LICENCIATURA EM SISTEMAS E TECNOLOGIAS DE INFORMAÇÃO

GESTÃO DO CONHECIMENTO EM PROCESSOS DE ENGENHARIA DE SOFTWARE EM ORGANIZAÇÕES EMPRESARIAIS

Projeto Final de Licenciatura

Elaborado por Paulo Sérgio dos Santos Carreteiro
Discente nº 20131827

Orientador Professor Doutor José Braga de Vasconcelos

Barcarena

Junho de 2016

Universidade Atlântica

Licenciatura em Sistemas e Tecnologias de Informação

GESTÃO DO CONHECIMENTO EM PROCESSOS DE ENGENHARIA DE SOFTWARE EM ORGANIZAÇÕES EMPRESARIAIS

Projeto Final de Licenciatura

Elaborado por Paulo Sérgio dos Santos Carreteiro
Discente nº 20131827

Orientador Professor Doutor José Braga de Vasconcelos

Barcarena

Junho de 2016

Agradecimentos

O desporto e a corrida ensinaram-me duas coisas, em primeiro lugar que o mais importante não é como se começa uma prova mas sim como se acaba e em segundo lugar que os desafios que se apresentam como mais difíceis, são os que nos ensinam mais e que acabam por nos tornar mais fortes. A minha motivação e a força de vontade foram importantes para superar esta maratona, mas o apoio e a ajuda de familiares, amigos, colegas e docentes tornaram com que o percurso que muitas vezes parecia uma subida íngreme se tornasse numa inclinação ligeiramente acentuada.

Agradeço ao Professor Doutor José Braga de Vasconcelos por todo o apoio, orientação e colaboração neste trabalho, pelos desafios que me foi lançando ao longo de toda a licenciatura, essenciais no meu crescimento académico, tornando-se sem dúvida um dos professores mais marcantes.

Ao Professor Doutor Alexandre Barão um agradecimento especial pela ajuda e orientação na elaboração do artigo científico, que foi aceite na conferência World Cist 2016. Ao restante corpo docente da Universidade Atlântica, o meu obrigado pelo conhecimento transmitido.

Uma palavra aos meus colegas de curso por toda a ajuda, trocas de opiniões, ideias e momentos divertidos que proporcionaram durante a licenciatura, de todos um agradecimento especial ao Nuno Silva pela ajuda, dicas e partilha do seu valioso conhecimento que foi essencial em muitas etapas da licenciatura.

À minha família e amigos pela força e palavras de incentivo o meu sincero obrigado, à minha mãe em especial que não pode assistir em vida a um dos seus grandes desejos, estou certo que esteve presente durante este percurso.

Por último à minha mulher Lígia, principal motivadora para o meu ingresso nesta licenciatura, aos meus filhos Afonso e Letícia, os três são os meus pilares, pelo amor, apoio incondicional e por me terem libertado para que me pudesse focar durante todo este tempo. A todos o meu obrigado!

"Objetiva a genialidade que possuis, senão impera a preguiça."

Luísa Carreteiro

Lista de Abreviaturas

ETL - Extraction Transformation Load

MS - Manutenção de Software

GC - Gestão do Conhecimento

ES - Engenharia de Software

CVDS - Ciclo de Vida de Desenvolvimento de Software

SWEBOK - Software Engineering Body of Knowledge

DSR - Design Science Research

CMMI - Capability Maturity Model Integration

RD - Requirement Definition

TS - Technical Solution

VAL - Validation

VER - Verification

PI - Product Integration

BRS - Business Requirements Specification

SRS - Software Requirements Specification

PRS - Project Requirements Specification

UML - Unified Modelling Language

Resumo

A Manutenção de *Software* é a ultima fase do ciclo de vida do desenvolvimento de *software* e a sua atividade visa a correção de incidentes. Os indivíduos que nela participam necessitam de dominar um conjunto significativo de competências. Adicionalmente muitos destes problemas quando ocorrem exigem intervenções rápidas por originarem indisponibilização do negócio ou causarem impactos significativos noutras aplicações ou sistemas.

Nas fases anteriores dos projetos de desenvolvimento de *software*, é gerado um número significativo de documentação sobre o *software* que está a ser produzido, onde são descritas funcionalidades, regras e condições referentes ao funcionamento do mesmo. Apesar da existência de toda essa documentação muito raramente ela é utilizada pelos indivíduos que efetuam a manutenção, que acabam por recorrer à interpretação da linguagem de programação do *software* ou a trocas informais de ideias, gastando grande parte do tempo da manutenção a tentar perceber o que está a ser mantido.

Neste trabalho é abordada a implementação da Gestão do Conhecimento na Manutenção de *Software*, em primeiro lugar como forma de tornar mais eficiente esta atividade e em segundo utilizá-la como ferramenta para validar, corrigir e atualizar o conhecimento produzido anteriormente, utilizando o "back flushing" dos processos de ETL do Data Warehousing.

Palavras-chave – Gestão do Conhecimento; Engenharia de Software; Ciclo de Vida do Desenvolvimento de Software; Manutenção de Software.

Abstract

Software Maintenance is the last phase of the software development lifecycle and its activity is aimed at correcting incidents. Individuals who participate need to master a significant set of skills. In addition when many of these problems occur require rapid interventions for yielding unavailability of business or cause significant impacts in other applications or systems.

In previous phases of software development projects, it is generated a significant number of documents of the software that is being produced, where are described features, rules and conditions related to its functioning. Despite the existence of all this documentation seldom it is used by individuals who perform maintenance, who end up resorting to interpretation of software programming language or informal exchanges of ideas with their peers, spending much of the maintenance time trying to understand what it is to be maintained.

This paper discusses the implementation of Knowledge Management in Software Maintenance, first as a way to become more efficient this activity and secondly to use it as a tool to validate, correct and update the knowledge produced previously using the "back flushing " a processes from ETL in Data Warehousing .

Keywords – Knowledge Management; Software Engineering; Software Life Cycle; Software Maintenance.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUÇÃO

A Manutenção de *Software* (MS) é uma atividade exigente e complexa, os indivíduos que nela participam precisam de dominar as linguagens de programação utilizadas no *software*, conhecer a arquitetura do sistema, entender os modelos de dados, ter um conhecimento funcional das aplicações e saber as atualizações que sobre estes são realizadas bem como os impactos associados. Adicionalmente e derivado da escassez de recursos, cada vez mais a MS é efetuada a várias aplicações, obrigando a gerir uma quantidade significativa e complexa de informação.

Esta atividade é habitualmente entregue a indivíduos com maior experiência dentro da organização, decorrente da sua senioridade e heurísticas obtidas ao longo dos anos nos diversos projetos de desenvolvimento de *software*. Esta preferência vem reforçar o elevado grau de exigência, conhecimento, sensibilidade e experiência necessária por parte de quem as realiza, devido à criticidade que representam e à urgência com que as mesmas devem ser realizadas.

No decorrer dos projetos de Engenharia de Software (ES) são produzidos um elevado número de documentos, que pela dificuldade em serem pesquisáveis e indexados, raramente são utilizados no auxílio à resolução de problemas e muitas vezes não se encontram devidamente atualizados, apesar de terem como principal objetivo descrever todo o conhecimento identificado durante a realização dos projetos.

Constata-se que os indivíduos que participam na MS, acabam maioritariamente por recorrer à interpretação da linguagem de programação ou a uma troca informal de ideias com outros membros da equipa, o que faz com que entre 40% a 60% do tempo consumido no processo de MS é gasto na compreensão do *software* que está a ser mantido.

A Gestão do Conhecimento (GC) surge com o objetivo de otimizar o conhecimento individual das organizações. Identifica o conhecimento existente nas organizações, quem o detém, quando e como foi gerado, a GC propõem-se assim a geri-lo mais eficazmente. É única porque incide sobre o indivíduo como um especialista e como portador do conhecimento relevante que ele pode partilhar sistematicamente com a organização.

Da experiência tida e da revisão da literatura efetuada, constata-se:

- A implementação da Gestão do Conhecimento na Engenharia de Software tem sido mais focada nas fases de requisitos, desenho e codificação tendo ficado a MS para segundo plano;
- O acesso a informação relevante, ou seja, de onde se possam obter respostas que contenham soluções concretas ou com a orientação para a resolução dos erros com que os indivíduos são confrontados, é um dos principais problemas na realização de tarefas de MS.

Como poderá a MS beneficiar da adoção de métodos de GC? A captura do conhecimento gerado na manutenção pode ser utilizado como meio para validar, atualizar e corrigir o conhecimento documentado nas fases anteriores?

Neste trabalho é apresentado um modelo onde é mapeado, catalogado e armazenado o conhecimento mais relevante para a MS, a partir dos documentos criados nas diversas fases do ciclo de vida da ES, tornando o acesso ao mesmo mais eficiente e fácil, visando reduzir o tempo gasto atualmente. O modelo irá também permitir que as equipas de MS sejam utilizadas para validar, corrigir e atualizar o conhecimento criado em fases anteriores, para tal no modelo é utilizado uma analogia com a técnica de "back flushing" utilizada nos processos de ETL[1].

Este trabalho encontra-se organizado da seguinte forma: No primeiro capítulo são apresentadas as definições e características da GC nas organizações, da ES e explicadas as fases do CVDS bem como as principais atividades nelas realizadas. No segundo capítulo demonstra-se a ligação entre os conceitos da GC e as atividades da ES no sentido de demonstrar a relação de proximidade entre ambos, provando que o desenvolvimento de *software* é uma atividade de conhecimento intensivo. Os dois primeiros capítulos pretendem expor os alicerces onde este trabalho assenta, essenciais para realizar o respetivo enquadramento. No terceiro capítulo é apresentada a metodologia seguida para a realização do trabalho e conceção do modelo. No capítulo quatro é efetuada uma introdução sobre a abordagem deste trabalho para a implementação de uma iniciativa de GC na ES. No capítulo quinto é apresentado o modelo, os

[1] Do inglês Extract Transform Load (Extração Transformação Carregamento), são ferramentas de *software* cuja função é a extração de dados de diversos sistemas, transformação desses dados conforme regras de negócio e por fim o carregamento dos dados.

seus atores e o seu funcionamento onde é introduzido o conceito de “back flushing”. Por último, é apresentada a conclusão do trabalho efetuado e identificado o trabalho futuro a realizar.

1 DEFINIÇÕES E CONCEITOS DA GESTÃO DO CONHECIMENTO E DA ENGENHARIA DE SOFTWARE

1.1 O Conhecimento e a sua Gestão

1.1.1 Definição e Níveis do Conhecimento

É consensual a existência de três níveis de refinamento nomeadamente dados, informação e por fim conhecimento (Davenport, 2010), (Rus & Lindvall, Knowledge Management in Software Engineering, 2002). Dados são os itens, valores, nomes, etc., informação que se encontra numa forma não estruturada. Informação diz respeito a dados já estruturados, isto é, que se encontram organizados de uma forma lógica. Conhecimento é quando já existe uma análise realizada sobre a informação que possibilita efetuar relações entre os diversos tipos de informação que permite efetuar classificações, em suma é informação sobre a informação.

1.1.2 Características do Conhecimento

O conhecimento pode ser explícito ou tácito e separado pela forma como está distribuído, ou seja, onde pode ser aplicado, a quem está acessível e que atividades são por ele suportadas. Significa que o conhecimento existente nas organizações pode residir nelas próprias, ao nível individual ou num grupo de pessoas.

É assim possível separar a informação propriamente dita, dos conhecimentos subjacentes às pessoas, ou seja o conhecimento como um corpo de informação, como um saber individual e como um saber em equipa (Cascão, 2014). Enquanto um corpo de informação, ele encontra-se de uma forma explícita e é codificável, nos outros dois ele é de natureza tácita e pessoal ou dependente da dinâmica das equipas, sendo no entanto consensual que o conhecimento existente nos indivíduos é a base para as restantes dimensões.

1.1.3 A Gestão do Conhecimento

A Gestão do Conhecimento visa através de ferramentas e processos otimizar o conhecimento individual das organizações para o gerir mais eficazmente, identificando onde é que ele está,

quem o detém, quando e como foi gerado. É única porque incide sobre o indivíduo como um especialista e como o portador do conhecimento importante que ele pode partilhar sistematicamente com a organização (Rus, Lidvall, & Sinha, Knowledge Managment in Sofware Engineering, 2001).

O seu ciclo de vida é constituído por um conjunto de processos/fases, conforme a Figura 1, que visam de uma forma sistemática e com o auxílio de ferramentas capturar o conhecimento tácito existente na organização, transformá-lo em conhecimento explícito e organizá-lo para que possa ser partilhado e utilizado, possibilitando a geração de novo conhecimento.

Figura 1 - Ciclo de Vida da Gestão do Conhecimento adaptado (Rus, Lidvall, & Sinha, Knowledge Managment in Sofware Engineering, 2001)

O conhecimento não é uma receita, é um procedimento definido para lidar com uma situação concreta, uma situação de rotina. O conhecimento deve permitir lidar com diferentes situações, antecipar implicações e avaliar os seus efeitos (Cascão, 2014).

É possível estabelecer uma relação entre a ES e a GC? Estas são questões que serão apresentadas no capítulo seguinte.

1.2 ENGENHARIA DE SOFTWARE

1.2.1 Definição

De acordo com o SWEBOK[2] a ES é a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de *software*.

A ES é uma disciplina onde é necessário dominar tanto competências sociais como técnicas (Alawneh, Hattab, & Al-Ahmad, 2008), nela são realizadas um conjunto complexo de tarefas, que vão desde a identificação de requisitos, passando pelo *design*, a construção, os testes, a implementação e finalmente a manutenção do *software*.

A ES visa a organização, a produtividade e a qualidade através da aplicação de tecnologias e práticas de gestão de projetos nas atividades relacionadas com o desenvolvimento de aplicações ou sistemas.

1.2.2 Características

De acordo com o SWEBOK a ES é composta por um conjunto de áreas de conhecimento que são:

- Requisitos de *software*;
- *Design* de *software*;
- Construção de *software*;
- Teste de *software*;
- Manutenção de *software*;
- Gestão de configuração de *software*;
- Gestão de engenharia de *software*;
- Processos de Engenharia de Software;

[2] Software Engineering Body Of Knowledge

- Ferramentas e Métodos de Engenharia de Software;
- Qualidade de *software*.

As seis primeiras áreas estão relacionadas com os processos de desenvolvimento de *software* enquanto as quatro últimas dizem respeito a métodos e metodologias, ou seja, práticas para a realização dos processos.

Focando a análise nos processos da ES, ou projetos de desenvolvimento de *software*, identificam-se diversos modelos, alguns deles são:

- Cascata;
- Iterativo;
- Modelo em V;
- Espiral;
- Ágil; entre outros.

Estes modelos, conhecidos como modelos de ciclo de vida do desenvolvimento de *software* definem as atividades que permitem especificar e transformar requisitos de *software* no produto final ou entregável (Bourque & Fairley, 2014).

1.2.3 Fases do Ciclo de Vida do Desenvolvimento

Não obstante a particularidade de cada projeto de desenvolvimento de *software* existem sempre entre cada um deles alguns pontos em comum. Analisando os diversos modelos do ciclo de vida dos projetos de desenvolvimento de *software*, verifica-se que maioritariamente são compostos pelas fases indicadas na

Figura 2.

Figura 2 - Fases do Ciclo de Vida dos Projetos de Desenvolvimento de Software

A *criação* do conhecimento é originado nos clientes ou *stakeholders*, através da identificação de necessidades que visam melhorar a sua atividade. Os peritos efetuam a *análise* realizando reuniões, entrevistas e questionários *capturando-o* em documentos próprios de requisitos de negócio. Estas anotações são depois *transformadas* em documentos de *desenho* que servem de transferência entre o conhecimento do negócio e o conhecimento técnico. A *codificação* e os *testes***,** que funcionam como provas de conceito, do código gerado são a *implantação* sobre a forma de *software* do conhecimento reunido. Depois de deviamente certificado o produto final é *implementado*, efetuando a sua *partilha* pelos diversos sistemas ou ambientes. Finalmente entra numa atividade de *manutenção* com o objetivo corrigir problemas pontuais com o seu *uso***.**

Há uma interação entre a equipa técnica e os clientes, que são os donos do conhecimento do negócio. Esses clientes devem transferir o conhecimento do negócio para os analistas, para que

eles possam desenvolver modelos que devem ajudar a transferir conhecimento do negócio ao conhecimento técnico em modelos que descrevem o *software* (Camacho, Sanches-Torres, & Galvis-Lista, 2013).

2 A GESTÃO DO CONHECIMENTO EM PROJETOS DE DESENVOLVIMENTO DE SOFTWARE

A dependência em tecnologia por parte das organizações continua a aumentar, fazendo com que por um lado, os custos do desenvolvimento de *software* consumam uma fatia significativa do orçamento e por outro aumentem o volume e a diversidade da informação com que as organizações têm de lidar (Rus & Lindvall, Knowledge Management in Software Engineering, 2002), (Natali & Falbo, 2005).

A capacidade para eliminar os excessos sem perca de eficiência e qualidade são os desafios que as organizações de conhecimento intensivo têm de ultrapassar e o fator principal para uma abordagem da GC sobre o CVDS. De seguida são apresentados alguns dos desafios para as organizações na implementação da GC, bem como o estabelecimento de uma relação entre os projetos de desenvolvimento de *software* e a GC como introdução à abordagem deste trabalho sobre a sua implementação na ES.

2.1 DESAFIOS ORGANIZACIONAIS

Durante as diversas fases do CVDS é produzido um volume significativo de documentos, que servem para registar o conhecimento adquirido no decorrer de cada projeto. Lidar com a sua utilização é um dos desafios das organizações. A dificuldade em conseguir obter deles informação significativa, aceder e pesquisar o que realmente é importante num determinado contexto, faz com que toda essa documentação não seja utilizada e difícil de partilhar, para além de que a forma como se encontra dispersa faz com que não seja atualizada (Rus & Lindvall, Knowledge Management in Software Engineering, 2002), (Natali & Falbo, 2005).

Enquanto algum do conhecimento é registado, existe outro que se mantém na mente dos indivíduos (Natali & Falbo, 2005), o que significa que no final do dia esse conhecimento deixa a organização. As heurísticas que os indivíduos adquirem ao longo de anos de experiência não é possível de transmitir para outros no espaço de meses, criando lacunas de conhecimento na organização quando estes indivíduos deixam a organização, algumas das vezes com consequências graves e complexas de resolver.

Gerir o conhecimento e os seus fluxos é de extrema importância para as organizações, no entanto a GC não é produto que se possa adquirir, a GC leva o seu tempo, envolve processos, tecnologia e acima de tudo pessoas e organizações, de forma a criarem uma cultura de partilha onde os primeiros devem participar e os segundos devem encorajar, GC = Pessoas + Processos + Tecnologia (Leistner, 2010).

2.2 A Reificação de Conhecimento nas Organizações

O desenvolvimento de *software* é uma atividade coletiva, complexa e criativa (Natali & Falbo, 2005). Os projetos de desenvolvimento de *software* são compostos por equipas de indivíduos multidisciplinares. São estas equipas que aplicam no seu dia-a-dia as fases do ciclo de vida do desenvolvimento de *software*.

Ao realizarem estas atividades estas equipas, efetuam uma apropriação significativa do conhecimento, regras e processos de negócio existentes nos seus clientes. O uso de anotações para capturar aspetos do conhecimento tácito não é particularmente novo, no entanto, a contribuição distintiva deste trabalho permite relacionar a anotação diretamente com a taxonomia da competência (Vasconcelos, Kimble, Miranda, & Henriques, 2009).

Este conhecimento em conjunto com a experiência em projetos anteriores é diariamente explicitado por estas equipas, das mais diversas formas, representando o conhecimento organizacional. Estas anotações podem ser vistas como um exemplo do conhecimento gerado, que se encontravam externalizadas e que são registadas num contexto específico, permitindo a captura do conhecimento individual e de grupo. Como atividades que se encontram incorporadas no dia-a-dia do grupo de trabalho possibilitam preservar esse conhecimento para reutilização futura (Vasconcelos, Kimble, Miranda, & Henriques, 2009).

A GC identifica as atividades relacionadas com o processo de conhecimento na organização, conforme já apresentado na Figura 1, por outro lado, os Projetos de Desenvolvimento de Software são os processos executados pelos indivíduos nas atividades relacionadas com o desenvolvimento de *software,* compostos pelo ciclo de vida do desenvolvimento de software identificado anteriormente na

Figura 2.

É possível criar uma relação quase direta entre os processos de um e as atividades de outro, Figura 3, evidenciando que as atividades exercidas no desenvolvimento de *software* são acima de tudo processos transformação de conhecimento tácito em explícito (Aurum & Ward, 2004).

Figura 3 - Relação entre o ciclo de vida da Gestão do Conhecimento e os Projetos de Desenvolvimento de Software

As atividades realizadas nos projetos de desenvolvimento de *software*, quando devidamente documentadas, são atividades de reificação de conhecimento organizacional. É importante que todo este conhecimento produzido pelas equipas de projetos seja devidamente preservado, valorizado e gerido de forma adequada nas organizações. A gestão deste conhecimento possibilita por exemplo:

Criar informação sobre lições aprendidas, identificando boas e más práticas que podem servir de guias para novos projetos;

Permite que novos elementos possam obter informação sobre artefactos e projetos utilizando-o como material de estudo;

- Identificar os peritos em determinadas áreas;
- Registar boas práticas e lições aprendidas em projetos anteriores;
- Minimizar o tempo consumido na leitura de linhas de código para saber as fórmulas e regras;
- Registar problemas frequentes ocorridos e como atuar sobre os mesmos durante os processos de manutenção.

Grupos de pessoas em organizações de conhecimento intensivo precisam criar mecanismos para induzir a inovação, encontrar fontes de informação, gerir as competências de forma eficiente e recolher ideias e sugestões, a fim de fazer o seu trabalho (Vasconcelos, Kimble, Miranda, & Henriques, 2009).

Na realidade, a maioria deste tipo de organizações já lida e dispõe de tecnologia e processos suficientes que podem ser utilizados na implementação da Gestão do Conhecimento na Engenharia de Software. As ferramentas baseadas no conhecimento podem acelerar a transferência de conhecimentos especializados de engenheiros e, desta forma, reduzir o ciclo de vida do conhecimento (Lowry, 1993).

3 METODOLOGIA DE INVESTIGAÇÃO

Metodologia é o estudo dos métodos, das etapas a seguir num determinado processo. É o conjunto de técnicas e processos utilizados para a pesquisa e elaboração de um trabalho científico. É um processo utilizado para orientar uma investigação, o estudo de uma ciência ou para alcançar um determinado fim. A metodologia de investigação aborda as principais regras para a produção científica, fornecendo as técnicas, os instrumentos e os objetivos para uma melhor execução e qualidade do trabalho científico.

Pela natureza deste trabalho das diversas metodologias existentes a opção recaiu sobre a *Design Science Research* (DSR), uma vez que esta metodologia visa a criação de artefatos para a resolução de problemas da vida real, com o objetivo de tornar mais eficazes e eficientes as organizações para as quais a investigação se destina.

3.1 Design Science Research

O paradigma da DSR tem nas suas raízes a engenharia e "ciência do artificial" assim designada por Herbert Simon[3], referido por Hevner, seguindo uma analogia às ciências comportamentais com raízes nas ciências naturais. A DSR é fundamentalmente um paradigma para resolução de problemas com o objetivo de criar inovações que definem as ideias, práticas, capacidades técnicas ou produtos que podem ser tornados mais eficazes e eficientes através da análise, *design*, implementação, gestão e utilização de tecnologias de informação (Hevner, Ram, March, & Park, 2004).

Definições como construtos, modelos, métodos e instanciações não só fazem parte da DSR mas são também utilizados como linguagem no dia-a-dia dos sistemas de informação nas organizações (Hevner, Ram, March, & Park, 2004). Tal como as atividades do ciclo de vida do desenvolvimento, que são amplamente abordadas neste trabalho, também a DSR tem o seu ciclo de investigação, tornando-a por este motivo como uma metodologia naturalmente utilizada na realização de trabalhos de investigação relacionados com sistemas de informação como este.

[3] Foi um pesquisador nos campos de psicologia cognitiva, informática, administração pública, sociologia económica, e filosofia. Por vezes, descreveram-no como um polímata.

A DSR direciona a investigação para os sistemas de informação, tendo por base a relação existente entre a estratégia de negócio e tecnologias de informação das organizações, a infraestrutura da organização e a infraestrutura das tecnologias de informação (Hevner, Ram, March, & Park, 2004), utilizando estes quatro pilares e as ligações entre eles, cada vez mais significativas nos dias que correm para sustentar o processo de investigação.

Entre alguns dos problemas que a DSR visa resolver podemos encontrar os que são caracterizados por uma dependência crítica sobre as capacidades de sociabilização dos indivíduos nas organizações, como por exemplo o trabalho em equipa ou sobre as capacidades cognitivas para produzir soluções, nomeadamente a criatividade (Hevner, Ram, March, & Park, 2004). Estes dois tipos de problemas identificados por Hevner são muito representativos da investigação efetuada neste trabalho, nomeadamente pela sua relação ao tema da gestão do conhecimento nas organizações e nos projetos de desenvolvimento de *software,* com especial incisão sobre a MS.

3.2 Orientações de Investigação

São sete as linhas orientadoras de investigação da DSR (Hevner, Ram, March, & Park, 2004), nomeadamente:

- Desenho do artefacto
- Importância do problema
- Avaliação do desenho
- Contribuição da investigação
- Rigor na investigação
- Desenho do artefacto como processo de pesquisa
- Comunicação da pesquisa

Estas servem de guião para a investigação a efetuar e assentam no propósito que esta metodologia visa a resolução de problemas e que o conhecimento dos problemas, a sua compreensão e resolução são resolvidos com a criação e implementação de um artefacto (Hevner, Ram, March, & Park, 2004).

De seguida é efetuado o respetivo enquadramento de cada uma das linhas orientadoras da metodologia, de acordo com o trabalho realizado até ao momento.

3.2.1 Desenho do Artefacto

A criação/desenho de um artefacto é a primeira de todas, um artefacto entenda-se, pode ser um modelo, um constructo, um método ou uma instanciação. No caso deste trabalho optou-se pela elaboração de um modelo, apresentado no capítulo 4, que representa os principais fluxos de conhecimento na organização, os principais atores e as áreas de atuação do artefacto. Neste modelo de alto nível está identificado o fluxo inicial e a origem do conhecimento, que decorre da definição de requisitos de negócio. Foi utilizada como ferramenta para o *design* deste a linguagem *Unified Modelling Language*[4], o modelo será posteriormente refinado e reavaliado até à sua instanciação.

3.2.2 Importância do problema

A segunda orientação da metodologia é que o artefacto deverá endereçar um problema de negócio que possa ser resolvido através da implementação de uma solução tecnológica. Neste trabalho o problema identificado endereçado através da questão de investigação "Como poderá a MS beneficiar da adoção de métodos de GC?". Nas organizações de conhecimento intensivo, como é o caso das que efetuam desenvolvimento de *software*, o consumo de tempo desperdiçado na reanálise dos processos de MS poderá representar cerca de 60% dessa atividade (Anquetil, Oliveira, Sousa, & Dias, 2007).

Para uma atividade que se pretende que seja eficiente e objetiva, como é o caso da manutenção de *software*, este é um tema com uma importância relevante, como experiência pessoal constato que numa semana de trabalho o tempo gasto por indivíduo em processos de MS pode equivaler a cerca de 50% do seu tempo, ou seja, em cerca de 2,5 dias por semana em atividades de manutenção. Se sobre esse tempo for consumir por volta de 1 dia a tentar perceber o que está a

[4] UML - É uma linguagem de modelagem que permite representar um sistema de forma padronizada.

ser mantido, sendo que 1 ano tem 52 semanas, teremos cerca de 50 dias, ou seja, quase dois meses de trabalho por ano que poderá ser gasto a tentar perceber o que está a ser mantido.

3.2.3 Desenho do artefacto como processo de pesquisa

DSR é uma investigação iterativa, uma vez que na génese do seu processo está a procura para criar um artefacto, muitas vezes a pesquisa incessante para criar o melhor artefacto possível acaba por ser contraproducente, fazendo com que o produto final acabe por não se encaixar na realidade da organização. Muitas vezes a experiência pessoal, as heurísticas do investigador, acabam por determinar a criação de um artefacto mais adequado à realidade da organização. A geração de versões e testes sobre elas que produzem novos requisitos e necessidades que por sua vez criam novas versões é um processo identificado por Hevner como o ciclo Geração/Teste (Hevner, Ram, March, & Park, 2004).

O processo de pesquisa para o desenho do artefacto, modelo MIMIR, foi realizado de forma iterativa, através da revisão da literatura e consequente redesenho do modelo. A minha experiência pessoal e o conhecimento dos processos existentes permitiram adequar o modelo à realidade organizacional, tendo sido para tal criadas diversas versões como forma de o otimizar e ajustar o mais possível. Seguindo a metodologia apresentada e como trabalho futuro será efetuado o seu refinamento, as reavaliações necessárias e a respetiva instanciação, tudo também efetuado de forma iterativa.

4 GESTÃO DO CONHECIMENTO NA ENGENHARIA DE SOFTWARE

O conhecimento em projetos de desenvolvimento de *software* é diverso e em constante crescimento. Organizações têm dificuldade em identificar o conteúdo, a localização e a melhor forma de tirar partido desse conhecimento (Rus & Lindvall, Knowledge Management in Software Engineering, 2002).

Um passo importante para um projeto de GC é a caracterização de um ativo de conhecimento. O conhecimento de domínio inclui processos de negócios, processos de decisão, competências empresariais, o conhecimento declarativo e processual, heurísticas e conhecimento informal (Vasconcelos, Kimble, Miranda, & Henriques, 2009).

Neste trabalho as atividades do CVDS foram segmentadas em três áreas de processo que derivaram da

Figura 2, tendo em conta a combinação dos seus output's com as suas necessidades. As áreas de processo são:

- Definição de Requisitos (DR);
- Desenvolvimento de *Software* (DS);
- Manutenção do *Software* (MS).

Esta segmentação visa facilitar a identificação de todos os documentos produzidos nas diversas fases e determinar para cada um deles a informação mais relevante, conforme se pode observar na Figura 4.

Figura 4 - Segmentação de CVDS de acordo com os seus outputs e necessidades

Constata-se que a introdução de métodos de Gestão do Conhecimento (GC) nos processos de Engenharia de *Software* (ES) nas Organizações é maioritariamente centrado nas fases do Ciclo de Vida do Desenvolvimento de S*oftware* (CVDS) anteriores à MS, mais concretamente na definição de requisitos, desenho e codificação. (Rus & Lindvall, Knowledge Management in Software Engineering, 2002), (Aurum & Ward, 2004), (Natali & Falbo, 2005), (Alawneh, Hattab, & Al-Ahmad, 2008), (Camacho, Sanches-Torres, & Galvis-Lista, 2013), (Isotani, Dermeval, Bittencourt, & Barbosa, 2015) .

Como tal, o foco deste trabalho é sobre a área MS, onde as abordagens da GC na ES têm sido menos exploradas, mas onde a importância de gerir o conhecimento poderá significar intervenções mais cirúrgicas, objetivas e eficazes com a consequente redução do tempo e respetivos custos para a organização.

Verifica-se que os documentos produzidos nas fases anteriores do CVDS raramente são utilizados como ajuda para a resolução de problemas no processo de MS (Carreteiro, Vasconcelos, Barão, & Rocha, 2016), onde as equipas acabam por recorrer à leitura do código fonte ou a trocas informais de ideias com os seus pares para encontrarem a resolução de problemas, estimando-se que 40% a 60% do esforço das equipas de MS é gasto a tentarem perceber o *software* que estão a manter. Os indivíduos envolvidos na MS lidam com problemas

reais e por isso a sua atividade pode ser utilizada para avaliar o conhecimento documentado nas fases anteriores do CVDS.

5 MODELO MIMIR

5.1 Introdução ao Modelo

A MS é uma atividade complexa e exigente, os indivíduos que dela fazem parte têm de dominar linguagens de programação, arquitetura do sistema, modelos de dados, conhecimento de processos, saber sobre os impactos das atualizações no *software* entre outros, sendo que por vezes este conhecimento é multiplicado por várias aplicações, concluindo-se que na MS tem de ser gerida uma quantidade significativa de informação (Rodríguez, Vizcaíno, & Martínez, 2004).

Os indivíduos mais experientes acabam por ser os escolhidos para esta tarefa devido à sua senioridade e heurísticas, tornando-os como tal os eleitos pelas organizações para assumir estas funções, realçando assim que esta atividade exige um elevado nível de conhecimento, sensibilidade e experiência devido à criticidade e urgência da mesma.

O modelo MIMIR apresentado em seguida, cujo nome deriva do deus da mitologia nórdica, o mais sábio dos deuses nórdicos e guardião da "Fonte de todo o Conhecimento", é uma abordagem para mapear a informação produzida com base nos documentos criados nas fases CVDS, com o objetivo de extrair deles a informação mais relevante para a MS.

O modelo é baseado nos documentos de referência do CMMI-DEV[5], que consiste num conjunto de boas práticas direcionadas para atividades de desenvolvimento de produtos e serviços. Para melhor compreender o modelo MIMIR é importante perceber os conceitos inerentes à produção dos documentos que derivam do CMMI-DEV, motivo pelo qual será de seguida realizada uma pequena apresentação da metodologia e da documentação adequada aos processos do CVDS.

5.1.1 CMMI

O modelo CMMI foi criado para satisfazer as necessidades do departamento de defesa dos Estados Unidos por volta dos anos 80. Com o aparecimento de sistemas operativos mais leves

[5] CMMI-DEV – Capability Maturity Model® Integration for Development é um modelo de boas práticas que ajuda as organizações a melhorar os seus processos.

e com utilização mais gráfica (ex. Windows) o *software* do departamento começou a ser desenvolvido por empresas externas com um controlo do próprio departamento.

Apesar de existir um controlo apertado no que dizia respeito à segurança com que o desenvolvimento era efetuado, a sua externalização acabou por levantar outras questões. Era essencial assegurar que o *software* era entregue no prazo definido, com o orçamento estabelecido e que as funcionalidades pedidas estavam asseguradas. Esta necessidade por parte do departamento de defesa acabou por se tornar num requisito de "qualidade no desenvolvimento de *software*" (Liberato, 2008).

A resposta a esta necessidade surgiu da Carnegie Mellon University's Software Engineering Institute que concebeu um modelo que conseguisse definir os elementos essenciais dos processos de desenvolvimento, que designaram de Capability Maturity Model. Decorrente da sua grande aceitação e resultados efetivos nas organizações este modelo tornou-se uma referência, contendo as melhores práticas para os projetos de desenvolvimento de *software*, uma vez que descreve os diferentes estágios de maturidade por onde as organizações passam na evolução dos ciclos de desenvolvimento.

O seu sucesso acabou por dar origem à criação de vários modelos de maturidade com aplicação noutras áreas, no entanto e apesar do valor que eles representam para as organizações a implementação destes vários modelos representava custos acrescidos tanto a nível financeiro como a nível de complexidade da sua implementação (Software Engineering Institute, Novembro 2010). Como resposta a este problema foi criado o Capability Maturity Model Integration, que integrava três modelos de maturidade, a saber:

- Capability Maturity Model for Software (SW-CMM) v2.0 draft C
- Systems Engineering Capability Model (SECM)
- Integrated Product Development Capability Maturity Model

Os modelos foram escolhidos por terem entre si diferentes abordagens ao processo de melhoria e porque eram extremamente populares nas comunidades de desenvolvimento de *software* e de engenharia de sistemas (Software Engineering Institute, Novembro 2010). A utilização de modelos já aceites e com provas dadas, permitiu que a integração tivesse um conjunto de modelos passíveis de continuar a ser adotados por quem já tinha implementado os modelos *per*

si, e também que quem ainda não tivesse adotado os modelos de maturidade tivesse referências com melhorias efetivas.

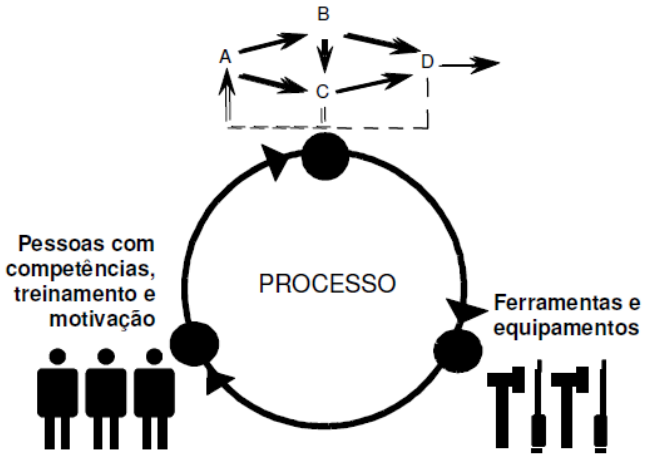


Figura 5 - As três dimensões críticas numa organização (Software Engineering Institute, Novembro 2010)

O modelo de maturidade permitiu a organizações aplicar normas a projetos de desenvolvimento, entregues a empresas externas garantindo um aumento nas taxas de sucesso dos mesmos. O CMMI funciona como um guia para a melhoria das três dimensões críticas numa organização, Figura 5, através da otimização dos processos das mesmas e capacidade de gerir o desenvolvimento, aquisição e manutenção de produtos e serviços (Liberato, 2008).

5.1.2 CMMI for Development

CMMI-DEV é composto por melhores práticas que abordam atividades de desenvolvimento de produtos e de serviços. Define práticas que cobrem o ciclo de vida do produto, desde a sua conceção até a entrega e manutenção. A ênfase é sobre o trabalho necessário para construir e manter o produto total (Software Engineering Institute, Novembro 2010). Assim, fornecendo as orientações necessárias para documentar e nortear o conhecimento produzido nos projetos de desenvolvimento de *software*, o modelo CMMI-DEV contempla quatro grandes áreas de processos, nomeadamente:

- Gestão de Processos;
- Gestão de Projetos;
- Engenharia;
- Suporte.

Os processos da área de Engenharia, que estão diretamente relacionados com o CVDS e que fornecem a base de análise para o modelo MIMIR, enquadram todas as atividades de requisitos, desenvolvimento e manutenção. São processos com a estratégia orientada à elaboração, implementação e otimização de um produto. As cinco áreas de processos da Engenharia, representadas na Figura 6, são:

- Definição de Requisitos (RD)
- Solução Técnica (TS)
- Validação (VAL)
- Verificação (VER)
- Integração de Produto (PI)

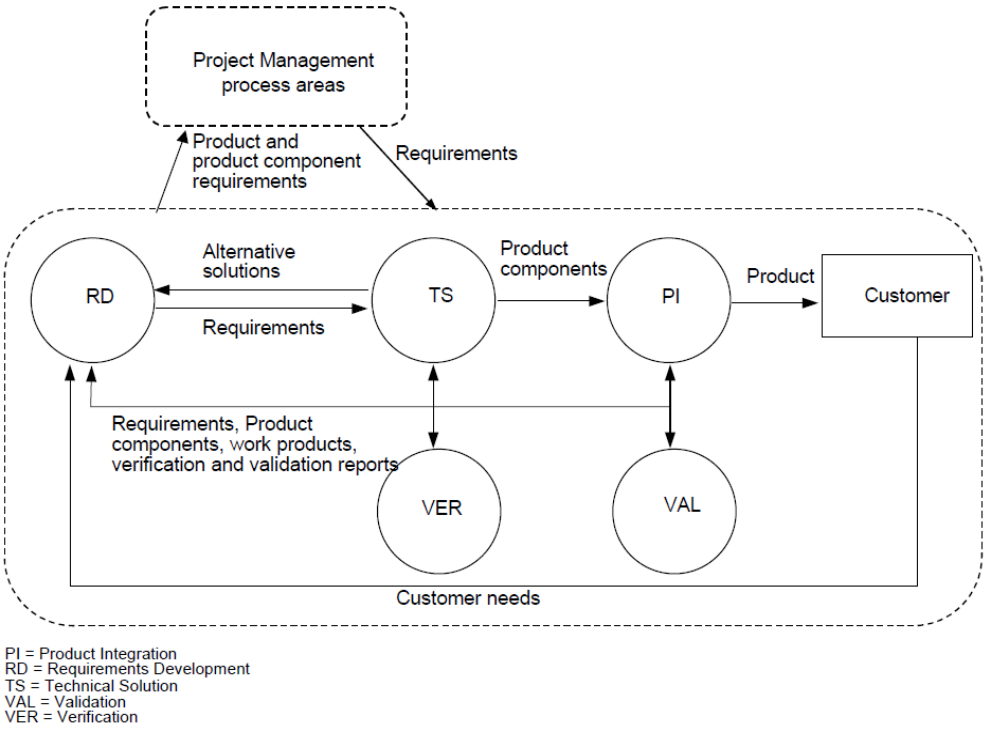


Figura 6 – Áreas de processo da Engenharia no modelo CMMI-DEV (Software Engineering Institute, Novembro 2010)

O processo de Definição de Requisitos (RD) diz respeito à elaboração, análise e levantamento de todas as necessidades dos stakeholders[6]. Sendo de extrema importância que as necessidades dos "patrocinadores" do projeto (no modelo MIMIR identificados como "Sponsors") sejam perfeitamente entendidas e delimitadas.

A correta definição e entendimento das necessidades, para além da importância que tem na qualidade do produto final, produz também informação essencial para outras fases do CVDS, como os critérios para as matrizes de testes ou determinados atributos do produto como por exemplo segurança, usabilidade e de manutenção (Software Engineering Institute, Novembro 2010), esta última essencial no foco do modelo MIMIR. São elaborados documentos onde se descrevem os requisitos de negócio (BRS), Figura 7, e documentos de requisitos funcionais (SRS ou PRS), Figura 8, que serão utilizados pelo modelo MIMIR para popular o modelo de dados de conhecimento.

3. REQUISITOS/NECESSIDADES DO SPONSOR

3.1. REQUISITOS/NECESSIDADES

<descrever os requisitos/necessidades, dando um id único a cada um>

ID BRS	BRS1	Nome	
Descrição do requisito/ Necessidade			
Regras de negócio Do requisito (rr)	Rr1: Rr2:		

3.2. REGRAS CONTABILÍSTICAS

ID BRS	Nome	Regra contabilística

3.3. NECESSIDADES DE INFORMAÇÃO DE GESTÃO

<nesta secção deverão ser identificadas as necessidades de informação de gestão, tais como **relatórios** ou **indicadores de gestão**, que devem ser articuladas com o dig e o do, **se não se aplica pôr n/a**.>

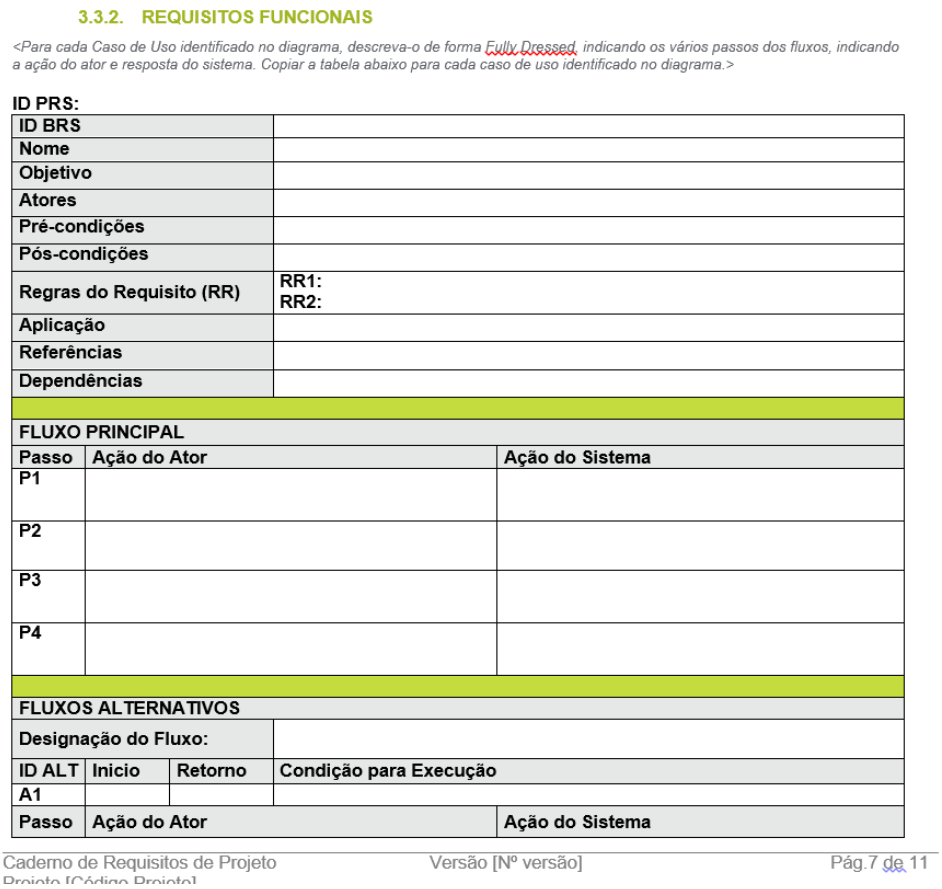
ID BRS	Nome	Dados necessários

3.4. REQUISITOS DE USABILIDADE

Figura 7 – Exemplo de parte do documento de requisitos de negócio

[6] Parte interessada ou interveniente no projeto.

São documentados todo o tipo de requisitos, desde os que dizem efetivamente respeito ao negócio, origem do projeto, mas também todos os outros requisitos colaterais, que sejam necessários existir derivado do objetivo a atingir.



3.3.2. REQUISITOS FUNCIONAIS

<Para cada Caso de Uso identificado no diagrama, descreva-o de forma Fully Dressed, indicando os vários passos dos fluxos, indicando a ação do ator e resposta do sistema. Copiar a tabela abaixo para cada caso de uso identificado no diagrama.>

ID PRS:

ID BRS			
Nome			
Objetivo			
Atores			
Pré-condições			
Pós-condições			
Regras do Requisito (RR)			**RR1:** **RR2:**
Aplicação			
Referências			
Dependências			
FLUXO PRINCIPAL			
Passo	**Ação do Ator**		**Ação do Sistema**
P1			
P2			
P3			
P4			
FLUXOS ALTERNATIVOS			
Designação do Fluxo:			
ID ALT	**Inicio**	**Retorno**	**Condição para Execução**
A1			
Passo	**Ação do Ator**		**Ação do Sistema**

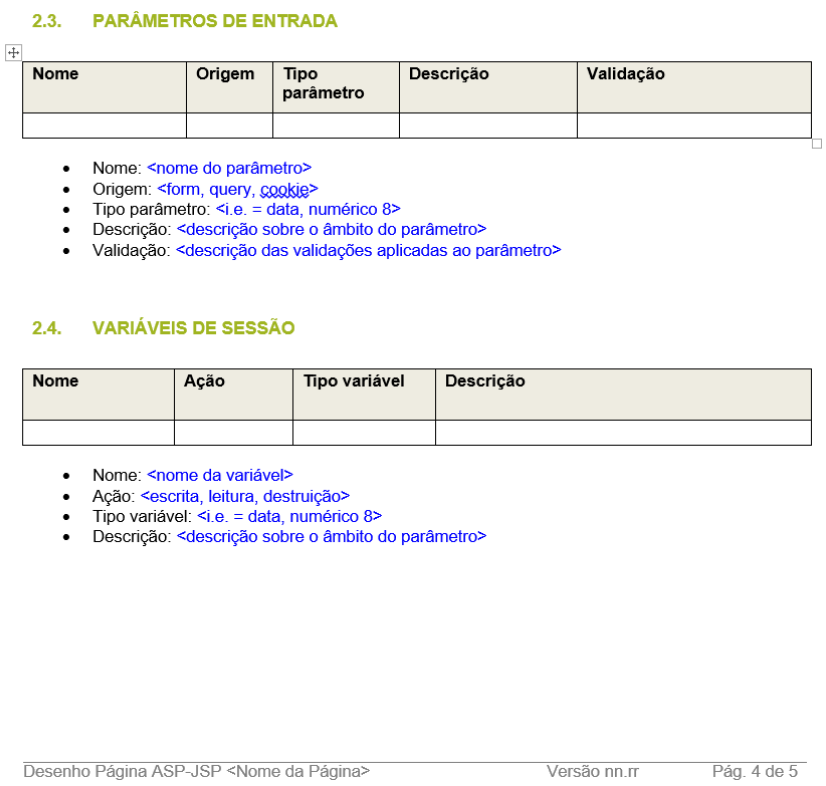
Caderno de Requisitos de Projeto Versão [Nº versão] Pág.7 de 11
Projeto [Código Projeto]

Figura 8 - Exemplo de parte do documento de requisitos de funcionais

A área de processo Solução Técnica (TS) consiste no desenvolvimento propriamente dito do produto que será posteriormente utilizado no processo de Integração de Produto. O modelo prevê neste processo o estudo de soluções alternativas para determinar a melhor solução técnica a implementar com base em determinados critérios. Critérios esses que podem ser significativamente diferentes de solução para solução, dependendo do tipo de produto, ambiente operacional, os requisitos de desempenho e de suporte de custo ou de calendário de entrega.

A área de processo de Verificação (VER) é utilizada como complemento à área de processo Solução Técnica (TS), para realizar a verificação do desenho e *peer review*[7] durante a fase de desenvolvimento da solução técnica e antes da versão final do produto.

Documentos como desenhos técnicos, conforme Figura 9, onde são descritas e representadas as alterações ou implementações a realizar decorrente dos requisitos especificados anteriormente, em linguagem natural ou em linguagens de modelação, como por exemplo o UML[8], são tipicamente documentos associados à área de processo Solução Técnica.



2.3. PARÂMETROS DE ENTRADA

Nome	Origem	Tipo parâmetro	Descrição	Validação

- Nome: <nome do parâmetro>
- Origem: <form, query, cookie>
- Tipo parâmetro: <i.e. = data, numérico 8>
- Descrição: <descrição sobre o âmbito do parâmetro>
- Validação: <descrição das validações aplicadas ao parâmetro>

2.4. VARIÁVEIS DE SESSÃO

Nome	Ação	Tipo variável	Descrição

- Nome: <nome da variável>
- Ação: <escrita, leitura, destruição>
- Tipo variável: <i.e. = data, numérico 8>
- Descrição: <descrição sobre o âmbito do parâmetro>

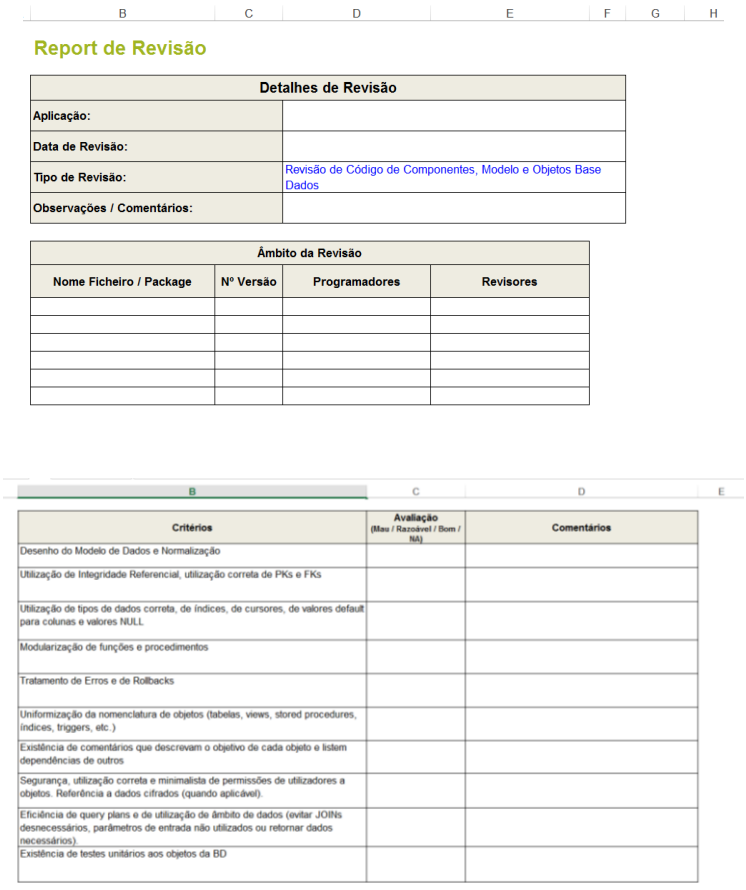
Desenho Página ASP-JSP <Nome da Página> Versão nn.rr Pág. 4 de 5

Figura 9 - Exemplo de parte do documento de técnico ASP/JSP

[7] Processo de avaliação do trabalho desenvolvido por outrem com competências similares ou superiores com o objetivo de manter *standards* de qualidade e performance.

[8] Unified Modeling Language

Os relatórios de *peer review*, Figura 10, relacionados com a área de Verificação, são utilizados para análise e avaliação por elementos com maior experiência e conhecimento técnico sobre o trabalho desenvolvido pelos elementos da equipa.



Report de Revisão

Detalhes de Revisão	
Aplicação:	
Data de Revisão:	
Tipo de Revisão:	Revisão de Código de Componentes, Modelo e Objetos Base Dados
Observações / Comentários:	

Âmbito da Revisão			
Nome Ficheiro / Package	Nº Versão	Programadores	Revisores

Critérios	Avaliação (Mau / Razoável / Bom / NA)	Comentários
Desenho do Modelo de Dados e Normalização		
Utilização de Integridade Referencial, utilização correta de PKs e FKs		
Utilização de tipos de dados correta, de índices, de cursores, de valores default para colunas e valores NULL		
Modularização de funções e procedimentos		
Tratamento de Erros e de Rollbacks		
Uniformização da nomenclatura de objetos (tabelas, views, stored procedures, índices, triggers, etc.)		
Existência de comentários que descrevam o objetivo de cada objeto e listem dependências de outros		
Segurança, utilização correta e minimalista de permissões de utilizadores a objetos. Referência a dados cifrados (quando aplicável).		
Eficiência de query plans e de utilização de âmbito de dados (evitar JOINs desnecessários, parâmetros de entrada não utilizados ou retornar dados necessários).		
Existência de testes unitários aos objetos da BD		

Figura 10 - Exemplo de parte do documento de *peer review*

A área de processo de Validação (VAL) está relacionada com a realização de testes efetuados sobre o produto, que permitam criar evidências de que o mesmo vai de encontro às necessidades identificadas nos requisitos. Estes testes comprovam o funcionamento do produto, funcionando como uma prova de conceito. A validação pode ser realizada no ambiente operacional ou num ambiente operacional simulado.

A elaboração de documentos como matrizes e planos de testes, Figura 11, onde no primeiro são descritos os testes que se pretendem efetuar sobre cada uma das componentes alteradas em

função dos requisitos definidos, e no segundo onde são estabelecidas as estratégias, necessidades e critérios para os testes a realizar.

INFORMAÇÃO DO DOCUMENTO

A. GESTÃO DE VERSÕES

Data entrega	Versão	Descrição	Capítulos alterados	Responsável
dd/mm/aaaa	Nn.rr			

1. ÂMBITO

O âmbito deste documento é estabelecer as estratégias, necessidades e critérios a serem seguidos para todas as atividades de teste para o projeto NNNNNN PDT vnn.rr ,para garantir que todos os requisitos presentes no documento <PRS nome documento> do projeto são abrangidos.

2. OBJETIVO

O objetivo deste documento inclui a estratégia de testes a seguir, a identificação dos produtos a serem testados, a definição dos critérios para a aceitação dos testes, a identificação dos entregáveis para a atividade, bem como as necessidades do ambiente, dados e estratégias a serem seguidas na execução dessas atividades.

3. DOCUMENTOS RELACIONADOS / REFERÊNCIAS

Os seguintes documentos serviram de input para a preparação deste documento:

Nome documento e nº versão	Descrição

3.1. ACRÓNIMOS, ABREVIATURAS E DEFINIÇÕES

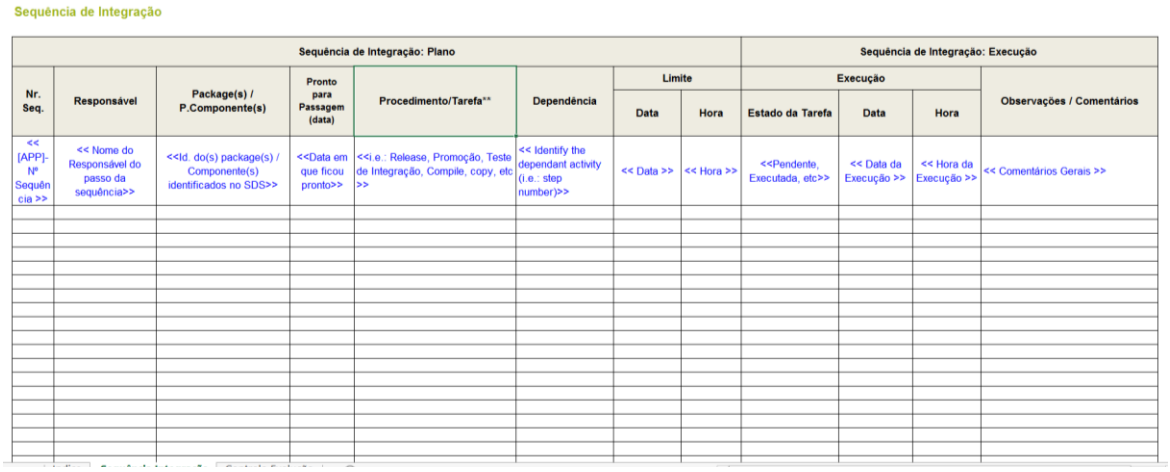
As seguintes abreviaturas, acrónimos e definições foram usados no documento.

Acrónimo/Abreviatura	Descrição

Figura 11 - Exemplo de parte do documento de plano de testes

Por último, a área de processo Integração de Produto (PI) contém as práticas específicas associadas à geração de uma estratégia de integração, integração de componentes do produto, e entregando o produto ao cliente. Esta área de processo é suportada por processos tanto da área de Verificação como da área de Validação, para identificar as componentes concluídas e consequentemente passíveis de serem entregues.

O Plano de Integração, Figura 12, é o documento relacionado com esta área de processo, apesar de ser um documento próprio de uma fase final do CVDS é de todo importante que o seu preenchimento se realize durante o decorrer das fases anteriores, minimizando assim o risco de falhas.



Sequência de Integração

Sequência de Integração: Plano								Sequência de Integração: Execução			
Nr. Seq.	Responsável	Package(s) / P.Componente(s)	Pronto para Passagem (data)	Procedimento/Tarefa**	Dependência	Limite		Execução			Observações / Comentários
						Data	Hora	Estado da Tarefa	Data	Hora	
<< [APP]-Nº Sequência >>	<< Nome do Responsável do passo da sequência>>	<<Id. do(s) package(s) / Componente(s) identificados no SDS>>	<<Data em que ficou pronto>>	<<i.e.: Release, Promoção, Teste de Integração, Compile, copy, etc >>	<< Identify the dependant activity (i.e.: step number)>>	<< Data >>	<< Hora >>	<<Pendente, Executada, etc>>	<< Data da Execução >>	<< Hora da Execução >>	<< Comentários Gerais >>

Figura 12 - Exemplo de parte do documento do plano de integração

Todos os documentos apresentados, principais referências associadas ao modelo CMMI-DEV, são a estrutura para alimentar a base de conhecimento do modelo MIMIR. Nos próximos capítulos será feita a apresentação da framework MIMIR, primeiro apresentando uma visão de alto nível onde se pretende mostrar o modelo na sua totalidade, para dar toda a abrangência do mesmo, e em seguida é detalhada a vertente relacionada com o segmento da Definição de Requisitos.

5.2 Visão de Alto-nível do Modelo e atores

O *driver* para a criação deste modelo é o facto de que as equipas de manutenção de *software* são confrontados com problemas efetivos e por esse motivo têm uma necessidade premente de ter acesso a conhecimento relevante que os possa auxiliar quando confrontados com situações de erro (Carreteiro, Vasconcelos, Barão, & Rocha, 2016). Esta necessidade possibilita com que estes indivíduos possam atuar como agentes validadores da informação existente, permitindo que corrijam ou atualizem o conhecimento obtido (Carreteiro, Vasconcelos, Barão, & Rocha, 2016). A partir dessa mesma atualização é aplicada a técnica de back flushing para corrigir os dados na origem, neste caso as bases de conhecimento onde fica registado o conhecimento das fases anteriores conforme representado na Figura 13.

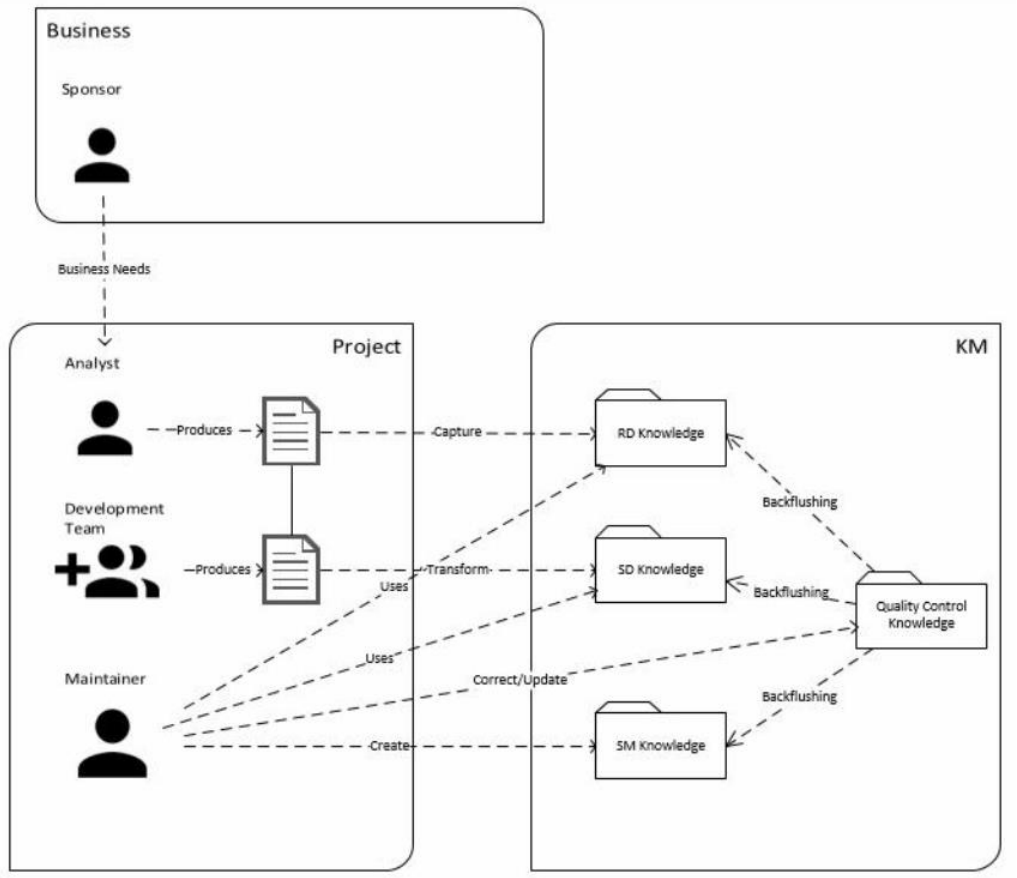


Figura 13 - Modelo MIMIR (Alto Nível) (Carreteiro, Vasconcelos, Barão, & Rocha, 2016)

Este modelo visa uma abordagem preliminar às fases do CVDS em organizações de conhecimento intensivo, baseada na segmentação em três áreas, de acordo com a Figura 4, e para o qual foram definidos um conjunto de atores, utilizando a representação UML, criando estereótipos dos atores nomeadamente: *Sponsor*, Analista, Equipa de Desenvolvimento e Equipa de Manutenção (Maintainer).

O papel do *Sponsor* é definir as necessidades do projeto e validador no que à qualidade dos requisitos diz respeito, estes podem ser internos ou externos à organização. O analista é o perito para este modelo, fornece os requisitos discutidos com o *Sponsor*, a análise aplicacional, sendo este ator a principal fonte do conhecimento no projeto. A equipa de desenvolvimento é composta pelos indivíduos que participam no projeto essencialmente na fase de codificação.

Em projetos menores o analista pode acabar por integrar a equipa de desenvolvimento. A equipa de manutenção são os indivíduos que dão suporte à aplicação ou sistema. Se o analista é exclusivamente um analista de negócio não terá qualquer participação na equipa de manutenção, no entanto se for um analista aplicacional fará com toda a certeza parte da equipa de manutenção.

No que às áreas de conhecimento definidas no modelo diz respeito, a "RD Knowledge", será onde ficará armazenado todo o conhecimento relacionado com a vertente de levantamento de requisitos. A "SD Knowledge" servirá para registar a informação das fases de desenvolvimento, como por exemplo especificações técnicas para desenvolvimento, ligações entre *software*, regras existentes no código, etc. A "SM Knowledge" irá guardar informação específica dos processos de manutenção, como por exemplo problemas rotineiros e a sua solução, sequência de execução do processos *batch*[9], sequência de navegação na aplicação, etc. Por último, a "Quality Control Knowledge" é onde residirão as correções e atualizações ao conhecimento existente nas outras bases. As correções ficarão aqui até que os peritos (analistas) façam uma avaliação das correções propostas para dar início ao processo de correção na fonte, *back flushing*.

5.3 Captura de conhecimento da Definição de Requisitos

O MIMIR centra-se na extração de conhecimento gerado nos documentos produzidos de acordo com metodologia CMMI-DEV pelos elementos das equipas de conhecimento intensivo durante as fases do CVDS. A elicitação das necessidades do *Sponsor* por parte do Analista, decorrente de reuniões e entrevistas, irão originar a criação do documento BRS, que irá conter uma descrição de alto-nível das necessidades do negócio.

Na Figura 14 encontra-se a representação do modelo MIMIR referente à criação, recolha e alimentação da base de dados de conhecimento da fase de Definição de Requisitos e cujo seu fluxo será de seguida descrito.

[9] É a execução de uma série de programas sem intervenção manual.

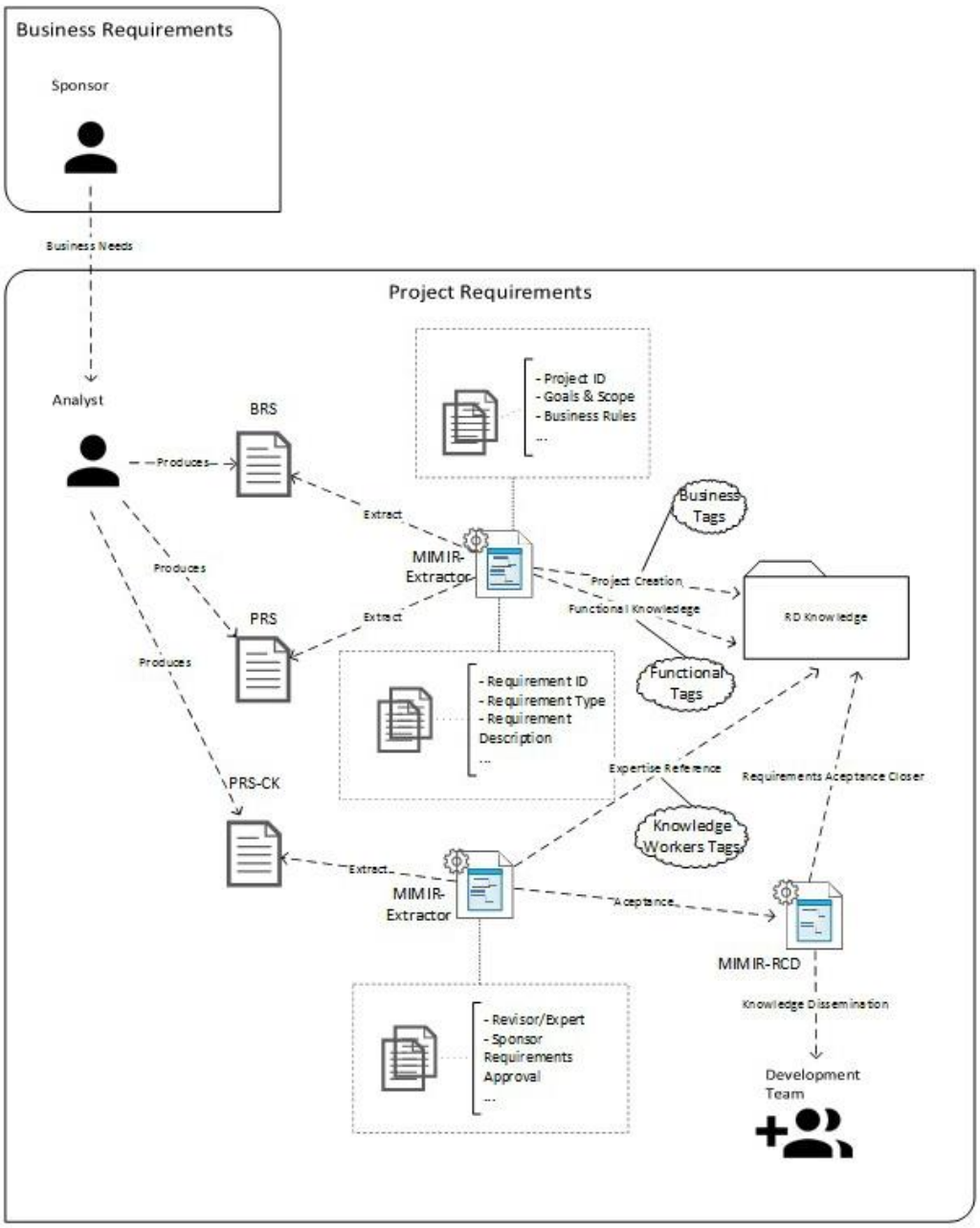


Figura 14 – Modelo MIMIR para a fase de Definição de Requisitos (Carreteiro, Vasconcelos, Barão, & Rocha, 2016)

Através da utilização de um módulo próprio e especificamente dedicado à extração de informação existente nos documentos, designado de MIMIR-*Extractor*, irá colher elementos do documento tais como a identificação (código unívoco por projeto), a sua descrição e o seu âmbito. Estes elementos desencadearão o momento de criação do projeto na base de dados de

conhecimento do MIMIR e adicionalmente permitirá a criação das primeiras *tags*[10] associadas ao projeto, que posteriormente irão facilitar as pesquisas e o fluxo de conhecimento entre os indivíduos que participam no projeto e também os das equipas de manutenção.

Também produzido pelo *Analista* nesta fase do CVDS é o documento das Especificações dos Requisitos Funcionais (PRS) do projeto. Elaborado por este, imediatamente após os requisitos de negócio estarem formalmente aceites pelos *Sponsors*, é um documento que não sendo especificamente de cariz técnico, já deverá conter menções a conceitos técnicos direcionados para a equipa de desenvolvimento, com o objetivo de permitir uma melhor compreensão dos elementos desta e facilitar a associação entre um requisito e o impacto que ele representa na estrutura técnica do sistema ou sistemas.

Deste documento o extrator retirará de cada requisito funcional o seu código unívoco de identificação, a sua descrição, o tipo de requisito funcional e a área funcional do sistema a que este se refere. Adicionando esta informação à base de dados do conhecimento do MIMIR pelo extrator, que a associará ao projeto criado anteriormente no tratamento do BRS, fornecerá conhecimento numa linguagem natural, rápida e simples de interpretar pelos indivíduos. Utilizando o seu código unívoco para cada requisito permitirá ao MIMIR, posteriormente na fase de desenvolvimento, mapear cada um dos diversos artefactos que são impactados no projeto.

O último documento desta fase e que será utilizado pelo MIMIR para considerar como concluída a fase de *Definição de Requisitos*, é a Checklist dos Requisitos Funcionais (PRS-CK). Daqui será identificado o Analista envolvido e responsável pela elicitação dos requisitos, referência para posterior identificação do especialista aplicacional que poderá prestar esclarecimentos e orientações sobre o projeto.

Este documento, elaborado apenas após a aceitação dos requisitos por parte dos *Sponsors* irá determinar a conclusão da fase de levantamento e definição de requisitos. Um novo módulo do modelo será despoletado, o MIMIR-RCD que alterará o *status* do projeto na base de dados de conhecimento da Definição de Requisitos e adicionalmente e de forma automática fará a

[10] Em português etiqueta, é uma palavra-chave (relevante) ou termo associado com uma informação (ex: uma imagem, um artigo, um vídeo) que o descreve e permite uma classificação da informação baseada em palavras-chave.

disseminação e distribuição do conhecimento aos indivíduos da equipa de desenvolvimento, para que iniciem o seu processo de apropriação.

CONCLUSÃO E TRABALHO FUTURO

O desenvolvimento de *software* é um processo de conhecimento intensivo e as atividades do ciclo de vida de desenvolvimento de software são a reificação do conhecimento organizacional. O conhecimento que resulta destas atividades é amplamente documentado, no entanto, muito pouco é reutilizado seja para novos projetos ou como auxílio na manutenção de *software,* para além daquele que ainda fica residente nos indivíduos, criando uma lacuna de conhecimento organizacional.

Como fazer uso de todo esse conhecimento explicitado e como explicitar o que ainda reside na mente dos indivíduos são desafios que as organizações têm pela frente. A proximidade entre as atividades do ciclo de vida de desenvolvimento e a gestão do conhecimento pode facilitar a implementação deste tipo de iniciativas.

Da revisão da literatura constata-se que muitas destas iniciativas incidem essencialmente nas fases iniciais do ciclo de vida de desenvolvimento de *software*, essencialmente na definição e levantamento de requisitos, deixando de parte a manutenção de *software*. Verifica-se também que a manutenção de *software* tem uma grande necessidade de acesso ao conhecimento criado no decorrer dos projetos, uma vez que até cerca de 60% do tempo gasto nesta atividade é a tentar perceber o que está a ser mantido.

Este trabalho visa precisamente a criação de um modelo, onde a partir de todos os documentos criados nas fases prévias do ciclo de vida de desenvolvimento, se possa extrair conhecimento útil que sirva de suporte às equipas que participam na atividade de manutenção de *software* e utilizar estes mesmos indivíduos como validadores desse mesmo conhecimento explicitado anteriormente.

Como trabalho futuro serão detalhadas as restantes áreas do modelo MIMIR, tanto no processo de alimentação da base de dados de conhecimento como no processo de *backflushing*, bem como a instanciação do mesmo, mantendo sempre o foco do modelo nas organizações de desenvolvimento de *software*, combinando a experiência em projetos de software corporativos como meio para avaliar os seus efeitos em pessoas, fluxos de trabalho e processos.

Adicionalmente, apesar de não se tratar do objetivo principal deste trabalho, pretende-se validar com a instanciação do modelo, se a implementação deste tipo de iniciativa na fase de manutenção pode funcionar como um facilitador para a adoção da gestão do conhecimento pela organização.

BIBLIOGRAFIA

Alawneh, A. A., Hattab, E., & Al-Ahmad, W. (2008). An Extended Knowledge Management Framework during the Software Development Life Cycle. *International Technology Management Review*.

Anquetil, N., Oliveira, K. M., Sousa, K. D., & Dias, M. G. (May de 2007). Software maintenance seen as a knowledge management issue. *INFORMATION AND SOFTWARE TECHNOLOGY*.

Aurum, A., & Ward, J. (2004). Knowledge Management in Software Engineering - Describing the Process. *2004 Australian Software Engineering Conference*.

Bourque, P., & Fairley, R. E. (2014). *SWEBOK v3.0 - Guide to the Software Engineering Body Of Knowledge*. New Jersey: IEEE Computer Society.

Camacho, J. J., Sanches-Torres, J. M., & Galvis-Lista, E. (Março de 2013). Understanding the Process of Knowledge Transfer in Software Engineering: A Systematic Literature Review.

Carreteiro, P., Vasconcelos, J. B., Barão, A., & Rocha, Á. (2016). A Knowledge Management Approach for Software Engineering Projects Development. *World Cist 2016*. Recife: Springer International.

Cascão, F. (2014). *Gestão de Competências, do conhecimento e do talento*. Lisboa: Edições Sílabo, Lda.

Davenport, T. H. (2010). Process Management for Knowledge Work. Em *Handbook on Business Process Management 1*. Springer.

Hevner, A. R., Ram, S., March, S. T., & Park, J. (March de 2004). Design Science Research In Information Systems Reserarch. *Mis Quartely*, pp. 75-105.

Isotani, S., Dermeval, D., Bittencourt, I., & Barbosa, E. (Março de 2015). Ontology Driven Software Engineering A Review of Challenges and Opportunities. *IEEE LATIN AMERICA TRANSACTIONS*.

Leistner, F. (2010). *Mastering Organizational Knowledge Flow.* New Jersey: John Wiley & Sons, Inc.

Liberato, M. E. (2008). *Implementação do Modelo CMMI na Espirito Santo Informática* . Vila Real.

Lowry, M. (1993). *Methodologies for Knowledge-Based Software Engineering.* California: Artificial Intelligence Research Branch.

Natali, A. C., & Falbo, R. d. (2005). *Knowledge Management in Software Engineering Environments.* Vitoria.

Rodríguez, O. M., Vizcaíno, A., & Martínez, A. I. (2004). How to Manage Knowledge in the Software Maintenance Process. *6th International Workshop LSO 2004* (pp. 78-87). Banff, Canada: Springer Berlin Heidelberg.

Rus, I., & Lindvall, M. (June de 2002). Knowledge Management in Software Engineering. *IEEE Software*.

Rus, I., Lidvall, M., & Sinha, S. S. (Novembro de 2001). Knowledge Managment in Sofware Engineering.

Software Engineering Institute. (Novembro 2010). *CMMI® para Desenvolvimento – Versão 1.3.*

Vasconcelos, J. B., Kimble, C., Miranda, H., & Henriques, V. (2009). A Knowledge-Engine Architecture for a Competence Management Information System.

ANEXOS

A KNOWLEDGE MANAGEMENT APPROACH FOR SOFTWARE ENGINEERING PROJECTS DEVELOPMENT


82-91.pdf