CI-CWR1, há uma grande preocupação com a eficiência do processo de engenharia de software. É necessário entender que o ambiente industrial funciona muito, com a necessidade de entregar os produtos, nos prazos estipulados. Isto resulta frequentemente, numa especificação de requisitos ad-hoc e ambígua. Estas constatações podem originar alguma resistência à mudança, devido a processo exigir mais recursos inicialmente, para se verificar uma diminuição do esforço futuramente.

Uma técnica de prioritização de requisitos que admita relações de requisitos seria útil para o processo concebido. No entanto, o autor do documento não encontrou nenhuma técnica para adaptar ao contexto.

5.3 Trabalho Futuro

Uma das propostas para trabalho futuro desta dissertação de mestrado é informatizar o processo concebido no capítulo 3. Uma informatização do processo resultaria provavelmente, numa minimização das possibilidades equivocar as funcionalidades estabelecidas. Por outro lado, a substituição dos artefactos sugeridos por um mecanismo onde fosse possível fazer tudo integrado e *online*, e que permitisse guardar a informação organizadamente, sem o utilizador ter essa preocupação seria uma mais valia.

Outra proposta de trabalho futuro está relacionada com a transformação do processo aplicável à Bosch, numa metodologia que permitisse satisfazer necessidades em domínios diferentes do domínio da engenharia de software. Apesar da engenharia de requisitos ser frequentemente definida como uma disciplina da engenharia de software (Fernandes & Machado, 2015), é fácil identificar requisitos em outras instâncias do mundo real. Imagine-se por exemplo, uma engenharia de requisitos adaptada à engenharia civil.

Por último, sugere-se o estudo da gestão de programas de projetos, para adaptar esta ao processo concebido, devido ao facto de frequentemente aparecer novos sistemas de informação, que são novas versões de sistemas que se tornam obsoletos.

REFERÊNCIAS

Amaral, L. (2005). Da Gestão ao Gestor de Sistemas de Informação: Expectativas Fundamentais no Desempenho da Profissão. *Sistemas de Informação Organizacionais*, 49–71.

Apache. (2006). Fundamentals of the V-Modell. In *V-Modell XT*. Retrieved from http://www.v-modell-xt.de/

Aurum, A., & Wohlin, C. (2003). The fundamental nature of requirements engineering activities as a decision-making process. *Information and Software Technology*, *45*(14), 945–954. http://doi.org/10.1016/S0950-5849(03)00096-X

Aurum, A., & Wohlin, C. (2005). *Engineering and Managing Software Requirements*. (A. Aurum & W. Claes, Eds.). Springer. http://doi.org/10.1007/3-540-28244-0

Berndtsson, M., Hansson, J., Olsson, B., & Lundell, B. (2008). *Thesis Guide - A Guide for Students in Computer Science and Information Systems* (Second Edi). Springer. http://doi.org/10.1007/978-1-84800-009-4

Blythe, J. (2005). *Essentials of Marketing* (Third Edit). Pearson Education. http://doi.org/9780273738442

Boehm, B. W. (1984). Verifying Validating Requirements Specifications. *IEEE Software*, *1*(1), 75–88.

Brown, P., Metz, R., & Hamilton, B. (2005). *Reference Model for Service Oriented* (No. 2). Retrieved from https://www.oasis-open.org/committees/download.php/15427/wd-soa-rm-10.pdf

Browning, T. R. (2002). Process Integration Using the Design Structure Matrix. In *Systems Engineering* (pp. 180–193).

Campos, C. (2013). *Estudo de Interdependências e Rastreabilidade no Processo RUP: Análise do Micro-Processo V-Model e do Método 4SRS*. Universidade do Minho.

Carlshamre, P., & Regnell, B. (2000). Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes. *REP2000*, *2000*(September), 0–4.

Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., & Dag, J. N. och. (2001). An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 84–92. http://doi.org/10.1109/ISRE.2001.948547

Chemuturi, M. (2013). Requirements Engineering and Management for Software Development Projects. In *Requirements Engineering and Management for Software* (pp. 177–201). Springer. http://doi.org/10.1007/978-1-4614-5377-2

Cheng, B. H. C., & Atlee, J. M. (2007). Research Directions in Requirements Engineering. *Future of Software Engineering*.

Chonoles, M. J., & Schardt, J. A. (2003). *UML 2 for Dummies*. John Wiley & Sons, 2011.

Referências

Christel, M., & Kang, K. (1992). *Issues in requirements elicitation*. Software Engineering Institute. http://doi.org/10.1016/S0140-6736(08)60207-X

Cleland-Huang, J., Chang, C. K., & Christensen, M. (2003). Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, *29*(9), 796–810. http://doi.org/10.1109/TSE.2003.1232285

CMMI Product Team. (2010). *CMMI for Development. Improving processes for developing better products and services*. Retrieved from http://repository.cmu.edu/sei/287/

Cooper, A. (1999). *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity*. http://doi.org/10.1007/978-3-322-99786-9_1

Davenport, T. H. (1993). *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business Press.

Davis, A. (2005). *Just enough requirements management: Where Software Development Meets Marketing*. *New York: Dorset House*. Retrieved from http://www.sigs.de/download/oop_09/Davis Mo4 brb WR A4.pdf

Dufresne, T., & Martin, J. (2003). *Process Modeling for E-Business*.

Endres, A., & Rombach, D. (2003). *A Handbook of Software and Systems Engineering*. (R. Dieter, Ed.). Pearson Education. http://doi.org/10.1109/MS.2004.1270773

Fernandes, J. M., & Machado, R. J. (2015). *Requirements in Engineering Projects*. (Springer, Ed.) (1st ed.). Guimarães, Braga, Portugal.

Frank, U. (1999). Conceptual Modelling as the Core of the Information Systems Discipline - Perspectives and Epistemological Challenges. *Proceedings of the Fifth Americas Conference on Information Systems (AMCIS 1999) : August 13-15, 1999, Milwaukee, Wisconsin*, (Amcis 99), 695 -698. Retrieved from http://www.uni-koblenz.de/~iwi/publicfiles/PublikationenFrank/AMCIS99.pdf%5CnZ:%5CDocuments %5CLiteratur%5CFrank_1999_Conceptual Modelling as the Core of the Information Systems Discipline.pdf%5Cnhttp://ais99.sba.uwm.edu/

Gilb, T. (1997). Towards the engineering of requirements. *Requirements Engineering*, *2*, 165–169. http://doi.org/10.1007/BF02802774

Goguen, J. A., & Charlotte, L. (1993). Techniques for Requirements Elicitation. In *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on* (Vol. 6, pp. 58–66). IEEE.

Gotel, O., & Finkelstein, A. (1993). Making Requirements Elicitation Traceable, (i), 1–3.

Haskins, C. (2006). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. (C. Haskins, Ed.)*INCOSE. Version*. INCOSE Systems Engineering Handbook v. 3. Retrieved from

http://smslab.kaist.ac.kr/Course/CC532/2012/LectureNote/2012/INCOSE Systems Engineering Handbook v3.1 5-Sep-2007.pdf

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. http://doi.org/10.2307/25148625

Hull, E., Jackson, K., & Dick, J. (2006). *Requirements Engineering*. *Requirements Engineering* (Second, Vol. 13). Springer. http://doi.org/10.1145/336512.336523

IEEE. (1990). IEEE standard glossary of software engineering terminology. In *IEEE Standards Board*. IEEE.

IEEE. (1993). IEEE Recommended Practice for Software Requirements Specification. In *IEEE Standards Board*. IEEE. Retrieved from http://www.utdallas.edu/~chung/RE/IEEE830-1993.pdf %5Cnhttp://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=15571&arnumber=720574&count=1&index=0

IEEE. (2014). *Guide to the software engineering body of knowledge version 3.0*. (P. Bourque & R. E. Fairley, Eds.) (3rd ed.). IEEE. Retrieved from www.swebok.org

IIBA. (2009). *Guide to the Business Analysis Body of Knowledge*. *BABOK* (2nd ed.). IIBA. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Guide+to+the+Business+Analysis+Body+of+Knowledge#0

Institute, P. M. (2013). *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK)* (Fifth). Global Stardard. http://doi.org/19073-3299

ISO/IEC/IEEE. (2008). *ISO/IEC 12207:2008*.

ISO/IEC/IEEE. (2011). *ISO/IEC/ IEEE 29148*.

Jalote, P. (2005). *An Integrated Approach to Software Engineering*. (D. Gries & F. B. Schneider, Eds.) (Third). Springer.

Kaindl, H., & Svetinovic, D. (2010). On confusion between requirements and their representations. *Requirements Engineering*, *15*(3), 307–311. http://doi.org/10.1007/s00766-009-0095-7

Karlsson, J., Wohlin, C., & Regnell, B. (1998). An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, *39*(14–15), 939–947. http://doi.org/10.1016/S0950-5849(97)00053-0

Keen, P. (1981). Information Systems and Organizational Change. *Communications of the ACM*, *24*(1), 24–33. http://doi.org/10.1145/358527.358543

Krishnamurthy, N., & Saran, A. (2008). *Building Software - A Practitioner's Guide*. *Auerbach Publications* (Vol. 1). Taylor & Francis Group. http://doi.org/10.1017/CBO9781107415324.004

Laplante, P. A. (2007). *What Every Engineer Should Know about Software Engineering*. Taylor & Francis Group. http://doi.org/10.1201/9781420006742

Referências

Leffingwell, D., & Widrig, D. (2003). *Managing Software Requirements: A Use Case Approach, Second Edition*. Addison Wesley.

Møller, C., & Ph, D. (2014). Business Process. In N. Damij & T. Damij (Eds.), *Process Management* (pp. 7–25). Springer. http://doi.org/10.4135/9781446251720.n28

Neiva, R. (2013). *Geração de Modelos de Rastreabilidade de Informação Industrial: Análise de um Caso Prático*. Universidade do Minho.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements Engineering: A Roadmap. *Proceedings of the Conference on The Future of Software Engineering*, *1*, 35–46. http://doi.org/10.1145/336512.336523

Ogland, P. (2009). Action Research and Design Science Research - More Similar Than Dissimilar.

Pandey, D., Suman, U., & Ramani, A. K. (2010). An Effective Requirement Engineering Process Model for Software Development and Requirements Management. *International Conference on Advances in Recent Technologies in Communication and Computing*, 287–291. http://doi.org/10.1109/ARTCom.2010.24

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. http://doi.org/10.2307/40398896

Pohl, K. (1995). Requirements Engineering: An Overview *. *Informatik V*, *36*, 1–40.

Polli, R., & Cook, V. (1969). Validity of the Product Life Cycle. *The Journal of Business*, *42*(4), 385–400. http://doi.org/10.1086/295215

Pressman, R. S. (2005). *Software Engineering, a Practitioner's Approach* (sixth). Higher Education.

Rajlich, V. T., & Bennett, K. H. (2000). A Staged Model for the Software Life Cycle. *Computer*, *33*(7). http://doi.org/10.1109/2.869374

Ramzan, S., & Ikram, N. (2006). Requirement Change Management Process Models: Activities, Artifacts and Roles. *Multitopic Conference*, 219–223.

Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. *Crown Business*. Creown Publishing. http://doi.org/23

Robertson, S., & Robertson, J. (2013). *Mastering the Requirement Process. Getting Requirements Right* (Third). Pearson Education. Retrieved from www.volere.co.uk

Sommerville, I. (2011). *Software Engineering*. (M. Horton, Ed.)*Software Engineering* (Ninth). Pearson Education. http://doi.org/10.1111/j.1365-2362.2005.01463.x

Stark, J. (2011). *Product Lifecycle Management*. *Journal of Chemical Information and Modeling* (Vol. 53). http://doi.org/10.1017/CBO9781107415324.004

Taylor, C., & Probst, C. (2003). Business Process Reference Model Languages: Experiences from BPI Projects. *INFORMATIK 2003 : Innovative Informatikanwendungen, Band 1, Beiträge Der 33. Jahrestagung Der Gesellschaft Für Informatik e.V. (GI), 29. September – 2. Oktober 2003 in Frankfurt Am Main*, 259–263.

Taylor, F. W. . (1995). *Princípios de Administração Científica*. (Atlas, Ed.)*Princípios de Administração Científica*.

van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006). A Reference Framework for Software Product Management. http://doi.org/10.1109/RE.2006.66

Wiegers, K., & Beatty, J. (2013). *Software Requirements*. (D. Musgrave & C. Dillinham, Eds.) (Third). Washington: Microsoft Press. Retrieved from http://www.amazon.com/dp/0735679665/ref=pe_385040_30332190_TE_M3T1_ST1_dp_3

Zave, P. (1995). Classification of Research Efforts in Requirements Engineering. IEEE.

(Página intencionalmente deixada em branco)

APÊNDICE I – ARTIFACT OF REQUIREMENTS HANDLING

O *artifact of requirements handling* é um artefacto, que auxilia o *software requirements engineer* a realizar as atividades de *elicitation* e *elaboration* do processo aplicável à Bosch.

O documento não tem uma estrutura estabelecida, de modo a não limitar o *software requirements engineer* (as atividade de *elicitation* e *elaboration* têm por norma características incompatíveis com uma estrutura padronizada). Apesar da estrutura do documento não ser padronizada, o *software requirements engineer* deve seguir as instruções do processo aplicável à Bosch, devido à necessidade deste documento servir de *input* de informação nas atividades de *lifecycle management* e *traceability* na componente de *management*.

Este artefacto não deve substituir o LHPH (ver anexo I) no desígnio de comunicar os requisitos, devendo ser só utilizado para suportar a atividade do *software requirements engineer*.

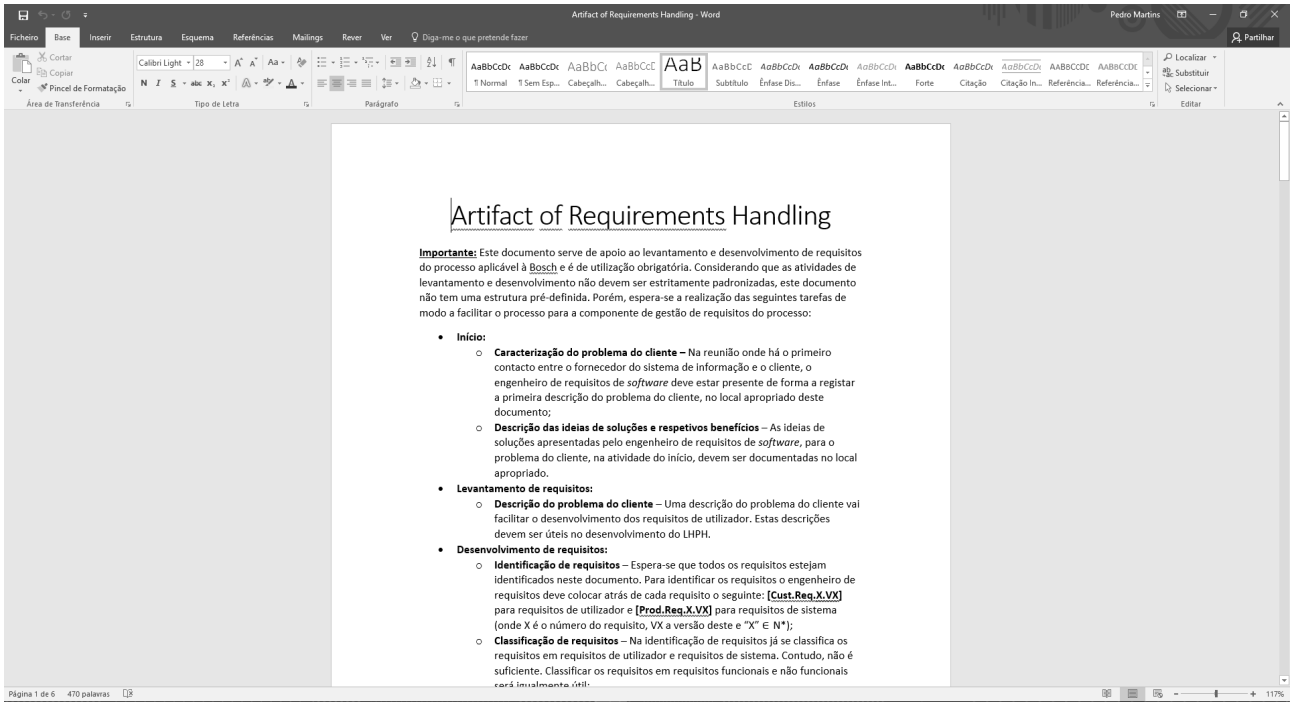


Ilustração 54: Artifact of Requirements Handling

Hiperligações para descarregar o artefacto

Artifact of Requirements Handling para utilização: https://goo.gl/4ABWfi

Artifact of Requirements Handling desenvolvido no projeto "GEIME" (1ª iteração): https://goo.gl/qJmQOr

Artifact of Requirements Handling desenvolvido no projeto "GEIME" (2ª iteração): https://goo.gl/h7ihr8

Apêndice I – Artifact of requirements handling

Artifact of Requirements Handling desenvolvido no projeto "ALR": https://goo.gl/xBq6l0

APÊNDICE II – STATE TABLE

A *state table* deve ser utilizada na atividade de *lifecycle management*, para permitir a gestão do ciclo de vida de requisitos na sua dimensão de estado.

Aqui, um requisito pode ter diferentes estados em diferentes versões de requisito, que podem estar associadas a um ou mais sistemas de informação. Isto justifica o facto de usar o padrão de atribuição de estado: **[nome_do_sistema_de_informação-VX]**. Consequentemente, só deve existir um artefacto deste tipo no ambiente aplicacional.

Este artefacto também serve de *input* de informação na atividade de *traceability*.

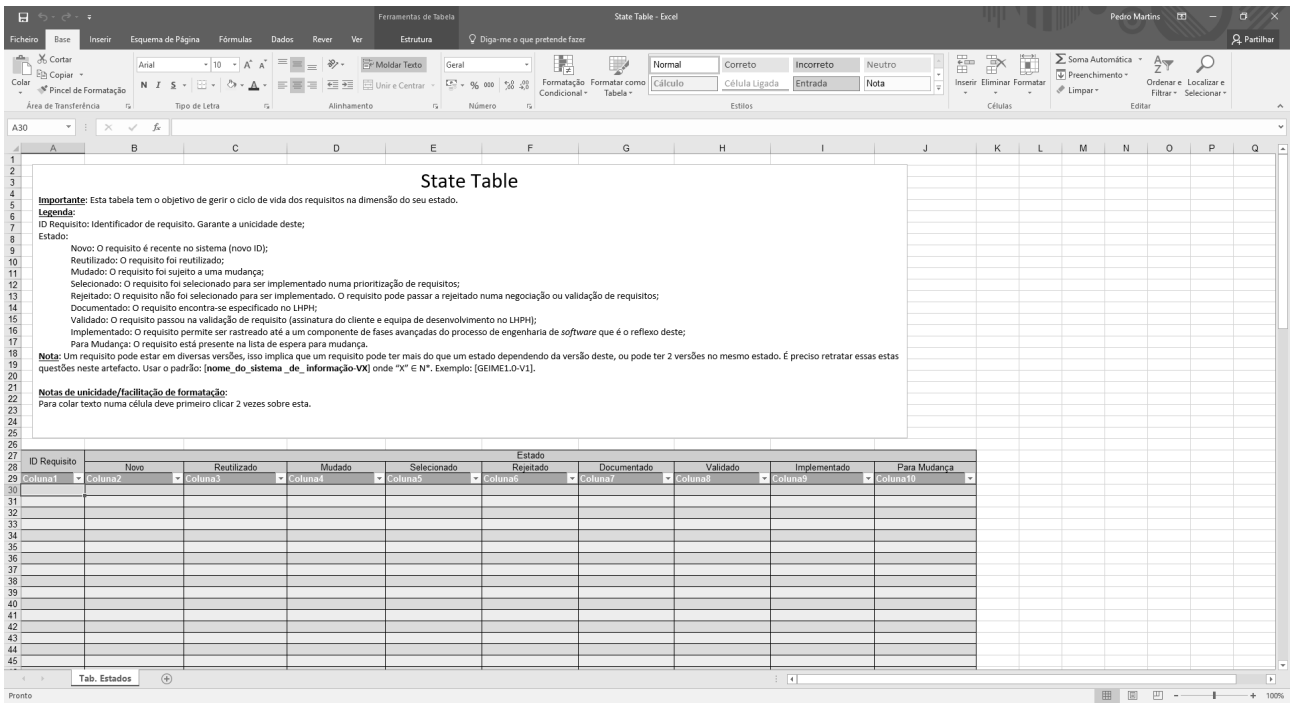


Ilustração 55: State Table

Hiperligações para descarregar o artefacto

State Table para utilização: https://goo.gl/xgzvON

State Table desenvolvida no projeto "GEIME" (1ª iteração): https://goo.gl/XdTRWm

State Table desenvolvida no projeto "ALR": https://goo.gl/zvYpmK

(Página intencionalmente deixada em branco)

APÊNDICE III – TABLES OF REQUIREMENTS RELATIONS

O *tables of requirements relations* é um dos artefactos disponibilizados pelo processo aplicável à Bosch, para a identificação de relações nos requisitos de um sistema de informação.

A abordagem deste artefacto é primeiro encontrar as relações para os requisitos de utilizador, ou seja, relações (utilizador – utilizador) e relações (utilizador – sistema). De seguida identificam-se relações para os requisitos de sistema, relações (utilizador – sistema) tendo em vista quais as precedências dos requisitos de sistema e relações (sistema - sistema).

O *software requirements engineer* deve usar o tipo de relação (deste modo: **<<tipo_de_relação>>**) para relações (utilizador – utilizador) e relações (sistema – sistema) e um "**x**" para relações (utilizador – sistema).

Este artefacto deve assumir mais a finalidade de comunicação de relações, por dar a facilidade de encontrar as relações para um determinado requisito.

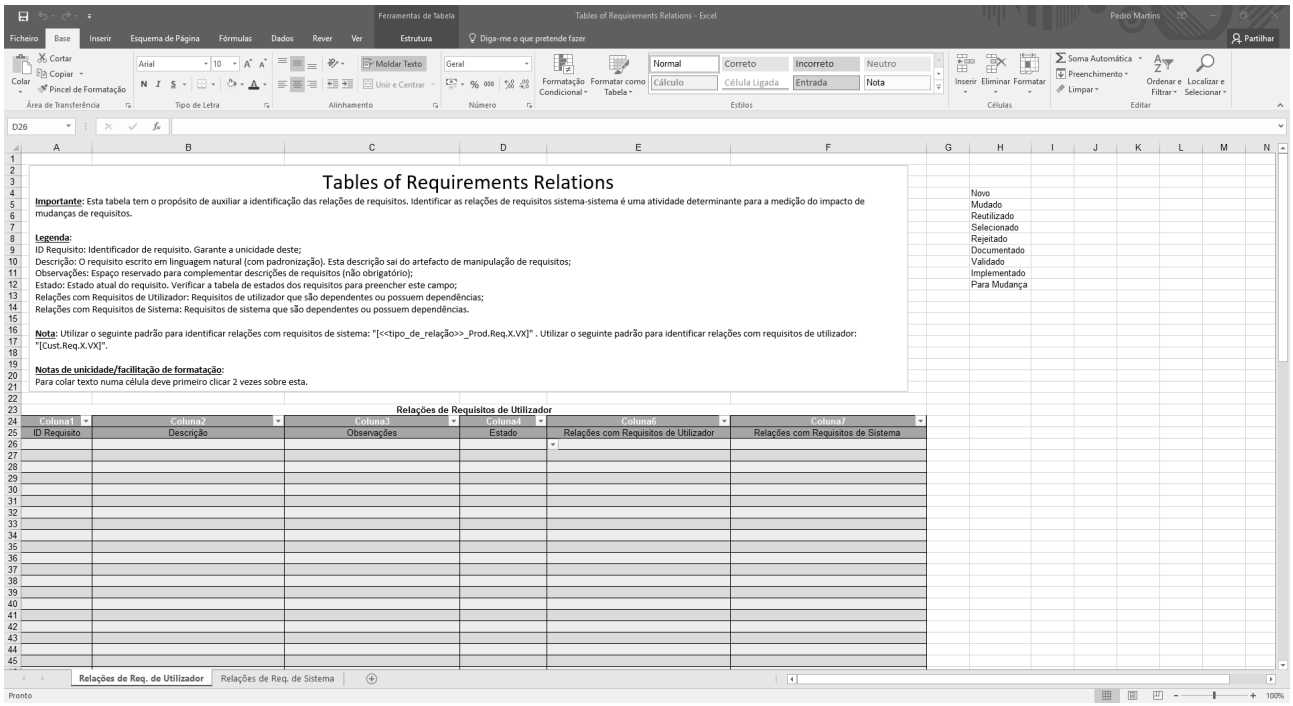


Ilustração 56: Tables of Requirements Relations

Hiperligações para descarregar o artefacto

Tables of Requirements Relations para utilização: https://goo.gl/MoLmbg

Tables of Requirements Relations desenvolvido no projeto "GEIME" (1ª iteração): https://goo.gl/ZP4f96

Apêndice III – Tables of requirements relations

Tables of Requirements Relations desenvolvido no projeto "GEIME" (2ª iteração): https://goo.gl/uFGFN5

Tables of Requirements Relations desenvolvido no projeto "ALR": https://goo.gl/pYoy08

APÊNDICE IV – MATRIX OF REQUIREMENTS RELATIONS

A *matrix of requirements relations* é um artefacto que permite a um *software requirements engineer* identificar relações através de *match* entre uma coluna (respeitante aos requisitos afetantes) e uma linha (referente aos requisitos afetados).

Para cada sistema de informação deve desenvolver este artefacto para relações (utilizador – utilizador), relações (utilizador – sistema) e relações (sistema – sistema).

Quando se trata de relações (utilizador – utilizador) e relações (sistema – sistema) deve identificar a relação com o tipo de relação da seguinte forma: **<<tipo_de_relação>>**. Quando se pretende identificar relações (utilizador – sistema) um "**x**" é suficiente.

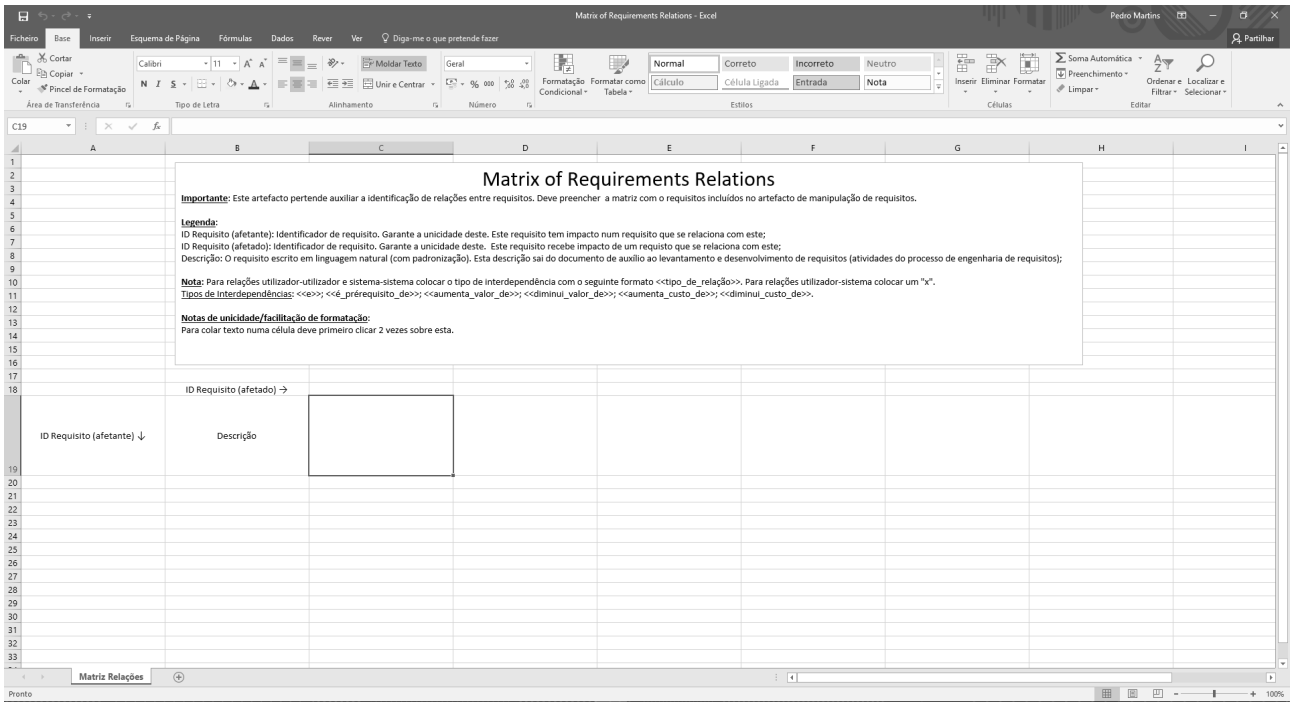


Ilustração 57: Matrix of Requirements Relations

Hiperligações para descarregar o artefacto

Matrix of Requirements Relations para utilização: https://goo.gl/fj8gBX

Matrix of Requirements Relations desenvolvido no projeto "ALR" (user - system): https://goo.gl/SCy2wz

Matrix of Requirements Relations desenvolvido no projeto "ALR" (system – system): https://goo.gl/hEbhVG

(Página intencionalmente deixada em branco)

APÊNDICE V – TABLE OF HORIZONTAL TRACEABILITY

A *table of horizontal traceability* é um artefacto, que dá ao processo aplicável à Bosch, a funcionalidade de gerir o ciclo de vida de requisitos na sua dimensão da versão, na atividade *lifecycle management*. Em adição, é o artefacto que permite a um *software requirements engineer*, reutilizar e mudar requisitos entre sistemas de informação.

Para cada versão de um requisito, o *software requirements engineer* deve adicionar uma descrição deste e uma eventual observação. Só deve haver uma *table of horizontal traceability* no domínio aplicacional que contenha todos os requisitos e versões destes.

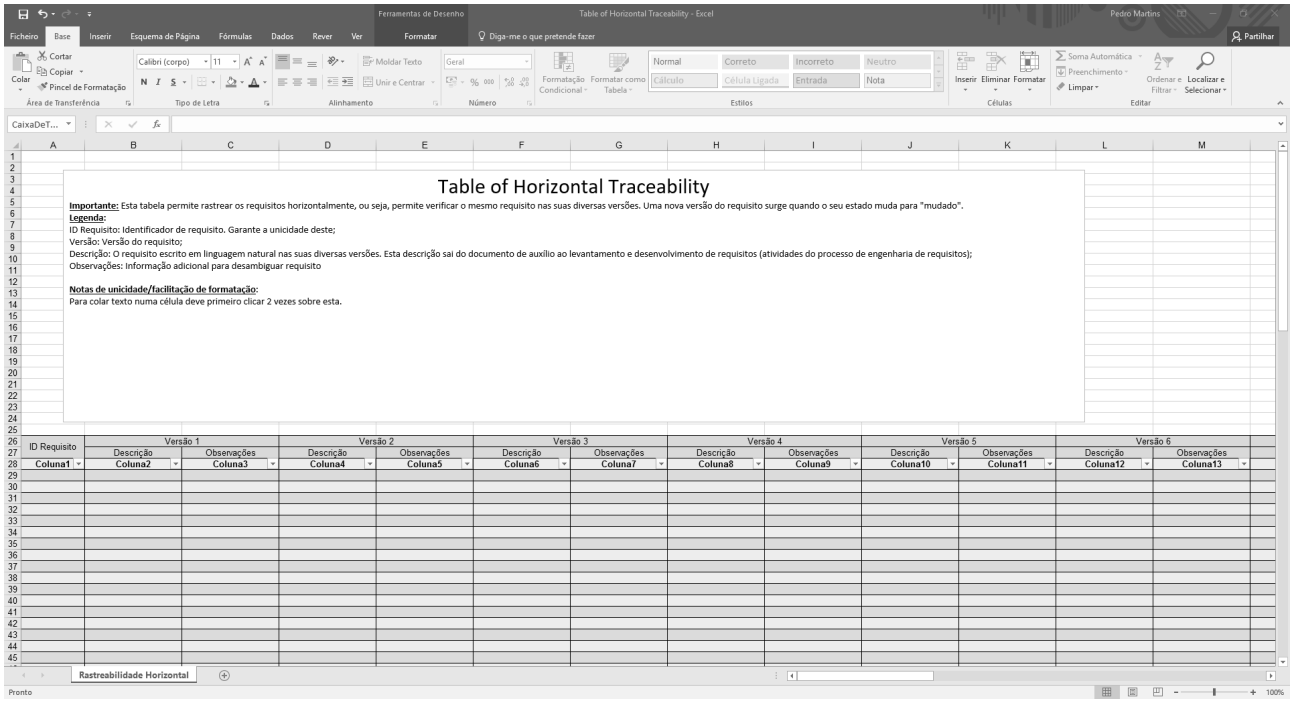


Ilustração 58: Table of Horizontal Traceability

Hiperligações para descarregar o artefacto

Table of Horizontal Traceability para utilização: https://goo.gl/caEocF

Table of Horizontal Traceability desenvolvido no projeto "GEIME" (1ª iteração): https://goo.gl/ZP4f96

Table of Horizontal Traceability desenvolvido no projeto "GEIME" (2ª iteração): https://goo.gl/uFGFN5

Table of Horizontal Traceability desenvolvido no projeto "ALR": https://goo.gl/DsqLff

(Página intencionalmente deixada em branco)

APÊNDICE VI – TABLE OF VERTICAL TRACEABILITY

A *table of vertical traceability* oferece ao processo aplicável à Bosch a funcionalidade de rastrear os diversos requisitos até SLOs das diversas fase do processo de engenharia de software.

Para identificar as relações com os SLOs, o *software requirements engineer* deve colocar um identificador do SLO na coluna referente à fase do processo de engenharia de software.

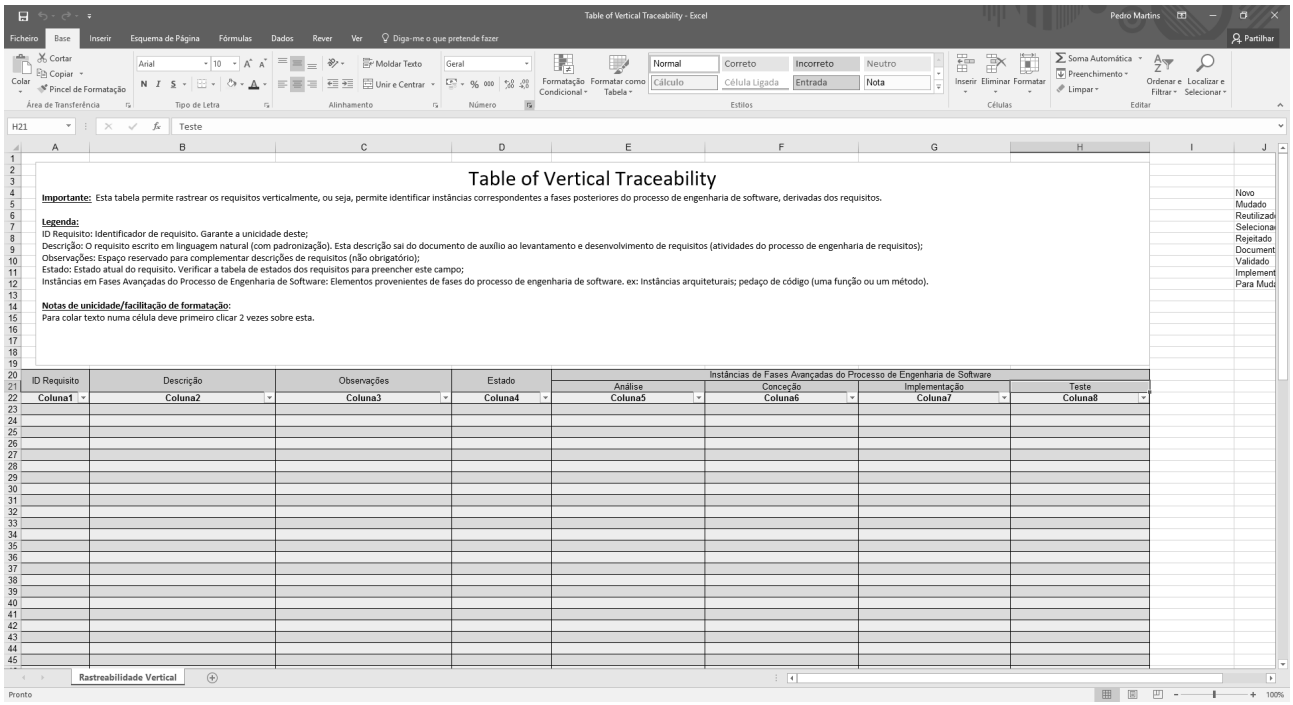


Ilustração 59: Table of Vertical Traceability

Hiperligações para descarregar o artefacto

Table of Vertical Traceability para utilização: https://goo.gl/NuAtS2

Table of Vertical Traceability desenvolvida no projeto "ALR": https://goo.gl/SqC6pV

(Página intencionalmente deixada em branco)

APÊNDICE VII – WAITING LIST OF CHANGES

A *waiting list of changes* é um artefacto onde se guardam mudanças de *change requests*, que surgem na atividade de *change management* do processo aplicável à Bosch, quando um novo projeto não inicia, nem vai iniciar.

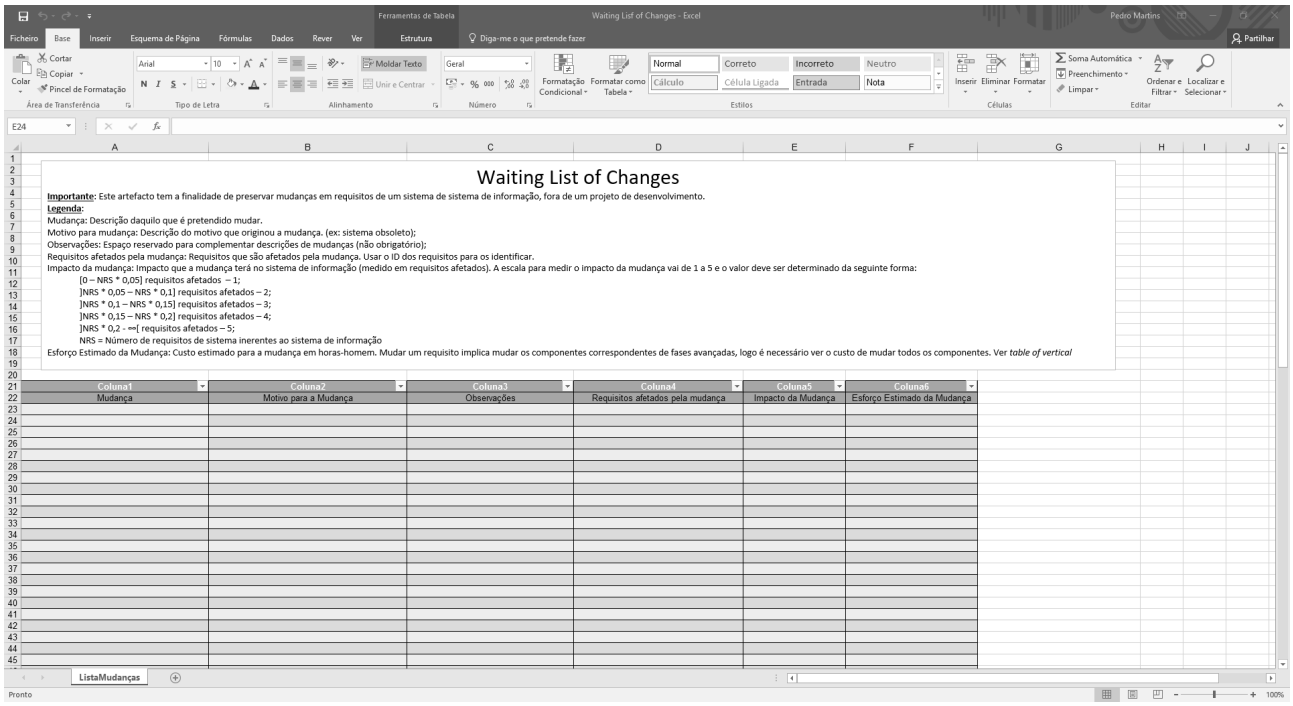


Ilustração 60: Waiting List of Changes

Hiperligações para descarregar o artefacto

Wainting List of Changes para utilização: https://goo.gl/vcF5Q9

(Página intencionalmente deixada em branco)

ANEXO I – LHPH

O LHPH é o documento de especificação de requisitos oficial na Bosch, logo é utilizado na atividade de *documentation* do processo aplicável à Bosch.

Este artefacto foi sujeito a diversas atualizações de forma a possuir uma estrutura que satisfaça todas as necessidades do *software requirements engineer* para a documentação de requisitos.

Este artefacto assume a função de comunicação de requisitos, portanto só inclui o sub-conjunto dos requisitos fundamentais do sistema.

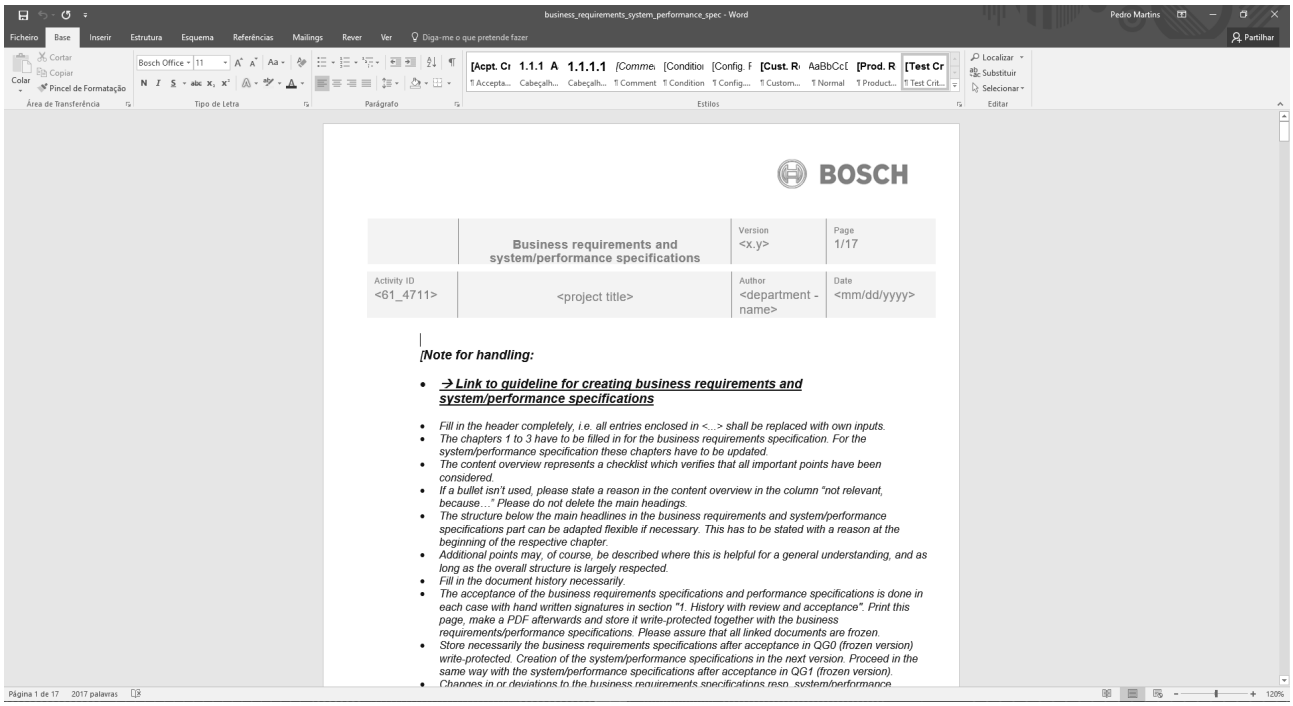


Ilustração 61: LHPH

Hiperligações para descarregar o artefacto

Por restrições impostas pela organização, nenhuma hiperligação pode ser disponibilizada para descarregar o LHPH.