PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

USO DE REPRESENTAÇÃO DE CONHECIMENTO PARA DOCUMENTAÇÃO EM METODOLOGIAS ÁGEIS

FERNANDO SELLERI SILVA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

S586u Silva, Fernando Selleri
Uso de representação de conhecimento para documentação em metodologias ágeis / Fernando Selleri Silva. – Porto Alegre, 2009.
149 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Marcelo Blois Ribeiro.

1. Informática. 2. Linguística Computacional. 3. Ontologia. 4. Documentação. I. Ribeiro, Marcelo Blois. II. Título.

CDD 006.35

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada **"Uso de Representação de Conhecimento para Documentação em Metodologias Ágeis"**, apresentada por Fernando Selleri Silva, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 24/09/09 pela Comissão Examinadora:

Prof. Dr. Marcelo Blois Ribeiro – Orientador — PPGCC/PUCRS

Profa. Dra. Ana Paula Terra Bacelo - — PPGCC/PUCRS

Prof. Dr. Marcelo Soares Pimenta – — UFRGS

Homologada em 15/12/09, conforme Ata No. 22/09. pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Para minha esposa, Lucélia Barroso, minha eterna gratidão pelo amor, compreensão e incentivo constantes em minha vida.

Para meus pais, José Carlos e Maria Aparecida, pelo exemplo de luta, perseverança e caráter, e para minhas irmãs, Giselli e Graziella.

Para todas as pessoas que deram sua colaboração ao longo desta caminhada.

AGRADECIMENTOS

Expresso meus agradecimentos ao orientador deste trabalho, Prof. Dr. Marcelo Blois Ribeiro, pelo incentivo, compreensão, críticas, contribuições e disposição em proporcionar os subsídios teóricos e práticos que foram fundamentais para a realização do trabalho e em muito contribuíram para meu crescimento pessoal e profissional.

Aos professores membros da banca examinadora por terem aceitado o convite e pelas contribuições que serão somadas a este trabalho. À Profª. Drª. Ana Paula Terra Bacelo agradeço especialmente pelas contribuições nas etapas de desenvolvimento.

Aos colegas do grupo de pesquisa ISEG: Rodrigo Perozzo Noll, por todo o conhecimento compartilhado no desenvolvimento deste trabalho e pelas ferramentas disponibilizadas, fundamentais para a execução prática da proposta; Anderson Yanzer Cabral, pelas discussões sobre representação de conhecimento e o material bibliográfico compartilhado; Ana Paula Lemke, pelas revisões, materiais e disposição em contribuir nesta jornada.

Aos professores e funcionários do Programa de Pós-Graduação em Ciência da Computação da PUCRS, sobretudo aqueles que atuaram no Mestrado Interinstitucional (MINTER). Professores: Avelino Francisco Zorzo; César Augusto Fonticielha De Rose, Duncan Dubugras Alcoba Ruiz; Fabiano Passuelo Hessel; Fernando Luís Dotti; João Batista Souza de Oliveira; Jorge Luis Nicolas Audy; Marcelo Blois Ribeiro; Milene Selbach Silveira; Ney Laert Vilar Calazans; Paulo Henrique Lemelle Fernandes; Toacy Cavalcante de Oliveira; *Vera Lúcia Strube de Lima. Funcionário: Thiago Lingener. Aos professores Avelino, Dotti e Paulo Fernandes agradeço por terem encampado a iniciativa do MINTER.*

A todos os colegas do MINTER, em especial, Luciano, Everton, Uelinton, Ivan, José Fernandes, Rodrigo, Alexandre e Diógenes, também colegas na UNEMAT, pelo companheirismo, vivência e aprendizado compartilhado. A todos vocês externo os meus agradecimentos e o desejo de tê-los como companheiros em outros desafios.

Ao Prof. Dr. Elias Januário, Coordenador da Faculdade Indígena Intercultural da UNEMAT, meus agradecimentos pelo incentivo e apoio ao longo das atividades do mestrado. À equipe da Faculdade, Sandra, Rivelino, Rosa, Itamar, Dener, Silvair, Lenin e Ivanildo, meus agradecimentos.

Ao Prof. Dr. Flavio Teles, Coordenador do MINTER na UNEMAT, pelo apoio na execução do mestrado e ao Prof. Dr. Ricardo de Oliveira Alves pelo apoio nas atividades do Departamento de Ciência da Computação da UNEMAT.

Aos estudantes do curso de Ciência da Computação da UNEMAT que participaram do experimento o meu muito obrigado pela disponibilidade e interesse em contribuir com o trabalho.

À Fundação de Amparo a Pesquisa do Estado de Mato Grosso – FAPEMAT, à Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS e à Universidade do Estado de Mato Grosso – UNEMAT pela oferta deste Mestrado Interinstitucional, que certamente trará inúmeros benefícios para o Estado de Mato Grosso.

Agradeço a Deus, fonte de sabedoria e persistência, pela oportunidade de desenvolver mais esse trabalho.

USO DE REPRESENTAÇÃO DE CONHECIMENTO PARA DOCUMENTAÇÃO EM METODOLOGIAS ÁGEIS

RESUMO

Este trabalho aponta para a compreensão dos aspectos envolvidos no emprego de representação de conhecimento em metodologias ágeis de desenvolvimento de software. As metodologias ágeis compreendem uma abordagem centrada na disponibilidade imediata do software ao cliente, sendo utilizadas principalmente em sistemas nos quais os requisitos mudam frequentemente. O objetivo deste trabalho é integrar um formalismo de representação de conhecimento a uma metodologia ágil, permitindo capturar o conhecimento e reduzir esforços no desenvolvimento de sistemas informatizados. Neste contexto, é apresentada uma proposta de representação de conhecimento, por meio de ontologias, que promove a associação dos artefatos produzidos aos conceitos do domínio do sistema e aos tipos de conhecimento expressados por esses artefatos. Um protótipo de ferramenta foi desenvolvido para auxiliar os desenvolvedores na realização das associações. A proposta é avaliada por meio de um experimento realizado para comparar o esforço na definição das associações e a precisão na recuperação dos artefatos com e sem a representação de conhecimento.

Palavras-Chave: Metodologias Ágeis. Representação de Conhecimento. Ontologia. Documentação.

USING KNOWLEDGE REPRESENTATION FOR DOCUMENTATION IN AGILE METHODOLOGIES

ABSTRACT

This work aims the understanding of issues involved in the use of knowledge representation in agile methodologies of software development. Agile methodologies include an approach focused on the immediate availability of software to the customer, being used mainly in systems where requirements change frequently. The goal is to integrate a knowledge representation formalism to an agile methodology, enabling knowledge capture and reducing efforts in information systems development. In this context, it is presented a proposal for knowledge representation, using ontologies, which promotes the association of artifacts produced to the system domain concepts and to the types of knowledge expressed by these artifacts. A prototype tool was developed to help developers in establishing the association between the concepts in the ontology and the artifacts produced. The proposal is evaluated by an experiment conducted to compare the effort in definition of associations and the precision in recovery artifacts in an approach with knowledge representation and without knowledge representation.

Keywords: Agile Methodologies. Knowledge Representation. Ontology. Documentation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

AM – Agile Modeling
ASD – Adaptive Software Development
AUP – Agile Unified Process
CMM – Capability Maturity Model
CMMI – Capability Maturity Model Integration
CRC – Class, Responsibility, Collaboration
DSDM – Dynamic Systems Development Methodology
EF – Experience Factory
ER – Entity Relationship
FAQ – Frequently Asked Questions
FDD – Feature-Driven Development
GQM – Goal Question Metric
IA – Inteligência Artificial
IBIS – Issue Based Information System
JSP – Java Server Pages
ODM – Ontology Definition Meta-Model
OMG – Object Management Group
OWG – Ontology Working Group
OWL – Web Ontology Language
PLN – Processamento da Linguagem Natural
PSP – Personal Software Planning
QOC – Questions, Options and Criteria
RDF – Resource Description Framework
RDFS – RDF Schema
RUP – Rational Unified Process
SPEM – Software Process Engineering Metamodel
TDD – Test-Driven Development
TOVE – Toronto Virtual Enterprise
TSP – Team Software Process
UML – Unified Modeling Language
W3C – Word Wide Web Consortium
XML – Extensible Markup Language
XP – Extreme Programming
XPO – Extreme Programming Ontology
XSD – XML Schema Definition

SUMÁRIO

1 INTRODUÇÃO

As metodologias ágeis (ou leves) são abordagens de desenvolvimento de software, em geral, empregadas por organizações que dão ênfase à colaboração de maneira flexível [BEC01]. Estas organizações geralmente lidam com projetos nos quais os requisitos mudam constantemente, em decorrência do mercado, da organização, do projeto e do conhecimento. *Extreme Programming* (XP) [BEC00] e *Scrum* [SCH02] são exemplos de metodologias ágeis.

As metodologias ágeis surgiram da necessidade de estabelecer processos que contemplassem o desenvolvimento de sistemas menores, de forma mais rápida e com elevada qualidade. Estas metodologias foram impulsionadas pela Aliança Ágil, em inglês *Agile Alliance* [AGI08], por meio do documento chamado Manifesto Ágil. A Aliança Ágil é constituída por especialistas da área de desenvolvimento de software. Suas principais ideias ressaltam a valorização de aspectos como indivíduos e interações, software funcionando, colaboração com o cliente e resposta a mudanças [BEC01].

Com uma abordagem que prioriza o conhecimento real sobre as funcionalidades do sistema, as metodologias ágeis estimulam a produção direta do software, a sua constante melhoria por meio de iterações e da troca de conhecimento e experiência entre os membros da equipe. Elas não pretendem representar as funcionalidades por meio de extensas documentações, antes de implementá-las, e segmentar o desenvolvimento como em uma linha de produção. A abordagem empregada pelas metodologias ágeis encara a produção de software como uma prestação de serviço e não como a montagem de um produto [TEL04].

Entretanto, as metodologias ágeis necessitam superar alguns desafios que impedem a obtenção de melhores resultados e a sua ampla utilização em projetos, incluindo aqueles de grande porte. Estes desafios envolvem mudanças na maneira como o software é comercializado, na postura do cliente ao contratar uma equipe para o desenvolvimento de um sistema, na organização desenvolvedora de software, na forma como o projeto é gerenciado e na equipe de desenvolvedores [TEL04].

Também são necessárias melhorias nas próprias metodologias, principalmente na forma como estas representam o conhecimento que foi adquirido para conduzir o desenvolvimento do sistema desde o início até a finalização. Esta representação é fundamental para socializar o conhecimento na equipe de desenvolvimento mesmo em caso de troca de seus membros. A representação do conhecimento tem como finalidade aprimorar a atividade de produção de

software, reduzindo esforços e facilitando o desenvolvimento e a manutenção de sistemas de informação.

Entre as formas de representação de conhecimento consideradas, as ontologias têm sido bastante estudadas recentemente. Uma ontologia trata da especificação formal de um conceito [GRU93], que pode ser compartilhado [BOR97]. Ontologias podem ser empregadas para representar a aquisição do conhecimento a partir de elementos abstratos, presentes no contexto do desenvolvimento de software, tais como objetos, requisitos, entidades e relacionamentos [NOL07a]. A integração de ontologias ao processo de desenvolvimento procura melhorar o desempenho de tarefas inerentes a produção de software, entre elas, a manutenção e a rastreabilidade de artefatos. Em [SEL08], considerou-se que a união entre ontologias e metodologias ágeis pode contribuir com a melhoria do desenvolvimento, sendo necessário, porém, concentrar esforços para agilizar o processo de representação do conhecimento, preferencialmente, tornando-o automatizado.

O propósito do trabalho é aprofundar o estudo de ontologias e de outros formalismos para representação de conhecimento, verificando a sua adaptabilidade na utilização junto a metodologias ágeis. Neste contexto, busca-se identificar uma técnica que combine a eficiência na representação do conhecimento com a dinâmica de desenvolvimento empreendida pelas metodologias ágeis.

1.1 Questão de pesquisa

A proposta de pesquisa aponta para a compreensão dos aspectos envolvidos no emprego de formalismos de representação de conhecimento em metodologias ágeis de desenvolvimento de software, identificando em quais sentidos tal emprego pode favorecer o desenvolvimento e a manutenção de sistemas informatizados. Neste sentido, surge a questão de pesquisa que norteia o presente trabalho: "**Como capturar o conhecimento com algum formalismo computacional de maneira a aumentar a precisão da representação de conhecimento nas metodologias ágeis de desenvolvimento de software sem aumento significativo do tempo gasto para a sua utilização?**".

1.2 Objetivos

Tendo como referência a questão de pesquisa definida, o trabalho constitui-se do objetivo geral apontado adiante. Estarão cooperando para alcançar este objetivo, os objetivos específicos listados na sequência.

1.2.1 Objetivo Geral

Identificar um formalismo de representação de conhecimento que melhor se adapte as características das metodologias ágeis e propor uma maneira de integrá-lo a uma metodologia ágil, permitindo capturar o conhecimento e reduzir esforços no desenvolvimento de projetos de sistemas informatizados.

1.2.2 Objetivos Específicos

Constituem objetivos específicos deste trabalho:

- Aprofundar o estudo sobre formalismos de representação de conhecimento.

- Identificar uma forma de representação que melhor se adapte as metodologias ágeis considerando critérios como o tempo gasto com a representação e a precisão da compreensão da documentação.

- Definir uma proposta de integração da forma de representação de conhecimento escolhida com uma metodologia ágil.

- Realizar um experimento aplicando a proposta de integração definida a um conjunto de pessoas.

- Avaliar os resultados obtidos com o experimento e descrevê-los na dissertação.

- Contribuir para aproximar metodologias ágeis de processos de desenvolvimento sistemáticos e receptíveis, sem comprometer a agilidade característica dessas metodologias.

1.3 Estrutura da Dissertação

O presente documento encontra-se estruturado em três partes: fundamentação teórica sobre metodologias ágeis e representação de conhecimento; proposta de representação de conhecimento em metodologias ágeis; avaliação da viabilidade por meio de um experimento.

O Capítulo 2 corresponde à fundamentação teórica do trabalho, abordando as metodologias ágeis de desenvolvimento de software. As metodologias *Extreme Programming* (XP) e *Scrum* são abordadas com maior detalhe neste capítulo.

O Capítulo 3, que também constitui a fundamentação teórica, apresenta os formalismos de representação de conhecimento estudados, com destaque para ontologias, que são utilizadas na proposta elaborada.

Os principais aspectos relacionados à representação de conhecimento em metodologias ágeis são tratados no Capítulo 4. Esse capítulo apresenta alguns trabalhos relacionados a representação de conhecimento no desenvolvimento de software e que contribuíram para a formulação da proposta.

No Capítulo 5 é descrita uma proposta conceitual de representação de conhecimento na metodologia ágil XP, por meio de ontologias, com destaque para as atividades desempenhadas durante o processo. Os artefatos produzidos nas metodologias ágeis e os tipos de conhecimento presentes no desenvolvimento de software são apresentados a partir de revisões sistemáticas. Uma ferramenta protótipo, desenvolvida com o objetivo de viabilizar a aplicação da proposta, é detalhada neste capítulo.

O Capítulo 6 descreve um experimento realizado, constando de um breve referencial teórico sobre Engenharia de Software Experimental, juntamente com a avaliação da aplicabilidade da proposta.

Por fim, no Capítulo 7 são apresentadas as considerações finais, com as principais contribuições do trabalho e indicações para trabalhos futuros.

2 METODOLOGIAS ÁGEIS

Este capítulo, que corresponde a base teórica da pesquisa desenvolvida, pretende fornecer uma visão geral sobre metodologias ágeis de desenvolvimento de software. Serão abordados os conceitos básicos, aspectos sobre documentação, manutenção e experiências, além de características específicas de duas metodologias ágeis de desenvolvimento: XP e *Scrum*.

2.1 Conceitos

Em virtude da ampla utilização de softwares, atendendo a diferentes áreas do conhecimento, principalmente na área comercial, passou-se a exigir deste cada vez mais qualidade, além de certa agilidade em seu desenvolvimento. No sentido de garantir a qualidade desejada, dentro de um espaço de tempo tolerável, foram estabelecidos padrões para o desenvolvimento de software. Esses padrões ou metodologias de desenvolvimento encontram-se inseridos no contexto da Engenharia de Software.

As metodologias ágeis (*agile*) ou leves (*lightweight*) foram propostas como alternativa as metodologias de desenvolvimento consideradas tradicionais, dirigidas a plano (*plan-driven*), pesadas (*heavy*) ou ainda sistemáticas (*tayloristics*). Os princípios comuns das metodologias ágeis são descritos em um documento denominado "Manifesto para o Desenvolvimento Ágil de Software" ou, simplesmente, "Manifesto Ágil" [BEC01]. Este documento foi proposto pela Aliança Ágil, entidade que congrega especialistas em desenvolvimento de software e que estimula a utilização dessas metodologias. De acordo com o manifesto, as metodologias ágeis ressaltam a valorização dos seguintes conceitos:

- Indivíduos e interações acima de processos e ferramentas;

- Software funcionando acima de documentação abrangente;

- Colaboração com o cliente acima de negociação de contratos;

- Resposta a mudanças acima de obediência a um plano.

Convém ressaltar que as metodologias ágeis não ignoram totalmente os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, porém elas partem do princípio de que o software em si, em especial o código, deve ser o foco principal do

desenvolvimento. Dessa forma, as metodologias ágeis consideram os indivíduos e interações, o software executando, a colaboração com o cliente e a resposta rápida a mudanças, como conceitos de maior valor.

2.2 Metodologias ágeis usadas como base do trabalho

Existem diversas metodologias ágeis disponíveis para serem utilizadas em projetos de desenvolvimento de software. Algumas delas são: *Extreme Programming* (XP) [BEC00], *Scrum* [SCH02], *Crystal* [COC00], *Feature-Driven Development* (FDD) [PAL02], *Lean Software Development* (LSD) [POP06], *Dynamic Systems Development Methodology* (DSDM) [STA03], *Agile Modeling* (AM) [AMB02], *Adaptive Software Development* (ASD) [HIG00], *Agile Unified Process* (AUP) [LAR02], *Test-Driven Development* (TDD) [BEC02], *Personal Software Planning* (PSP) [HUM97], *Team Software Planning* (TSP) [HUM00] e *Iconix* [ROS05]. A seguir serão abordadas as metodologias XP e *Scrum*. Essas metodologias foram usadas no trabalho por serem populares e muitas vezes utilizadas em conjunto na indústria, tendo disponível um maior número de relatos na literatura.

2.2.1 *Extreme Programming* (XP)

O *Extreme Programming* (XP), segundo [BEC00], resultou da experiência do desenvolvimento do projeto *C3 Payroll* na *Chrysler*, projeto que consistia no desenvolvimento de um sistema de folha de pagamento. A partir do sucesso de XP no desenvolvimento do referido sistema, o processo passou a ser mais disseminado e popularizado, principalmente, no ambiente da orientação a objetos. Anteriormente ao projeto *C3*, o processo já recebia importantes contribuições oriundas do desenvolvimento utilizando *Smalltalk* e de autores da área como Kent Beck, Martin Fowler e Ward Cunningham

Grande parte do sucesso de XP é devido a sua simplicidade e objetividade, possibilitando a disponibilidade do software ao cliente de forma rápida e eficiente, sem descartar, no entanto, a possibilidade de mudanças, visando cada vez mais o melhoramento do sistema.

A metodologia adota quatro valores que norteiam a equipe envolvida no desenvolvimento de software, sendo eles [BEC00]:

- Comunicação: foca a comunicação direta entre os envolvidos no projeto, reduzindo ao máximo a documentação formal;

- Simplicidade: objetiva realizar as tarefas da maneira mais simples possível e com menor custo, inclusive para mudanças futuras, evitando o desenvolvimento prévio de funcionalidades que posteriormente poderão não ser utilizadas;

- *Feedback*: permite uma maior interação e conhecimento sobre o sistema entre os envolvidos (programadores e cliente);

- Coragem: a melhoria de um projeto pode estar relacionada a atitudes corajosas como alterar o código escrito, reescrever o código, entre outras.

O núcleo principal de XP compõe-se de doze práticas, que vão de acordo aos valores e princípios do processo. Percebe-se que algumas dessas práticas também são empregadas em outros processos de software, contudo, em XP as mesmas possuem uma abordagem mais coletiva. As práticas e um breve comentário sobre cada uma delas são relatadas a seguir [BEC00]:

- O jogo do planejamento: o planejamento utiliza-se de casos de uso simplificados (estórias) levantados com a participação da equipe de desenvolvimento (técnico) e do cliente (negócio), estimulados pela figura do gerente, que utiliza como métrica a razão entre o tempo estimado para desenvolver e o tempo realmente gasto (que deve ser menor).

- *Releases* pequenos: os *releases* devem ser de curta duração, estimulando sua frequência e, em decorrência disto, a constante comunicação entre os envolvidos e o melhoramento do sistema.

- Metáfora: utilização de uma metáfora (exemplo ou modelo) que auxilie na compreensão do sistema pelos envolvidos (programadores e clientes), oferecendo uma visão geral sobre o sistema.

- Projeto simples: o projeto deve satisfazer os problemas atuais, implementando as funcionalidades já definidas e dispensando o investimento na resolução prévia de problemas que ainda estão por vir.

- Testes constantes: inclui a aplicação de testes para verificação de cada parte produzida pelos programadores (teste de unidade) e testes para verificação do sistema como um todo, junto ao cliente (teste funcional).

- Refatoramento: melhoria do projeto do código já definido, por meio da aplicação de uma série de passos, visando uma melhor adaptação do projeto a mudanças.

- Programação em pares: o código é produzido por um par de programadores em um mesmo computador, os quais se alternam nos papéis de codificador, que elabora os algoritmos e a lógica de programação, e de observador, que pensa em melhorar, simplificar e corrigir o código produzido.

- Propriedade coletiva do código: a equipe trabalha de forma unida, prevalecendo a busca pela qualidade e melhoria do código e do sistema como um todo.

- Integração contínua: integrações podem ser realizadas a qualquer tempo, desde que não existam erros nas novas funcionalidades.

- Semana de quarenta horas: existe a possibilidade de que a carga horária da equipe exceda quarenta horas semanais, observando, porém, a carga tolerável que não gere insatisfação entre seus membros.

- Cliente no local: requer a inclusão na equipe de uma pessoa da parte do cliente, que domine ou, ao menos, detenha os conceitos principais sobre o negócio ao qual o sistema se destina, para auxiliar na definição das funcionalidades e na utilização do software.

- Padrões de codificação: a utilização de padrões é imprescindível para garantir o desenvolvimento em equipe, já que todos possuem acesso e poder de alteração sobre o código.

As práticas de XP se relacionam entre si. A combinação dessas práticas possibilita a equipe poupar esforços com a concepção (projeto) do sistema, mantendo simultaneamente a flexibilidade diante de mudanças nos requisitos.

Ainda segundo [BEC00], a gerência de projetos com XP envolve basicamente dois papéis: o *treinador* (*coach*), que se preocupa principalmente com a execução técnica e a evolução do processo; e o *rastreador* (*tracker*), que coleta métricas sobre o que está sendo desenvolvido e verifica possíveis divergências com as métricas estabelecidas. A equipe ainda é composta pelos seguintes papéis: *programador*, que ocupa o papel principal, analisando, projetando, testando, codificando e integrando o sistema, a fim de produzir rapidamente código de alta qualidade; *cliente*, que escolhe o que irá agregar valor ao seu negócio, o que deve ter prioridade de desenvolvimento e participa na definição dos testes funcionais; *testador*, que ajuda o cliente na definição e escrita dos

testes funcionais; *consultor*, que possui elevado nível de conhecimento em determinada tecnologia ou assunto que não é de compreensão dos envolvidos no projeto.

2.2.1.1 O Ciclo de vida em XP

No ciclo de vida da metodologia XP, definido por [BEC00], o desenvolvimento de software encontra-se organizado nas seguintes fases: Exploração; Planejamento; Iterações do *Release*; Produção; Manutenção e Morte.

Um projeto com XP tem início na fase de *Exploração*. Nesta fase o cliente escreve as primeiras estórias sobre o sistema, enquanto a equipe prepara uma arquitetura prototipal a partir da qual o sistema será desenvolvido e busca estabelecer uma metáfora que represente o funcionamento do sistema, fornecendo uma possível solução para o problema.

Os requisitos extraídos das estórias escritas pelo cliente e a metáfora desenvolvida pela equipe serão utilizados como entrada para a fase de *Planejamento*. Nela a equipe estabelece estimativas de desenvolvimento para as estórias, com base na experiência de sistemas anteriores e na arquitetura prototipal considerada, e o cliente estabelece prioridades, considerando as características de seu negócio. São definidas as estórias que serão implementadas em cada *release*.

Tendo como ponto de partida o plano do *release*, a fase de *Iterações do Release* será iniciada. As estórias pertencentes ao *release* são implementadas por meio de iterações, com duração média de uma a duas semanas. Realiza-se o projeto, a codificação e a refatoração em cada iteração. Durante a codificação das funcionalidades, são realizados testes de unidade para cada componente produzido. Ao final de cada iteração são realizados testes de aceitação (ou testes funcionais), considerando cenários de testes produzidos pelos clientes, com o auxílio da equipe, a partir das estórias. Parte-se então para a próxima iteração, ocasião em que também serão corrigidos possíveis problemas detectados nos testes de aceitação.

Durante o desenvolvimento das iterações as estórias poderão ser modificadas, desmembradas, excluídas ou surgirem novas estórias, sendo aplicados os procedimentos de planejamento do *release* (estimativa e prioridade), acrescentando as alterações ao plano do *release*. As alterações nas estórias terão reflexos na velocidade do projeto.

Na fase de *Produção*, as funcionalidades desenvolvidas durante as iterações passam pela aprovação do cliente e o *release* é colocado em operação. Parte-se então para o desenvolvimento de um novo *release*.

A fase de *Manutenção* envolve as fases de planejamento, iterações do *release* e produção. Procura-se produzir novas funcionalidades, com o sistema existente funcionando, incorporando novas pessoas a equipe e melhorando o código. A *Morte* consiste no término do projeto seja pela satisfação total do cliente ou pela inviabilidade técnica ou econômica de se continuar o desenvolvimento. A Figura 2.1 apresenta os elementos gerais e as fases presentes no ciclo de vida do XP.

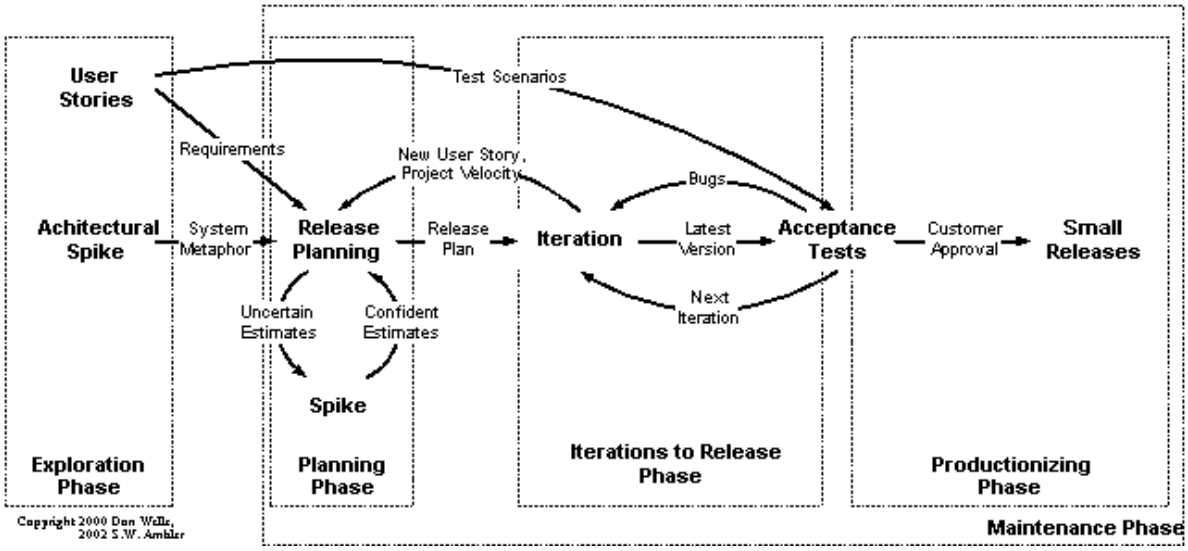


Figura 2.1: Ciclo de vida do XP
Fonte: Extraído de [AMB02]

Embora em XP não sejam definidas disciplinas de forma tão explícita quanto em outras metodologias, como no Processo Unificado, Sampaio em [SAM04] apresenta um conjunto de doze disciplinas para XP, criadas a partir de uma análise interpretativa da literatura. As disciplinas compreendem: Fazer Explorações Iniciais; Definir e Revisar Requisitos; Planejar *Release*; Planejar a Iteração; Escrever Testes Funcionais; Fazer Projeto; Escrever Testes de Unidade; Codificar; Testar; Integrar; Fazer Testes Funcionais; Colocar em Produção. As atividades inerentes a estas disciplinas se manifestam ao longo do ciclo de vida, por meio de tarefas a serem desenvolvidas e seus respectivos artefatos gerados.

Com base nestas disciplinas é apresentada uma modelagem, utilizando os conceitos e a notação do meta-modelo SPEM (*Software Process Engineering Metamodel*), versão 2.0 [OMG08]. A escolha de SPEM para realizar a modelagem se justifica pelo fato deste ser um padrão oficial da OMG, pelo apoio que este possui de grandes empresas na área de software e por se basear no padrão UML [SAM04]. No Quadro 2.1 são apresentados os elementos utilizados do SPEM e seus respectivos significados.

Quadro 2.1: Notação dos elementos utilizados do SPEM

Elemento	Notação do SPEM
Papel (*ProcessRole*)	
Atividade (*WorkDefinition*)	
Tarefa (*TaskDefinition*)	
Artefato (*WorkProduct*)	

Convém ressaltar que a modelagem não tem o objetivo de alterar a forma como se emprega XP, nem sua estrutura. Ela apenas visa trazer benefícios como [SAM04]: simplificar o entendimento dos elementos de XP; simplificar a adoção de XP por parte de uma organização; simplificar o trabalho da organização que já usa XP; e não alterar a característica ágil da metodologia. A dinâmica do processo e as disciplinas são mostradas por meio de um diagrama de atividades na Figura 2.2. Em [SAM04] pode ser conferido para cada disciplina uma modelagem estática, descrita por meio de diagramas de classe UML.

Com relação aos artefatos, observando o ciclo de vida de XP, exibido na Figura 2.1, é possível identificar alguns artefatos presentes no processo, entre eles: Estórias; Metáfora; Plano de *Release*; e Cenários de Testes. Contudo, nota-se a importância de se realizar um levantamento para identificar estes artefatos em maior detalhe, conforme será apresentado na Seção 5.4.1.

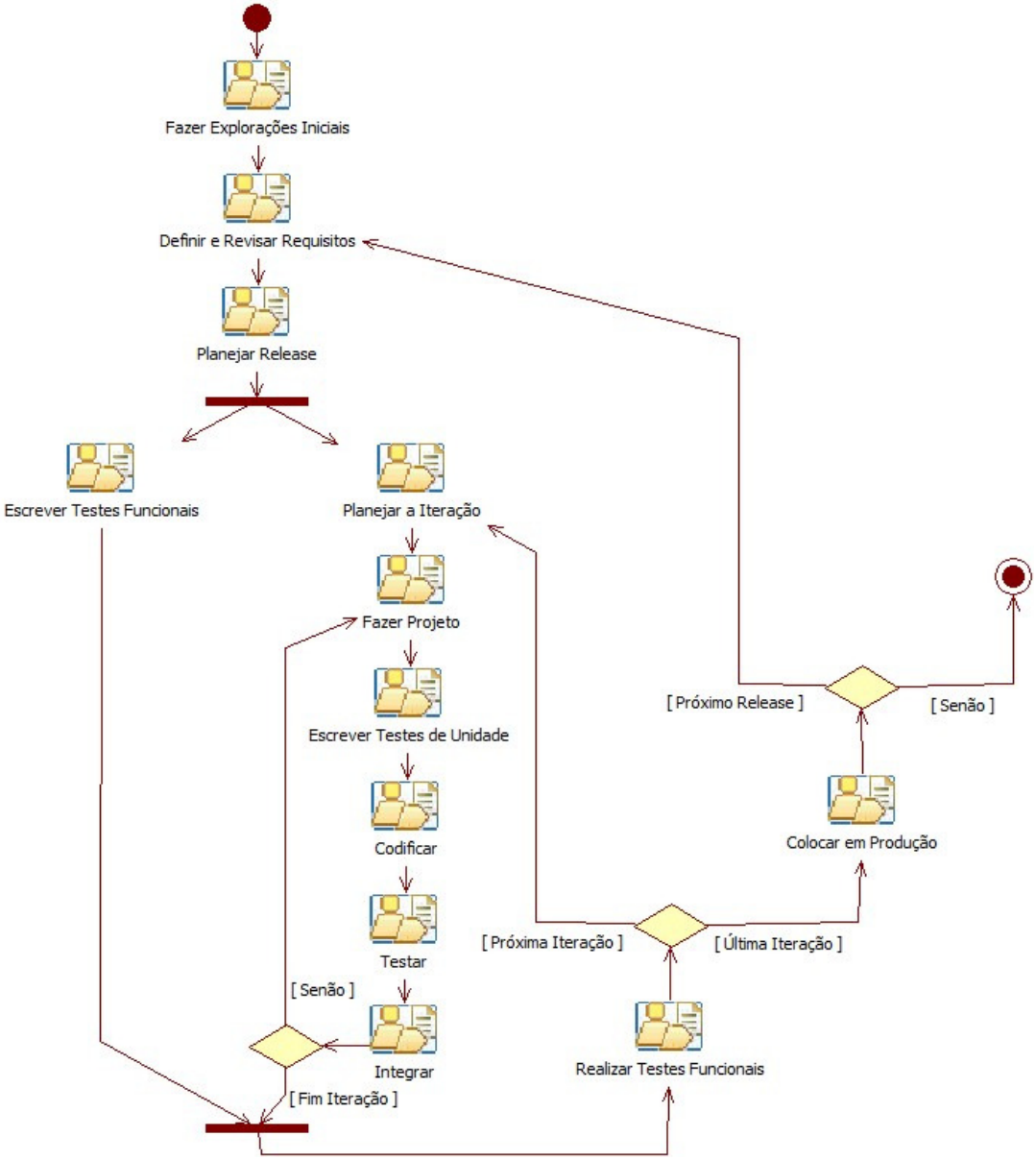


Figura 2.2: Modelagem de XP com diagrama de atividade UML e notação SPEM
Fonte: Extraído de [SAM04]

2.2.2 *Scrum*

O *Scrum* é considerado por [SCH02] como um método ágil dirigido ao gerenciamento e desenvolvimento de projetos de software orientados a objetos. A concepção do *Scrum* está relacionada a sua aplicabilidade em empresas de fabricação de automóveis, a exemplo da *Toyota*, como uma ferramenta de gerenciamento de projetos. A formalização do *Scrum* para ser utilizado no desenvolvimento de software veio posteriormente com os trabalhos de Ken Schwaber e Mike Beedle em [SCH95] e [SCH02].

Considerando as características das metodologias ágeis, o *Scrum* tem como base uma abordagem iterativa e incremental, com foco nas pessoas que compõem a equipe de desenvolvimento, indicado para projetos nos quais os requisitos aparecem e mudam rapidamente. Trata-se de uma implementação que objetiva aumentar a produtividade e reduzir o tempo de desenvolvimento de um software ou produto [ZAN05].

Entre os princípios de *Scrum* destacam-se: *equipes pequenas*, recomenda-se que esta seja composta de aproximadamente sete pessoas; *requisitos pouco estáveis ou desconhecidos*, sujeitos a mudanças repentinas; *iterações curtas*, com entrega do produto (ou versão deste) para o cliente.

O desenvolvimento é divido em intervalos de tempos, denominados *Sprints*. Estes intervalos costumam ter, no máximo, 30 dias de duração. Não são estabelecidas diretrizes específicas para a etapa de desenvolvimento de software, o processo é mais voltado para as regras e as práticas gerenciais a serem utilizadas na condução do projeto.

As seguintes práticas gerenciais são adotadas pelo *Scrum* [SCH02]:

- ***Product Backlog***: é o ponto de partida; prática responsável pelo levantamento de requisitos. São realizadas reuniões com os envolvidos no projeto (*stakeholders*) para apontamento das necessidades do negócio e dos requisitos técnicos que deverão ser desenvolvidos, resultando em uma lista das atividades a serem alcançadas para o projeto.

- ***Sprint***: as atividades definidas no *Product Backlog* são implementadas pela equipe, em períodos de tempo com duração de uma a quatro semanas (30 dias). Como no *Sprint* o objetivo é o desenvolvimento em si, este pode utilizar as etapas clássicas do desenvolvimento de software, tais como requisitos, análise, projeto e entrega.

- ***Sprint Planning Meeting***: procura definir e identificar o que será desenvolvido durante o período em que a equipe irá trabalhar no *Sprint*.

- ***Sprint Backlog***: consiste na análise do *Product Backlog*, com a elaboração de um conjunto de requisitos aceitos e a posterior análise dos resultados obtidos. Este princípio gera também informações técnicas sobre como a equipe irá implementar os requisitos.

- ***Daily Scrum***: possibilita um controle dos requisitos que estão sendo desenvolvidos por meio de reuniões diárias entre os envolvidos no projeto.

- ***Sprint Review Meeting***: reunião para revisão dos resultados obtidos no *Sprint* por parte dos envolvidos no projeto. Caso novos requisitos apareçam durante a revisão, estes são adicionados ao

Product Backlog para serem desenvolvidos no *Sprint*. O desenvolvimento de um projeto é concluído quando são atingidos todos os requisitos definidos no *Product Backlog*.

Os seguintes papéis têm destaque em uma equipe que trabalha com *Scrum*: o *Scrum Master*, que tem como objetivo contornar as dificuldades que impedem a equipe de atingir o objetivo do *Sprint* e apresentar os resultados do trabalho com os envolvidos para avaliação durante as reuniões; e o *Product Owner*, que representa o proprietário do produto e tem como função definir prioridades e verificar se os requisitos estão sendo desenvolvidos conforme solicitados pelo cliente.

2.3 Documentação

Como pontua o Manifesto Ágil [BEC01], as metodologias ágeis são favoráveis à documentação. Entretanto, ressalta-se que a documentação deve envolver apenas o que é realmente necessário para prover o desenvolvimento do software garantindo-se a sua qualidade. A produção de extensas documentações é desencorajada por essas metodologias, que alegam que documentações extensas acabam se tornando difíceis de serem utilizadas e atualizadas. Em XP, por exemplo, os documentos são utilizados principalmente para registrar o trabalho já produzido ao contrário de descreverem de forma exaustiva o que deverá ser feito [TEL04].

Por ter como filosofia a elaboração de uma documentação objetiva e que reflete o estado atual do software, o próprio código fonte acaba exercendo o papel de parte dessa documentação. Ao considerar, por exemplo, práticas de XP como propriedade coletiva do código, refatoramento, padrões de codificação e programação em pares, observa-se que a implementação delas possui estreita relação com o código fonte do software que está sendo produzido.

De acordo com as características do projeto, a documentação pode estar relacionada aos testes realizados e estes contribuem para a sua elaboração. Em [TEL04] prevê-se que em uma equipe de XP seja desempenhado o papel de *redator técnico*, responsável por manter as documentações do sistema devidamente atualizadas. No entanto, ressalta-se que este papel pode ser plenamente desempenhado pelo *testador*. A documentação deve refletir o código e os processos de negócio que serão suportados pelo sistema. Documentar não é o objetivo principal do projeto nas metodologias ágeis, deve-se gerar apenas o suficiente para a compreensão do código, possibilitando que outros desenvolvedores possam dar manutenção no sistema futuramente. A documentação, ao

contrário do que ocorre nas metodologias tradicionais, é produzida em grande parte após a implementação das funcionalidades do sistema.

2.4 Manutenção

A possibilidade de manutenção posterior ao desenvolvimento do sistema é um aspecto que merece atenção quanto à adoção de processos de desenvolvimento de software. Considerando que em grande parte das metodologias ágeis o sistema é desenvolvido de maneira incremental, a manutenção é realizada continuamente, a fim de criar novas funcionalidades.

Contudo, é necessário que a metodologia forneça mecanismos que possibilitem uma manutenção adequada do software após a finalização do projeto, seja para fazer correções de problemas não detectados durante o desenvolvimento, acrescentar novas funcionalidades ou alterar funcionalidades já existentes. Os sistemas refletem a realidade na qual se encontram inseridos e quando esta realidade se altera o sistema precisa ser alterado consequentemente [SOU05]. A possibilidade de realizar manutenções no sistema deve ser resguardada, mesmo que esta seja desenvolvida por uma equipe distinta da equipe que atuou no projeto original.

As questões referentes à manutenção em projetos de software têm um amparo considerável nas metodologias tradicionais como o Processo Unificado, em inglês *Rational Unified Process* (RUP) [KRU03], por meio da documentação, e é extremamente importante que esta também seja considerada nas metodologias ágeis, a fim de garantir a melhoria do software. É necessário que os responsáveis pela manutenção do sistema consigam compreender os processos de negócio e o código para alterá-lo de maneira rápida e segura [TEL04].

A preocupação com manutenções futuras é um dos principais incentivos para a produção de documentação nas metodologias ágeis. E a representação de conhecimento aplicada a documentação produzida pode contribuir com a manutenção. Nos casos em que a documentação não está disponível, a manutenção pode se tornar tão complexa quanto o desenvolvimento de um novo sistema. O conhecimento sobre as funcionalidades do sistema e a maneira como estas foram produzidas fica comprometido sem a existência de uma documentação de apoio ao código fonte.

Por outro lado, o excesso de documentação também pode colocar em risco o desenvolvimento e a manutenção do software, como apontado em [SOU05]. Dessa forma é essencial encontrar um equilíbrio na documentação gerada, para garantir a continuidade do

desenvolvimento do sistema. Na concepção de [AMB02] a documentação deve facilitar a comunicação durante o projeto e auxiliar no entendimento das atividades de manutenção.

Uma pesquisa inicial, relatada em [SOU05], procurou investigar junto a profissionais que atuam na manutenção de sistemas, qual a documentação essencial para o propósito de auxiliar o entendimento na fase de manutenção. Os seguintes artefatos foram apontados como mais importantes: código fonte; comentários no código fonte; modelo lógico de dados; diagrama de classes; modelo físico de dados; diagrama de caso de uso; especificação de caso de uso e plano de teste de aceitação. Uma pesquisa posterior retornou como documentos de fato mais utilizados: código fonte; protótipo não funcional; comentários no código fonte; modelo lógico de dados; especificação de caso de uso; plano de teste unitário; protótipo funcional; e diagrama de caso de uso. O código e os comentários continuaram sendo os principais documentos. Os modelos de dados e as informações sobre os requisitos perderam um pouco da importância, mas continuaram relevantes.

A pesquisa de [SOU05] considerou artefatos utilizados tanto na análise estruturada quanto na análise orientada a objetos, contudo, na descrição anterior foram considerados apenas os artefatos orientados a objetos, dada a aplicabilidade destes no contexto das metodologias ágeis. A pesquisa também retornou que nem todos os documentos produzidos foram efetivamente utilizados, dentre eles, os documentos que fornecem uma visão geral sobre o sistema como documento de visão e o modelo arquitetural. Nesse ponto, a pesquisa diverge de estudos anteriores que levam a conclusão de que a arquitetura do sistema é o documento mais importante para a manutenção de software, fundamental para o entendimento da estrutura do sistema, e reforça a necessidade de se produzir apenas a documentação essencial.

Apesar de terem reduzido significantemente a quantidade de documentos, as metodologias ágeis não removem a necessidade da documentação como um mecanismo de comunicação ao longo do tempo, que permita aos desenvolvedores comunicar informações importantes sobre o sistema para futuros mantenedores [SOU05]. A capacidade de inter-relacionar e recuperar os artefatos de documentação que se identificam também facilita atividades de manutenção. Esta associação pode ser auxiliada por formalismos de representação de conhecimento.

2.5 Desvantagens

As características apresentadas neste capítulo têm demonstrado a viabilidade da aplicação de metodologias ágeis em projetos de desenvolvimento de software. No entanto, algumas desvantagens ainda estão presentes nestas metodologias, que podem imprimir na equipe de desenvolvimento o receio de utilizá-las.

A dificuldade em aplicar metodologias ágeis em organizações que dispõem de equipes maiores faz com que seu uso não seja estimulado quando se trata de equipes com mais de 20 pessoas. Algumas práticas, a exemplo da propriedade coletiva do código, e valores, como *feedback* constante e comunicação face a face, tornam-se difíceis de serem realizados quando se lida com grandes equipes. A documentação enxuta, principalmente na fase inicial, também dificulta a socialização do projeto. A recomendação que se tem para contornar essa dificuldade consiste em dividir a equipe de desenvolvimento em equipes menores, em torno de 12 pessoas, e que cada equipe assuma o desenvolvimento de um projeto distinto.

A limitação em torno do tamanho da equipe também dificulta a aplicação de metodologias ágeis em projetos de grande porte, considerando que a maioria destes demanda grandes equipes atuando em seu desenvolvimento [SOA04]. Em virtude dessa limitação e também por se tratarem de metodologias recentes, têm-se poucos relatos de experiências de utilização de metodologias ágeis em projetos de grande porte, cujo domínio do conhecimento seja específico e de difícil compreensão.

A presença do cliente junto a equipe de desenvolvimento, ao mesmo tempo em que facilita a obtenção dos requisitos e funcionalidades do sistema, torna-se uma dificuldade nos casos em que o cliente não dispõe de tempo, não pode ou ainda não quer, seja qual for o motivo, participar deste processo. Além da mudança de comportamento por parte dos membros da equipe, a utilização de metodologias ágeis, também demanda uma mudança de comportamento por parte do cliente. A própria maneira como são efetuadas as contratações para o desenvolvimento de software precisa ser flexibilizada [TEL04]. Contratos com escopo variável são mais interessantes para metodologias ágeis, por permitirem mudanças nos projetos, do que contratos de escopo fechado.

Como o cliente tem acesso ao sistema produzido a cada iteração, pois o *release* serve para verificar se as funcionalidades já desenvolvidas estão corretas, existe o risco de que o cliente se dê por satisfeito com uma versão inacabada do sistema, que ele pensa já atender todas as suas necessidades. Dessa forma, o cliente dá por encerrado o desenvolvimento do projeto, acreditando

estar poupando tempo e recursos com o mesmo, porém, posteriormente, descobre que o sistema não implementa funcionalidades essenciais para o seu negócio. A equipe de desenvolvimento, sobretudo a função gerencial, precisa estar atenta e discutir junto ao cliente questões referentes às expectativas e funcionalidades do projeto. Quanto maior for a interação do cliente com a equipe melhor será a dimensão deste sobre o desenvolvimento do projeto.

A análise de requisitos nas metodologias ágeis costuma ser bastante informal e essa característica pode não ser desejável para todos os projetos. Ela pode ser entendida pelo cliente como falta de profissionalismo por parte da equipe tornando-o inseguro com relação a qualidade do software. Em metodologias como XP, a preocupação com a gestão de riscos do projeto não é evidente [SOA04], porém riscos podem acontecer nos projetos de desenvolvimento e devem ser considerados sem sobrecarregar a metodologia.

Embora o desenvolvimento de um sistema com metodologias ágeis estimule a constante manutenção do software, a realização de manutenção posterior ao desenvolvimento do sistema, sobretudo em sistemas que lidam com um domínio pouco conhecido, torna-se difícil. Possíveis mudanças nos membros que compõe a equipe de desenvolvimento agravam ainda mais essa dificuldade. As metodologias ágeis demandam uma maneira efetiva de representar o conhecimento vivenciado por ocasião do desenvolvimento do sistema, para que outros desenvolvedores possam absorvê-lo e contribuir com modificações posteriores a entrega do software sem, no entanto, sobrecarregar o processo a ponto de torná-lo burocrático. Nesse aspecto, a documentação possui grande importância nas metodologias ágeis, pois os documentos contribuem para o aprendizado sobre o sistema.

2.6 Experiências de utilização

A primeira experiência da qual se tem relato da utilização do XP foi o projeto *C3* desenvolvido na *Chrysler*, em 1996. Esse projeto serviu para reunir um conjunto de práticas que já vinham sendo discutidas em torno do desenvolvimento de software, dando origem a essa metodologia. De acordo com [TEL04], o projeto tinha como objetivo unificar quatro sistemas legados diferentes, que controlavam a folha de pagamento da *Chrysler* com 86.000 funcionários, começando pelo sistema responsável pelo pagamento mensal de 10.000 empregados gerenciais e técnicos. O projeto envolveu em torno de 20 profissionais e foi concluído com sucesso em um tempo de 14 meses.

As metodologias ágeis não devem ser confundidas com a ausência de processo de desenvolvimento de software, muito menos como uma retomada ao denominado processo de desenvolvimento "caótico" que prevalecia antes do estabelecimento dos processos tradicionais. Elas podem ser utilizadas inclusive por organizações que buscam certificações para o seu processo de desenvolvimento. A *Boeing* utilizou XP antes de implementar ideias do CMM (*Capability Maturity Model*) e verificou que não foram necessárias muitas alterações nos processos para que esta fosse certificada com o nível 5 do CMM [COH03].

Em [FRI07] é feita uma análise das áreas do CMMI (*Capability Maturity Model Integration*) que podem ser abrangidas por métodos ágeis, como XP e *Scrum*. O estudo considera que a melhoria do processo com CMMI também pode ser realizada quando se usa métodos ágeis. Os métodos ágeis podem ser utilizados sem quaisquer adaptações importantes até o nível 2 e até o nível 3 com algumas pequenas mudanças propostas no estudo. Contudo, algumas áreas do CMMI, principalmente aquelas dos níveis de maturidade 4 e 5, estão em conflito com os princípios ágeis, o que demanda maior adaptação dos métodos.

2.7 Considerações

Neste capítulo foram apresentados os principais aspectos relacionados às metodologias ágeis, tais como os valores iniciais que motivaram a origem, as metodologias mais conhecidas, o objetivo da documentação, a maneira como lidam com a manutenção, projetos que utilizaram metodologias ágeis no desenvolvimento e desvantagens. Também foram abordadas em maior detalhe as características das metodologias *Extreme Programming* (XP) e *Scrum*, que estão em evidência no atual cenário de desenvolvimento ágil de software. Os aspectos discutidos procuraram dar fundamentação teórica ao trabalho.

Por meio dos argumentos e experiências apresentados, considera-se que as metodologias ágeis podem oferecer recursos capazes de conduzir o desenvolvimento de software com qualidade. Contudo, desafios como adaptação da equipe de desenvolvimento e da maneira como os projetos são gerenciados, mudanças nas relações entre equipe e cliente, melhor organização e disponibilidade do conhecimento, abordagens mais efetivas para garantir a manutenção, entre outros, necessitam ser superados para a obtenção de melhores resultados e utilização dessas metodologias em projetos que lidam com domínios mais complexos.

Melhorias na forma como as metodologias ágeis organizam o conhecimento empregado durante o desenvolvimento, sobretudo no que se refere aos artefatos que compõem a documentação produzida, são requeridas. Para tal, sugere-se identificar os tipos de conhecimento a que se referem os artefatos gerados e ampliar a capacidade de inter-relacionar os artefatos que se referem a uma mesma funcionalidade ou a um mesmo conceito do domínio. Abordagens de representação de conhecimento podem contribuir para representar e socializar esse conhecimento entre a equipe, principalmente em manutenções.

Neste sentido, o próximo capítulo abordará conceitos relacionados à representação de conhecimento e aos formalismos que a viabilizam. Estes formalismos de representação de conhecimento serão estudados, considerando as características das metodologias ágeis, com o objetivo de melhorar as atividades na produção de software, entre elas a documentação e a manutenção. A proposta é conhecer as características da representação de conhecimento para verificar a sua adaptabilidade na utilização junto a metodologias ágeis.

3 REPRESENTAÇÃO DE CONHECIMENTO

Este capítulo também integra a base teórica do trabalho desenvolvido e apresenta as características relacionadas à representação de conhecimento e os principais formalismos de representação presentes na área computacional. Dentre os formalismos apresentados, será dada maior ênfase às ontologias e os aspectos referentes à utilização destas no desenvolvimento de software.

3.1 Conceitos

A Engenharia de Software é um processo intensivo em conhecimento, englobando a obtenção dos requisitos, o projeto, o desenvolvimento, o teste, a implantação, a manutenção, a coordenação de projeto e o gerenciamento de atividade [CHA03a]. Ao se desenvolver um sistema informatizado, são empregados processos, técnicas, ferramentas e conhecimento sobre um determinado domínio ou realidade, que juntos colaboram com a equipe de desenvolvimento que atua em sua produção.

Para [RUS02] o conhecimento gerado na Engenharia de Software é variado e suas proporções imensas, estão crescendo constantemente. Logo, este conhecimento demanda abordagens que possam representá-lo, organizá-lo e colocá-lo a disposição da equipe de desenvolvimento. Os membros da equipe de projeto adquirem experiência individual valiosa com cada projeto. Assim, a equipe e os membros poderiam ganhar muito mais se pudessem compartilhar esse conhecimento. Novos desenvolvedores em uma equipe, por exemplo, precisam de conhecimento sobre a base de software existente e convenções locais de programação.

Existem várias definições para o conceito de conhecimento na literatura e estas costumam variar de acordo com a área que o aborda (computação, administração, filosofia, psicologia, sociologia, educação, entre outras). Na área computacional, a maioria das definições sobre conhecimento passa pela definição de dados e informação.

Na definição de [RUS02], os dados consistem em fatos discretos e objetivos sobre eventos, mas não sobre a importância ou relevância própria desses eventos; é uma matéria-prima para criar informação. A informação é o dado que é organizado para ser feito aproveitável para usuários finais que desenvolvem tarefas e tomam decisões. O conhecimento é mais amplo do que dados e

informação, requer entendimento da informação, das relações entre itens de informação, sua classificação e metadado. Experiência é o conhecimento aplicado. Por sua vez, o conhecimento é dinâmico e um conceito geralmente encontra-se associado a outro conceito. Dessa forma, o conhecimento está em constante atualização, associando novos conceitos que geram novos conhecimentos.

O conhecimento costuma ser dividido em duas categorias, com relação à representação, como propõe Nonaka e Takeuchi em [NON97] e Rus e Lindvall em [RUS02]:

- **Conhecimento explícito**: é o conhecimento que está representado em documentos, manuais, artigos, leis e de maneira mais ampla em livros, jornais e revistas. É de fácil articulação, manipulação e transmissão.

- **Conhecimento tácito**: é o conhecimento que está na mente dos indivíduos, adquirido por meio da experiência de cada um e vivência de situações ocorridas durante a vida. É de difícil captura e transmissão, porém é extremamente valioso.

O conhecimento tácito é particularmente importante em equipes que atuam com metodologias ágeis de desenvolvimento de software [CHA03a]. Nestas metodologias, grande parte do conhecimento adquirido e empregado está na mente dos desenvolvedores (conhecimento tácito), sendo estabelecido na interação entre os membros da equipe e o cliente.

A representação de conhecimento busca transformar o conhecimento tácito em conhecimento explícito. Para [ALV03] a representação do conhecimento é simbólica. Representar significa o "ato de colocar algo no lugar de". A representação feita pelos autores no momento da expressão dos resultados de seus pensamentos utiliza-se das linguagens disponíveis no contexto da produção e comunicação de conhecimentos. Nas metodologias tradicionais além da linguagem natural é marcante a utilização da *Unified Modeling Language* (UML) [OMG09a] e seus diagramas para representação do conhecimento referente ao desenvolvimento de um sistema.

De acordo com [BAR97], a representação do conhecimento é um "ramo da Organização do Conhecimento que compreende o conjunto dos processos de simbolização notacional ou conceitual do saber humano no âmbito de qualquer disciplina". Na representação de conhecimento estão envolvidas a classificação, a indexação e o conjunto de aspectos informáticos e linguísticos, que se encontram relacionados com a tradução do conhecimento.

Sowa em [SOW00] considera a representação de conhecimento como um ramo da Inteligência Artificial. Contudo, pondera que a representação de conhecimento é um tema

multidisciplinar que aplica teorias e técnicas oriundas: da *lógica*, que provê a estrutura formal e as regras de inferência; das *ontologias*, que definem os tipos de coisas que existem no domínio da aplicação; e da *computação*, que suporta a aplicação e distingue a representação de conhecimento da pura filosofia.

3.2 Uso de ferramentas para representação

Conforme [RUS02] as organizações que desejam melhorar a capacidade de Engenharia de Software da equipe podem conduzir tarefas que garantam que o conhecimento obtido durante o projeto não se perca. Essas tarefas podem ser realizadas durante o projeto e brevemente após a sua conclusão. Elas podem abordar tanto a aquisição do conhecimento que não foi documentado como parte das atividades centrais quanto à análise de documento para criar novo conhecimento. Incluso nessa tarefa estão: formas de lições aprendidas e análises posteriores que identificam o que foi certo ou errado para o software e o processo; análise dos dados do projeto (custos e esforço estimado e atual, cronograma estimado e atual, histórias que refletem os eventos do projeto). Todo o conhecimento registrado por meio dessas tarefas pode ser armazenado em repositórios e bases de experiência.

Assim, é sugerido o uso de algumas ferramentas para suporte a atividade de gerenciamento, entre elas: ferramentas para gerenciamento de documentos; ferramentas para gerenciamento de competências e habilidades; ferramentas de controle de versões e o uso de mecanismos que controlem a rastreabilidade. Os requisitos do sistema conduzem ao desenvolvimento do software, porém a conexão entre o sistema final e os requisitos é nebulosa. A rastreabilidade é uma abordagem que explicitamente conecta os requisitos e o sistema de software final.

Na visão de [RUS02] os desenvolvedores devem ser estimulados a compartilhar o conhecimento e a utilizar o conhecimento compartilhado, colocando essa característica como um dos fatores de sucesso da gestão do conhecimento. O uso de técnicas presentes em metodologias ágeis, como programação em pares, também são favoráveis a criação de uma cultura de compartilhamento, assim como o estímulo para participação em comunidades ou fóruns de discussão. A representação do conhecimento também se utiliza de instrumentos como registros em atas e *wikis*, desenhos ou esboços, quadros de anotações, *flip charts*, fotografias e gravações de áudio e vídeo.

3.3 Formalismos de representação

Durante a pesquisa foram identificados alguns dos principais formalismos de representação de conhecimento, na expectativa de que a análise destes contribuísse com o processo de captura do conhecimento integrado a metodologias ágeis.

Hinz em [HIN06] destaca os seguintes formalismos para a representação do conhecimento como sendo os mais utilizados: Lógica, Redes Semânticas, *Frames*, Lógica Descritiva, Taxonomias, *Tesauros* e Ontologias. Além dos formalismos que estão diretamente relacionados com a área da computação, observou-se formas de representação de conhecimento pertencentes a outras áreas como a psicologia, a filosofia e a administração, tais como: Mapas Conceituais, Mapas Mentais, Mapas Cognitivos e *Design Rationales*. Considerando o interesse pela utilização de um formalismo computável e a capacidade de comunicação entre humanos e padronização dos formalismos existentes, a seguir será apresentada uma breve descrição sobre ontologias.

3.3.1 Ontologias

A palavra ontologia provém do grego *óntos*, "ser", + *logos*, "estudo", que corresponde ao estudo ou conhecimento do ser, da existência. Teve sua origem na filosofia de Aristóteles, como um ramo da metafísica, que objetiva o estudo das características do ser ou da existência propriamente [GUA98].

A introdução da palavra ontologia na área da Ciência da Computação esteve voltada primeiramente ao ambiente da Inteligência Artificial, no qual uma ontologia encontra-se associada a ideia de representação do conhecimento, assim como outros formalismos existentes como a própria lógica, as redes semânticas, as taxonomias, entre outros. Esse mesmo conceito possibilitou a introdução dessa palavra no contexto da Engenharia de Software, pela sua possibilidade de representar conhecimento, conceitos e domínios, no decorrer da produção de um software.

De acordo com [GRU93], ontologia é "uma especificação formal de uma conceitualização". Uma ampliação deste conceito coloca que essa conceitualização também pode ser compartilhada [BOR97].

No que tange o desenvolvimento de software, o emprego de ontologias passou a ser considerado a partir dos trabalhos de Gruber, como em [GRU93], que coloca que aspectos

semânticos podem facilitar a atividade de produção de software. Ontologias são empregadas com o propósito de realizar deduções ou inferências levando em consideração os artefatos e as propriedades de um determinado cenário, também considerado como domínio.

Para [FAL04] ontologias aplicadas ao desenvolvimento de software tendem a oferecer uma maneira de se trabalhar com a representação de recursos de informação. Desse modo, "o modelo de domínio descrito por uma ontologia pode ser usado como uma estrutura unificadora para dar semântica e uma representação comum à informação" [FAL04].

Em [LIN05] acrescenta-se que as ontologias são estruturas de representação de conhecimento úteis para a especificação de abstrações de software de alto nível. Elas fornecem uma terminologia única que pode ser compartilhada por todos os envolvidos em um processo de desenvolvimento. As ontologias também facilitam a reutilização do conhecimento, considerando que são extensíveis e adaptáveis.

3.3.1.1 Vantagens da utilização de ontologias

O emprego de ontologias no desenvolvimento de software tem apresentado, na visão de [BUL06] e de [HIN06], as seguintes vantagens:

- Prevenção de diferentes interpretações sobre a semântica dos termos: como as ontologias são escritas em uma linguagem formal, se o sistema interpreta que um indivíduo pertence a uma classe em um determinado domínio, ele não pode pertencer a outra classe, se estas são disjuntas, por exemplo.

- Modelagem do conhecimento independente de implementação: a abordagem declarativa permite que um domínio seja descrito sem compromisso com a implementação de um sistema de software em específico.

- Interoperabilidade entre sistemas de software: integração transparente de serviços entre sistemas, como armazenamento, busca e correlação de informações.

- Compartilhamento público do conhecimento: por requerer concordância quanto a conceitos, ontologias podem ser compartilhadas como arquivos, inclusive na web.

- Capacidade de reusar e estender definições de termos: é possível importar definições de outras ontologias de mesmo domínio evitando o re-trabalho ou ainda estender o vocabulário de uma ontologia com termos não tratados por esta.

- Capacidade de dedução: as características de formalidade e semântica explícita permitem que um software deduza novos fatos a partir de fatos declarados em uma ontologia.

3.3.1.2 Desvantagens da utilização de ontologias

Apesar das vantagens citadas anteriormente, o uso de ontologias associado ao desenvolvimento de software possui algumas desvantagens. O processo de construção de uma ontologia é bastante formal e demanda tempo significativo. Mesmo quando se utiliza uma ferramenta para edição de ontologias, a exemplo do *Protégé* [GEN03], o desenvolvimento requer disponibilidade de tempo e de aprendizado sobre a ferramenta para que seja desenvolvida uma ontologia que represente conhecimento verdadeiramente útil sobre o domínio.

Outras desvantagens relacionadas ao uso de ontologias compreendem [BIE06]:

- Listas de ontologias não uniformes, vários vocabulários: um mesmo conceito pode ser descrito de diferentes maneiras, o que dificulta o compartilhamento da ontologia.

- Dificuldade de modelar a adaptação do conhecimento: embora ontologias permitam a representação do conhecimento, modelar a adaptação deste consiste em um desafio.

- Complexidade de transformação entre modelos existentes e ontologias: o mapeamento de um modelo em UML, por exemplo, para uma ontologia equivalente pode se tornar uma tarefa complexa, dependendo do domínio da aplicação. Existem características que estão presentes somente em ontologias e outras que estão somente no modelo considerado, assim, é possível perder informação com o mapeamento.

- Imaturidade das ferramentas para criação e manipulação de ontologias: as ferramentas estão em constante evolução, porém ainda demandam melhorias para tornar mais eficiente a criação e a manipulação de ontologias.

3.4 Considerações

Este capítulo procurou apresentar os principais conceitos relacionados a representação de conhecimento, destacando a importância de sua aplicação no contexto da Engenharia de Software. A representação de conhecimento tem sido viabilizada por meio de formalismos, geralmente tendo seu emprego auxiliado por ferramentas computacionais. Neste sentido, foram pesquisados os principais formalismos de representação de conhecimento apresentados na literatura.

Nota-se que os formalismos de representação de conhecimento provenientes da Inteligência Artificial (IA) em sua maioria conduzem a uma representação computacional. Dessa forma, o trabalho procurou considerar formalismos passíveis de serem representados em um modelo computável. Esses formalismos possibilitam que um artefato de documentação, por exemplo, seja representado em um modelo capaz de ampliar a compreensão em situações que envolvam manutenções e a recuperação deste artefato.

Dentre os formalismos pesquisados foi dada ênfase às ontologias, apresentando os principais conceitos relacionados a este formalismo no contexto da Ciência da Computação. Ontologias são consideradas úteis ao propósito deste trabalho por serem compreensíveis tanto por humanos quanto por computadores. Para tal compreensão, uma ontologia pode ser representada por uma linguagem formal, que contenha os principais requisitos necessários para descrevê-la. A *Web Ontology Language* (OWL) [W3C04] representa uma das principais linguagens para a descrição de ontologias, tendo seu desenvolvimento ancorado pelo W3C.

A construção de ontologias é amparada por metodologias que estabelecem as diretrizes a serem seguidas neste processo. Entre elas cita-se: *Toronto Virtual Enterprise* (TOVE) [GRU95], que descreve uma ontologia a partir de cenários de motivação em linguagem natural; *Enterprise Ontology* [USC95], que também descreve uma ontologia a partir da especificação da informação a ser representada; e *Methontology* [FER97], que trabalha com o conceito de evolução de protótipos para o desenvolvimento de ontologias.

O trabalho com ontologias pode ser facilitado com o uso de ferramentas que permitem a representação do conhecimento em alto nível e, posteriormente, a geração da ontologia em uma linguagem formal, preferencialmente a OWL. O uso de ferramentas evita a descrição manual de toda a sintaxe da linguagem utilizada, porém requer conhecimento sobre a ferramenta utilizada para gerar as representações desejadas. Dentre as ferramentas existentes, o *Protégé* tem sido bastante popular, sobretudo em relatos de trabalhos acadêmicos que lidam com ontologias.

O uso de ontologias junto ao desenvolvimento de software, em estudos realizados, indica que estas podem facilitar atividades relacionadas à compreensão do domínio da aplicação e das características dos artefatos que a compõe. Entretanto, adequar a construção de uma ontologia sobre um domínio à dinâmica de desenvolvimento empreendida pelas metodologias ágeis representa um desafio a ser contornado. Neste sentido, no próximo capítulo, será abordada em maior detalhe a questão da representação de conhecimento em metodologias ágeis.

4 REPRESENTAÇÃO DE CONHECIMENTO EM METODOLOGIAS ÁGEIS

Neste capítulo, pretende-se introduzir a problemática da representação de conhecimento aplicada ao desenvolvimento de software, tendo como foco o desenvolvimento utilizando metodologias ágeis. Também são apresentados alguns trabalhos que se encontram relacionados com o tema de pesquisa e que contribuíram para a elaboração da proposta de representação apresentada posteriormente.

4.1 Conceitos

A importância em representar conhecimento nas metodologias ágeis decorre do fato de que pouco do conhecimento tácito torna-se explícito. E pouco do conhecimento explícito pode ser documentado em detalhes porque os desenvolvedores na maioria dos casos resistem em fazê-lo devido às limitações de tempo e ao esforço que é necessário para se documentar o que eles sabem [CHA03a]. O autor ainda destaca que as metodologias ágeis não possuem suporte explícito para o aprendizado inter-equipe dentro de uma organização. Esse aprendizado demanda formas eficientes de representação, que permitam interpretar o conhecimento e dar continuidade ao projeto mesmo com uma a equipe distinta da equipe que o iniciou.

Ao discorrer sobre as características da documentação VanFosson em [VAN06] propõe que sejam observados dois critérios, os quais também podem ser aplicados no contexto da representação de conhecimento em metodologias ágeis: eficácia e eficiência. A representação de conhecimento precisa ser eficaz, o que corresponde a refletir o que o projetista deseja comunicar e, posteriormente, permitir que outros indivíduos, que dependem da documentação, compreendam o que está sendo comunicado. Uma vez representado o conhecimento precisa estar acessível e utilizável ao longo do desenvolvimento para ser eficaz. Outro aspecto de eficácia a ser considerado é que a forma de representação deve estimular a vontade e o interesse do desenvolvedor em utilizá-la para representar a compreensão sobre as decisões do projeto. Estes sentidos também devem ser despertados naqueles que usarão a representação para compreender o problema.

O emprego de menos tempo e menos recursos adicionais para capturar e compreender o conhecimento refere-se à eficiência da forma de representação. Uma preocupação que norteia este estudo é não sobrecarregar o processo, uma vez que o foco das metodologias ágeis é a produção de software de maneira rápida e sem excesso de burocracia em torno da documentação. Outra

característica é que o conhecimento também precisa ser passível de atualizações, inclusive por parte de outros desenvolvedores que não o seu autor inicial, para que se mantenha útil. Contudo, mesmo após as atualizações, o conhecimento precisa manter suas características anteriores, permitindo que iniciantes compreendam rapidamente o processo que culminou para que o conhecimento chegasse ao seu estágio corrente.

Considerando os aspectos sugeridos por [VAN06], a seguir serão apresentados alguns trabalhos relacionados a representação de conhecimento na Engenharia de Software. Estes trabalhos contribuíram com a proposta de representação de conhecimento em metodologias ágeis, elaborada neste documento.

4.2 Trabalhos relacionados

Na literatura são encontrados vários trabalhos relacionados à representação de conhecimento em Engenharia de Software. Dentre estes, o trabalho de [BAS01] contribui com a proposta desse trabalho por apresentar uma abordagem prática para o gerenciamento da experiência organizacional.

No trabalho de [CER03] a representação de conhecimento é apoiada pela definição de uma ontologia para a metodologia XP. Esta abordagem é importante por apresentar um modelo formal, especificando os principais conceitos usados em XP e suas propriedades, e por sugerir a indexação de artefatos como forma de estabelecer uma base de conhecimento sobre o desenvolvimento ágil.

A proposta de [NOL07a], apesar de estar relacionada ao Processo Unificado, apresenta uma abordagem interessante por não provocar alterações na metodologia. A representação de conhecimento é colocada como uma atividade adicional, prevendo a inclusão de um novo papel para desenvolvê-la. Esta característica, embora represente um esforço adicional para o processo de desenvolvimento, pode ser viável para a representação de conhecimento em uma metodologia ágil, pela pouca interferência que sugere.

4.2.1 Sistema de gerenciamento de experiência

Basili *et al* em [BAS01] apresenta uma abordagem leve para a gestão do conhecimento organizacional, denominada *Knowledge Dust to Pearls*. A abordagem é baseada na proposta de

Fábrica de Experiência, em inglês *Experience Factory* (EF), que é direcionada a equipes que precisam aprender com experiências passadas.

Como resultado a abordagem apresenta uma ferramenta, denominada *Knowledge Dust Collector*. A ferramenta suporta o compartilhamento do conhecimento via *peer-to-peer* por meio da captura do conhecimento que os desenvolvedores trocam e usam diariamente.

Os diálogos relacionados a problemas técnicos, por exemplo, considerados poeira de conhecimento (*knowledge dust*), são capturados, analisados e transformados em perguntas frequentes, em inglês *Frequently Asked Questions* (FAQs), consideradas pérolas de conhecimento (*knowledge pearls*). Estas perguntas frequentes são então analisadas e transformadas em melhores práticas, as chamadas pérolas de conhecimento estendidas (*extended knowledge pearls*).

A abordagem definida por [BAS01] conduz a organização a gerenciar o conhecimento gradualmente e melhorá-lo passo a passo. Possibilita a equipe investir menos no início e já começar a colher os resultados, habilitando-a a avaliar a abordagem e melhorá-la com base nos resultados.

Entre as ferramentas utilizadas pela equipe estão:

- Email: desenvolvedores se comunicam por email para formular e responder questões. Estas questões são armazenadas e disponibilizadas para a equipe em um sistema de FAQ. A equipe de EF analisa as questões e pode gerar uma descrição de processo a ser armazenada na base de experiência.

- **Banco de Dados de Lições Aprendidas**: os desenvolvedores são encorajados a registrar em um banco de dados de lições aprendidas os incidentes que ocorrem, para futuras consultas a fim de superar situações semelhantes. Quando a equipe de EF percebe que existe uma considerável quantidade de lições aprendidas sobre um determinado tópico, esta elabora um documento de boas práticas que é disponibilizado na base.

- ***Bug-tracking***: usado para documentação e rastreamento de defeitos durante o desenvolvimento de software. A equipe de EF analisa os defeitos reportados e estabelece práticas de desenvolvimento.

- ***Chat***: usado para condução de discussões direcionadas e *eWorkshops*. Os *logs* e resumos são analisados pela equipe de EF para gerar documentos de boas práticas e lições aprendidas que são disponibilizados na base de experiência e podem ser usados pela organização.

O método de análise qualitativa escolhido para a equipe de EF modelar a técnica de *dust-to-pearls* é o método de comparação constante [MIL94], concebido para revelar hipóteses (com base em tendências, padrões, entre outros) a partir de dados qualitativos. O emprego do método é auxiliado pela ferramenta *Nvivo*.

A base que armazena os pacotes de experiência (tais como as perguntas frequentes do sistema de FAQ, os documentos de boas práticas, entre outros) é um banco de dados informatizado. Uma ferramenta, denominada *Visual Query Interface*, foi desenvolvida para analisar e desenhar conclusões sobre o conteúdo e o crescimento desta base de experiência, o *feedback* dos usuários e a utilização de ferramentas. A análise qualitativa é utilizada no processo de sintetizar os pacotes de experiência em *checklists*, melhores práticas, diretrizes, processos e outras formas de conhecimento derivado.

O coletor de conhecimento em forma de poeira no caso do FAQ é implementado como uma aplicação web composta por um *front-end* (para usuários finais) e um *back-end* (gerenciador de conteúdo). A ferramenta é integrada com uma interface de email para responder as perguntas que são formuladas.

4.2.2 Ontologia para XP

Ceravolo *et al* em [CER03] apresenta uma ontologia de processo de desenvolvimento de software que especifica os principais conceitos usados em *Extreme Programming* (XP) e suas propriedades. O trabalho discute a utilização dessa ontologia em um processo semântico de aquisição e mineração de dados sobre atividades da equipe e repositórios de conteúdos, visando extrair novos conceitos e identificar fatores críticos ao desenvolvimento ágil.

Na definição da ontologia para XP, uma abordagem *top-down* foi adotada, definindo-se primeiramente os conceitos do domínio que mais se destacavam por meio de um conjunto de classes e então estas foram especializadas até a obtenção de um modelo de domínio completo. Foram destacadas três classes principais, a partir das quais os conceitos da metodologia XP passaram a ser agrupados, sendo elas:

- Papel Organizacional (*Organization Role*): que possui as subclasses Agente (*Agent*) e Papel (*Role*), referentes aos membros da equipe e aos papéis que desempenham.

- Fase (*Phase*): que possui as subclasses Fase do Projeto (*Project Phase*), Atividade do Processo (*Process Activity*) e Evento (*Event*), que representam conceitos gerais comuns ao processo, a partir dos quais foram definidas subclasses específicas de XP.

- Produto (*Product*): que possui as subclasses Entregável (*Deliverable*) e Produto Interno (*Internal Product*).

Os detalhes de cada subclasse foram suprimidos evitando estender a descrição, porém no Anexo A elas são definidas em detalhe. Os artefatos de XP são indexados às subclasses de acordo com o conhecimento a que se referem. A cada artefato indexado, um conceito, descrito por meio de código RDF (*Resource Description Framework*) e RDFS (*RDF Schema*) [W3C04a], é disponibilizado para consulta junto a este, incluindo alguns de seus atributos. Dessa forma, pode-se gerar arquivos RDF e RDFS descrevendo uma ontologia sobre uma Estória, por exemplo. Utilizando um sistema de representação de conhecimento é possível recuperar características como autor do artefato, data de criação, data de término, bem como as alterações agregadas a este artefato ao longo de sua existência.

4.2.3 Integração de ontologias no Processo Unificado

Noll em [NOL07a] apresenta uma proposta de integração de ontologias junto ao Processo Unificado, tendo como objetivo facilitar a rastreabilidade entre os artefatos gerados em projetos de software, sobretudo durante as disciplinas de Requisitos e de Análise e Projeto, relacionando-os com os conceitos presentes no domínio.

A proposta estabelece a criação de uma nova disciplina denominada “Modelagem do Conhecimento”. É essa disciplina que se ocupará de capturar o domínio durante o processo de desenvolvimento do sistema, representá-lo por meio de uma ontologia e, posteriormente, reutilizar o conhecimento quando da manutenção do sistema ou desenvolvimento de um sistema similar. A disciplina é estruturada em três atividades, sendo elas:

- Projeto: produz uma versão inicial da ontologia a partir do Modelo de Domínio, produzido durante a disciplina de Modelagem de Negócio. A ontologia é gerada a partir do mapeamento das classes, atributos, associações e generalizações em UML para estruturas equivalentes em OWL.

- Manutenção: busca refinar a ontologia, adicionando novas regras aos conceitos definidos, e gerar os elos de rastreabilidade entre os artefatos produzidos sobre o sistema e os conceitos representados pela ontologia.

- Avaliação: corresponde a avaliar a completude e corretude do conhecimento sobre o domínio do problema, expresso pela ontologia.

As atividades relacionadas à Modelagem do Conhecimento são executadas pelo Engenheiro do Conhecimento, que representa um novo papel a ser desempenhado no Processo Unificado.

Para prover a rastreabilidade ontológica entre os artefatos, bem como o mapeamento destes com os conceitos do domínio, é proposto um conceito denominado *ONTraceBasis*, que se encontra estruturado sobre outros dois conceitos, sendo eles: *artifacts*, que representa uma ocorrência de um elemento UML (como um caso de uso ou uma classe do diagrama de classe) e a relaciona com os conceitos do domínio e com o tipo do artefato; e *model*, que organiza hierarquicamente os tipos de artefatos previstos pela UML.

Uma ferramenta denominada *ArgoUML+ONTrace* foi desenvolvida para suportar a modelagem do conhecimento e a definição dos elos de rastreabilidade ontológica. Neste protótipo, agregou-se à ferramenta de modelagem *ArgoUML* uma aplicação capaz de recuperar os elos de rastreabilidade ontológica por meio de consultas (*ONTrace*).

Com o objetivo de avaliar a proposta, sob a perspectiva da viabilidade da integração de ontologia ao Processo Unificado e da viabilidade do processo de utilização da rastreabilidade ontológica sob a rastreabilidade por requisitos, foi conduzido um experimento, seguido de uma pesquisa de opinião. Considerando as características do estudo, a avaliação retornou resultados positivos para a proposta de rastreabilidade ontológica.

4.3 Considerações

A representação de conhecimento em metodologias ágeis tem como objetivo minimizar a lacuna de conhecimento explícito existente nestas metodologias ágeis, as quais são naturalmente orientadas ao conhecimento tácito. Entre os conceitos apresentados ressalta-se a eficácia em se representar o conhecimento desejado e a eficiência em realizar a representação com o mínimo de esforço e impacto na metodologia.

Foram apresentados alguns trabalhos relacionados a representação de conhecimento no processo de desenvolvimento de software. Dentre estes, o trabalho de Basili *et al* em [BAS01], que remete aos conceitos da Fábrica de Experiência, representa uma das mais relevantes contribuições no âmbito da gestão de conhecimento no desenvolvimento de software. Sua contribuição no âmbito deste trabalho está relacionada ao fato de disponibilizar a representação de conhecimento como parte do desenvolvimento, produzindo resultados ao longo do projeto. O trabalho de Ceravolo *et al* em [CER03] traz uma contribuição essencialmente voltada para representação de conhecimento na metodologia ágil XP, por meio da criação de uma ontologia para a metodologia. Dessa forma, seus conceitos apontam a viabilidade da utilização de ontologias junto a metodologias ágeis e a importância de se indexar os artefatos produzidos, proposta considerada neste trabalho. Por fim, o trabalho de Noll em [NOL07a] apresenta uma proposta de integração de ontologias ao Processo Unificado, que por causar o mínimo de impacto no desenvolvimento, se torna uma referência importante para ser adaptada ao contexto de metodologias ágeis, a exemplo das metodologias abordadas neste trabalho.

Com base nos conceitos apresentados por esses trabalhos, juntamente com os conceitos discutidos sobre as metodologias ágeis e a representação de conhecimento, no próximo capítulo será dada forma a uma proposta de representação de conhecimento para o desenvolvimento ágil de software.

5 PROPOSTA DE REPRESENTAÇÃO

Neste capítulo, apresenta-se uma proposta de representação de conhecimento em metodologias ágeis. Para a referida proposta, definiu-se *Extreme Programming* (XP) como metodologia ágil considerada, não descartando a utilização em conjunto com *Scrum*, e ontologias como formalismo para representação do conhecimento. As justificativas que favoreceram a escolha por XP e por ontologias, bem como as características da proposta de representação serão apresentadas a seguir.

5.1 Cenário da Proposta

Em uma metodologia ágil, como XP ou *Scrum*, os desenvolvedores vão produzindo o sistema e os artefatos correspondentes, de acordo com as funcionalidades definidas pelo cliente. A produção dos artefatos requer um Esforço Inicial (Ef_i). Os artefatos produzidos pelos desenvolvedores, envolvendo o código fonte e demais documentos de apoio ao código, são armazenados em uma ferramenta (gerenciador de projetos ou repositório). Os artefatos na ferramenta podem receber uma organização taxonômica, de acordo com o propósito para o qual foram produzidos (gerenciamento do projeto, requisitos, testes, projeto e arquitetura, implantação, padrões, codificação, entre outros). Ao final do *release*, os artefatos são organizados, prioritariamente os que compreendem o código fonte, para constituição do sistema a ser disponibilizado ao cliente.

A presente proposta visa inserir um formalismo de representação de conhecimento que realize o mapeamento entre os artefatos que estão disponíveis na ferramenta, permitindo associá-los aos conceitos do domínio para o qual o sistema foi desenvolvido e aos tipos de conhecimento aos quais estes artefatos se referem ou possuem. A atividade de mapeamento, que pode ser auxiliada por uma ferramenta automatizada, irá requerer um Esforço Adicional (Ef_{AD}) dos desenvolvedores, membros da equipe. Este esforço adicional não deve prejudicar a agilidade do processo. A Figura 5.1 ilustra o cenário correspondente a proposta.

Como pode ser visto na Figura 5.1, o objetivo do mapeamento é possibilitar, por meio de consultas, a recuperação do conhecimento necessário para a manutenção do sistema disponibilizado ao cliente. Esse mapeamento também poderá ser utilizado para o desenvolvimento de outro sistema

cujo domínio seja parecido com algum sistema já desenvolvido ou ainda que utilize alguns artefatos já produzidos para sistemas anteriores.

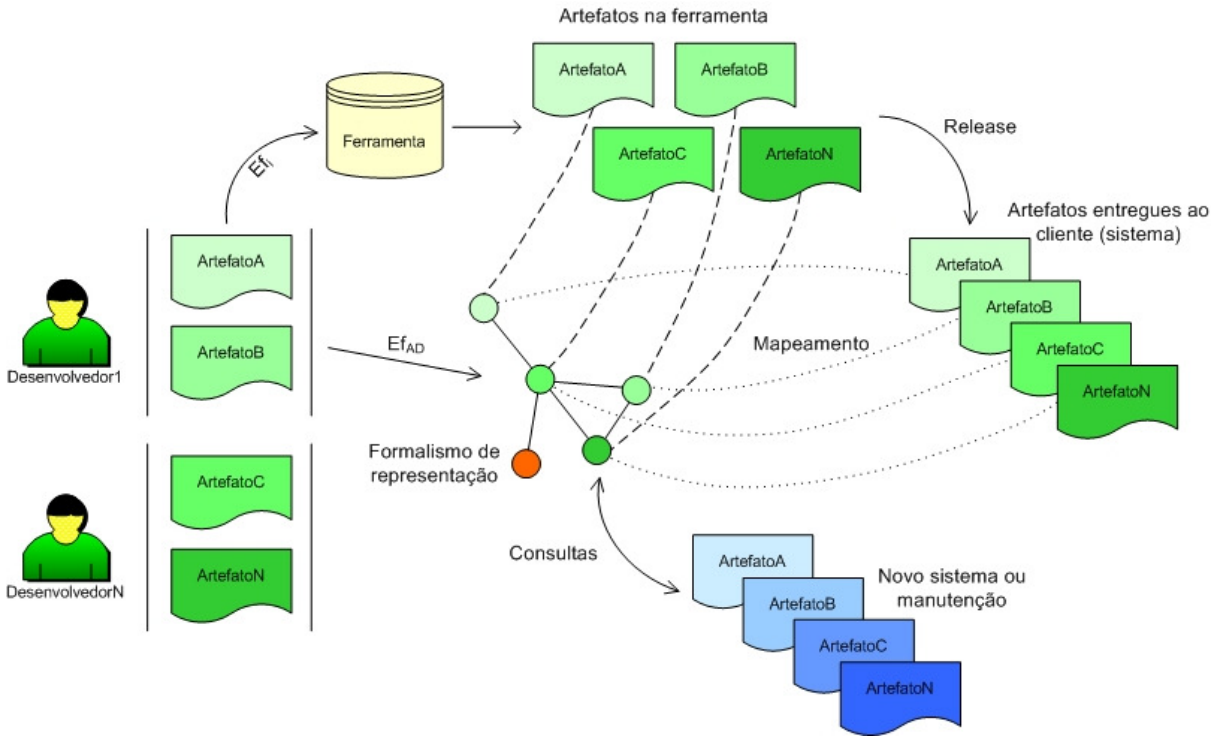


Figura 5.1: Cenário geral da proposta de trabalho

Para implementação da referida proposta, considera-se que os desenvolvedores façam uso de alguma ferramenta de apoio a aplicação da metodologia e gerenciamento dos artefatos produzidos. Esta ferramenta pode ser uma ferramenta de gerenciamento da metodologia adotada, como o *XPlanner*, para o XP, ou *ScrumWorks*, no caso do *Scrum*.

A proposta ainda prevê que seja desempenhado um novo papel na metodologia, o do *Engenheiro do Conhecimento*, responsável por realizar o mapeamento entre os artefatos produzidos. A criação desse novo papel visa a não alteração da metodologia e dos papéis já desenvolvidos nesta, o que poderia torná-la menos ágil. Desse modo, a representação do conhecimento é feita como uma etapa adicional no ciclo de vida da metodologia, assim como em [NOL07a]. Contudo, o papel do *Engenheiro de Software* pode ser desenvolvido pela mesma pessoa que desempenha o papel de *Testador* em metodologias como XP e *Scrum*, pois de acordo com [TEL04], é comum essa mesma pessoa também desenvolver o papel de *Redator Técnico*, responsável por organizar a documentação do sistema.

O trabalho do Engenheiro do Conhecimento deve ser auxiliado por um formalismo de representação de conhecimento (como ontologias) que permita o mapeamento entre os artefatos produzidos durante o desenvolvimento do sistema, como sugerido por [CER03]. Esse formalismo pode ser implementado em uma ferramenta que dê suporte ao mapeamento e que facilite a realização de consultas na base de conhecimento. Neste aspecto, a proposta se identifica com o trabalho de [BAS01] uma vez que a representação do conhecimento e a consulta a este é realizada juntamente com o desenvolvimento do projeto. A Figura 5.2 ilustra em detalhe o cenário descrito.

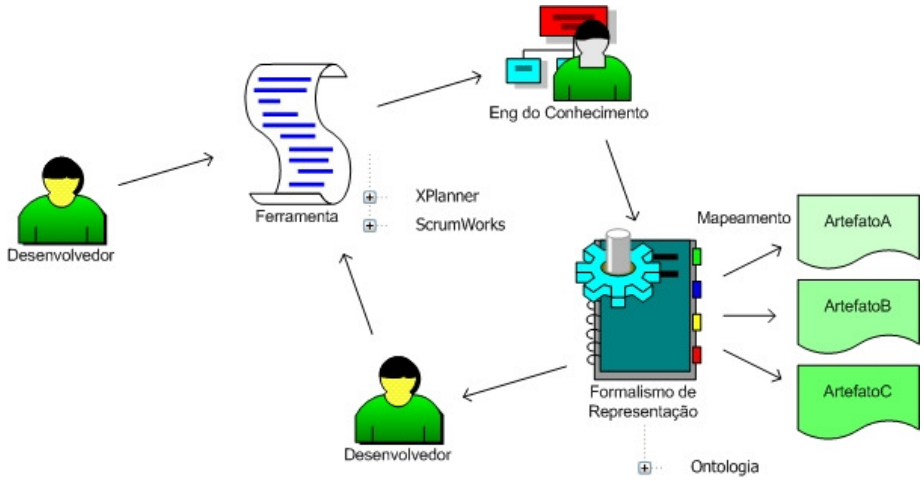


Figura 5.2: Cenário detalhado da proposta de trabalho

Nas próximas seções serão apresentadas a metodologia ágil e o formalismo de representação de conhecimento considerados, juntamente com a proposta de representação de conhecimento, por meio do mapeamento por conceitos do domínio e por tipo de conhecimento.

5.2 Escolha da metodologia ágil

Neste trabalho os estudos foram direcionados para as metodologias XP e *Scrum*, apesar da existência de várias metodologias ágeis. Tal escolha é justificada, por serem essas as metodologias mais utilizadas no contexto do desenvolvimento de software comercial e por serem as metodologias que apresentam o maior número de referências bibliográficas e estudos relacionados [SAL08].

Considerando as características de XP e *Scrum*, nota-se que as duas metodologias podem inclusive ser utilizadas em conjunto. Enquanto o *Scrum* é mais preocupado com o gerenciamento do

projeto de maneira global, o XP aborda os procedimentos adotados para a produção dos artefatos e como estes são produzidos. Em outras palavras, o *Scrum* determina como o processo será conduzido numa perspectiva gerencial e o XP como os artefatos serão criados dentro do processo estipulado.

Essa relação de complementaridade das metodologias XP e *Scrum* é abordada por [KUT02], quando afirma que as metodologias para desenvolvimento ágil de software que foram introduzidas durante os últimos anos podem ser classificadas como metodologias de meta-processo ou métodos de desenvolvimento específico. Dessa maneira, metodologias de meta-processo não descrevem abordagens de desenvolvimento específicas, mas se concentram nos procedimentos gerais para apoiar a equipe de desenvolvimento no estabelecimento de uma abordagem de desenvolvimento para um projeto específico. *Adaptive Software Model* (ASD), *Crystal* e *Scrum* pertencem a esta categoria. Em contraponto às metodologias de meta-processo, os métodos de desenvolvimento específico descrevem práticas comprovadas ou dão sugestões concretas para orientar a equipe de desenvolvimento para implementar o software que é adaptável às mudanças. São exemplos: *Dynamic Systems Development Methodology* (DSDM), *Extreme Programming* (XP) e *Feature Driven Development* (FDD).

Dessa forma, considerando que este trabalho se concentra em torno dos métodos de desenvolvimento e os artefatos gerados, será considerada a representação de conhecimento junto à metodologia *Extreme Programming* (XP). Contudo, a metodologia *Scrum* também será considerada na perspectiva do gerenciamento de projeto.

5.3 Escolha do formalismo de representação

Dentre os formalismos estudados têm-se aqueles passíveis de serem representados computacionalmente como redes semânticas, *frames*, ontologias, entre outros e aqueles associados a representações mentais tais como mapas mentais, mapas cognitivos e outros. Considerando o foco deste trabalho, um dos critérios para escolha do formalismo de representação é a capacidade de ser computacionalmente representado. Essa característica permitirá agregar agilidade no processo de busca, recuperação e inferência sobre o conhecimento representado, bem como estabelecer ligações entre os artefatos gerados durante o processo de desenvolvimento do software.

Tendo este critério como base, o universo de escolha restringe-se aos seguintes formalismos: ontologias, que agregam características inerentes a lógica, redes semânticas, *frames* e taxonomias;

mapas conceituais, que também podem ser organizados em um modelo computável; e *design rationales*, que armazenam informações referentes a decisões tomadas e os motivos que levaram a estas decisões.

Design rationales são passíveis de representação computacional, como demonstrado em [SAU03]. Abordagens como IBIS [KUN70] e QOC [MAC91] favorecem a captura e representação de conhecimento, com base em argumentação. A abordagem IBIS (*Issue Based Information System*), considerada a primeira representação explícita para raciocínio em um contexto de design, tem como propósito principal registrar a deliberação do design. As questões de design são organizadas como *temas* (*issues*), em um processo de deliberação aberta. Cada tema é seguido por uma ou mais *posições* que procuram respondê-lo, podendo estas serem adotadas como solução para o tema ou serem rejeitadas. Fatores que apóiam ou contrariam as posições são definidos como *argumentos*. Já a abordagem QOC (*Questions, Options and Criteria*) consiste em uma notação semiformal de análise do espaço de design, que busca explicar com base em critérios e argumentos, porque determinadas opções foram escolhidas dentre um espaço de possibilidades. Tem como foco os seguintes conceitos: *questões*, que identificam os principais temas para estruturar o escopo de alternativas; *opções*, que viabilizam possíveis respostas a estas questões; e *critérios* que formam as bases para avaliação e escolha entre as opções.

Contudo, as representações geradas por essas abordagens constituem-se em sua essência de representação textual, tornando trabalhoso o processo de captura e representação. A realização de inferência posteriormente sobre o conhecimento representado depende da implementação em ambientes externos ou em conjunto com outro formalismo, o que dificulta a sua utilização no contexto desta proposta, porém não inviabiliza o uso de *design rationales* em trabalhos futuros.

Os mapas conceituais também representam uma alternativa computável. Entretanto, são poucos os relatos de estudos que empregam mapas conceituais junto ao processo de desenvolvimento de software. Por gerarem uma representação específica, no caso das metodologias ágeis torna-se necessário um esforço adicional para a geração do mapa, que se constituirá em um novo artefato sobre o sistema. O processo de inferência posteriormente também demandaria estudos mais aprofundados no sentido de identificar tal possibilidade.

Quanto às ontologias, ainda que estas também demandem um relativo esforço no processo de concepção, encontram-se estudos, como os realizados por [FAL04] e [NOL07a], que empregam o formalismo no desenvolvimento de software. O uso de ontologias, bem como sua evolução, vem sendo apoiado por organismos de padronização como o W3C, visando sua aplicação principalmente

na Web Semântica. Também se encontram relatos do uso de ontologias no desenvolvimento de Sistemas Multiagentes. Essas características demonstram se tratar de um formalismo com possibilidade de evolução e solidez. Em decorrência disso, este trabalho irá considerar a representação de conhecimento junto às metodologias ágeis por meio de ontologias, como será abordado adiante.

5.4 Representação por conceito do domínio

O escopo da referida proposta compreende a representação de conhecimento em XP com ontologias, visando o mapeamento entre os artefatos produzidos por meio de conceitos do domínio do sistema. A ideia tem como fundamento adaptar para XP a solução apresentada por Noll em [NOL07a], que representa o conhecimento por meio de uma ontologia gerada a partir do Modelo de Domínio. O objetivo é não alterar as fases ou disciplinas já definidas para XP, mas sim acrescentar a representação de conhecimento como uma atividade adicional. Antes, porém, foi realizado um levantamento a fim de identificar os artefatos presentes nas metodologias ágeis, para posteriormente estabelecer a associação destes com conceitos do domínio, como relatado a seguir.

5.4.1 Levantamento de artefatos

Com o objetivo de identificar os artefatos gerados para documentação durante o desenvolvimento com métodos ágeis, foi realizado um levantamento envolvendo estudos de caso, *surveys* e outros trabalhos que abordam à temática. O levantamento seguiu como embasamento teórico conceitos de uma Revisão Sistemática, descritos em [KIT04] e [BIO05].

A questão básica de pesquisa buscou identificar quais artefatos são utilizados para a documentação de sistemas desenvolvidos com metodologias ágeis. A *string* de busca utilizada foi: "*(artifact OR artefact OR documentation OR document) AND (extreme programming OR scrum OR agile) AND (case study OR survey)*".

A pesquisa se concentrou em trabalhos disponibilizados nas seguintes bibliotecas digitais: *IEEE Xplore*; *ACM Digital Library; ScienceDirect*; *SpringerLink*. Também foram considerados livros e trabalhos publicados pelas editoras *John Wiley*, IBM e *MS Press*, além de trabalhos acadêmicos. Outros detalhes da revisão foram suprimidos em função da extensão do trabalho,

porém podem ser conferidos no protocolo apresentado no Apêndice A. A seguir é apresentado o Quadro 5.1 com os trabalhos incluídos no levantamento dos artefatos.

Quadro 5.1: Trabalhos considerados no levantamento de artefatos de documentação ágil

Estudo	Fonte	Pesquisa	Metodologia
Ambler [AMB02]	Livro John Wiley	Proposta	Ágil
Kutschera e Schäfer [KUT02]	Paper RTO-MP	Estudo de Caso	Ágil
Forward e Lethbridge [FOR02]	Paper ACM	Survey	Ágil
Rüping [RUP03]	Livro John Wiley	Proposta	Ágil
Hansson *et al* [HAN06]	Paper Elsevier	Estudo de Caso	Ágil
Law e Charron [LAW05]	Paper ACM	Estudo de Caso	Ágil
Hanks *et al* [HAN08]	Paper ACM	Proposta	Ágil
Huang e Holcombe [HUA09]	Paper Elsevier	Experimento	Ágil
Schwaber [SCH04]	Livro MS Press	Proposta	Scrum
Marchesi *et al* [MAR07]	Paper Springer	Estudo de Caso	Scrum
Paelke e Nebe [PAE08]	Paper ACM	Estudo de Caso	Scrum
Pikkarainen *et al* [PIK08]	Paper Springer	Estudo de Caso	Scrum
Hedin *et al* [HED03]	Paper Elsevier	Proposta	XP
Smith [SMI03]	Paper IBM	Comparativo	XP
Abrahamsson e Koskela [ABR04]	Paper IEEE	Estudo de Caso	XP
Dulipovici e Robillard [DUL04]	Paper IEEE	Experimento	XP
Teles [TEL05]	Dissertação UFRJ	Estudo de Caso	XP
Kokkoniemi [KOK08]	Paper IEEE	Exemplo de uso	XP

Foram identificados um total de 75 artefatos nos oito trabalhos que envolvem metodologias ágeis em geral (classificados como “Ágil” na coluna “Metodologia” do Quadro 5.1). Entre os artefatos mais citados estão: Casos de teste; Notas de reunião; Código/comentários no código; Manual do usuário; Estórias; Requisitos; Requisitos não-funcionais; Testes de unidade; Documentação de manutenção; e Manual de instalação e administração. Especificamente sobre *Scrum* o levantamento identificou 23 artefatos nos quatro trabalhos considerados. Os mais citados foram: *Product backlog*; *Sprint backlog*; *Burndown chart*; Casos de teste; Modelos; e Código/comentários no código. Sobre XP foram identificados um total de 57 artefatos nos seis trabalhos considerados. Dentre os mais citados estão: Estórias; Metáfora; Código/comentários no código; Tarefas; Plano de *release*; Estimativas das estórias; Dados de teste; Descrição arquitetural; Padrões de codificação; Ferramentas de gerenciamento de código; e Manual do usuário.

A relação completa dos artefatos encontrados é listada no Apêndice B. Os artefatos foram organizados de acordo com classificações apresentadas em [KUT02] e [RUP03], sendo elas: Definição e gerenciamento do projeto; Requisitos e especificação; Projeto e arquitetura;

Codificação; Testes; Implantação e operação; Padrões e diretrizes; Manutenção; e Utilização. Para essa listagem não foram eliminadas redundâncias semânticas, preservando-se o nome original dos artefatos conforme citados nos trabalhos.

Considerando a *string* de busca padrão utilizada na pesquisa, procurou-se identificar trabalhos relacionados a estudos de caso ou *surveys*, que apresentassem além da citação do artefato a sua estrutura. Entretanto, observou-se que a maioria dos trabalhos apenas cita os artefatos, sendo poucos os que apresentam em detalhes a estrutura e exemplos de utilização. Também pôde ser observado na relação de artefatos que alguns itens se referem ao mesmo tipo de documentação e conteúdo apesar de apresentarem uma grafia diferente. É o caso dos artefatos: relatórios e notas sobre reuniões, atas de reuniões e anotações de conversas, na relação apresentada sobre XP. Em outros casos, também são apresentados artefatos genéricos que certamente envolvem artefatos mais específicos, como: dados de teste, que incluem testes de unidade e testes de aceitação. Essas relações motivaram a classificação dos artefatos de acordo com o propósito pretendido, a fim de melhor identificá-los, porém ainda demandam estudos mais aprofundados. No Apêndice C é apresentada uma lista com os artefatos ágeis encontrados, somando-se as sublistas de cada metodologia e retirando-se as redundâncias semânticas.

5.4.2 Elaboração do Modelo de Domínio

Como definido em [NOL07a], uma ontologia inicial é gerada a partir do Modelo de Domínio. Este artefato descreve os conceitos do domínio e seus relacionamentos, por meio da notação do diagrama de classes UML. Embora não seja um artefato característico da metodologia XP, Ambler em [AMB04] propõe o Modelo de Domínio como um dos artefatos iniciais para a modelagem de requisitos no desenvolvimento ágil.

No levantamento de documentação apresentado na Seção 5.4.1 a Metáfora aparece como um artefato de importância para XP. São objetivos da metáfora descrever como o sistema funciona e ampliar a comunicação entre cliente e desenvolvedores sobre as funcionalidades do sistema. Também objetiva fornecer uma visão conceitual sobre o sistema, permitindo que os desenvolvedores tenham uma visão de integridade sobre o mesmo [TEL05].

O Modelo de Domínio pode contribuir com a Metáfora para o entendimento dos conceitos do sistema. Assim, este deve ser elaborado durante a disciplina Fazer Explorações Iniciais, proposta

em [SAM04], na fase de Exploração no ciclo de vida, visando facilitar a geração da ontologia. A Figura 5.3 apresenta a modelagem da referida disciplina, acrescentando o desenvolvimento do Modelo de Domínio como artefato. Foi utilizada a sintaxe do diagrama de classe, com a notação SPEM.

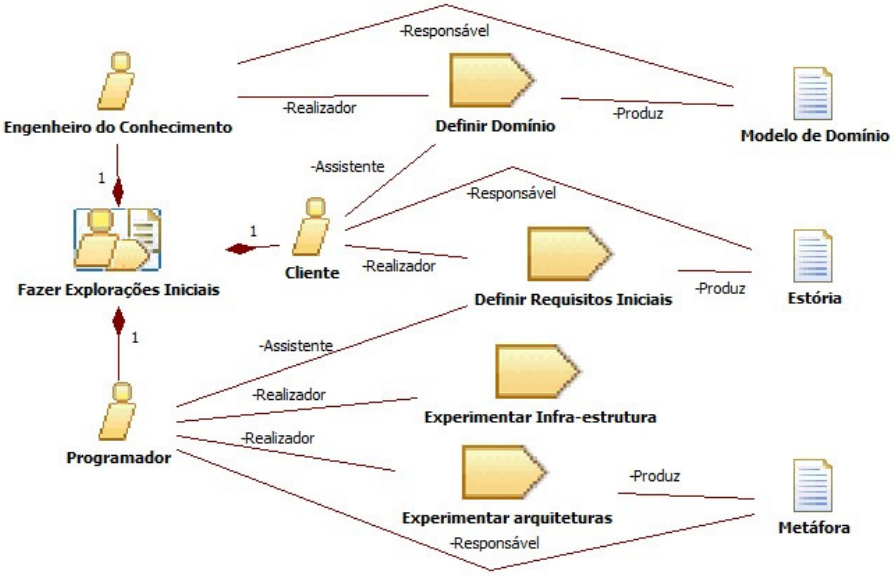


Figura 5.3: Adaptação da disciplina Fazer Explorações Iniciais para inserção do Modelo de Domínio
Fonte: Adaptado de [SAM04]

Para ilustrar a elaboração do Modelo de Domínio será considerada a realidade de uma Clínica de Atendimento Médico, para a qual a administração deseja um sistema que gerencie agendamentos de consultas, informações sobre pacientes, atendimentos dos médicos que nela atuam, entre outras atividades. A Figura 5.4 representa o Modelo de Domínio definido para essa realidade. Embora seja bastante simples e conhecida, esta realidade possui os elementos necessários para a demonstração da proposta de representação.

Definido o Modelo de Domínio, a ontologia é então gerada a partir do mapeamento deste, conforme proposto adiante. Na fase de Planejamento em XP, a ontologia pode ser refinada a partir das estórias e dos demais artefatos produzidos durante as iterações.

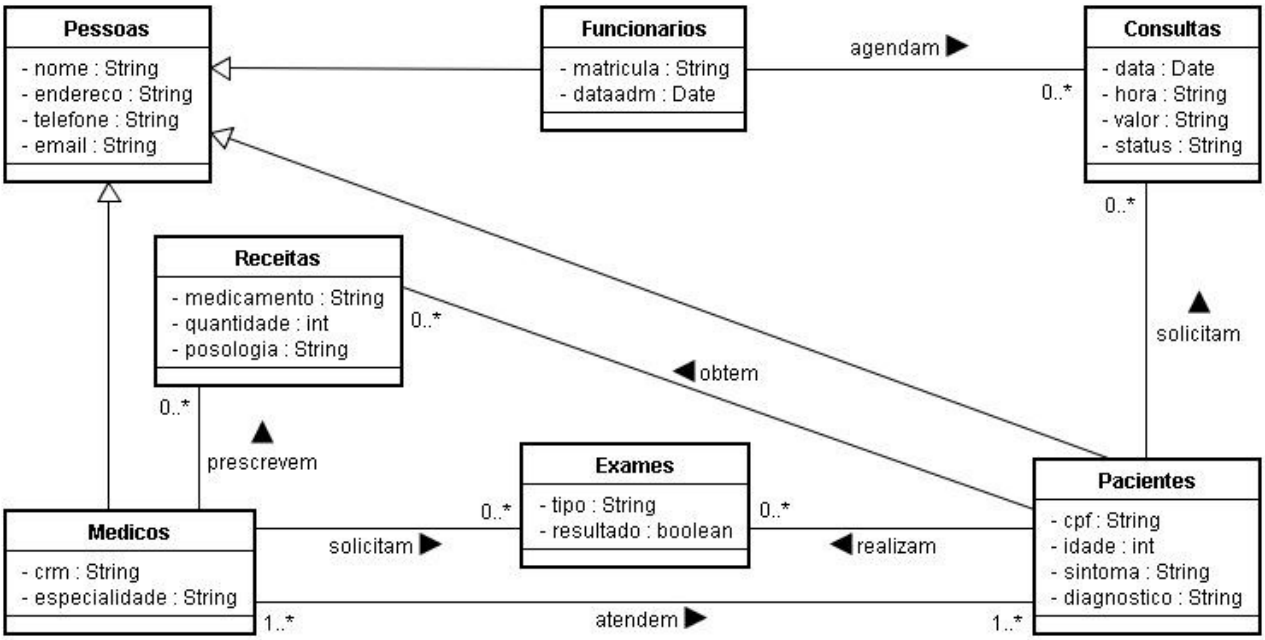


Figura 5.4: Exemplo de Modelo de Domínio para uma Clínica de Atendimento Médico

5.4.3 Disciplina representar conhecimento

Seguindo a proposta de [NOL07a], para representação do conhecimento, será incluída uma nova disciplina em XP, denominada *Representar Conhecimento*. Esta disciplina produz um modelo lógico que representa os conceitos do domínio do sistema e que serão relacionados com os artefatos produzidos. Ela recebe como entrada o Modelo de Domínio e artefatos produzidos em XP e terá como saída uma ontologia. A ênfase da disciplina é maior nas fases de Exploração e Planejamento.

Como mencionado no início deste capítulo, a abordagem sugere o desempenho de um novo papel em XP, o do Engenheiro do Conhecimento, responsável por elaborar o Modelo de Domínio, definir a ontologia e refiná-la ao longo do desenvolvimento do projeto.

Para definição do processo da disciplina também será utilizada a notação do SPEM, ilustrada no Quadro 2.1. As principais atividades da disciplina são definidas por [NOL07a], sendo elas: Projeto; Manutenção; e Validação.

5.4.3.1 Projeto

A atividade de Projeto consiste em definir uma versão preliminar da ontologia de conceitos considerando o Modelo de Domínio, produzido na disciplina Fazer Explorações Iniciais. O objetivo é delimitar o escopo e explicitar os conceitos do domínio do sistema e os relacionamentos (taxonômicos ou não), organizando-os em um formalismo lógico.

O Engenheiro do Conhecimento é o responsável por realizar a atividade, que tem a ontologia como o artefato gerado. A partir do Modelo de Domínio é feita a extração da ontologia. A versão inicial da ontologia pode passar por um refinamento de conceitos, taxonomia, *object properties* e *datatype properties*. Este refinamento visa manter a ontologia atualizada, caso sejam descobertos novos conhecimentos sobre o domínio, a partir de artefatos como a metáfora e as estórias, produzidos durante as disciplinas Fazer Explorações Iniciais e Definir e Revisar Requisitos. A Figura 5.5 ilustra a adaptação para XP do modelo de processo da atividade de projeto definido por [NOL07a].

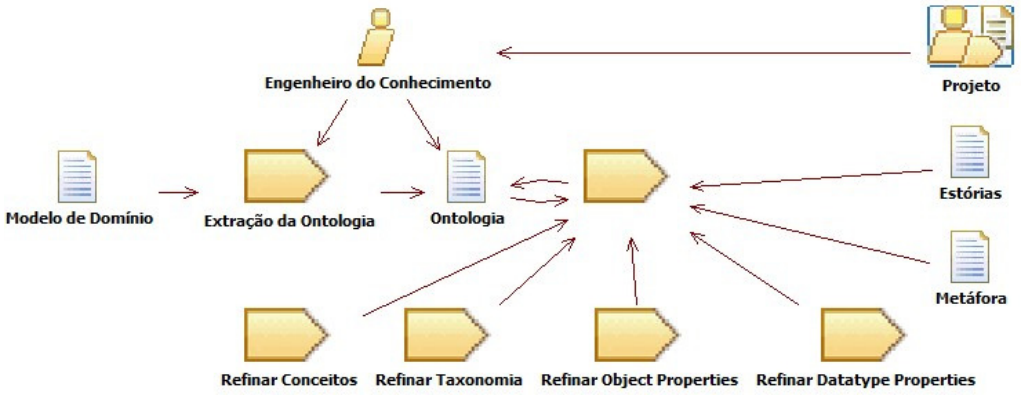


Figura 5.5: Processos da atividade de Projeto adaptada para XP
Fonte: Adaptado de [NOL07a]

A extração da ontologia pode ser automatizada, segundo [NOL07a], realizando-se o mapeamento de estruturas como classes, atributos, associações e generalizações em UML para estruturas em OWL [W3C04b] com equivalência semântica. Este mapeamento tem como base a proposta *Ontology Definition Meta-Model* (ODM) definida pelo *Ontology Working Group* (OWG) da OMG [OMG09b], segundo a qual é possível mapear um modelo de classes para um modelo lógico. Para cada estrutura UML, presente no Modelo de Domínio, tem-se o seguinte mapeamento em OWL:

- Classe: uma classe em UML (*Pessoas*, *Medicos*, *Funcionarios*) corresponde a uma *OWL Class* em OWL;

- Atributo: cada atributo (*matricula*, *dataadm*) é mapeado como um *Datatype Property*, possuindo como domínio a classe pai (*Funcionarios*) e como imagem o tipo XSD definido pelo *XML Schema* (*strin*g, *data*);

- Associação: cada associação (*Medicos atendem Pacientes*) corresponde a um *Object Property*, cujo domínio é a classe inicial da associação (*Medicos*) e imagem é a classe final da associação (*Pacientes*). Como restrições têm-se: a propriedade é *simétrica* se a associação inicial e final é navegável; cria-se uma restrição de *cardinalidade* se a ocorrência das associações é igual, do contrário criam-se restrições de *cardinalidade mínima e máxima*;

- **Generalização:** cada generalização estabelece uma relação taxonômica entre as classes (*Medicos*, *Pacientes* e *Funcionarios* são subclasses de *Pessoas*).

Para exemplificar o mapeamento, a Figura 5.6 apresenta parte da estrutura do Modelo de Domínio proposto anteriormente para a Clínica de Atendimento Médico e o código equivalente da ontologia gerada. Para realizar o mapeamento automático pode ser utilizada uma ferramenta como *ArgoUML+ONTrace* proposta por [NOL07a].

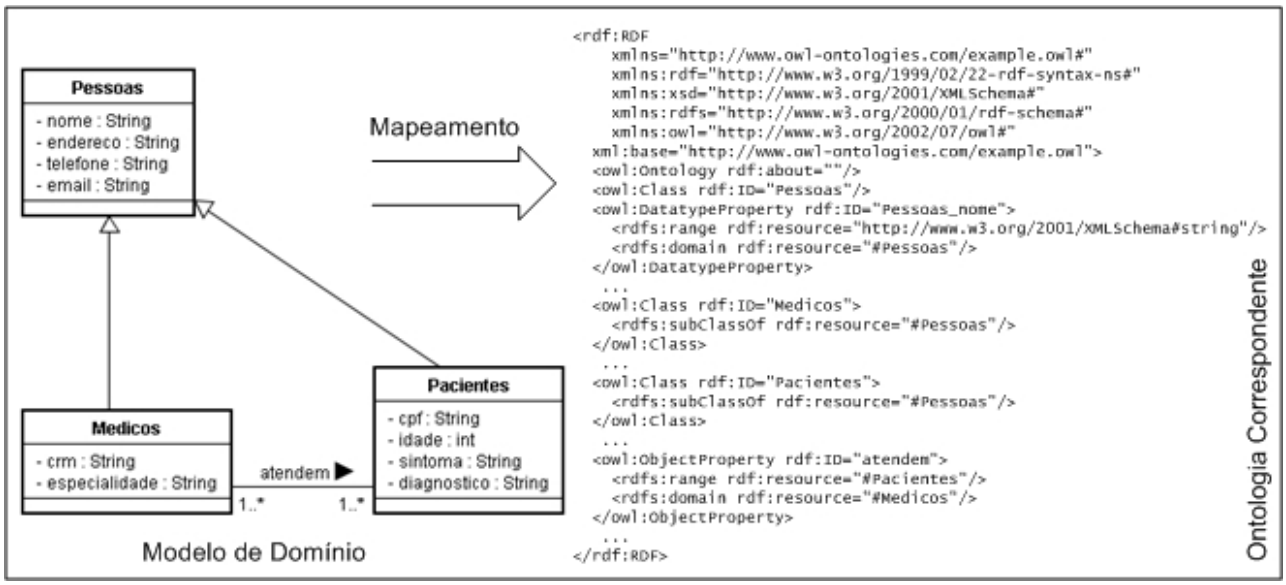


Figura 5.6: Mapeamento do Modelo de Domínio para uma Ontologia correspondente

5.4.3.2 Manutenção

A ontologia gerada na atividade de Projeto pode ser refinada para representar o conhecimento sobre o domínio de forma mais abrangente, como proposto anteriormente. Podem ser adicionados a ontologia novos conceitos, novos relacionamentos e regras lógicas. Para tal, o Engenheiro do Conhecimento pode utilizar uma ferramenta específica para a edição de ontologias, a exemplo do *Protégé* [GEN03], considerando que a presente proposta busca utilizar ferramentas já existentes. Recomenda-se seguir uma metodologia de construção de ontologias, como a *On-To-Knowledge* [SUR02], como abordagem para o refinamento. A Figura 5.7 mostra a área de trabalho do *Protégé* em que foi importada a ontologia gerada a partir do Modelo de Domínio da Clínica de Atendimento Médico.

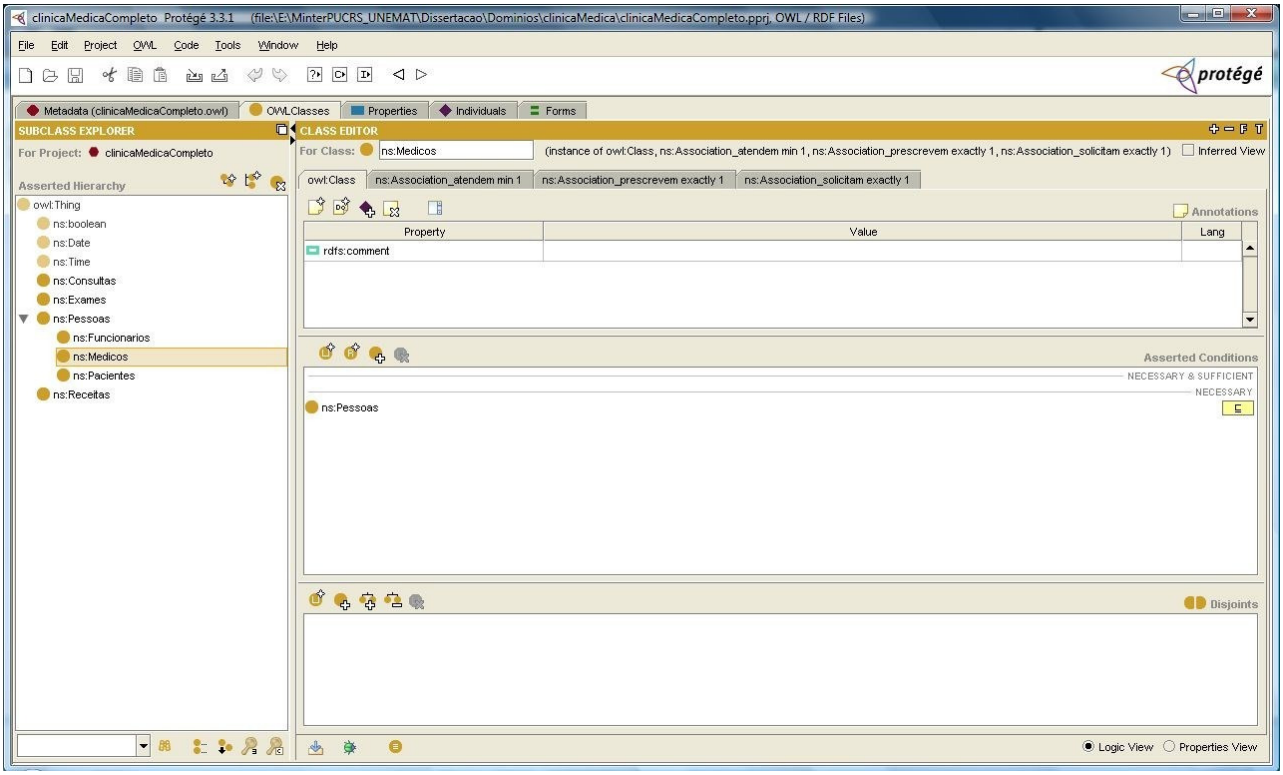


Figura 5.7: Tela do *Protégé* para edição da Ontologia gerada do Modelo de Domínio

Conforme [NOL07a], a atividade de Manutenção tem como objetivo principal a associação entre os artefatos produzidos durante o desenvolvimento do software com os conceitos pertencentes ao domínio, representados por meio da ontologia. Essa associação gera elos de rastreabilidade que integram o conhecimento sobre os componentes dos artefatos do projeto do software com os

conceitos do domínio, implementando de fato a representação por conceito. As principais atividades associadas à manutenção são [NOL07a]: refinar conceitos e taxonomia; identificar novos *object properties* e *datatype properties*; adicionar as restrições lógicas e relacionar os elementos dos artefatos de XP com os conceitos da ontologia.

Na proposta adaptada para XP, são considerados como entrada para a atividade de Manutenção, além da ontologia, os artefatos produzidos durante as disciplinas Definir e Revisar Requisitos, Planejar *Release*, Planejar a Iteração, Escrever Testes Funcionais, e Fazer Projeto, sendo eles: Estórias; Plano de *Release*; Tarefas (em cada iteração); Testes Funcionais (cenários de teste); Projetos (descrição arquitetural). Como em XP o projeto e a codificação ocorrem em paralelo, adicionalmente será estudada a possibilidade de incluir na associação artefatos gerados durante a codificação, como: Testes de Unidade; Código Gerado; Padrões de Codificação. E também artefatos referentes à implantação como Tutoriais e Guias (Manuais) do Usuário. A atividade de Manutenção será executada em XP ao final de cada iteração e por ocasião da entrega do *release*. A Figura 5.8 mostra os processos da atividade de Manutenção adaptada para XP.

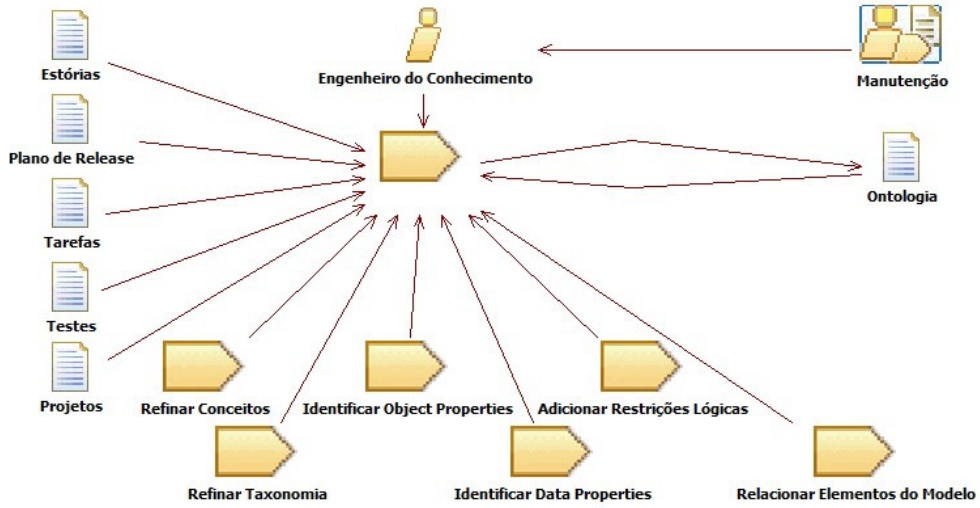


Figura 5.8: Processos da atividade de Manutenção adaptada para XP
Fonte: Adaptado de [NOL07a]

Para ilustrar a associação entre os elementos dos artefatos do sistema e os conceitos do domínio, que é o objetivo principal da atividade de Manutenção, serão consideradas as seguintes estórias (*User Stories – US*), que compõem os artefatos do sistema da Clínica de Atendimento Médico, citado anteriormente:

*US01 – Manter Cadastro de **Pacientes***
*Descrição: O sistema manterá um cadastro dos **pacientes** que são atendidos na clínica, registrando suas informações pessoais e dados de contato.*

Prioridade: Alta	*Tempo Estimado: 40 horas*
Número: 1	*Tempo Atual: 20 horas*
Cliente: Selleri	*Tempo Restante: 20 horas*
Rastreador: Fernando	*Tipo da Atividade: Planejada*
Data: 07/04/2009 17:34	*Status: Andamento*

*US02 – Manter Cadastro de **Médicos***
*Descrição: O sistema manterá um cadastro dos **médicos** que atendem na clínica, registrando informações pessoais e profissionais, contato e especialidade. Na clínica atendem pediatras, cardiologistas e psicólogos.*

Prioridade: Alta	*Tempo Estimado: 40 horas*
Número: 2	*Tempo Atual: 10 horas*
Cliente: Selleri	*Tempo Restante: 30 horas*
Rastreador: Fernando	*Tipo da Atividade: Planejada*
Data: 07/04/2009 17:40	*Status: Andamento*

*US03 – Agendar **Consultas***
*Descrição: Os **funcionários** realizarão o agendamento das **consultas** no sistema. O agendamento deverá conter o **paciente**, o **médico**, o valor e o status da **consulta**.*

Prioridade: Alta	*Tempo Estimado: 40 horas*
Número: 3	*Tempo Atual: 00 horas*
Cliente: Selleri	*Tempo Restante: 40 horas*
Rastreador: Fernando	*Tipo da Atividade: Planejada*
Data: 07/04/2009 17:45	*Status: Em espera*

Ao verificar estes artefatos o Engenheiro do Conhecimento irá associá-los aos conceitos do Modelo de Domínio, já representados pela ontologia do domínio. Desse modo:

- A estória *US01 – Manter Cadastro de Pacientes* será associada ao conceito do domínio *Pacientes*;

- A estória *US02 – Manter Cadastro de Médicos* terá associação com *Medicos*;

- A estória *US03 – Agendar Consultas* estará associada aos conceitos *Funcionarios*, *Pacientes*, *Medicos* e *Consultas*.

A Figura 5.9 apresenta o cenário descrito, com as respectivas associações. As associações dessas estórias com outros conceitos do domínio foram suprimidas, evitando estender o exemplo.

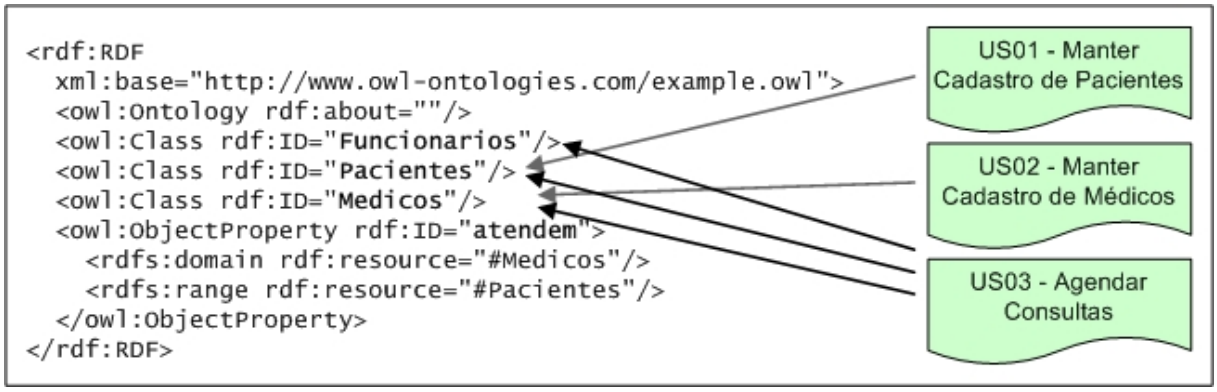


Figura 5.9: Cenário de associação de artefatos (estórias) com conceitos do domínio
Fonte: Adaptado de [NOL07a]

Tal como relatado em [NOL07a], no cenário apresentado na Figura 5.9, que considera artefatos de XP, também é possível realizar consultas e recuperar elos de rastreabilidade entre os artefatos associados aos conceitos, tais como:

- A estória *Manter Cadastro de Médicos* relaciona-se diretamente com a estória *Agendar Consultas*, pois ambas estão associadas ao conceito *Medicos*.

- A estória *Manter Cadastro de Médicos* relaciona-se indiretamente com a estória *Manter Cadastro de Pacientes*, pois o conceito *Medicos* relaciona-se com o conceito *Pacientes* por meio de uma propriedade denominada *atendem*. Esta informação surge da integração entre os artefatos e a representação de conhecimento.

Na Seção 5.6 será apresentada uma estrutura ontológica que permite automatizar a representação de conhecimento e a rastreabilidade por conceitos nas metodologias ágeis.

5.4.3.3 Validação

Para [NOL07a] a atividade de Validação consiste em avaliar constantemente o conhecimento sobre o domínio do sistema. Prevê-se que o Especialista do Domínio seja o responsável por avaliar a integridade, completude e corretude do conhecimento representado pelo Engenheiro do Conhecimento. Em XP o Especialista do Domínio geralmente é uma pessoa da parte do cliente, senão o próprio, que acompanha a equipe durante todo o desenvolvimento. Somente em casos muito específicos sugere-se a contratação de um especialista para auxiliar a equipe de desenvolvimento.

Recomenda-se utilizar abordagens em que se verifica a consistência e coerência da ontologia e de seu vocabulário referente ao domínio a cada iteração do ciclo de desenvolvimento, a exemplo da metodologia *On-To-Knowledge*. São propostos três tipos de validação para cada etapa da criação da ontologia [NOL07a]: Teste Unitário e Teste de Integração, executados pelo Engenheiro do Conhecimento; e Teste de Aceitação, executado por aplicações. Como exemplo de teste de unidade, pode-se definir visões sobre a ontologia para resolver tarefas de inferência específicas, considerando os conceitos com os quais determinados usuários lidam frequentemente no desenvolvimento de suas ações. Testes de integração enfocam, por exemplo, a integração da ontologia do domínio com a ontologia referente aos artefatos das metodologias ágeis. Testes de aceitação podem ser realizados para verificar a consistência da ontologia como um todo na representação do conhecimento. A ferramenta *OntoEdit* [SUR02] pode ser utilizada para este tipo de teste, bem como ferramentas que utilizem a ontologia para recuperação do conhecimento, a exemplo da ferramenta protótipo proposta neste trabalho.

5.5 Representação por tipo de conhecimento

Como abordado na Seção 5.4, a representação de conhecimento por conceitos pode facilitar a rastreabilidade entre os artefatos produzidos durante o desenvolvimento ágil, além de suprir a necessidade das metodologias ágeis referente a uma modelagem do domínio mais formal. Entretanto, considerando que as metodologias ágeis introduzem artefatos novos, alguns utilizados apenas por determinada metodologia em específico, torna-se necessário uma análise mais detalhada acerca do tipo de conhecimento ao qual cada artefato se refere. Tal análise poderá facilitar a equipe a identificar em quais artefatos pode estar certo conhecimento a ser consultado, bem como quais artefatos ágeis utilizar quando se deseja representar determinado tipo de conhecimento, no caso de se adaptar uma metodologia ágil de acordo com as características de um projeto.

Contudo, torna-se necessário primeiramente identificar os principais tipos de conhecimento presentes no desenvolvimento de sistemas. Desse modo, foi realizado um levantamento conforme relatado adiante.

5.5.1 Levantamento de tipos de conhecimento

Tendo como objetivo encontrar os tipos de conhecimento envolvidos no desenvolvimento e manutenção de software, foi realizado um levantamento em trabalhos relacionados ao assunto. A exemplo do levantamento sobre artefatos apresentado na Seção 5.4.1, o levantamento sobre os tipos de conhecimento também teve como embasamento teórico os conceitos de Revisão Sistemática, apresentados por [KIT04] e [BIO05].

A seguinte questão básica de pesquisa foi definida para o levantamento: “Quais os tipos de conhecimento associados ao desenvolvimento e manutenção de sistemas?”. A *string* padrão utilizada nas buscas foi: *("type of knowledge" OR "kind of knowledge") AND ("software development" OR "software engineering")*. A *string* padrão precisou de alguns ajustes a fim de refinar os resultados de acordo com o mecanismo de busca, porém atentou-se para que esta não perdesse o foco do levantamento.

O levantamento considerou como fontes de pesquisa os trabalhos disponibilizados nas seguintes bibliotecas digitais: *IEEE Xplore*; *ACM Digital Library*; *ScienceDirect*; *SpringerLink*; *SBC*. Outros detalhes do levantamento foram suprimidos para não estender o relato, mas podem ser conferidos no protocolo apresentado no Apêndice D. No Quadro 5.2 são apresentados os trabalhos incluídos no levantamento sobre os tipos de conhecimento.

Quadro 5.2: Trabalhos considerados no levantamento de tipos de conhecimento

	Autor	**Fonte**	**Pesquisa**
Estudo 1	Villela, Travassos e Rocha [VIL04]	Paper SBC	Survey
Estudo 2	Santos *et al* [SAN05]	Paper Springer	Survey
Estudo 3	Sousa, Anquetil e Oliveira [SOU04]	Paper SBC	Experimento
Estudo 4	Anquetil *et al* [ANQ07]	Paper Elsevier	Experimento
Estudo 5	Kobayashi [KOB04]	Paper IEEE	Proposta
Estudo 6	Ramal, Meneses e Anquetil [RAM02]	Paper IEEE	Experimento
Estudo 7	Boh [BOH08]	Paper Elsevier	Survey
Estudo 8	Rodríguez-Elias *et al* [ROD08]	Paper Elsevier	Estudo de Caso

Por meio da leitura dos trabalhos selecionados identificou-se um total de 82 tipos de conhecimentos presentes no desenvolvimento de software. Entre os tipos de conhecimento mais encontrados nos trabalhos observados estão: Domínio da Aplicação; Codificação; Testes; Documentação de Software; Análise e Especificação de Requisitos de Software; Ferramentas

usadas; Linguagens e técnicas de programação; Distribuição de Competências entre os Profissionais da Organização; Processos Organizacionais e Estrutura Organizacional de Clientes; Projeto (design) de Software; Processo de Software da Organização; Estrutura Organizacional. Estes foram listados em pelo menos quatro dos oito trabalhos considerados.

A relação completa dos tipos de conhecimento é apresentada no Apêndice E, organizada de acordo com o grupo de conhecimento ao qual pertencem, conforme [SAN05] e [ANQ07], sendo eles: Conhecimento sobre Clientes; Conhecimento sobre Dados Históricos da Organização; Conhecimento sobre o Domínio; Conhecimento sobre Parceiros Técnicos; Conhecimento sobre a Organização; Conhecimento sobre Melhores Práticas no Desenvolvimento; Conhecimento sobre Manutenção; e Outros Conhecimentos.

5.5.2 Associação de artefatos aos tipos de conhecimento

Após a realização do levantamento dos tipos de conhecimento, procurou-se estabelecer uma associação entre os artefatos e os tipos de conhecimento a que se referem. Para essa tarefa, foi realizada uma revisão dos trabalhos utilizados no levantamento dos artefatos das metodologias ágeis, descrito na Seção 5.4.1. Para cada artefato, associou-se o tipo de conhecimento expressado, de acordo com as informações contidas sobre o artefato nos trabalhos que o abordava. Essa associação foi realizada para os artefatos das metodologias XP e *Scrum*, por serem essas as metodologias abordadas neste trabalho. Por meio da revisão realizada foi possível identificar, por exemplo, os tipos de conhecimento presentes nos artefatos de XP listados a seguir (a associação completa dos artefatos aos tipos de conhecimento encontra-se no Apêndice F):

- Estórias: *Gerência de Projeto*: as estórias contribuem para a gerência e são decompostas em tarefas a serem desenvolvidas para implementação dos requisitos [SMI03]; por meio de atributos como tempo estimado, prioridade e outros, as estórias possuem relação com o tempo de desenvolvimento, favorecendo a gerência de projeto [ABR04]; as estórias rastreiam o progresso do projeto e das iterações [KOK08]; *Domínio da Aplicação*: as estórias por meio de suas descrições, bem como prioridades estabelecidas pelo cliente, favorecem a compreensão do domínio para o qual o sistema será desenvolvido [HED03]; *Análise e Especificação de Requisitos*: para [HED03], [SMI03], [ABR04], [DUL04] e [TEL05], as estórias representam os requisitos capturados para o sistema; *Planos de Projeto*: além de favorecer o planejamento [SMI03], as estórias proveem um processo no qual estratégias de *release* incremental são desenvolvidas para evoluir o produto ao

longo do prazo; *Estimativa de Custos*: conforme [SMI03] e [ABR04], por meio do tempo estimado pela equipe e da prioridade definida pelo cliente, as estórias possibilitam a estimativa de custos no projeto; *Negociação entre área do setor de tecnologia*: possibilitam a negociação com o cliente com relação as funcionalidades do sistema [SMI03].

- Tarefas: *Distribuição de Competências entre os Profissionais da Organização*: as tarefas são distribuídas entre os membros da equipe [ABR04], assim uma consulta as tarefas de um projeto pode fornecer um quadro sobre a distribuição do trabalho na equipe; *Gerência de Projeto*: assim como as estórias, as tarefas permitem que o progresso do projeto e das iterações seja rastreado [KOK08]; *Análise e Especificação de Requisitos*: para [ABR04] e [DUL04], as tarefas são requisitos decompostos em nível gerencial; *Implementação da Aplicação*: as tarefas representam as atividades desenvolvidas para implementação dos requisitos estabelecidos para a aplicação [ABR04]; *Projeto (design) de Software*: as tarefas representam o projeto em essência [SMI03], pois expressam a maneira como os requisitos serão atendidos.

- Plano de *Release*: *Gerência de Projeto*: de acordo com [SMI03] e [DUL04], o plano de *release* consiste em um artefato clássico da gerência de projeto; *Processo de Fornecimento de Produto*: prevê para cada *release* a disponibilidade de valor ao negócio do cliente, por meio de funcionalidades a serem desenvolvidas [HED03]; *Gerência de Configuração*: o plano de *release* favorece a gerência de configuração e projeto [DUL04], incorporando aspectos como a duração de um *release*; *Planos de Projeto*: fornece um plano para o desenvolvimento do software [SMI03].

- Metáfora: *Processos Organizacionais de Clientes*: uma metáfora pode descrever processos organizacionais do cliente, visando ampliar a compreensão sobre os requisitos obtidos [KOK08]; *Domínio da Aplicação*: permite ampliar o conhecimento sobre o domínio por fornecer uma visão compartilhada sobre determinada situação [HED03]; *Arquitetura do Sistema*: conforme [HED03], [SMI03] e [DUL04], uma metáfora amplia a compreensão da arquitetura, explicando a essência de como o sistema funciona; *Projeto (design) de Software*: para [SMI03] e [DUL04], a metáfora relaciona-se com o projeto do sistema, demonstrando como determinada funcionalidade pode ser implementada.

- Descrição Arquitetural: *Codificação*: uma descrição arquitetural bem formulada pode fornecer uma visão em alto nível sobre a codificação do sistema [HED03]; *Documentação de Software*: segundo [HED03] e [ABR04], a descrição arquitetural é fundamental na documentação do software, incluindo detalhes técnicos; *Arquitetura do Sistema*: a descrição arquitetural provê

uma descrição sobre a arquitetura do sistema [ABR04]; *Projeto (design) de software*: pode demonstrar quais soluções foram empregadas no projeto do software [ABR04].

- Código / Comentários no Código: *Codificação*: de acordo com [HED03], [SMI03] e [ABR04], o código fonte refere-se a codificação/programação do sistema; *Documentação de Software*: um código bem estruturado, aliado comentários coerentes, é uma documentação fundamental sobre o sistema em metodologias ágeis [SMI03]; *Implementação da aplicação*: o código é a representação explícita de como a aplicação foi implementada [SMI03].

Essas associações são transformadas em uma ontologia, que contém os tipos de conhecimento presentes no desenvolvimento e manutenção de sistema associados aos artefatos produzidos nas metodologias XP e *Scrum*. Por meio dessa ontologia, os elementos dos artefatos produzidos, como as estórias *US01 – Manter Cadastro de Pacientes* e *US03 – Agendar Consultas*, também estarão previamente relacionados aos tipos de conhecimento, como ilustra o cenário da Figura 5.10.

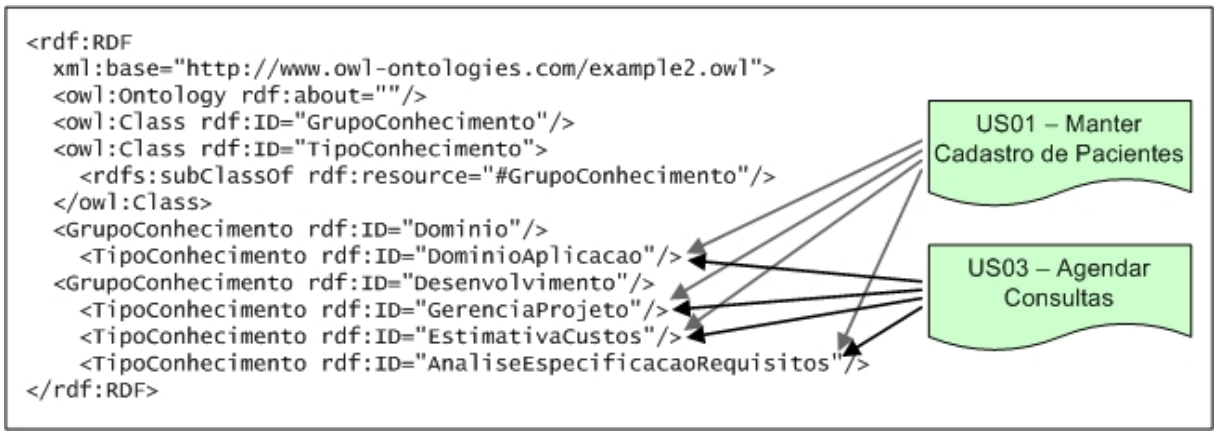


Figura 5.10: Cenário de associação de estórias com tipos de conhecimento

Para o cenário da Figura 5.10, é possível realizar consultas e recuperar relacionamentos entre os artefatos associados aos tipos de conhecimento. A seguir será apresentada a estrutura ontológica que contém a associação entre artefatos e tipos de conhecimento e também possibilita a associação entre artefatos e conceitos do domínio.

5.6 Estrutura ontológica para a representação de conhecimento

Considerando as seções anteriores, nota-se que por meio de ontologias é possível a representação de conceitos do domínio do sistema e de tipos de conhecimento presentes no desenvolvimento. Como relatado, a associação de artefatos das metodologias ágeis a estas representações favorecem a rastreabilidade por conceitos e por tipo de conhecimento. Contudo, torna-se necessária uma estrutura ontológica que viabilize a associação dos artefatos aos conceitos do domínio e aos tipos de conhecimento.

A estrutura ontológica proposta neste trabalho utilizará como referência a estrutura apresentada por [NOL07a] para o Processo Unificado. Ela tem como ponto de partida um conceito denominado *AgileKnOWLedgeConcept*, que é similar ao conceito *ONTraceBasis* [NOL07a]. Sob este conceito são propostos outros três conceitos que proveem os mecanismos para que as associações sejam de fato implementadas, sendo eles:

- *Elements*: representa uma ocorrência (um elemento) de um determinado tipo de artefato, tal como uma Estória, Tarefa ou Iteração. Este conceito se assemelha ao conceito *Artifacts* proposto em [NOL07a], e como tal também é caracterizado por duas propriedades do tipo *objectProperty*:

- *recoverDomain*: que associa uma ocorrência de um tipo de artefato ágil, definida na classe *Elements*, aos conceitos do domínio do sistema, definidos por classes OWL.

- *recoverArtifact*: que associa uma ocorrência de um tipo de artefato ágil, definida na classe *Elements*, ao seu respectivo tipo de artefato, definido pelo conceito *Artifact*.

- *Artifact*: organiza hierarquicamente os artefatos, conforme metodologia ágil a que pertencem (subclasse *Methodology*), categoria (subclasse *Category*), propósito (subclasse *Purpose*) e tipo (subclasse *Kind*), encontrados por meio do levantamento de artefatos ágeis realizado (ver Apêndice G). É caracterizado por uma propriedade do tipo *objectProperty*:

- *recoverKnowledge*: que associa um tipo de artefato ágil, definido pela classe *Artifact*, aos tipos de conhecimento, definidos pela classe *Knowledge*.

- *Knowledge*: organiza hierarquicamente o conhecimento referente ao desenvolvimento de software, encontrado por meio do levantamento realizado, de acordo com o grupo de conhecimento (subclasse *Group*) e o tipo de conhecimento (subclasse *Type*).

Para exemplificar a Figura 5.11 apresenta a estrutura ontológica proposta, com uma instanciação, utilizando a notação de representação de ontologias definida pela [OMG09b].

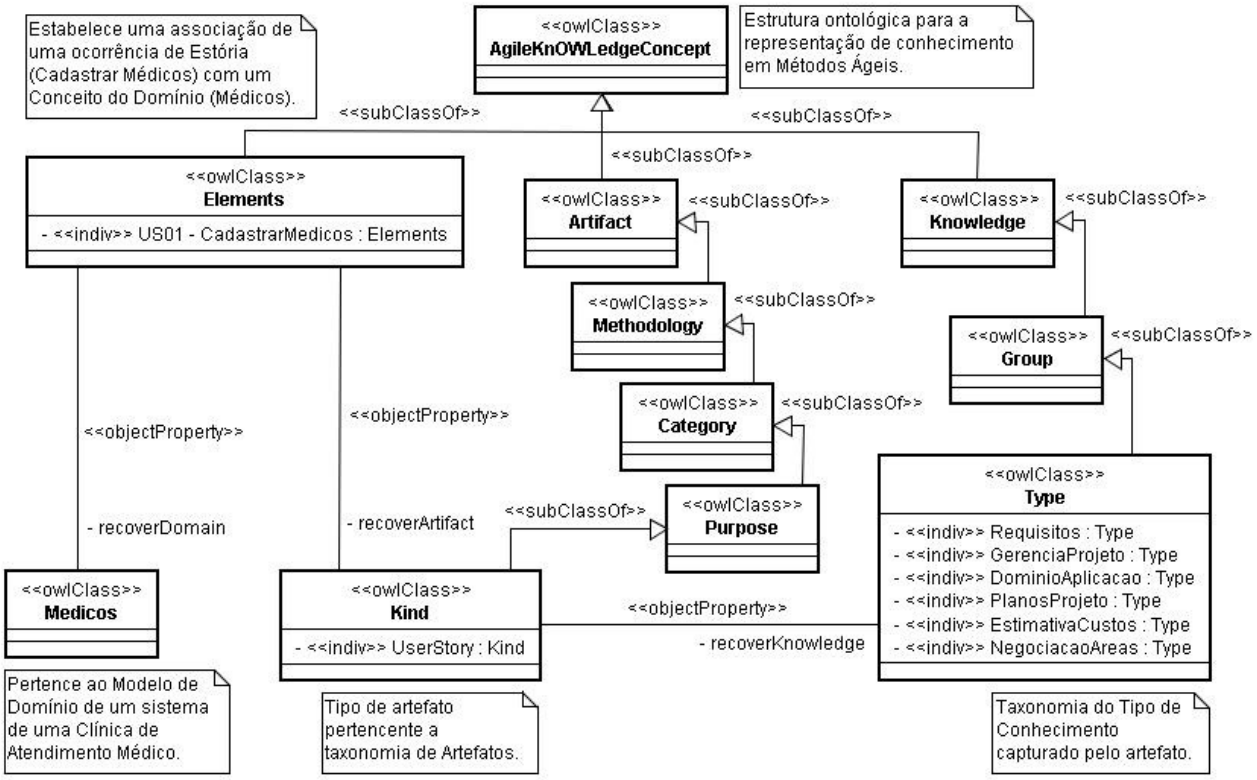


Figura 5.11: Estrutura ontológica para representação de conhecimento em métodos ágeis
Fonte: Adaptado de [NOL07a]

Na Figura 5.11, a estrutura ontológica instanciada ilustra uma associação entre o elemento *US01 – CadastrarMedicos*, que é um artefato do tipo *Estória*, como recuperado pela propriedade *recoverArtifact*. Este elemento encontra-se associado ao conceito *Medicos* pela propriedade *recoverDomain*. O conceito *Medicos* provém do domínio de um sistema de Clínica de Atendimento Médico. Por meio da propriedade *recoverKnowledge* é possível identificar que o artefato do tipo *Estória* é associado aos tipos de conhecimento *Requisitos*, *Gerência de Projeto*, *Domínio da Aplicação*, *Planos de Projeto*, *Estimativa de Custos* e *Negociação entre Áreas*. Portanto, o elemento *US01 – CadastrarMedicos* também está associado a estes conhecimentos.

A estrutura apresentada permite a recuperação de elos de rastreabilidade entre elementos de artefatos e conceitos do domínio e entre elementos de artefatos e tipos de conhecimento. Também é possível recuperar, a partir de um conceito de domínio, quais artefatos encontram-se associados a

ele, bem como recuperar a partir de um tipo de conhecimento, quais são os artefatos que o representa.

A possibilidade de associar artefatos, conceitos do domínio e tipos de conhecimento representa um primeiro passo para concepção de um processo de desenvolvimento de software voltado para o conhecimento e é particularmente útil nas seguintes situações:

- Durante a manutenção de determinado sistema, o desenvolvedor ao consultar um elemento do tipo estória do usuário (ex. *US01 – CadastrarMedicos*) poderá obter como resultado o tipo de conhecimento ao qual este artefato está associado (conhecimento sobre domínio da aplicação, requisitos, entre outros). Este tipo de conhecimento já estará mapeado na ontologia de conhecimento, sendo possível também verificar quais outros artefatos se encontram relacionados a ele (*Tarefa*). Por outro lado, o elemento do tipo estória também estará associado a um conceito do domínio da aplicação (*Medicos*). Desse modo, tem-se um cenário em que é possível verificar o artefato, os tipos de conhecimento e os conceitos do domínio associados a ele, bem como recuperar por meio de conceitos do domínio ou por tipo de conhecimento, quais outros artefatos encontram-se associados ao artefato que será impactado. Esse conhecimento permite uma visão mais ampla sobre o tipo de conhecimento que será abordado, os artefatos impactados e os conceitos do domínio relacionados durante a manutenção.

- Ao iniciar o desenvolvimento de um novo sistema com uma metodologia ágil, considerando que a documentação deverá abordar um determinado tipo de conhecimento (ex. conhecimento sobre arquitetura do sistema) a equipe responsável poderá identificar quais artefatos deverão ser produzidos durante o desenvolvimento (*Plano de Iteração*, *Notas de Reuniões*, *Metáfora*, *Descrição Arquitetural*, entre outros) para que se represente o conhecimento desejado (ou requerido pelo cliente). Assim, tem-se uma primeira abordagem para customização do ciclo de desenvolvimento ágil orientado ao conhecimento.

Dessa forma, a proposta visa promover de maneira automática algumas diretrizes para a utilização dos artefatos pertencentes as metodologias ágeis, facilitando a compreensão sobre seu significado, representação, relação com outros artefatos, conteúdo e uso. Essas diretrizes, segundo [SMI03], são amplamente cobertas por metodologias como RUP, contudo, costumam ser deficientes em métodos ágeis como XP. Em uma proposta futura, também poderão ser envolvidos os *stakeholders*, associando-os aos elementos de artefatos gerados e aos tipos de conhecimento.

5.7 Ferramenta para representação de conhecimento

Tendo como propósito viabilizar a implementação da estrutura ontológica de representação de conhecimento proposta anteriormente, bem como analisar a sua viabilidade, constatou-se a necessidade de estender as funcionalidades das atuais ferramentas de suporte à utilização de metodologias ágeis. Para a identificação dessas ferramentas foi realizado um levantamento, que retornou como ferramentas mais populares, de código aberto (*open source*), as seguintes (o levantamento completo das ferramentas está descrito no Apêndice H):

- *Agilefant* [AGI09];

- *IceScrum* [ICE09];

- *XP Studio* [XPS09];

- *XPlanner* [XPL09];

- *XPWeb* [XPW09].

Considerando aspectos como popularidade, suporte ao desenvolvimento ágil, qualidade do código fonte, plataforma de execução e desenvolvimento, a ferramenta *XPlanner* foi escolhida como a ferramenta a ser utilizada neste trabalho.

Entretanto, considerando que o Modelo de Domínio não é um artefato característico das metodologias ágeis será necessário a utilização de uma ferramenta específica para o seu desenvolvimento e, posterior, obtenção da ontologia com os conceitos do domínio. Dessa forma, junto com o levantamento das ferramentas de suporte ao desenvolvimento ágil, também se procurou identificar uma ferramenta que facilitasse a obtenção da ontologia do domínio, como descrito a seguir.

5.7.1 Mapeamento do Modelo de Domínio para Ontologia

Com relação às ferramentas capazes de realizar o mapeamento de UML para OWL, conforme ilustrado na Figura 5.6, por meio de um levantamento foram identificadas as seguintes:

- *ArgoUML+ONTrace* [NOL07a]: adiciona às funcionalidades da ferramenta de modelagem *ArgoUML* uma estrutura para rastreabilidade ontológica.

- *IBM Integrated Ontology Development Toolkit* [IBM09]: conjunto de ferramentas da IBM para definição e manipulação de ontologias, incluindo a transformação entre modelos UML e OWL.

- *Visual Ontology Modeler* [SAN09]: ferramenta de modelagem baseada em UML que permite o desenvolvimento e o gerenciamento de ontologias para uso em aplicações colaborativas.

Em função da disponibilidade do código fonte, a ferramenta *ArgoUML+ONTrace*, apresentada por [NOL07a], foi escolhida para o trabalho. Contudo, uma alteração em suas funcionalidades fora implementada permitindo a realização apenas do mapeamento de UML para OWL, dispensando as classes referentes a rastreabilidade ontológica gerada pela ferramenta. A seguir será apresentada a funcionalidade desenvolvida para a ferramenta *XPlanner*.

5.7.2 *XPlannerKnOWLedge*

A ferramenta protótipo *XPlannerKnOWLedge* acrescenta a representação por conceitos e por tipo de conhecimento à ferramenta de desenvolvimento ágil *XPlanner*. *XPlanner* é uma ferramenta web de código aberto, desenvolvida para dar suporte a projetos que utilizem, preferencialmente, a metodologia XP. Contudo, também pode ser utilizada em projetos com *Scrum* ou outras metodologias. Em seu desenvolvimento são utilizadas as linguagens *Java* e *Java Server Pages* (JSP), os *frameworks Struts* e *Hibernate* e o sistema de gerência de banco de dados *MySQL*.

A interface inicial da ferramenta *XPlannerKnOWLedge* é apresentada na Figura 5.12. Para a interface da ferramenta são definidas três páginas JSP:

- *formImportOntology*: apresenta o formulário para importação do arquivo contendo a ontologia com os conceitos do domínio.

- *formRepresentKnowledge*: apresenta o formulário com os conceitos da ontologia do domínio que podem ser associados aos elementos de artefatos definidos no *XPlanner*.

- *formRecoverKnowledge*: apresenta o formulário que recupera o conhecimento representado por conceitos e por tipo de conhecimento.

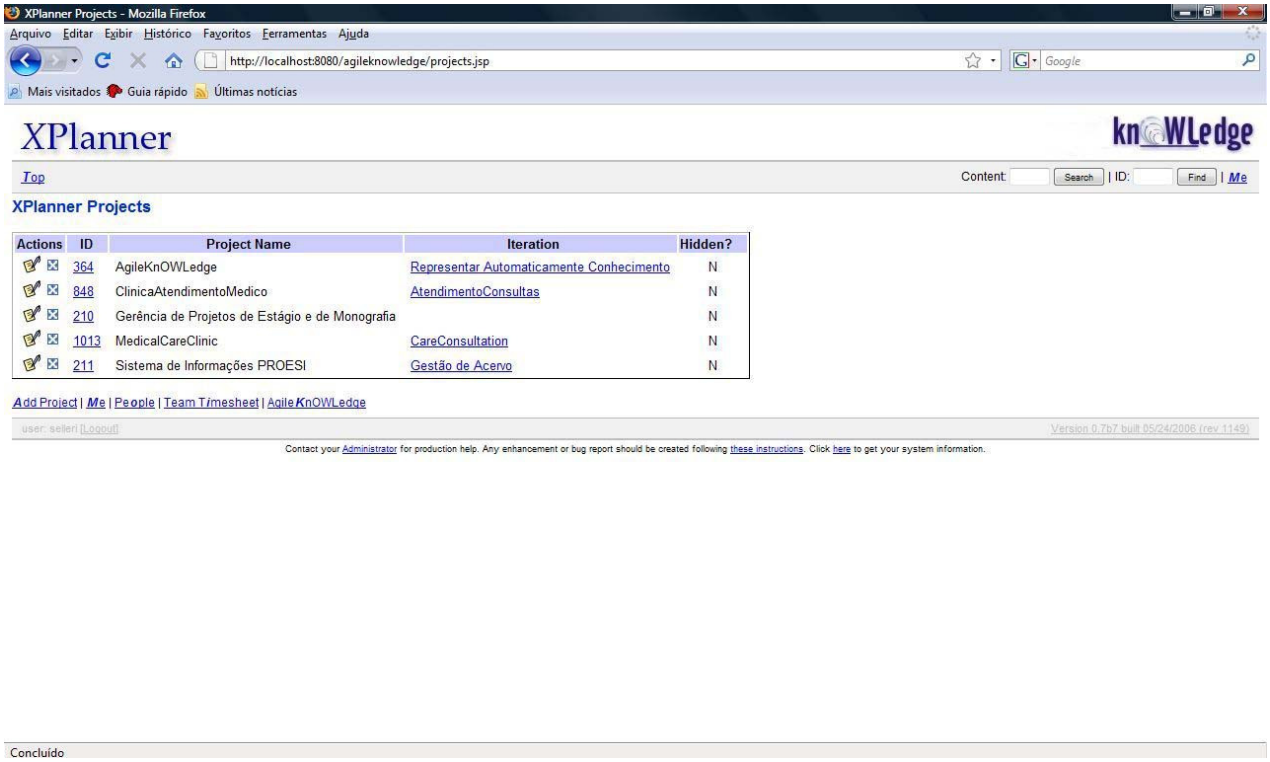


Figura 5.12: Interface da ferramenta *XPlannerKnOWLedge*

A ferramenta possui a arquitetura representada pelo diagrama de classes da Figura 5.13. A classe *ImportOntology* importa o arquivo contendo a ontologia com os conceitos do domínio. A classe *RepresentKnowledge* cria uma instância da classe *OntModel*, que se refere a *Application Programming Interface* (API) do *Jena* [JEN09], implementando em OWL a estrutura ontológica para representação de conhecimento, com os conceitos do domínio, os tipos de artefatos, os tipos de conhecimento e os elementos de artefatos oriundos do *XPlanner*.

As classes *RecoverKnowledge* e *AgileKnowledge* realizam a inferência sobre a ontologia, recuperando as associações entre elementos de artefatos, conceitos do domínio e tipos de conhecimento. Por meio da associação entre elementos e conceitos é possível recuperar relacionamentos diretos e indiretos entre elementos de artefatos. Também é possível recuperar relacionamentos diretos e indiretos por meio dos tipos de conhecimento. O modelo de inferência utilizado é o *InfModel*, pertencente a API do *Jena*.

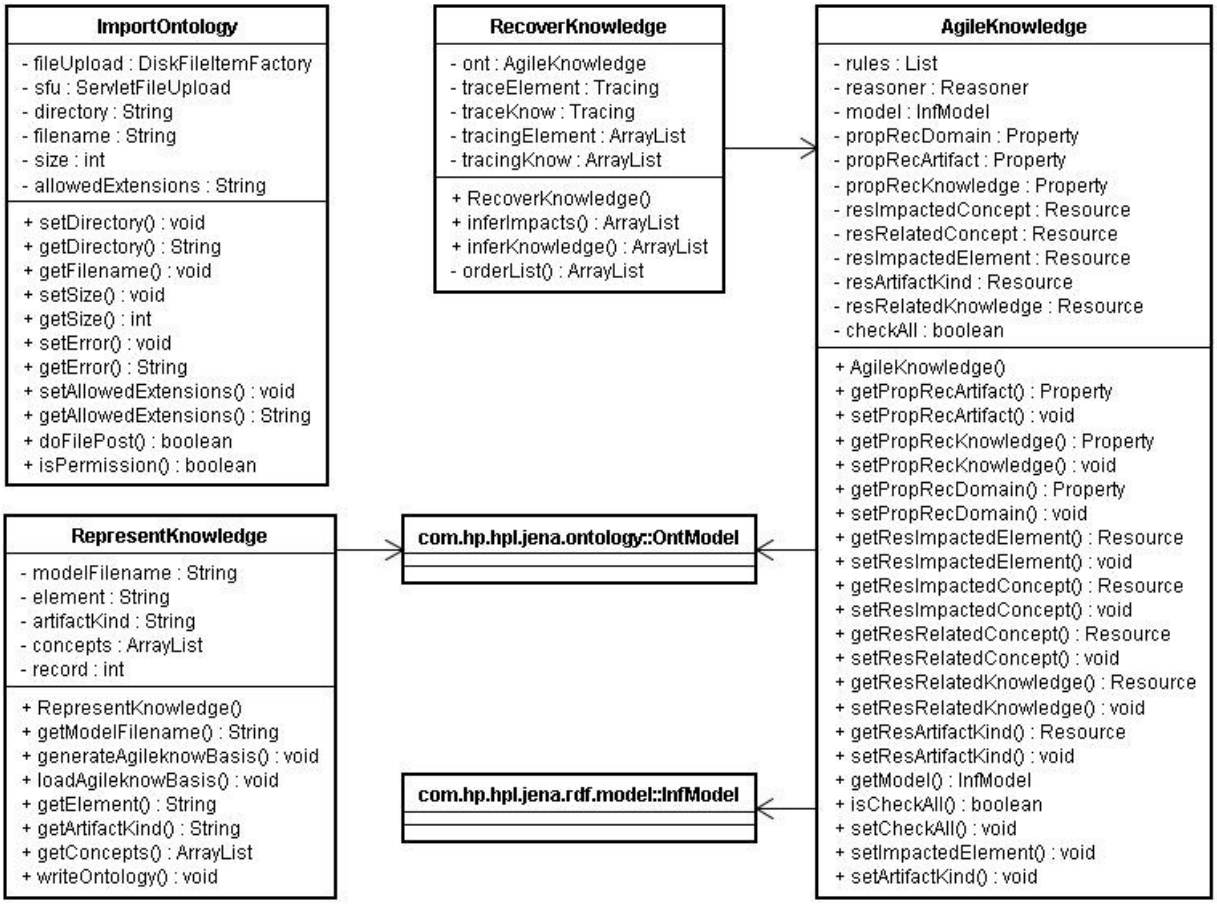


Figura 5.13: Diagrama de Classes da ferramenta *XPlannerKnOWLedge*

Para ilustrar o emprego da ferramenta protótipo apresentada, será considerado como exemplo o domínio do sistema da Clínica de Atendimento Médico e seu respectivo mapeamento para uma ontologia que represente os conceitos do domínio.

O primeiro passo para a utilização da ferramenta corresponde a importação da ontologia com os conceitos do domínio, por meio da página *formImportOntology.jsp* apresentada na Figura 5.14. Por meio dessa interface o usuário irá selecionar o arquivo que contém a ontologia com os conceitos do domínio e submetê-lo para possibilitar as associações com os elementos de artefatos produzidos no *XPlanner*.

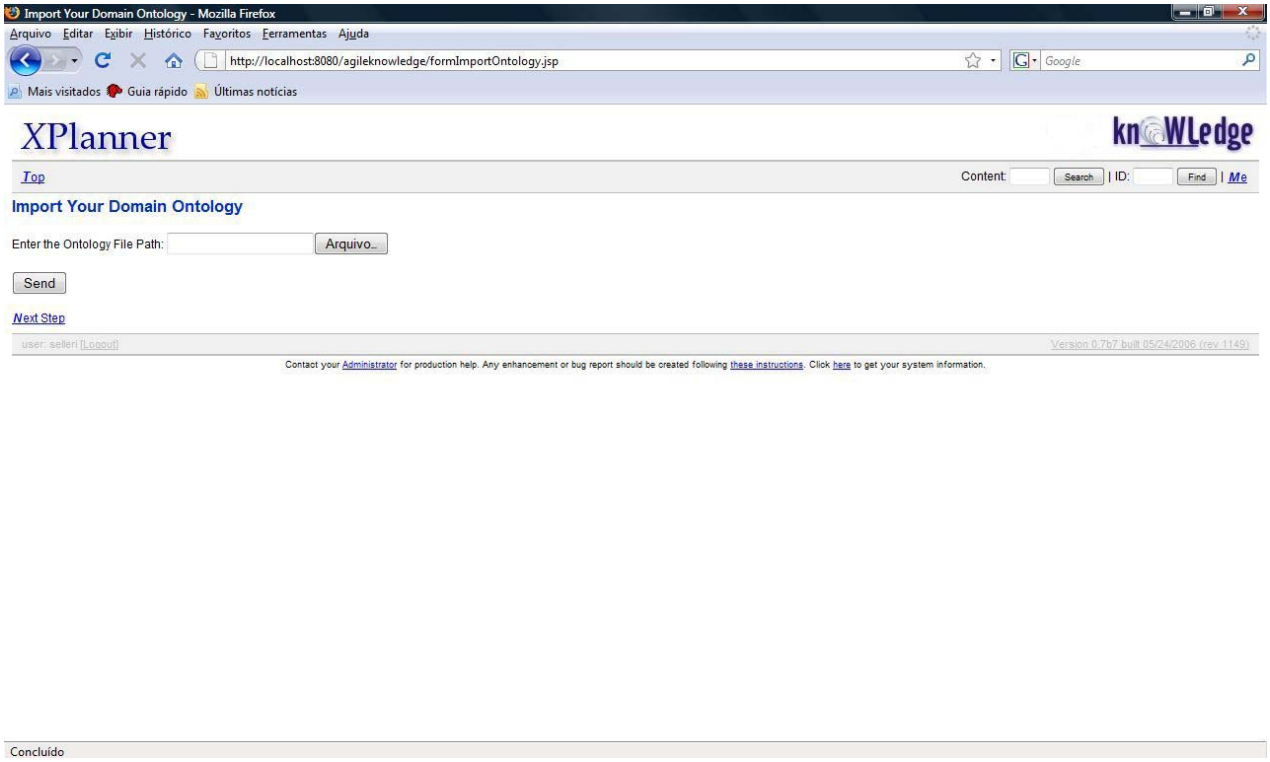


Figura 5.14: Interface para importação da ontologia do domínio

Após a importação, os elementos dos artefatos poderão ser associados aos conceitos do domínio, por meio da página *formRepresentKnowledge.jsp* apresentada na Figura 5.15.

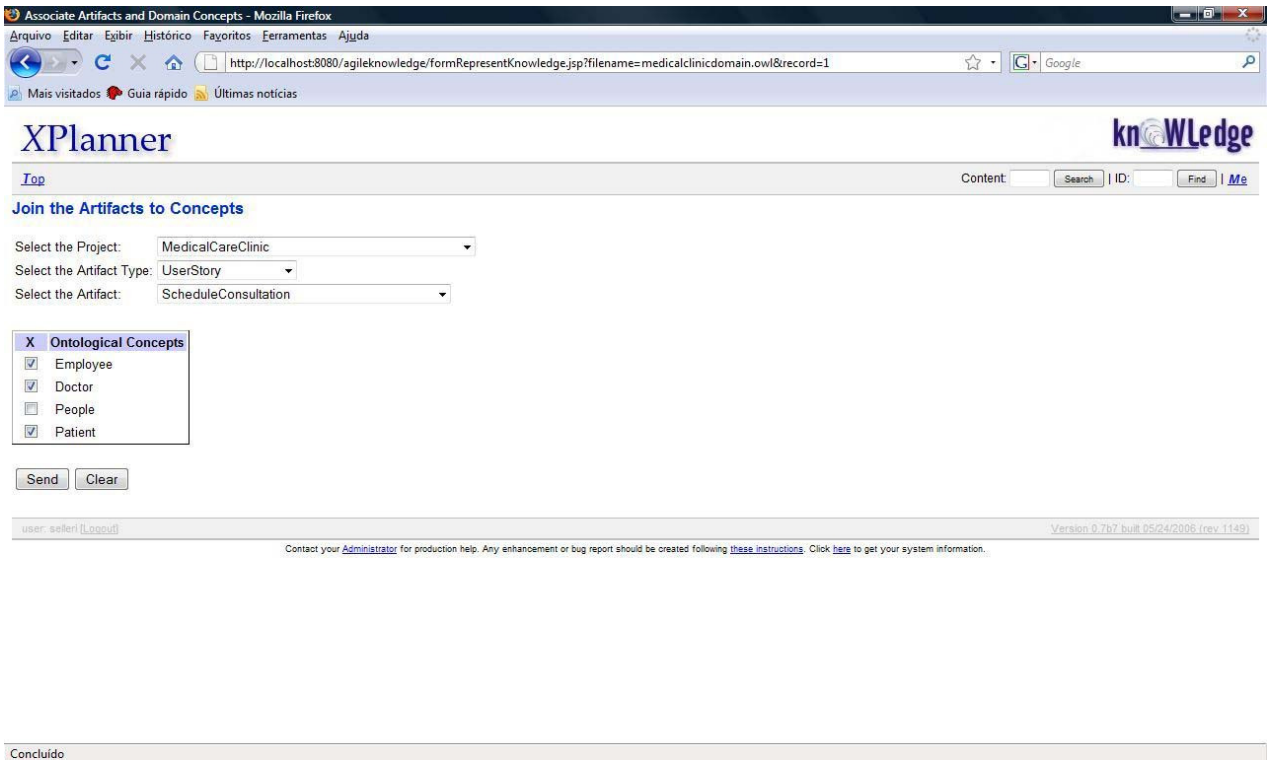


Figura 5.15: Interface para associação entre elementos de artefatos e conceitos do domínio

Os conceitos oriundos do domínio são listados, juntamente com os elementos de artefatos oriundos do banco de dados do *XPlanner*, possibilitando a definição das associações.

A Figura 5.16 apresenta a interface com o resultado da submissão da associação de um elemento, a partir da qual é possível optar pela associação de um novo elemento ou pela recuperação do conhecimento representado.

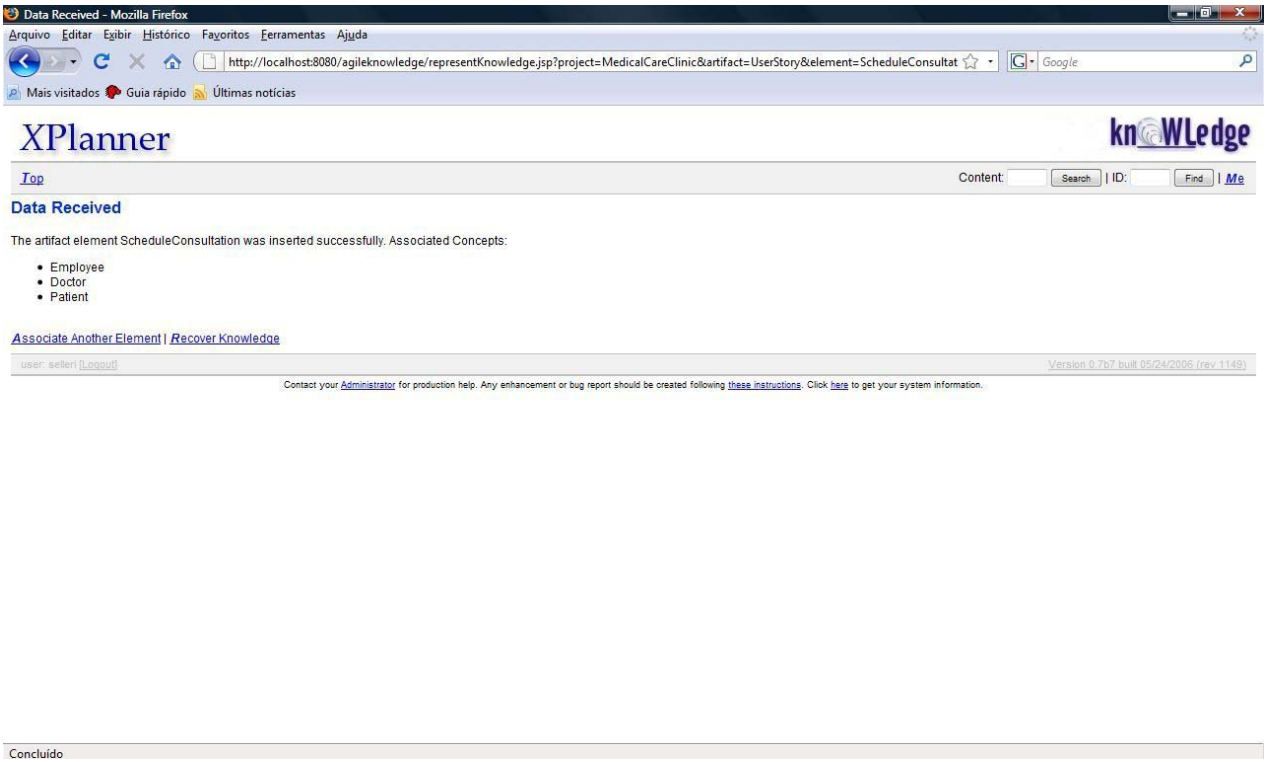


Figura 5.16: Interface informando a associação entre elemento de artefato e conceitos

A recuperação das associações, juntamente com os relacionamentos entre elementos de artefatos e tipos de conhecimento, é realizada por meio da página *formRecoverKnowledge.jsp* ilustrada na Figura 5.17. O usuário seleciona o elemento impactado e o filtro por tipo de artefato. A partir de então, são listados os relacionamentos primários, secundários e os tipos de conhecimento associados.

A recuperação também pode ser realizada a partir dos tipos de conhecimento. O usuário irá selecionar o tipo de conhecimento a ser filtrado, sendo exibida a relação de elementos de artefatos associado ao tipo em questão.

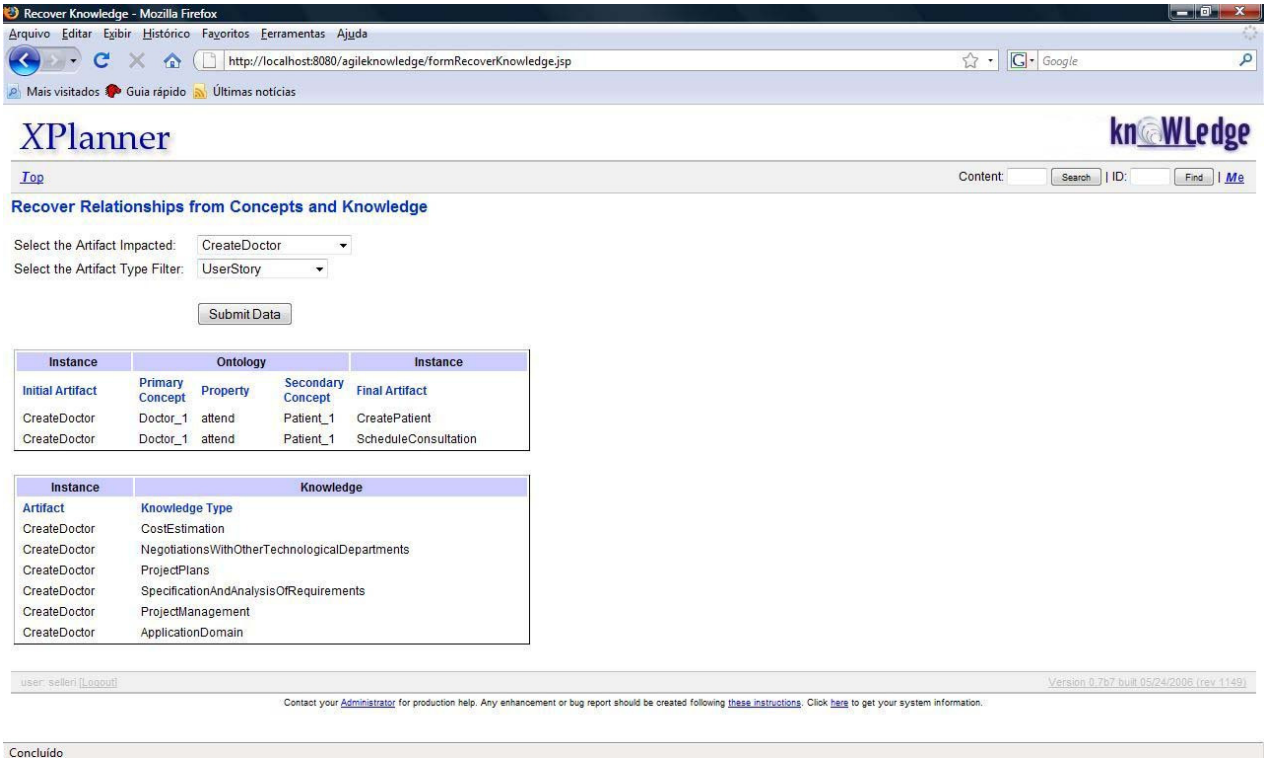


Figura 5.17: Interface para recuperação dos relacionamentos e tipos de conhecimento

5.7.3 Associação automatizada entre artefatos e conceitos

A associação entre os elementos de artefatos e os conceitos do domínio representa uma etapa fundamental da proposta de representação de conhecimento discutida neste trabalho. Contudo, o processo de associação manual, como apresentado anteriormente, selecionando o elemento de artefato e clicando nos conceitos correspondentes tende a se tornar uma tarefa exaustiva em sistemas maiores, representando um gargalo para o desenvolvimento ágil.

Considerando essa dificuldade, buscou-se alternativas para aperfeiçoar essa associação, preferencialmente, tornando-a automatizada. Uma solução encontrada foi a utilização de um *Parser* que emprega recursos de Processamento da Linguagem Natural (PLN), tendo sua implementação realizada a partir da ferramenta *The Matcher* proposta por [NOL07b] para análise da similaridade léxica e semântica entre documentos XML e ontologias definidas em OWL.

O *Parser* recebe como entrada os elementos de artefatos, definidos na ferramenta *XPlanner*, e a ontologia do domínio, contendo os conceitos referentes a realidade do sistema em questão, realizando a associação entre ambos. O algoritmo de *Stemming* [CHA03b] é utilizado para a verificação da similaridade léxica, avaliando a sequência de caractere de cada palavra a partir da

remoção de seus prefixos e sufixos, que a reduz ao seu *stem*. Para a verificação da similaridade semântica é utilizada a base de dados da *WordNet* [PRI06], que relaciona substantivos, verbos, adjetivos e advérbios em um conjunto de sinônimos cognitivos sob a perspectiva léxica e semântica, representando a maior base léxica da língua inglesa. Para desenvolvimento da solução foi utilizada a linguagem de programação *Java* e as APIs:

- *Stemming* [STE09]: responsável pela avaliação léxica, por meio da recuperação dos *stems* das palavras.

- *WordNet* [WOR09]: responsável pela avaliação semântica, por meio da recuperação dos sinônimos.

- *JWNL* [JWN09]: responsável pelo acesso ao dicionário relacional definido pela *WordNet*.

- *Jena* [JEN09]: responsável pela manipulação dos documentos OWL.

Para exemplificar a utilização do *Parser*, será considerada como entrada o elemento de artefato *CreateDoctor*, que é do tipo *User Story* e está disponível no banco de dados do *XPlanner*, e os conceitos *People*, *Doctor*, *Patient* e *Employee*, que estão descritos em um arquivo OWL e pertencem ao domínio do sistema da Clínica de Atendimento Médico. Ao processar os dados de entrada, o *Parser* retornará como resultado a associação do elemento *CreateDoctor* com o conceito *Doctor*, como ilustra a Figura 5.18.

Esse resultado corresponde a inclusão de um indivíduo *CreateDoctor* na classe ontológica *Elements* e a sua associação com a classe ontológica *Doctor*, por meio da propriedade *RecoverDomain*, de acordo com a estrutura ontológica proposta. A associação por meio do *Parser* pode ser estendida aos demais artefatos presentes na base de dados do *XPlanner*, considerando um determinado projeto.

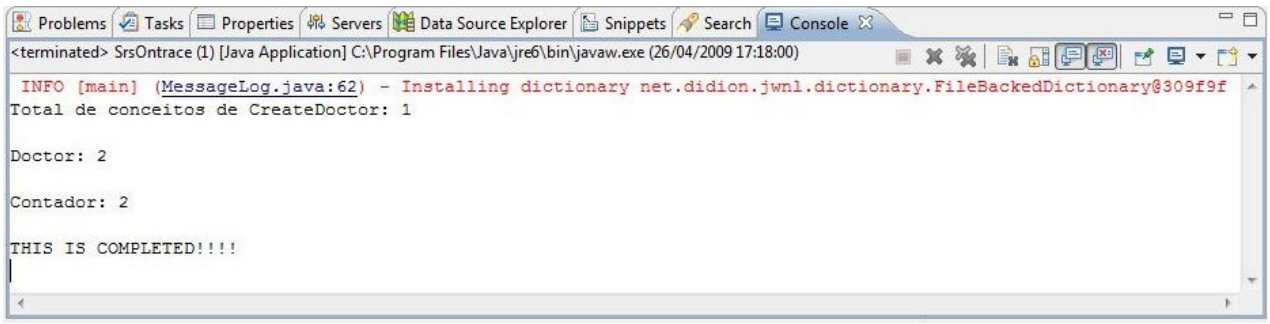


Figura 5.18: Associação entre elemento de artefato e conceito via *Parser*

Os resultados obtidos pelo *Parser* não dispensam uma checagem por parte do Engenheiro do Conhecimento, a fim de constatar a relevância das associações. Contudo, o fato de tornar automatizada a associação entre elementos de artefatos e conceitos do domínio adiciona uma significativa agilidade ao processo de representação, contribuindo para a viabilidade da proposta aqui apresentada.

5.8 Considerações

Neste capítulo foi apresentada uma proposta de representação de conhecimento em metodologias ágeis, considerando os conceitos identificados na fundamentação teórica e as contribuições de trabalhos relacionados à área de pesquisa.

Esta proposta de representação de conhecimento tem como foco os artefatos produzidos durante o desenvolvimento ágil e encontra-se ancorada em dois princípios: a representação por conceitos e a representação por tipo de conhecimento. Na representação por conceitos os artefatos ágeis são associados aos conceitos pertencentes ao domínio do sistema desenvolvido. Por meio dessa associação é possível mapear relações entre artefatos produzidos ao longo do desenvolvimento. Com a representação por tipo de conhecimento, os artefatos são associados aos tipos de conhecimento mais comuns de ocorrerem no desenvolvimento de software. Para tal, foi realizado um levantamento para identificar esses tipos de conhecimento e os artefatos que os representam. Com essa associação, pode-se recuperar artefatos que exprimem um determinado conhecimento, bem como verificar quais conhecimentos são expressos por cada artefato produzido.

Com o propósito de tornar utilizável a proposta apresentada foi desenvolvida uma ferramenta protótipo, que implementa a representação de conhecimento em metodologias ágeis por conceitos e por tipos de conhecimento, utilizando uma ontologia como base para a representação. Por padrão a ferramenta já possui as associações entre artefatos e tipos de conhecimento. Pretende-se que a ferramenta também permita a edição destas associações, com o objetivo de adequá-los a realidade de projetos específicos.

Quanto às associações entre artefatos e conceitos do domínio, a ferramenta é capaz de realizar o mapeamento automaticamente a partir da entrada de uma ontologia contendo os conceitos e acesso a uma base de dados onde se encontram os artefatos desenvolvidos. Esse mapeamento automático visa agilizar o processo, evitando-se o esforço em definir as associações manualmente.

Entretanto, as associações também podem ser definidas manualmente nos casos em que a equipe julgar necessárias associações específicas, não capturadas pela ferramenta.

No próximo capítulo, um experimento é relatado com o objetivo de avaliar a proposta de representação de conhecimento, em sua dimensão por conceitos. A avaliação também engloba a ferramenta protótipo desenvolvida, e sua precisão na associação automática entre artefatos e conceitos do domínio, comparando-a com uma indexação manual, utilizando uma matriz de rastreabilidade para os artefatos.

6 AVALIAÇÃO DA PROPOSTA

Neste capítulo, pretende-se analisar a viabilidade da proposta de representação de conhecimento, abordada anteriormente, por meio da realização de um experimento. De acordo com [PFL98], um *experimento* consiste em um tipo de estudo controlado, geralmente executado em laboratórios, no qual os valores de variáveis independentes (entradas) são manipulados para se observar as mudanças nos valores de variáveis dependentes (saídas), com posterior análise, interpretação, apresentação e empacotamento dos resultados.

Numa perspectiva inicial, o experimento tem como objetivo avaliar a ferramenta quanto à viabilidade da representação de conhecimento em metodologias ágeis. Numa perspectiva secundária, o experimento também buscará avaliar o processo de utilização da proposta. Em um aspecto mais amplo o experimento irá investigar o método proposto para representação de conhecimento, comparando as variáveis *menor tempo gasto* e *maior precisão da compreensão* com e sem a proposta de representação apresentada. De acordo com os requisitos definidos pela Engenharia de Software Experimental, haverá um controle sobre as variáveis de entrada a fim de constatar os resultados obtidos nas variáveis de saída.

Por se tratar da avaliação de um processo, considerando o envolvimento com o fator humano, justifica-se a utilização do experimento como abordagem de avaliação da proposta. Entretanto, o experimento possibilita uma avaliação essencialmente quantitativa [WOH00]. Dessa forma, também foi realizada, uma pesquisa de opinião integrada ao experimento, como abordagem para avaliação qualitativa da proposta. Conforme [PFL98], uma *pesquisa de opinião (survey)* consiste em um estudo que objetiva documentar relações e resultados de uma situação, registrando-se as informações e comparando-as com informações semelhantes.

6.1 Estudo experimental realizado

A realização de um experimento requer preparo, com a finalidade de conduzir e analisar de forma correta o objeto de estudo. O controle sobre as variáveis independentes conduz a conclusões relevantes ao experimento a partir das variáveis dependentes, considerando que um experimento provê uma resposta específica para determinada configuração. Neste aspecto, é necessário o controle dos participantes do experimento em cada abordagem considerada, sendo ela com ou sem a proposta de representação de conhecimento.

Como guia para a realização do experimento foram utilizadas as propostas de [WOH00], [TRA02] e [NOL07a] e para a avaliação as contribuições de [ARA06] e [BAR04]. De acordo com [TRA02], o processo de experimentação é composto pelas seguintes fases: definição; planejamento; execução; análise e interpretação; apresentação e empacotamento. Nas seções a seguir, cada fase é apresentada em maior detalhe.

6.1.1 Definição

A fase de definição determina a fundamentação do experimento, constando o motivo pelo qual o mesmo é conduzido. São estabelecidos os objetivos, o escopo, os objetos e os grupos envolvidos no experimento.

O estudo experimental realizado utilizou para a fase de definição a abordagem *Goal Question Metric* (GQM), proposta por [BAS94]. Esta abordagem assume que para realizar uma medição significativa, deve-se primeiramente especificar os objetivos para a medição e seu projeto, em seguida, mapear esses objetivos para os dados que se destinam a defini-los operacionalmente e, por fim, fornecer um quadro para a interpretação dos dados recuperados por meio de métricas que corresponde aos objetivos definidos [BAS94]. Desse modo, foram definidos:

- Objetivo do estudo: *comparar* no *Extreme Programming* a compreensão (recuperação) da documentação com a representação de conhecimento com a compreensão (recuperação) sem a representação, *com o propósito* de caracterizar o tempo gasto no uso e os elementos recuperados por cada abordagem, *com foco* no esforço e na precisão, *sob o ponto de vista* do arquiteto de software, *no contexto* de manutenção de um sistema de informação desenvolvido por estudantes (*toy example*) no domínio de um gerenciador de projetos de estágio e monografia de uma universidade.

- Objetivo da medição: caracterizar em uma manutenção de um sistema:

- Qual o *esforço* (medido em minutos por participante) necessário para indexação (representação) dos artefatos de documentação com e sem o uso da representação de conhecimento;

- Qual a *precisão* definida por meio da corretude e completude dos artefatos recuperados com e sem o uso da representação de conhecimento, quanto aos relacionamentos no sistema.

- Métricas: as métricas relacionadas ao estudo foram:

- Métrica 1: tempo gasto em minutos por cada participante para indexar (representar) a documentação em cada abordagem;

- Métrica 2: precisão na compreensão (recuperação) da documentação em cada abordagem para identificar os artefatos a serem impactados. Por precisão entende-se a razão entre a quantidade de elementos do conjunto correto de artefatos recuperados e a quantidade de elementos do conjunto total de artefatos relacionados. O conjunto total é definido por um especialista com base no estudo do sistema.

6.1.2 Planejamento

Para [WOH00], a fase de planejamento refere-se a fase que determina como o experimento é conduzido. O experimento deve ser bem planejado para que se estabeleça o controle sobre o mesmo, evitando distorções nos resultados. São definidos os seguintes passos para a fase de planejamento: seleção do contexto; formulação das hipóteses; seleção das variáveis; seleção dos indivíduos; projeto; instrumentação; e avaliação da validade.

6.1.2.1 Seleção do contexto

Foi selecionado o contexto de uma universidade para a condução do experimento. Em função dos recursos disponíveis, não foi possível a seleção de um ambiente relacionado a indústria de software. Foram consideradas as seguintes dimensões:

- Processo: abordagem *In-vitro*, com os participantes executando o experimento em um ambiente controlado (laboratório de informática). Por não ser aplicado à realidade de uma organização desenvolvedora de software industrial, o processo é considerado *off-line*.

- Participantes: os participantes compreendem estudantes do último período da graduação em Ciência da Computação da UNEMAT – Barra do Bugres.

- Realidade: problema de sala de aula (*toy example*) e consiste em um sistema modelado e desenvolvido por alunos da graduação, durante uma disciplina do Curso de Ciência da

Computação da UNEMAT, no domínio de um gerenciador de projetos de estágio e monografia desenvolvidos em uma universidade.

- Generalidade: o experimento é específico com validade no escopo deste estudo.

6.1.2.2 Seleção das variáveis

A escolha das variáveis deve ser realizada criteriosamente, com base no conhecimento sobre o domínio da situação. As *variáveis independentes* (entrada) além de serem controladas, devem influenciar as *variáveis dependentes* (saída). As seguintes variáveis foram assumidas:

- Variáveis independentes (entrada):

    - Experiência do time de manutenção;

    - Abordagem para representação de conhecimento.

- Variáveis dependentes (saída):

    - Esforço para indexação (representação) da documentação;

    - Precisão para compreensão (recuperação) da documentação.

6.1.2.3 Formulação das hipóteses

Para a definição formal do experimento é necessário a definição de uma hipótese fundamental, denominada *Hipótese Nula* (H_0), e de uma ou mais *Hipóteses Alternativas* (H_1, H_2, ..., H_n). A Hipótese Nula representa a não derivação dos objetivos do experimento. As Hipóteses Alternativas são as hipóteses em favor das quais a Hipótese Nula pode ser rejeitada. Foram definidas as seguintes hipóteses de acordo com cada uma das métricas identificadas:

1. **Hipóteses relacionadas à variável *esforço***:

    a) **Medidas**: o esforço representa o tempo gasto em minutos com a definição das associações entre os elementos de artefatos em cada abordagem, compreendendo a diferença entre o tempo final e o tempo inicial de cada abordagem, sendo:

i. Δt_{crep}: variação do tempo gasto em minutos para indexação com a abordagem de representação de conhecimento.

ii. Δt_{srep}: variação do tempo gasto em minutos para indexação sem a abordagem de representação de conhecimento.

b) **Hipótese Nula, H_0**: $\Delta t_{crep} = \Delta t_{srep}$: o esforço gasto para a indexação (representação) da documentação utilizando a abordagem de representação de conhecimento é igual ao esforço para a indexação (representação) sem a representação.

c) **Hipótese Alternativa, H_1**: $\Delta t_{crep} > \Delta t_{srep}$: o esforço para a indexação da documentação com a abordagem de representação de conhecimento é maior do que o esforço para a indexação sem a representação.

d) **Hipótese Alternativa, H_2**: $\Delta t_{srep} > \Delta t_{crep}$: o esforço para a indexação da documentação sem a abordagem de representação de conhecimento é maior do que o esforço para a indexação com a representação.

2. **Hipóteses relacionadas à variável *precisão***:

a) **Medidas**: a precisão corresponde a razão entre o conjunto correto de artefatos recuperados e o conjunto total, retornada em cada abordagem, sendo:

i. P_{crep}: precisão associada a recuperação com a abordagem de representação de conhecimento.

ii. P_{srep}: precisão associada a recuperação sem a abordagem de representação de conhecimento.

b) **Hipótese Nula, H_0**: $P_{crep} = P_{srep}$: a precisão da compreensão (recuperação) da documentação utilizando a abordagem de representação de conhecimento é igual à precisão da compreensão sem a representação

c) **Hipótese Alternativa, H_1**: $P_{crep} > P_{srep}$: a precisão da compreensão (recuperação) da documentação com a abordagem de representação de conhecimento é maior do que a precisão da compreensão sem a representação.

d) **Hipótese Alternativa, H_2**: $P_{srep} > P_{crep}$: a precisão da compreensão (recuperação) da documentação sem a abordagem de representação de conhecimento é maior do que a precisão da compreensão com a representação.

6.1.2.4 Seleção dos indivíduos

Define o conjunto de participantes do experimento. Deve-se selecionar uma população representativa, que garanta a geração de resultados relevantes. Para a seleção dos indivíduos foram consideradas:

- **População**: constituída por um total de dezesseis estudantes do último período da graduação em Ciência da Computação da UNEMAT, dos quais oito estudantes concluem o curso no semestre 2009/1 e oito estudantes concluem o curso em semestres futuros devido a pendências em algumas disciplinas.

- **Amostragem**: considerando a escolha de pessoas com maior disponibilidade para realização do experimento, foi utilizada uma *amostragem por conveniência*. Foram alocados estudantes que concluem o curso no semestre atual com estudantes que concluem o curso posteriormente, em uma *amostragem por quota*, pressupondo que a experiência desses grupos seja diferenciada. Na Seção 6.3 será apresentada uma avaliação qualitativa sobre a experiência dos participantes.

6.1.2.5 Projeto do experimento

Um experimento deve ser cuidadosamente planejado e projetado. Um projeto apropriado também possibilita a replicação do experimento posteriormente. Considerando esse aspecto, foram atendidos os seguintes *princípios genéricos de projeto* para o experimento:

- **Aleatoriedade**: a definição de quais participantes irão executar cada abordagem (com ou sem representação de conhecimento) foi aleatória.

- **Obstrução**: divergência quanto a experiência profissional dos participantes. Para minimizar o efeito da experiência utilizou-se o critério de quota.

- **Balanceamento**: cada abordagem será executada pelo mesmo número de participantes, distribuídos aleatoriamente. Dessa forma, para cada abordagem foram definidos quatro estudantes que concluem o curso no semestre atual e quatro que concluem o curso futuramente.

- **Técnica**: o experimento procura investigar se a abordagem com representação de conhecimento possui o mesmo esforço e precisão que a abordagem sem representação de conhecimento. Para tal, será empregada uma técnica denominada **um fator** (a abordagem que é considerada) **com dois tratamentos** (a abordagem com e sem representação de conhecimento). As seguintes notações serão utilizadas:

  - μ_{crep} Abordagem com representação de conhecimento.

  - μ_{srep} Abordagem sem representação de conhecimento.

- **Projeto**: optou-se por conduzir um *projeto aleatório*, no qual cada participante executará apenas uma abordagem definida aleatoriamente. A Tabela 6.1 representa a distribuição do fator sobre os dois tratamentos, com a ordem definida de maneira aleatória e balanceada entre os fatores.

Tabela 6.1: Distribuição dos participantes por abordagem

Participante	μ_{crep}	μ_{srep}
1		X
2		X
3	X	
4		X
5		X
6	X	
7	X	
8		X
9	X	
10	X	
11	X	
12	X	
13		X
14		X
15	X	
16		X

- **Teste das hipóteses**: de acordo com a natureza da amostra, serão utilizados os seguintes tipos de testes:

  - Teste paramétrico de significância: *Teste T*;

  - Teste não-paramétrico de significância: *Mann-Whitney*.

  - Para definição do teste a ser aplicado será considerada:

    - Análise da normalidade: teste de *Shapiro-Wilk*;

    - Análise da variância dos dados: teste de *Levene*.

6.1.2.6 Instrumentação

A instrumentação estabelece os recursos para realização do experimento e sua posterior análise. Foram utilizados os seguintes instrumentos:

- Objetos: foi disponibilizado o acesso aos seguintes artefatos pertencentes a um projeto de sistema de informação: Plano de Iteração, Estórias, Tarefas, Notas e Modelo de Domínio. Para a indexação (representação) da documentação foi fornecido o seguinte aporte ferramental: uma ferramenta de gerência de projeto (*XPlannerKnOWLedge*) e uma ferramenta CASE (*ArgoUML+ONTrace*) para a abordagem com representação de conhecimento; uma ferramenta de gerência de projeto (*XPlanner*) e uma matriz de rastreabilidade, elaborada no MS Excel, para a abordagem sem representação de conhecimento.

- Guias: os participantes receberam um treinamento, apresentando os principais conceitos sobre metodologias ágeis, com enfoque na metodologia XP, o contexto e a motivação do experimento, e as duas abordagens de indexação/recuperação. Adicionalmente, foi disponibilizado um tutorial com as orientações para o experimento.

- Métricas: os dados foram recuperados por meio de formulários preenchidos pelos participantes, com base na ferramenta *XPlannerKnOWLedge* e na matriz elaborada no MS Excel.

6.1.2.7 Análise da validade

A preocupação com a análise da validade dos resultados do experimento deve estar presente durante o planejamento. Foram considerados quatro tipos de validação, conforme sugerido pela literatura:

- Validade interna: tendo como critérios:

- Histórico: data de realização conveniente aos participantes.

- Maturação: utilização de técnica de motivação aos participantes.

- Seleção dos grupos: nivelamento do conhecimento dos participantes por meio de um treinamento. A execução das atividades do experimento foi individual.

- Difusão: evitou-se a comunicação entre os participantes durante o experimento.

- Validade externa: procurou-se selecionar participantes que apresentassem conhecimento prévio sobre processo de desenvolvimento de software e modelagem de sistemas.

- Validade de construção: os seguintes aspectos foram avaliados:

- Inadequada explicação pré-operacional: explicação operacional do experimento, com ênfase na forma como foram representadas e recuperadas as associações entre os artefatos.

- Adivinhação de hipóteses: possível interação do participante com o experimento, propondo novas hipóteses e exercitando a criatividade. Procurou-se manter o foco no estudo planejado.

- Expectativas do condutor do experimento: possível influência do condutor sobre as variáveis envolvidas e sobre o material elaborado. Avaliação do material também foi realizada por outro responsável.

- Validade da conclusão: com avaliação das seguintes perspectivas:

- Manipulação dos dados: possível variação nos dados pelo pesquisador, que os manipulou durante o processo.

- Confiabilidade das medidas: medidas subjetivas podem ser influenciadas pelo pesquisador. Desse modo foram definidas medidas objetivas para o experimento.

- Confiabilidade na implementação dos tratamentos: diferentes participantes podem implementar distintamente os processos do experimento. Não se pode interferir no caráter subjetivo de indexação e recuperação dos artefatos e, possivelmente, alguns participantes definiram associações distintas.

- Configurações do ambiente do experimento: interferências externas ao ambiente podem exercer influência sobre o experimento. O experimento foi realizado em um laboratório, sem interação externa.

- Heterogeneidade aleatória dos participantes: as diferentes experiências que cada participante possui podem ter exercido influência sobre a variação dos resultados.

6.1.3 Execução

Durante a execução o experimento entra em sua fase operacional, com os participantes desempenhando as atividades propostas, possibilitando a obtenção das métricas definidas. A validade dos resultados do experimento está associada ao comprometimento dos indivíduos com as atividades propostas, devendo estes executá-las com responsabilidade. Os seguintes passos compreenderam a execução do experimento:

- Preparação: durante os preparativos para a execução foram considerados:

- Consenso com o experimento: procurou-se assegurar que os participantes tivessem concordância com os objetivos da pesquisa e do experimento, por meio do treinamento, fornecendo o embasamento teórico necessário.

- Resultados sensitivos: possível influência no experimento por questões pessoais dos participantes por estarem sendo avaliados. Foi assegurado o sigilo dos participantes na descrição do experimento.

- **Instrumentação**: oferta de um treinamento específico para cada grupo, contextualizando os objetivos, a técnica, a motivação, o procedimento técnico e a ferramenta para a execução. O conteúdo usado no treinamento de ambos os grupos está disponível no Apêndice I. O tutorial sobre a execução das atividades para a abordagem com representação de conhecimento está no Apêndice J e o tutorial para a

abordagem sem representação de conhecimento no Apêndice K. O formulário para preenchimento dos resultados por parte dos participantes está no Apêndice L.

- Execução: o pesquisador esteve envolvido em todos os detalhes da execução do experimento, que ainda considerou que:

- A coleta de dados foi de responsabilidade dos participantes, por meio do preenchimento dos formulários;

- O pesquisador ficou disponível para responder aos eventuais questionamentos durante o experimento.

6.1.4 Análise e interpretação

As conclusões são extraídas a partir da interpretação dos dados gerados durante a execução [WOH00]. Como observação inicial sobre os dados obtidos no experimento, sugere-se atentar para a medida de suas escalas. São as escalas que determinam as operações que podem ser aplicadas aos valores das variáveis. A partir da verificação das medidas das escalas de cada variável, parte-se para uma análise numérica dos dados, empregando recursos da estatística descritiva. Com base na análise da distribuição geral do conjunto de dados, eventuais distorções como dados anormais ou valores extremos (*outliers*) que comprometam a validade das conclusões devem ser desconsideradas. Por fim, obtêm-se as conclusões do experimento por meio do teste das hipóteses.

Seguindo o exposto, como análise inicial, apresenta-se a classificação das escalas das variáveis definidas no experimento, sendo elas:

- Variáveis dependentes:

- Tempo: escala *Razão*;

- Precisão: escala *Razão*.

- Variáveis independentes:

- Técnica: escala *Nominal*.

6.1.4.1 Análise tabular e gráfica

O experimento obteve os resultados conforme apresentados na Tabela 6.2.

Tabela 6.2: Tabulação dos valores obtidos com o experimento

Abordagem	Participante	Esforço	Precisão
Com Representação	C01	5,00	1,00
Com Representação	C02	4,00	1,00
Com Representação	C03	8,00	1,00
Com Representação	C04	5,00	1,00
Com Representação	C05	7,00	1,00
Com Representação	C06	6,00	1,00
Com Representação	C07	6,00	0,97
Com Representação	C08	5,00	1,00
Sem Representação	S01	6,00	0,91
Sem Representação	S02	16,00	0,37
Sem Representação	S03	6,00	0,91
Sem Representação	S04	8,00	0,91
Sem Representação	S05	8,00	0,78
Sem Representação	S06	11,00	0,44
Sem Representação	S07	19,00	0,50
Sem Representação	S08	7,00	0,91

Na Figura 6.1 é apresentado o gráfico de barras com os resultados referentes ao esforço de cada participante para a indexação (representação) da documentação, utilizando as abordagens com e sem representação de conhecimento.

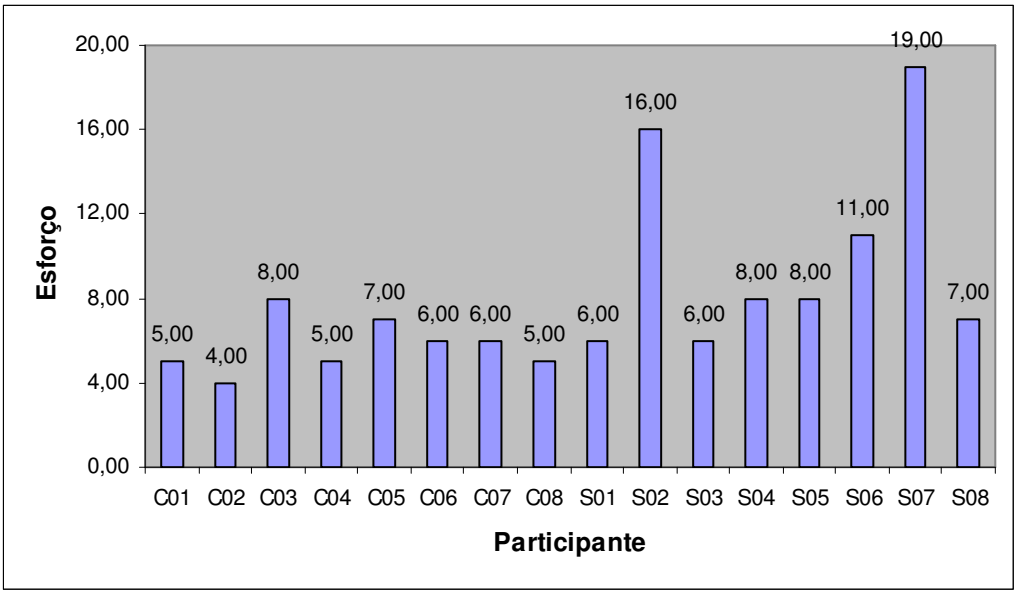


Figura 6.1: Gráfico de barras referente ao esforço para indexação da documentação

Os resultados referentes à precisão dos artefatos recuperados por cada participante, utilizando a respectiva abordagem, são representados na Figura 6.2 por meio de um gráfico de barras.

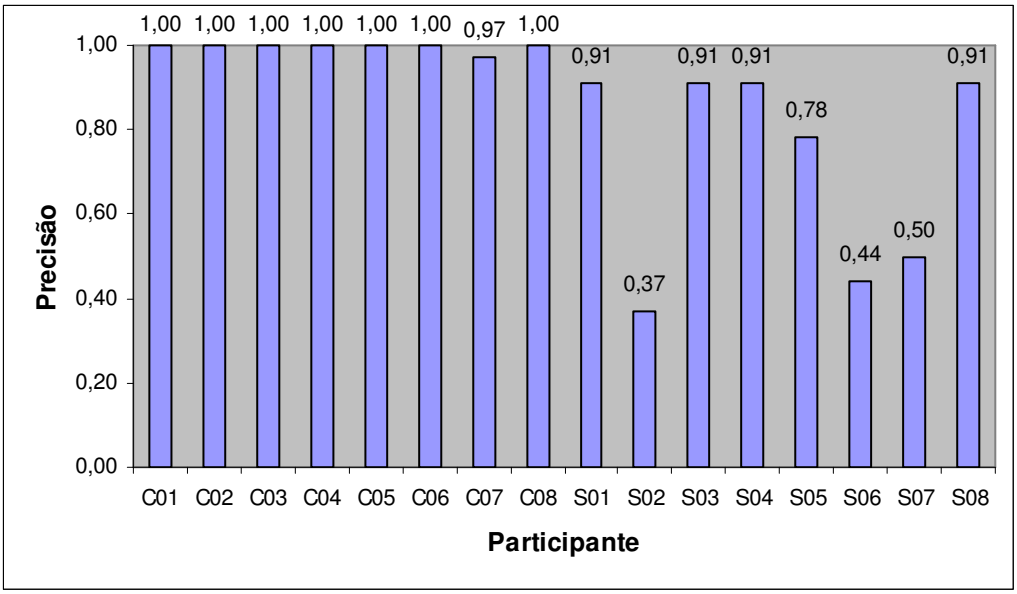


Figura 6.2: Gráfico de barras referente à precisão na recuperação da documentação

6.1.4.2 Estatística descritiva

Considerando que as variáveis dependentes (esforço e precisão) referem-se a escala razão, é possível o cálculo da normalidade e da homocedasticidade, a fim de definir se o tipo de teste das hipóteses é paramétrico ou não-paramétrico. Para tal foram considerados os seguintes padrões:

- Cálculo da normalidade e homocedasticidade: define a utilização do *Teste T*, se paramétrico, ou *Mann-Whitney*, se não-paramétrico.

- Nível de significância adotado (*p-value*): 5% para todos os testes, que corresponde ao menor nível de significância para rejeição da hipótese nula.

6.1.4.3 Análise dos resultados para a variável *esforço*

- Distribuição: para análise da distribuição, avaliando o comportamento das amostras, foi utilizado o gráfico de dispersão *boxplot* para identificação dos possíveis *outliers*, conforme ilustra a Figura 6.3.

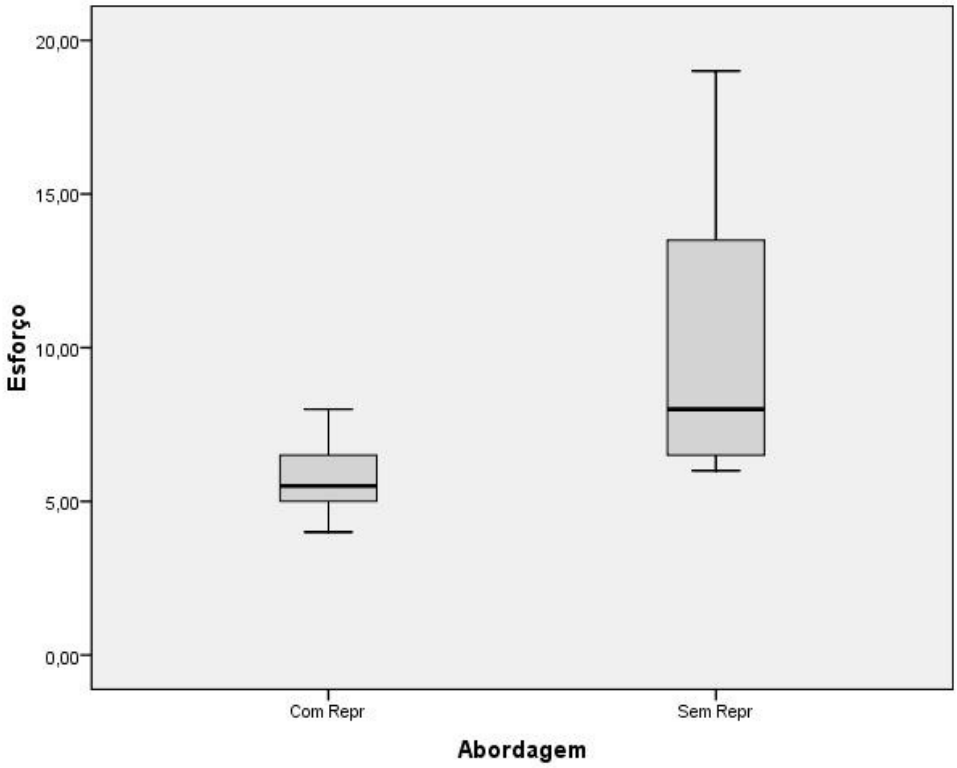


Figura 6.3: Gráfico de dispersão para a variável *esforço*

- Seleção da Amostra: como pode ser observado na Figura 6.3, a variável *esforço* não possui *outliers*. Dessa forma, os valores de todos os participantes serão considerados na amostra.

- Análise da normalidade: identificar se os dados seguem uma distribuição normal. Para tal, foi definida uma hipótese nula e uma hipótese alternativa, sendo:

- $\mathbf{H_0}$: a distribuição é normal;

- $\mathbf{H_1}$: a distribuição não é normal.

Considerando que a variável esforço possui menos de 50 valores, para se avaliar a distribuição dos dados foi utilizado o teste de *Shapiro-Wilk*. Na Tabela 6.3 são apresentados os resultados do teste para a amostra selecionada.

Tabela 6.3: Resultados do teste de normalidade *Shapiro-Wilk* para a variável *esforço*

Variável	Abordagem	Estatística	Grau de Liberdade	Significância
Esforço	Com Representação	0,938	8	0,592
	Sem Representação	0,824	8	0,052

Considerando os valores da Tabela 6.3, nota-se que a significância dos dados do teste de *Shapiro-Wilk* é superior ao nível de significância determinado (0,05 ou 5%). Dessa forma, consegue-se o requisito inicial para utilização de teste paramétrico.

- Análise da homocedasticidade: consiste em analisar a variância das duas amostras. Para tal, também foram definidas duas hipóteses, sendo:

- $\mathbf{H_0}$: as variâncias são iguais;

- $\mathbf{H_1}$: as variâncias não são iguais.

O teste da hipótese definida foi realizado por meio do *Teste de Levene*, que é utilizado para testar se *k* amostras possuem a mesma variância. Na Tabela 6.4 são listados os resultados obtidos.

Tabela 6.4: Resultados do teste de *Levene* para a variável *esforço*

Variável	Variâncias Iguais	Significância
Esforço	Assumindo	0,007
	Não assumindo	0,000

De acordo com a Tabela 6.4, nota-se que o nível de significância para variâncias iguais (0,007) é inferior ao nível de significância considerado (0,05 ou 5%). Com esse resultado, o segundo requisito para utilização do teste paramétrico não se confirma, ou seja, rejeita-se a hipótese nula.

- Resultado: os resultados obtidos na análise da normalidade e da homocedasticidade, não preenchem os requisitos referentes ao teste paramétrico, devendo ser aplicado, portanto, o

teste de *Mann-Whitney* para avaliar se as diferenças entre as médias são estatisticamente significativas.

- Aplicação do Teste de *Mann-Whitney*: a partir das hipóteses definidas inicialmente, foram consideradas as seguintes possibilidades:

- $\mathbf{H_0}$: Não há diferença entre as médias ($\mu_{crep} = \mu_{srep}$);

- $\mathbf{H_1}$: Há diferença entre as médias ($\mu_{crep} \neq \mu_{srep}$).

Os resultados obtidos com a realização do teste de *Mann-Whitney* estão representados na Tabela 6.5.

Tabela 6.5: Resultados do teste não-paramétrico de *Mann-Whitney* para a variável *esforço*

Variável	U de *Mann-Whitney*	W de *Wilcoxon*	Z	Sig. Assimpt. (bilateral)	Sig. Exata [2*(Sig. Unilateral)
Esforço	8,500	44,500	-2,503	0,012	0,010[a]

(a) Não Corrigido para os Empates

Conforme a Tabela 6.5, o grau de significação associado (Sig. Assimpt.) é 0,012 e, portanto, menor que a significância assumida (0,050). Dessa forma é possível rejeitar H_0 e considerar que para a variável *esforço* existe diferença de média entre a abordagem com a representação e sem a representação de conhecimento.

- Conclusões: o teste de *Mann-Whitney* rejeitou a hipótese nula definida inicialmente, mas não avalia as hipóteses alternativas. Essas podem ser avaliadas comparando-se a análise descritiva das médias da amostra. Observando que a média da variável *esforço* para abordagem com representação é igual a 5,7500 e que a média para a abordagem sem representação é igual a 10,1250, considera-se que o esforço na associação dos artefatos (indexação) é menor na abordagem com a representação de conhecimento.

6.1.4.4 Análise dos resultados para a variável *precisão*

- Distribuição: novamente utilizou-se o gráfico de dispersão *boxplot* para identificação de *outliers*, conforme representado na Figura 6.4.

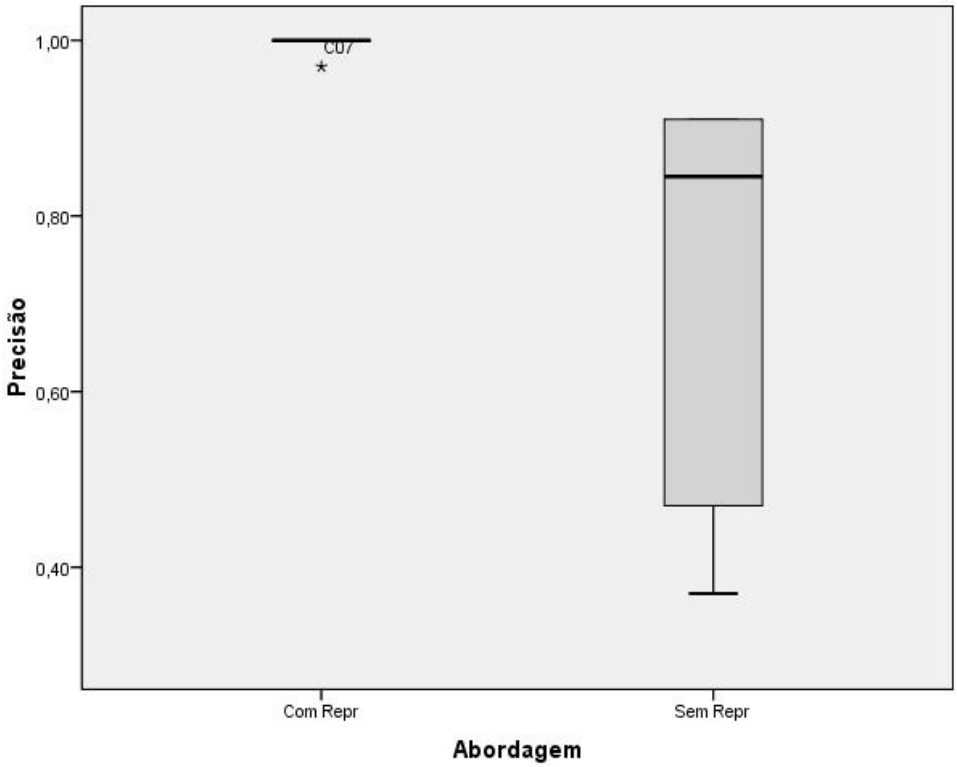


Figura 6.4: Gráfico de dispersão para a variável *precisão*

- Seleção da Amostra: conforme a Figura 6.4, a variável precisão possui apenas um *outlier*, o participante C07. Nota-se que para a abordagem com representação, a variável precisão possui as seguintes medidas estatísticas: Média = 0,9963; Desvio Padrão = 0,01061. Considerando como critério de projeto que apenas os valores que não atingirem a média com mais de dois desvios padrões serão eliminados da amostra, o participante C07 foi mantido no conjunto de dados.

- Análise da normalidade: para identificar se os dados seguem uma distribuição normal foram definidas as seguintes hipóteses:

- $\mathbf{H_0}$: a distribuição é normal;

- $\mathbf{H_1}$: a distribuição não é normal.

Como a variável precisão possui menos de 50 valores, novamente será utilizado o teste de *Shapiro-Wilk* para avaliar a distribuição dos dados. A Tabela 6.6 apresenta os resultados do teste para a amostra selecionada.

Tabela 6.6: Resultados do teste de normalidade *Shapiro-Wilk* para a variável *precisão*

Variável	Abordagem	Estatística	Grau de Liberdade	Significância
Precisão	Com Representação	0,418	8	0,000
	Sem Representação	0,774	8	0,015

De acordo com os valores da Tabela 6.6, nota-se que a significância dos dados do teste de *Shapiro-Wilk* é inferior ao nível de significância determinado (0,05 ou 5%) para as duas abordagens. Dessa forma, não se atinge o requisito inicial para utilização de teste paramétrico, devendo-se rejeitar a hipótese nula.

- Resultado: os resultados obtidos na análise da normalidade não preenchem os requisitos referentes ao teste paramétrico, definindo, portanto, o teste como não-paramétrico. Dessa forma, será aplicado o teste de *Mann-Whitney* para duas amostras independentes, visando identificar se as diferenças entre as médias são estatisticamente significativas.

- Aplicação do Teste de *Mann-Whitney*: considerando as hipóteses definidas, foram consideradas as seguintes possibilidades:

- $\mathbf{H_0}$: Não há diferença entre as médias ($\mu_{crep} = \mu_{srep}$);

- $\mathbf{H_1}$: Há diferença entre as médias ($\mu_{crep} \neq \mu_{srep}$).

Os resultados obtidos com a realização do teste de *Mann-Whitney* estão representados na Tabela 6.7.

Tabela 6.7: Resultados do teste não-paramétrico de *Mann-Whitney* para a variável *precisão*

Variável	U de *Mann-Whitney*	W de *Wilcoxon*	Z	Sig. Assimpt. (bilateral)	Sig. Exata [2*(Sig. Unilateral)
Precisão	0,000	36,000	-3,537	0,000	0,000[a]

(a) Não Corrigido para os Empates

Conforme a Tabela 6.7, o grau de significação associado (Sig. Assimpt.) é 0,000 e, portanto, menor que a significância assumida (0,050). Dessa forma é possível rejeitar H_0 e considerar que para a variável *precisão* existe diferença de média entre a abordagem com a representação e sem a representação de conhecimento.

- Conclusões: o teste de *Mann-Whitney* rejeitou a hipótese nula, mas não avalia as hipóteses alternativas. Essas podem ser avaliadas comparando-se a análise descritiva das médias da amostra. Considerando que a média para abordagem com representação é igual a 0,9963 e que a média para a abordagem sem representação é igual a 0,7163, nota-se que a precisão na recuperação do conhecimento (compreensão) é maior na abordagem com a representação de conhecimento.

6.1.5 Empacotamento

A fase de empacotamento visa documentar os aspectos relacionados ao experimento, possibilitando a replicação em outros contextos e a constatação de resultados adicionais. A Seção 6.1 de modo geral corresponde ao empacotamento do experimento realizado para avaliação da proposta de representação de conhecimento e contém as informações para a replicação deste.

6.2 Avaliação qualitativa das abordagens

Com o objetivo de agregar uma análise qualitativa ao experimento realizado, foi realizada uma pesquisa de opinião com os participantes, ao término da execução. Cada participante respondeu um questionário, de acordo com o Apêndice M. A escala *Likert* foi utilizada no questionário com 5 pontos para o grau de satisfação de cada resposta.

A pesquisa teve como objetivo coletar a opinião dos participantes com relação à usabilidade, utilidade e esforço em cada abordagem. Os dados obtidos como resultados foram tabulados de acordo com a Tabela 6.8.

Tabela 6.8: Tabulação dos resultados obtidos com a pesquisa de opinião sobre as abordagens

Abordagem	Participante	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Com Representação	C01	4	5	2	5	2	4	4
Com Representação	C02	3	5	4	4	5	4	5
Com Representação	C03	5	5	4	5	5	4	4
Com Representação	C04	4	5	5	3	4	5	4
Com Representação	C05	5	5	4	4	5	5	5
Com Representação	C06	4	5	3	4	5	4	4
Com Representação	C07	5	5	5	5	5	4	5
Com Representação	C08	4	4	3	4	2	3	4
Sem Representação	S01	4	4	4	4	4	4	4
Sem Representação	S02	4	4	4	3	4	4	4
Sem Representação	S03	1	4	2	2	1	1	3
Sem Representação	S04	3	5	4	3	2	4	4
Sem Representação	S05	4	4	2	2	3	4	4
Sem Representação	S06	4	4	3	4	3	5	4
Sem Representação	S07	3	4	4	2	2	2	3
Sem Representação	S08	5	5	4	4	3	4	5

Como análise preliminar, é apresentada a média aritmética das respostas das questões considerando cada abordagem, conforme Tabela 6.9.

Tabela 6.9: Média das respostas das questões sobre as abordagens

Abordagem	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Com Representação	4,25	4,88	3,75	4,25	4,13	4,13	4,38
Sem Representação	3,50	4,25	3,38	3,00	2,75	3,50	3,88

Nota-se que as médias para as respostas referentes a abordagem com representação de conhecimento é maior do para a abordagem sem representação de conhecimento, para todas as questões. Considerando a utilização da escala de *Linkert* variando de 1 a 5, sendo 1 para o grau de satisfação mais baixo e 5 para o grau de satisfação mais alto, verifica-se que a avaliação qualitativa da abordagem com representação de conhecimento foi superior a abordagem sem representação de conhecimento.

6.3 Avaliação qualitativa da experiência dos participantes

Como abordado na descrição do experimento, para a escolha dos participantes foram consideradas as pessoas com maior disponibilidade (amostragem por conveniência). Dessa forma, participaram do experimento dezesseis estudantes do último período da graduação em Ciência da Computação da UNEMAT, dos quais oito concluem o curso no semestre corrente (2009/1) e os demais concluem o curso em semestres futuros. Visando atender aos princípios de aleatoriedade e balanceamento, para cada abordagem avaliada foram alocados aleatoriamente quatro estudantes concluintes e quatro que concluem o curso futuramente.

Embora a população do experimento tenha englobado apenas estudantes de graduação, não sendo possível o envolvimento de estudantes de pós-graduação ou de profissionais que atuam na área, pressupõe-se que esses estudantes tenham diferentes experiências relacionadas ao desenvolvimento e manutenção de software. Visando considerar os possíveis impactos ou interferências causados por essa experiência na condução do experimento, foi realizada uma segunda pesquisa de opinião com o objetivo de analisar qualitativamente a experiência dos participantes de cada abordagem. Novamente cada participante respondeu um questionário, conforme o Apêndice N. A escala *Likert* com 5 pontos para o grau de satisfação de cada resposta foi considerada, atribuindo-se 1 para o menor valor e 5 para o maior. Os dados obtidos por meio dos questionários foram tabulados de acordo com a Tabela 6.10.

Tabela 6.10: Tabulação dos resultados obtidos com a pesquisa de opinião sobre a experiência dos participantes

Abordagem	Participante	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Com Representação	C01	2	1	2	2	1	2	5	3	3	2
Com Representação	C02	2	2	5	3	1	3	4	3	2	1
Com Representação	C03	2	2	2	2	2	3	4	2	2	1
Com Representação	C04	3	2	1	1	1	3	4	4	4	3
Com Representação	C05	1	1	1	3	1	2	5	1	1	1
Com Representação	C06	2	2	2	1	1	1	4	3	2	2
Com Representação	C07	5	5	2	2	1	2	4	1	2	2
Com Representação	C08	1	1	1	1	1	1	3	1	1	1
Sem Representação	S01	4	1	4	1	1	1	4	4	2	2
Sem Representação	S02	1	1	3	3	4	3	3	3	3	4
Sem Representação	S03	3	1	1	1	1	2	4	3	2	1
Sem Representação	S04	3	3	4	4	4	3	4	3	3	3
Sem Representação	S05	2	1	1	1	2	3	4	3	2	3
Sem Representação	S06	3	3	1	1	2	2	4	2	2	2
Sem Representação	S07	2	1	2	1	1	2	4	3	2	1
Sem Representação	S08	3	3	4	4	3	2	3	2	2	2

Em uma primeira análise apresenta-se a média aritmética das respostas das questões considerando cada abordagem, conforme Tabela 6.11.

Tabela 6.11: Média das respostas das questões sobre a experiência dos participantes

Abordagem	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Com Representação	2,25	2,00	2,00	1,88	1,13	2,13	4,13	2,25	2,13	1,63
Sem Representação	2,63	1,75	2,50	2,00	2,25	2,25	3,75	2,88	2,25	2,25

Observa-se que as médias para as respostas referentes a abordagem sem representação de conhecimento em oito das questões aplicadas é maior do que para a abordagem com representação de conhecimento. Estas questões abordam respectivamente: Q1) experiência no desenvolvimento de sistema em anos; Q3) frequência de atuação no desenvolvimento de sistemas no trabalho; Q4) frequência de atuação na manutenção de sistemas no trabalho; Q5) frequência na utilização de metodologias ágeis no trabalho; Q6) preocupação quanto a manter a rastreabilidade entre artefatos desenvolvidos; Q8) conhecimento sobre desenvolvimento de sistemas; Q9) conhecimento sobre manutenção de sistemas; Q10) conhecimento sobre metodologias ágeis. Essas condições podem ter favorecido os resultados obtidos na abordagem sem representação de conhecimento, mas apesar disso ainda obteve resultados piores.

De maneira global, a experiência dos participantes manteve-se em torno de um grau de satisfação regular, com os valores referentes a abordagem sem representação um pouco superiores aos da abordagem com representação. Considerando esses resultados, sugere-se para replicações futuras um melhor equilíbrio na distribuição dos participantes conforme experiência, e o envolvimento de participantes com maior experiência, a fim de verificar se os resultados mantêm correspondência com os obtidos no experimento atual.

6.4 Considerações

Uma avaliação da proposta de representação de conhecimento e da ferramenta protótipo desenvolvida foi aplicada neste capítulo, por meio da realização de um experimento. Para tal, buscou-se fundamentar com base na literatura as principais fases e princípios a serem considerados, com o objetivo de garantir a validação do experimento.

Embora tenha se procurado seguir essas etapas rigorosamente, o envolvimento do pesquisador na organização do experimento e na manipulação dos dados obtidos para se chegar aos resultados finais pode ter exercido alguma influência sobre o experimento, interferindo ainda que sensivelmente nesses resultados. Essa interferência, mesmo indesejada, está sujeita a ocorrer em experimentos conforme prevê a literatura.

Diante dessa possibilidade, recomenda-se a replicação do referido experimento em trabalhos futuros com outra população. É fortemente estimulada a aproximação com populações que atuem no ambiente industrial, envolvendo um projeto real (processo *in vivo*), durante o desenvolvimento de software ágil, avaliando os resultados neste contexto. Para possibilitar essa replicação, neste capítulo também foi inserido o empacotamento, contendo a estruturação do experimento e a documentação dos resultados.

Os resultados obtidos na análise quantitativa demonstraram a viabilidade da abordagem com representação de conhecimento, tanto para a variável *esforço* quanto para a variável *precisão*, sobre a abordagem sem representação de conhecimento. Embora não tenha sido possível a utilização de testes *paramétricos* para testar as hipóteses de cada variável, por meio de testes *não paramétricos* pode-se concluir que o *esforço*, medido em minutos, para indexação (representação) dos artefatos com representação de conhecimento foi menor do que a indexação sem representação de conhecimento. Também se verificou que a *precisão*, medida por meio das associações entre os artefatos recuperadas corretamente, para compreensão (recuperação) dos artefatos foi maior com representação de conhecimento do que sem a representação de conhecimento.

No que se refere a avaliação qualitativa (pesquisa de opinião), a abordagem com representação de conhecimento também apresentou médias superiores a abordagem sem representação de conhecimento, para todas as questões aplicadas.

Convém destacar que os resultados obtidos podem ter sido influenciados pela natureza das tarefas executadas pelos sujeitos, favorecendo o aparente excesso de precisão obtido pela abordagem com representação de conhecimento. A ferramenta protótipo desenvolvida e utilizada nos testes também pode ter influenciado a precisão, por estar adaptada ao tipo de tarefa colocada para os sujeitos. A documentação utilizada no teste refere-se a um sistema de pequeno porte, desenvolvido por alunos de graduação (*toy example*), e precisou ser traduzida para o inglês para viabilizar a utilização da ferramenta. Esse processo de tradução pode ter favorecido o estabelecimento das associações entre as estórias e os conceitos do domínio, influenciando diretamente na precisão obtida por meio da ferramenta, por minimizar as possibilidades de erro. O

esforço no estabelecimento das associações também é melhor atacado pela ferramenta, uma vez que grande parte do processo é automatizado e não depende exclusivamente do desempenho do indivíduo. Tais considerações reforçam a necessidade de replicação do experimento em trabalhos futuros.

O experimento realizado teve cobertura sobre a proposta de representação de conhecimento em metodologias ágeis em sua dimensão por conceitos. É recomendável a realização de um experimento adicional com cobertura sobre a representação em sua dimensão por tipo de conhecimento, analisando as associações recuperadas. Em virtude de sua extensão, essa experimentação não pode ser considerada neste trabalho. Contudo, a mesma é incentivada para trabalhos futuros.

7 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma proposta de representação de conhecimento para documentação em metodologias ágeis, tendo como meta estabelecer relacionamentos entre os artefatos produzidos durante o desenvolvimento de um sistema. Dessa forma, a proposta encontra-se diretamente associada ao conhecimento explícito, que é representado durante o desenvolvimento ágil.

Além da proposta de representação de conhecimento relacionada aos artefatos produzidos, este trabalho agrega como contribuição um levantamento por meio de uma revisão da literatura sobre os artefatos produzidos pelas metodologias ágeis, em especial nas metodologias ágeis XP e *Scrum*. A proposta permite a associação destes artefatos aos conceitos do domínio, por meio da construção de um Modelo de Domínio, introduzindo um nível maior de formalidade na maneira como as metodologias ágeis lidam com a realidade para a qual o sistema é desenvolvido. Com a associação entre os artefatos e os conceitos presentes no domínio, a proposta contribui para manter a rastreabilidade nas metodologias ágeis.

Um levantamento sobre os tipos de conhecimento que mais se destacam no desenvolvimento de software também foi realizado. Foram estabelecidas associações iniciais entre os artefatos e os tipos de conhecimento que estes exprimem. Essas associações permitem ampliar a compreensão dos desenvolvedores sobre qual documentação deve ser produzida e quais artefatos devem ser utilizados, para que se exprima um conhecimento em específico, importante de ser transmitido e recuperado em futuras interações com o sistema construído.

As principais ferramentas disponíveis para viabilizar suporte para o trabalho com metodologias ágeis foram identificadas, por meio de outro levantamento realizado. A partir desse levantamento o trabalho selecionou a ferramenta *XPlanner* e disponibilizou a ela a ferramenta protótipo *XPlannerKnOWLedge* capaz de implementar automaticamente a proposta de representação de conhecimento apresentada.

Em virtude das especificidades encontradas ao longo da pesquisa e da abrangência do tema em estudo, não se direcionou o foco deste trabalho para incluir o conhecimento tácito. Esse conhecimento, geralmente manifestado por meio de reuniões diárias e conversas entre os membros da equipe de desenvolvimento, é característico nas metodologias ágeis, além do conhecimento que é representado (conhecimento explícito). Dada a sua relevância, a representação do conhecimento

tácito tem sido alvo de uma pesquisa em nível de doutorado no contexto do grupo de pesquisa ISEG, neste programa de pós-graduação.

Como trabalhos futuros, além do foco na representação do conhecimento tácito, é sugerida a melhoria da ferramenta protótipo desenvolvida, ampliando suas funcionalidades tanto na definição quanto na recuperação das associações artefato-conceito e artefato-tipo de conhecimento. Investir em uma ferramenta que contemple um número maior de artefatos, incluindo o suporte para o código fonte, para uma determinada metodologia ágil, também permitirá ampliar a capacidade de representação e recuperação do conhecimento estabelecido durante o desenvolvimento.

Uma pesquisa aprofundada sobre os tipos de conhecimento presentes no desenvolvimento de software, considerando ambientes industriais com projetos reais e seus artefatos, também pode contribuir para associação entre conhecimento e artefatos produzidos de forma mais adequada aos objetivos dos desenvolvedores.

Por fim, o aprofundamento na pesquisa sobre formalismos de representação, passíveis de serem representados computacionalmente e associados às metodologias ágeis, pode apontar novas contribuições para o desenvolvimento de software e a socialização do conhecimento.

REFERÊNCIAS BIBLIOGRÁFICAS

[ABR04] ABRAHAMSSON, P.; KOSKELA, J. "Extreme Programming: A Survey of Empirical Data from a Controlled Case Study". In: International Symposium on Empirical Software Engineering, 2004, pp. 73-82.

[AGI08] AGILE ALLIANCE. "Agile Alliance Home". Capturado em: http://www.agilealliance.org/, Outubro 2008.

[AGI09] AGILEFANT.ORG. "Agilefant: The Finnish Ferrari for backlog management". Capturado em: http://www.agilefant.org/, Março 2009.

[ALV03] ALVARENGA, L. "Representação do conhecimento na perspectiva da ciência da informação em tempo e espaço digitais". *Encontros Bibli: Revista Eletrônica de Biblioteconomia e Ciência da Informação*, vol. 8, Jan-Jun 2003, pp. 18-40.

[AMB02] AMBLER, S. W. "Agile Modeling: Effective Practices for Extreme Programming and the Unified Process". Wiley, 2002, 400p.

[AMB04] AMBLER, S. W. "The Object Primer: Agile Model-Driven Development with UML 2.0". Cambridge University Press, 2004, 572p.

[ANQ07] ANQUETIL, N.; OLIVEIRA, K. M.; SOUSA, K. D.; BATISTA DIAS, M. G. "Software maintenance seen as a knowledge management issue". *Information and Software Technology*, vol. 49-5, Maio 2007, pp. 515-529.

[ARA06] ARAÚJO, M. A.; BARROS, M.; TRAVASSOS, G.; MURTA, L. "Métodos Estatísticos aplicados em Engenharia de Software Experimental". In: XX Simpósio Brasileiro de Engenharia de Software, 2006, pp. 325-326.

[BAR97] BARITÉ, M. "Glosario sobre organización y representación del conocimiento, clasificación, indización, terminología". Montevideo: Comisión Sectorial de Investigación Científica, 1997, 170p.

[BAR04] BARBETA, P. A.; REIS, M. M.; BORNIA, A. C. "Estatística para Cursos de Engenharia e Informática". São Paulo: Atlas, 2004, 410p.

[BAS94] BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. "The Goal Question Metric Approach; Encyclopedia of Software Engineering". New York: Wiley-Interscience, 1994, 10p.

[BAS01] BASILI, V.; COSTA, P.; LINDVALL, M.; MENDONÇA, M.; SEAMAN, C.; TESORIERO, R.; ZELKOWITZ, M. "An Experience Management System for a Software Engineering Research Organization". In: 26th Annual NASA Goddard Software Engineering Workshop, 2001, 7p.

[BEC00] BECK, K. "Extreme Programming explained: embrace change". Boston: Addison-Wesley, 2000, 224p.

[BEC01] BECK, K. et al. "Manifesto for Agile Software Development". 2001. Capturado em: http://www.agilemanifesto.org/, Outubro 2007.

[BEC02] BECK, K. "Test-Driven Development: by example". Addison-Wesley, 2002, 240p.

[BIE06] BIELIKOVÁ, M.; MORAVÈÍK, M. “Modeling the Content of Adaptive Web-Based System Using an Ontology”. In: 1st International Workshop on Semantic Media Adaptation and Personalization, 2006, 6p.

[BIO05] BIOLCHINI, J.; MIAN, P.; NATALI, A.; TRAVASSOS, G. “Systematic Review in Software Engineering: Relevance and Utility”. Technical Report, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2005, 30p.

[BOH08] BOH, W. F. “Reuse of knowledge assets from repositories: A mixed methods study”. *Information and Management*, vol. 45-6, Setembro 2008, pp. 365-375.

[BOR97] BORST, W. N. “Construction of engineering ontologies for knowledge sharing and reuse”. Tese de doutorado, University of Twente, Enschede, 1997, 227p.

[BUL06] BULCÃO N., R. F. “Um processo de software e um modelo ontológico para apoio ao desenvolvimento de aplicações sensíveis ao contexto”. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, USP, 2006, 219p.

[CER03] CERAVOLO, P.; DAMIANI, E.; MARCHESI, M.; PINNA, S.; ZAVATARELLI, F. “A Ontology-based Process Modelling for XP”. In: 10th Asia-Pacific Software Engineering Conference, 2003, 7p.

[CHA03a] CHAU, T.; MAURER, F.; MELNIK, G. “Knowledge Sharing: Agile Methods vs. Tayloristic Methods”. In: 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003, 6p.

[CHA03b] CHAVES, M. S. “Mapeamento e comparação de similaridade entre estruturas ontológicas”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2003, 119p.

[COC00] COCKBURN, A. “Agile Software Development”. Addison-Wesley, 2000, 304p.

[COH03] COHEN, D.; LINDVALL, M.; COSTA, P. “Agile Software Development: a DACS State-of-the-Art Report”. New York: DACS, 2003, 65p.

[DUL04] DULIPOVICI, M.; ROBILLARD, P. N. “Cognitive aspects in a project-based course in software engineering”. 5th International Conference on Information Technology Based Higher Education and Training, 2004, pp. 353-359.

[FAL04] FALBO, R. A.; RUY, F. B.; PEZZIN, J.; MORO, R. D. “Ontologias e ambientes de desenvolvimento de software semânticos”. In: IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento, 2004, 16p.

[FER97] FERNÁNDEZ, M.; GÓMEZ-PÉREZ, A.; JURISTO, N. “Methontology: From Ontological Art towards Ontological Engineering”. In: AAAI Spring Symposium, 1997, pp. 33-40.

[FOR02] FORWARD, A.; LETHBRIDGE, T. C. “The relevance of software documentation, tools and technologies: a survey”. In: ACM Symposium on Document Engineering, 2002, pp. 26-33.

[FRI07] FRITZSCHE, M.; KEIL, P. “Agile Methods and CMMI: Compatibility or Conflict?”. *e-Informatica Software Engineering Journal*, vol. 1-1, Março 2007, pp. 9-26.

[GEN03] GENNARI, J. H.; MUSEN, M. A.; FERGERSON, R. W.; GROSSO, W. E.; CRUBÉZY, M.; ERIKSSON, H.; NOY, N. F.; TU, S. W. "The Evolution of Protégé: An Environment for Knowledge-Based Systems Development". *International Journal of Human-Computer Studies*, vol. 58-1, Janeiro 2003, pp. 89-123.

[GRU93] GRUBER, T. R. "A Translation Approach to Portable Ontology Specifications". *Knowledge Acquisition*, vol. 5-2, Junho 1993, 1993, pp. 199-220.

[GRU95] GRÜNINGER, M.; FOX, M. S. "Methodology for the Design and Evaluation of Ontologies". In: Workshop on Basic Ontological Issues in Knowledge Sharing, 1995, 10p.

[GUA98] GUARINO, N. "Formal Ontology and Information System". In: Formal Ontology in Information Systems, 1998, pp. 3-15.

[HAN06] HANSSON, C.; DITTRICH, Y.; GUSTAFSSON, B.; ZARNAK, S. "How agile are industrial software development practices?". *The Journal of Systems and Software*, vol. 79-9, Setembro 2006, p. 1295-1311.

[HAN08] HANKS, B.; WELLINGTON, C.; REICHLMAYR, T.; COUPAL, C. "Integrating agility in the CS curriculum: practices through values". In: 39th SIGCSE Technical Symposium on Computer Science Education, 2008, pp. 19-20.

[HED03] HEDIN, G.; BENDIX, L.; MAGNUSSON, B. "Teaching extreme programming to large groups of students". *Journal of Systems and Software*, vol. 74-2, Janeiro 2003, pp. 133-146.

[HIG00] HIGHSMITH, J. A. "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems". New York: Dorset House, 2000, 392p.

[HIN06] HINZ, V. T. Proposta de criação de uma ontologia de ontologias. Trabalho Individual I, Programa de Pós-Graduação em Informática, UCP, 2006, 68p.

[HUA09] HUANG, L.; HOLCOMBE, M. "Empirical investigation towards the effectiveness of Test First programming". *Information and Software Technology*, vol. 51-1, Janeiro 2009, pp. 182-194.

[HUM97] HUMPHREY, W. S. "Introduction to the Personal Software Process". Boston: Addison-Wesley, 1997, 304p.

[HUM00] HUMPHREY, W. S. "Introduction to the Team Software Process". Boston: Addison-Wesley, 2000, 496p.

[IBM09] IBM. "IBM Integrated Ontology Development Toolkit". Capturado em: http://www.alphaworks.ibm.com/tech/semanticstk, Abril 2009.

[ICE09] ICESCRUM. "IceScrum: Your open source agile tool". Capturado em: http://www.icescrum.org/, Março 2009.

[JEN09] JENA. "Jena: A Semantic Web Framework for Java". Capturado em: http://jena.sourceforge.net/, Março 2009.

[JWN09] JWNL. "API for Java WordNet Library". Capturado em: http://jwordnet.sourceforge.net, Março 2009.

[KIT04] KITCHENHAM, B. “Procedures for Performing Systematic Reviews”. Technical Report, Department of Computer Science, Keele University, National ICT Australia, 2004, 28p.

[KOB04] KOBAYASHI, O. “What Should Software Practitioners Know for Adopting Product Line Software Engineering?”. In: 11th Asia-Pacific Software Engineering Conference, 2004, 2p.

[KOK08] KOKKONIEMI, J. K. “Gathering Experience Knowledge from Iterative Software Development Processes”. In: 41st Annual Hawaii International Conference on System Sciences, 2008, 10p.

[KRU03] KRUCHTEN, P. “The Rational Unified Process: An Introduction”. Addison-Wesley, 2003, 3.ed, 320p.

[KUN70] KUNZ, W.; RITTEL, H. “Issues as elements of information systems”. Working paper no. 131, Institute of Urban and Regional Development, University of California, 1970, 10p.

[KUT02] KUTSCHERA, P.; SCHÄFER, S. “Applying Agile Methods in Rapidly Changing Environments”. In: 1st RTO Symposium on Technology for Evolutionary Software Development, 2002, 8p.

[LAR02] LARMAN, C. “An Agile UP: Introduction”. 2002. Capturado em: http://www.craiglarman.com, Março 2008.

[LAW05] LAW, A.; CHARRON, R. “Effects of agile practices on social factors”. *ACM SIGSOFT Software Engineering Notes*, vol. 30-4, Julho 2005, pp. 1-5.

[LIN05] LINDOSO, A. N.; GIRARDI, R. “Uma técnica baseada em ontologias para o reuso de padrões de software e de frameworks no projeto de aplicações multiagente”. In: 1st Workshop on Software Engineering for Agent-oriented Systems, 2005, pp. 31-40.

[MAC91] MACLEAN, A.; YOUNG, R.; BELLOTTI, V.; MORAN, T. “Questions, options and criteria: Elements of design space analysis”. *Human-Computer Interaction*, vol. 6-3, Setembro 1991, pp. 201-250.

[MAR07] MARCHESI, M.; MANNARO, K.; URAS, S.; LOCCI, M. “Distributed Scrum in Research Project Management”. *Agile Processes in Software Engineering and Extreme Programming*, Julho 2007, pp. 240-244.

[MIL94] MILES, M. B.; HUBERMAN, M. “Qualitative Data Analysis: An Expanded Sourcebook”. Sage Publications, 1994, 2.ed., 352p.

[NOL07a] NOLL, R. P. “Rastreabilidade ontológica sobre o processo unificado”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2007, 132p.

[NOL07b] NOLL, R. P. “Uma proposta para análise de similaridade entre documentos XML e ontologias definidas em OWL”. Monografia de Especialização, Curso de Especialização em Web e Sistemas de Informação, UFRGS, 2007, 26p.

[NON97] NONAKA, I.; TAKEUCHI, H. “Criação de conhecimento na empresa: como as empresas japonesas geram a dinâmica da inovação”. Rio de Janeiro: Campus, 1997, 4.ed., 358p.

[OMG08] OBJECT MANAGEMENT GROUP. “Software Process Engineering Meta-Model”. Capturado em: http://www.omg.org/spec/SPEM/2.0/, Outubro 2008.

[OMG09a] OBJECT MANAGEMENT GROUP. “Introduction to OMG's Unified Modeling Language (UML)”. Capturado em: http://www.omg.org/spec/UML/2.2/, Abril 2009.

[OMG09b] OBJECT MANAGEMENT GROUP. “Ontology Definition Metamodel (ODM)”. Capturado em: http://www.omg.org/spec/ODM/1.0/, Maio 2009.

[PAE08] PAELKE, V.; NEBE, K. “Integrating agile methods for mixed reality design space exploration”. In: 7th ACM Conference on Designing Interactive Systems, 2008, pp. 240-249.

[PAL02] PALMER, S. R.; FELSING, J. M. “A Practical Guide to Feature-Driven Development”. Prentice Hall, 2002, 304p.

[PFL98] PFLEEGER, S. L. “Software Engineering: Theory and Practice”. Amsterdam: Prentice Hall, 1998, 659p.

[PIK08] PIKKARAINEN, M.; HAIKARA, J.; SALO, O.; ABRAHAMSSON, P.; STILL, J. “The impact of agile practices on communication in software development”. *Empirical Software Engineering*, vol. 13-3, Junho 2008, pp. 303-337.

[POP06] POPPENDIECK, M.; POPPENDIECK, T. “Implementing Lean Software Development: From Concept to Cash”. Addison-Wesley, 2006, 304p.

[PRI06] PRINCETON UNIVERSITY. “WordNet: a lexical database for the English language”. 2006. Capturado em: http://wordnet.princeton.edu/, Abril 2009.

[RAM02] RAMAL, M.F.; MENESES, R.D.; ANQUETIL, N. “A Disturbing Result on the Knowledge Used during Software Maintenance”. In: 9th Working Conference on Reverse Engineering, 2002, 10p.

[ROD08] RODRÍGUEZ-ELIAS, O.M.; MARTÍNEZ-GARCÍA, A.I.; VIZCAÍNO, A.; FAVELA, J.; PIATTINI, M. “A framework to analyze information systems as knowledge flow facilitators”. *Information and Software Technology*, vol. 50-6, Maio 2008, pp. 481-498.

[ROS05] ROSENBERG, D.; COLLINS-COPE, M.; STEPHENS, M. “Agile Development with ICONIX Process: People, Process and Pragmatism”. Apress, 2005, 261p.

[RUP03] RÜPING, A. “Agile documentation: a pattern guide to producing lightweight documents for software projects”. John Wiley & Sons Ltd, 2003, 244p.

[RUS02] RUS, I.; LINDVALL, M. “Knowledge Management in Software Engineering”. *IEEE Software*, vol. 19-3, Mai-Jun 2002, pp. 26-38.

[SAL08] SALO, O.; ABRAHAMSSON, P. “Agile methods in European embedded software development organizations: a survey on the actual use and usefulness of Extreme Programming and Scrum”. *IET Software*, vol. 2-1, Fevereiro 2008, pp. 58-64.

[SAM04] SAMPAIO, A. T. “XWEBPROCESS: um processo ágil para o desenvolvimento de aplicações web”. Dissertação de Mestrado, Pós-Graduação em Ciência da Computação, UFPE, 2004, 136p.

[SAN05] SANTOS, G.; VILLELA, K.; MONTONI, M.; ROCHA, A.R.; TRAVASSOS, G.H.; FIGUEIREDO, S.; MAFRA, S.; ALBUQUERQUE, A.; PARET, B.D.; AMARAL, M. "Knowledge Management in a Software Development Environment to support Software Processes Deployment". In: Professional Knowledge Management Conference, 2005, 10p.

[SAN09] SANDPIPER SOFTWARE. "Visual Ontology Modeler". Capturado em: http://www.sandsoft.com/products.html, Abril 2009.

[SAU03] SAUER, T. "Using Design Rationales for Agile Documentation". In: 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003, pp. 326-331.

[SCH95] SCHWABER, K. "The Scrum Development Process". In: Workshop on Business Object Design and Implementation, 1995, 23p.

[SCH02] SCHWABER, K.; BEEDLE, M. "Agile Software Development with Scrum". Prentice Hall, 2002, 158p.

[SCH04] SCHWABER, K. "Agile Project Management with Scrum". Microsoft Press, 2004, 192p.

[SEL08] SELLERI, F. "Ontologias em Metodologias Ágeis". Trabalho Individual I, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2008, 60p.

[SMI03] SMITH, J. "A Comparison of the IBM Rational Unified Process and eXtreme Programming". A technical discussion of RUP, Rational Software, IBM, 2003, 21p.

[SOA04] SOARES, M. S. "Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software". *Journal of Computer Science*, vol. 3-2, Jul-Dez 2004, pp. 8-13.

[SOU04] SOUSA, K. D.; ANQUETIL, N.; OLIVEIRA, K. M. "Captura de conhecimento durante a manutenção de software". In: Simpósio Brasileiro de Qualidade de Software, 2004, 10p.

[SOU05] SOUZA, S. C.; ANQUETIL, N.; OLIVEIRA, K. M. "A Study of the Documentation Essential to Software Maintenance". In: 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information, 2005, pp. 68-75.

[SOW00] SOWA, J. F. "Knowledge representation: logical, philosophical and computational foundations". Brooks/Cole, 2000, 594p.

[STA03] STAPLETON, J. "DSDM: Business Focused Development". Pearson Education, 2003, 2.ed., 272p.

[STE09] STEMMING. "The Lancaster Stemming Algorithm". Capturado em: http://www.comp.lancs.ac.uk/computing/research/stemming/, Maio 2009.

[SUR02] SURE, Y.; STUDER, R. "On-To-Knowledge Methodology". Project Deliverable D18 (WP5), Institute AIFB, University of Karlsruhe, 2002, 84p.

[TEL04] TELES, V. M. "Extreme Programming". São Paulo: Novatec, 2004, 320p.

[TEL05] TELES, V. M. “Um estudo de caso da adoção das práticas e valores do Extreme Programming”. Dissertação de Mestrado, Núcleo de Computação Eletrônico, UFRJ, 2005, 179p.

[TRA02] TRAVASSOS, G. H.; GUROV, D.; AMARAL, G. “Introdução a Engenharia de Software Experimental”. Relatório Técnico, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2002, 52p.

[USC95] USCHOLD, M.; KING, M. “Towards a Methodology for Building Ontologies”. In: International Joint Conferences on Artificial Intelligence, 1995, 13p.

[VAN06] VANFOSSON, T. “Plan-driven vs. Agile Software Engineering and Documentation: a Comparison from the Perspectives of both Developer and Consumer”. Submitted for the PhD Qualifying Examination, Computer Science, The University of Iowa, 2006, 20p.

[VIL04] VILLELA, K.; TRAVASSOS, G. H.; ROCHA, A. R. “Definição e Construção de Ambientes de Desenvolvimento de Software Orientados a Organização”. In: Simpósio Brasileiro de Qualidade de Software, 2004, 15p.

[W3C04a] WORD WIDE WEB CONSORTIUM. “Resource Description Framework (RDF) Model and Syntax Specification”. Fevereiro 2004. Capturado em: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, Março 2008.

[W3C04b] WORD WIDE WEB CONSORTIUM. “OWL Web Ontology Language Overview”. Fevereiro 2004. Capturado em: http://www.w3.org/TR/owl-features/, Dezembro 2008.

[WOH00] WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B.; WESSLÉN, A. “Experimentation in software engineering: an introduction”. Boston: Kluwer Academic Publishers, 2000, 228p.

[WOR09] WORDNET. “A lexical database for the English language”. Capturado em: http://wordnet.princeton.edu/, Maio 2009.

[XPL09] XPLANNER. “A project planning and tracking tool for eXtreme Programming (XP) teams”. Capturado em: http://www.xplanner.org, Março 2009.

[XPS09] XPSTUDIO. “An open source project available for eXtreme Programming (XP) or other Agile methodologies”. Capturado em: http://xpstudio.sourceforge.net/, Março 2009.

[XPW09] XPWEB. “A web-based tool to manage eXtreme Programming projects”. Capturado em: http://xpweb.sourceforge.net/, Março 2009.

[ZAN05] ZANATTA, A. L.; VILAIN, P. “Uma análise do método ágil Scrum conforme abordagem nas áreas de processo Gerenciamento e Desenvolvimento de Requisitos do CMMI”. In: Workshop em Engenharia de Requisitos, 2005, 12p.

APÊNDICE A – PROTOCOLO DA REVISÃO SISTEMÁTICA SOBRE ARTEFATOS DAS METODOLOGIAS ÁGEIS

PROTOCOLO DA REVISÃO SISTEMÁTICA

Mestrando: Fernando Selleri Silva
Orientador: Marcelo Blois Ribeiro
Tema: Uso de Representação de Conhecimento para Documentação em Metodologias Ágeis
PUCRS – Setembro de 2008

1 Formulação da Questão de Pesquisa

1.1. Foco da Questão:

- A proposta para realização desta revisão sistemática tem como escopo a identificação dos artefatos utilizados para documentação de sistemas desenvolvidos com metodologias ágeis. Neste sentido, a questão principal que norteará a presente revisão sistemática foi definida.

1.2. Amplitude e Qualidade da Questão:

- Problema: Encontrar os artefatos gerados para documentação durante o desenvolvimento com métodos ágeis.

- Questão de pesquisa: Quais artefatos são utilizados para a documentação de sistemas desenvolvidos com metodologias ágeis?

- Palavras-chave e sinônimos: artefato, documentação, documento, *extreme programming*, *scrum*, metodologia ágil, estudo de caso, pesquisa de opinião. (em inglês: *artifact, artefact, documentation, document, extreme programming, scrum, agile, case study, survey*.

- Intervenção: documentação em metodologias ágeis.

- Efeito: artefatos utilizados para documentação em metodologias ágeis.

- População: projetos desenvolvidos com metodologias ágeis.

- Projeto Experimental: serão considerados trabalhos que utilizem como abordagem estudos de caso ou pesquisas de opinião.

2. Seleção das Fontes

2.1. Definição dos Critérios de Seleção das Fontes:

- Pesquisa em bases de dados eletrônicas, incluindo *journals*, anais de conferências, livros e bibliotecas virtuais, que publicam estudos relacionados à metodologias ágeis.

2.2. Língua dos Estudos: inglês e português (língua utilizada nas fontes de pesquisa)

2.3. Identificação das Fontes:

- Método de Busca das Fontes: mecanismos de pesquisa na web.

- String de Busca: termos utilizados na pesquisa:

- *artifact, artefact, documentation, document*.
- *extreme programming, scrum, agile*.
- *case study, survey*.
- String definida:

(artifact OR artefact OR documentation OR document) AND (extreme programming OR scrum OR agile) AND (case study OR survey)

- Lista de Fontes: Periódicos disponíveis no portal da CAPES: *IEEE Xplore* (*http://ieeexplore.ieee.org*); *ACM Digital Library* (*http://portal.acm.org/dl.cfm*); *ScienceDirect* (*http://www.sciencedirect.com/*); e *SpringerLink* (*http://springerlink.metapress.com*). Livros sobre

desenvolvimento ágil publicados pelas editoras *John Wiley*, IBM e *MS Press*, além de trabalhos acadêmicos on-line.

2.4. Seleção das Fontes após Avaliação:

- Não serão definidas restrições com relação à data inicial de publicação para os artigos nas fontes de informação.

3. Seleção dos Estudos

3.1. Definição dos Estudos:

- Critérios de Inclusão de Estudos: estudo sobre documentação em metodologias ágeis.

- Critérios de Exclusão de Estudos: estudo sobre documentação em outras metodologias não consideradas ágeis.

- Procedimentos para Seleção dos Estudos: um pesquisador executará a pesquisa aplicando a estratégia definida para identificar os estudos iniciais de interesse. Os resultados iniciais serão revisados por, no mínimo, outro pesquisador. As divergências entre os revisores quanto a inclusão ou exclusão de estudos serão discutidas e caso não se chegue a um consenso o estudo deverá ser incluído.

3.2. Execução da Seleção:

- Avaliação da Qualidade dos Estudos: os estudos deverão descrever a utilização do artefato junto à metodologia ágil, visando a documentação e eventualmente a manutenção.

- Revisão da Seleção: os artigos selecionados nos estudos iniciais serão revisados na íntegra por um dos pesquisadores, para então se proceder a extração das informações.

4. Extração das Informações

4.1. Definição dos Critérios de Inclusão e Exclusão da Informação: serão extraídos de cada estudo incluído na seleção:

- Informações para a referência e identificação;
- Especificação do artefato
- Especificação da metodologia ágil utilizada;
- Descrição do processo de utilização ou finalidade do artefato.

Referências

[BIO05] BIOLCHINI, J.; MIAN, P.; NATALI, A.; TRAVASSOS, G. "Systematic Review in Software Engineering: Relevance and Utility". Technical Report, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2005, 30p.

[KIT04] KITCHENHAM, B. "Procedures for Performing Systematic Reviews". Technical Report, Department of Computer Science, Keele University, National ICT Australia, 2004, 28p.

APÊNDICE B – RELAÇÃO DE ARTEFATOS DAS METODOLOGIAS ÁGEIS IDENTIFICADOS NO LEVANTAMENTO

Artefatos do Desenvolvimento Ágil em Geral

Classificação	Artefatos
Definição e gerenciamento do projeto	Modelos de contrato; Resumo de gerenciamento; Plano de *release*; Plano de entrega; Tarefas; Listas de ação; Manual do projeto/diretrizes da equipe; Relatório de progresso; Relatórios mensais de *status* do projeto; Relatório final.
Requisitos e especificação	Documento de visão; Definição do projeto; Visão geral do projeto; Visão geral do sistema; Estórias (*user stories*); Requisitos; Requisitos não-funcionais; Requisitos de sistema; Especificações; Especificação de requisitos; Documento de requisitos; Especificação funcional; Especificação da interface do usuário; Comportamento temporal; Modelo de caso de uso ou estórias; Casos de uso; Glossário.
Projeto e arquitetura	Projeto do sistema; Documentação de sistema; Documentos de projeto detalhado; Diagrama de contexto do sistema; Hierarquia de classes; Diagrama de interação de classes; Diagramas de objeto; Modelo de dados; Acesso ao banco de dados / transações; Diagramas de sequência; Documentos de arquitetura; Visão geral da arquitetura; Diagrama de revisão da arquitetura; Projeto da interface do usuário; Integração com sistemas vizinhos; Modelo de componente; Descrições de componente.
Codificação	Código/comentários no código.
Testes	Estratégia de teste; Plano de teste de aceitação; Testes de unidade; Casos de uso de testes; Casos de teste; Concepção dos testes.
Implantação e operação	Plano de implantação; Implantação; Metas de incremento; Documentação de operação; Diretrizes de operações; Modelo operacional; *Trouble-shooting*; Migração de funcionalidade; Migração de dados.
Padrões e diretrizes	Decisões de projeto; Lista de padrões; Diretrizes de codificação; Diretrizes e convenções nominadas; Notas de reunião; Notas dos membros da equipe; Documentação de suporte.
Manutenção	Documentação de manutenção.
Utilização	Documentação do usuário; Manual de instalação e administração; Diretrizes / Conceitos de uso; Arquivos de ajuda; Manual do usuário; Livro de receitas (*cookbook*); Tutorial.

Artefatos de *Scrum*

Classificação	Artefatos
Definição e gerenciamento do projeto	Dados do projeto; *Product backlog*; *Sprint backlog*; Plano de iteração; *Project increment*; Tarefas do projeto; *Burndown chart*; Relatório diário de status; *Project reporting*; *Workshops* de reflexão; *Impediment list*.
Requisitos e especificação	Requisitos de usabilidade; Requisitos de *workflow*; Requisitos de interface de usuário; Documentos de especificação.
Projeto e arquitetura	Esboços do projeto; Protótipos em papel; Imagens de representações; Documentos; Modelos; Programas.
Codificação	Código/comentários no código.
Testes	Casos de teste.

Artefatos de *Extreme Programming* (XP)

Classificação	Artefatos
Definição e gerenciamento do projeto	Estimativas das tarefas; Tarefas; Tarefas técnicas; Restrições; Plano de *release*; *Releases*; Instruções sobre *release*; Plano de iteração; Plano global de orçamento; Relatórios sobre progressos; Registro de tempo de trabalho na tarefa; Outras métricas; Monitoramento de resultados; Relatórios e notas sobre reuniões; Atas de reuniões; Anotações de conversas; Documentação adicional a partir de conversas.
Requisitos e especificação	Estórias (*user stories*); Estimativas das estórias; Novas versões das estórias; Estórias compartilhadas (*shared stories*); Metáfora.
Projeto e arquitetura	Projeto – CRC, UML (esboços); Cartão CRC; Esboço UML; Diagrama de classes (UML); Descrição de construção; Modelo de dados (ER); Descrição do banco de dados; Descrição sobre formatos de arquivo interno; Documentos de projeto (produzidos ao final); Defeitos e dados associados; Descrição arquitetural; Protótipo de arquitetura; Protótipo (*spike*).
Codificação	Código de unidade; Código de sistema; Código/comentários no código.
Testes	Descrição de estórias de testes; Plano de testes de aceitação; Testes de aceitação; Testes de unidade; Dados de teste; Código de teste; Código de casos de teste; Resultados de teste.
Padrões e diretrizes	Documentação de Suporte; Padrões de codificação; Regras próprias; *Workspace* (desenvolvimento e facilidades); Ferramentas de *framework* de testes; Ferramenta de gerenciamento de código; Instruções sobre ferramentas; Javadoc.
Utilização	Ajuda on-line; Tutorial; Manual do Usuário.

APÊNDICE C – SÍNTESE DOS ARTEFATOS ENCONTRADOS EM MÉTODOS ÁGEIS

Relação obtida somando-se as sublistas dos artefatos ágeis encontrados em cada metodologia e retirando-se as redundâncias semânticas:

Nome do Artefato (em Português)	Nome do Artefato (em Inglês)
Definição e Gerenciamento do Projeto	
1) Modelos de contrato	Contract models
2) Resumo de gerenciamento	Management summary
3) Plano global de orçamento	Overall plan – budget
4) Plano de release	Release plan
5) Releases	Release
6) Instruções sobre release	Release instructions
7) Product backlog	Product backlog
8) Sprint backlog	Sprint backlog
9) Plano de iteração	Iteration plan
10) Tarefas	Tasks
11) Manual do projeto/diretrizes da equipe	Project manual / team guidelines
12) Relatório de progresso	Progress reports
13) Burndown chart	Burndown chart
14) Relatório final	Final report
15) Relatórios e notas sobre reuniões	Reports and notes on meetings
16) Outras métricas	Other metrics
17) Project increment	Project increment
18) Impediment list	Impediment list
19) Workshops de reflexão	Reflection workshops
Requisitos e Especificação	
20) Documento de visão	Executive Overview/Vision Statement
21) Visão geral do projeto	Project overview
22) Estórias	User stories
23) Requisitos	Requirements
24) Especificações	Specifications
25) Restrições	Constraints
26) Comportamento temporal	Timed behaviour
27) Casos de uso	Use cases
28) Metáfora	Metaphor
29) Glossário	Glossary
Projeto e Arquitetura	
30) Projeto do sistema	System design
31) Diagrama de contexto do sistema	System Context Diagram
32) Diagrama de classes (UML)	Class Diagram (UML)
33) Diagramas de objeto	Object diagrams
34) Modelo de dados (ER)	Data model
35) Acesso ao banco de dados / transações	Database access / transactions
36) Diagramas de sequência	Sequence diagrams
37) Descrição arquitetural	Architectural description
38) Diagrama de revisão da arquitetura	Architecture Overview Diagram
39) Projeto da interface do usuário	User interface design / event management
40) Imagens de representações	Mock-ups
41) Integração com sistemas vizinhos	Integration with neighbouring systems
42) Modelo de componente	Component Model
43) Cartão CRC	CRC card
44) Esboço UML	UML sketch
45) Descrição de construção	Build description

46) Descrição sobre formatos de arquivo interno	Description over internal file formats
47) Defeitos e dados associados	Defects and associated data
48) Protótipo de arquitetura	Prototype architecture
49) Protótipos em papel	Paper prototypes
Codificação	
50) Código/comentários no código	Source code
Testes	
51) Estratégia de teste	Test Strategy
52) Testes de aceitação	Acceptance tests
53) Testes de unidade	Unit tests
54) Resultados de teste	Test results
Implantação e Operação	
55) Plano de implantação	Deployment Plan
56) Metas de incremento	Increment Goals
57) Documentação de operação	Operations documentation
58) Modelo operacional	Operational Model
59) Trouble-shooting	Trouble-shooting
60) Migração de funcionalidade	Functionality migration
61) Migração de dados	Data migration
Padrões e Diretrizes	
62) Decisões de projeto	Design decisions
63) Lista de padrões	Standards
64) Diretrizes de codificação	Coding Guidelines
65) Diretrizes e convenções nominadas	Guidelines and naming conventions
66) Notas dos membros da equipe	Team member minutes
67) Documentação de suporte	
68) Instruções sobre ferramentas	Instructions about tools
69) Javadoc	Javadoc
Manutenção	
70) Documentação de manutenção	Maintenance documentation
Utilização	
71) Manual de instalação e administração	Installation and administrator manual
72) Diretrizes / Conceitos de uso	Usage guidelines / concepts
73) Arquivos de ajuda	Help files
74) Manual do usuário	User manual
75) Livro de receitas (cookbook)	Cookbook
76) Tutorial	Tutorial

APÊNDICE D – PROTOCOLO DA REVISÃO SISTEMÁTICA SOBRE TIPOS DE CONHECIMENTO

PROTOCOLO DA REVISÃO SISTEMÁTICA

Mestrando: Fernando Selleri Silva
Orientador: Marcelo Blois Ribeiro
Tema: Uso de Representação de Conhecimento para Documentação em Metodologias Ágeis
PUCRS – Janeiro de 2009

1 Formulação da Questão de Pesquisa

1.1. Foco da Questão:

- A proposta para realização desta revisão sistemática tem como escopo a identificação dos tipos de conhecimento presentes no desenvolvimento de software. Neste sentido, a questão principal que norteará a presente revisão sistemática foi definida.

1.2. Amplitude e Qualidade da Questão:

- Problema: Encontrar os tipos de conhecimento envolvidos no desenvolvimento e manutenção de software.

- Questão de pesquisa: Quais os tipos de conhecimento associados ao desenvolvimento e manutenção de sistemas?

- Palavras-chave e sinônimos: tipo de conhecimento, desenvolvimento, engenharia, software, sistema. (em inglês: *type of knowledge, kind of knowledge, software, system, development, engineering*).

- Intervenção: desenvolvimento ou manutenção de software.

- Efeito: tipos de conhecimento relacionados ao desenvolvimento ou manutenção de software.

- População: projetos de desenvolvimento ou manutenção de software.

- Projeto Experimental: serão considerados trabalhos que utilizem como abordagem estudos de caso ou pesquisas de opinião, preferencialmente.

2. Seleção das Fontes

2.1. Definição dos Critérios de Seleção das Fontes:

- Pesquisa em bases de dados eletrônicas, incluindo *journals*, anais de conferências, livros e bibliotecas virtuais, que publicam estudos relacionados à Engenharia de Software e Inteligência Artificial.

2.2. Língua dos Estudos: inglês e português (línguas utilizadas nas fontes de pesquisa)

2.3. Identificação das Fontes:

- Método de Busca das Fontes: mecanismos de pesquisa na web.

- String de Busca: termos utilizados na pesquisa:

- *type of knowledge, kind of knowledge*.
- *software, system*.
- *development, engineering*.
- Strings definidas:

String 1: *("type of knowledge" OR "kind of knowledge") AND (software OR system) AND (development OR engineering)*

String 2: *("type of knowledge" OR "kind of knowledge") AND ("software development" OR "software engineering")*

- Lista de Fontes: Periódicos disponíveis no portal da CAPES: *IEEE Xplore* (*http://ieeexplore.ieee.org*); *ACM Digital Library* (*http://portal.acm.org/dl.cfm*); *ScienceDirect* (*http://www.sciencedirect.com/*); e *SpringerLink* (*http://springerlink.metapress.com*). Biblioteca Digital da SBC (*http://www.sbc.org.br/bibliotecadigital/*).

2.4. Seleção das Fontes após Avaliação:

- Não serão definidas restrições com relação à data inicial de publicação para os artigos nas fontes de informação.

3. Seleção dos Estudos

3.1. Definição dos Estudos:

- Critérios de Inclusão de Estudos: estudo que relata o conhecimento envolvido no desenvolvimento ou manutenção de software.

- Critérios de Exclusão de Estudos: estudo que não aborda o conhecimento envolvido no desenvolvimento ou manutenção de software.

- Procedimentos para Seleção dos Estudos: um pesquisador executará a pesquisa aplicando a estratégia definida para identificar os estudos iniciais de interesse. Os resultados iniciais serão revisados por, no mínimo, outro pesquisador. As divergências entre os revisores quanto a inclusão ou exclusão de estudos serão discutidas e caso não se chegue a um consenso o estudo deverá ser incluído.

3.2. Execução da Seleção:

- Avaliação da Qualidade dos Estudos: os estudos deverão descrever o tipo de conhecimento envolvido e em que etapa do desenvolvimento ou manutenção este é requerido ou utilizado.

- Revisão da Seleção: os artigos selecionados nos estudos iniciais serão revisados na íntegra por um dos pesquisadores, para então se proceder a extração das informações.

4. Extração das Informações

4.1. Definição dos Critérios de Inclusão e Exclusão da Informação: serão extraídos de cada estudo incluído na seleção:

- Informações para a referência e identificação;
- Especificação do tipo de conhecimento;
- Especificação da etapa ou artefatos relacionados ao tipo de conhecimento;
- Descrição do processo de utilização ou finalidade.

Referências

[BIO05] BIOLCHINI, J.; MIAN, P.; NATALI, A.; TRAVASSOS, G. “Systematic Review in Software Engineering: Relevance and Utility”. Technical Report, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2005, 30p.

[KIT04] KITCHENHAM, B. “Procedures for Performing Systematic Reviews”. Technical Report, Department of Computer Science, Keele University, National ICT Australia, 2004, 28p.

APÊNDICE E – RELAÇÃO DE TIPOS DE CONHECIMENTO IDENTIFICADOS NO LEVANTAMENTO

Relação completa dos tipos de conhecimento presentes no desenvolvimento de software, organizada de acordo com o grupo de conhecimento ao qual pertencem:

Grupo de Conhecimento	Tipo de Conhecimento
Conhecimento sobre Clientes	Processos Organizacionais; Estrutura Organizacional; Alocação de Profissionais à Estrutura Organizacional; Diretrizes e Normas; Missão da Organização; Informação sobre Clientes e Competidores.
Conhecimento sobre Dados Históricos da Organização	Gerência de Projeto; Métricas de Produto; Gerência de Qualidade; Métricas de Processo; Gerência de Riscos.
Conhecimento sobre o Domínio	Domínio da Aplicação; Regras de Negócio; Processo de Negócio.
Conhecimento sobre Referências da Literatura Técnica	Relatos sobre as melhores práticas da Indústria de Software; Indicação de Referências sobre Inovações Tecnológicas em Engenharia de Software; Relatos sobre as lições aprendidas da Indústria de Software; Indicação de Referências sobre Conhecimento Teórico em Engenharia de Software; Apresentações sobre clientes e pesquisa sobre tendências da indústria.
Conhecimento sobre Parceiros Técnicos	Restrições, Deficiências e Potenciais de Parceiros Técnicos.
Conhecimento sobre a Organização	Distribuição de Competências entre os Profissionais da Organização; Objetivos e Metas Organizacionais; Áreas Críticas para o Alcance dos Objetivos Organizacionais; Restrições, Deficiências e Potenciais da Organização; Processos Organizacionais; Missão da Organização; Diretrizes e Normas da Organização; Estrutura Organizacional; Relações Informais existentes entre os Profissionais da Organização; Contratos/Acordos com Fornecedores, Parceiros Técnicos e Clientes.
Conhecimento sobre Melhores Práticas no Desenvolvimento	Codificação; Testes; Técnicas de Testes utilizadas; Documentação de Software; Gerência de Qualidade; Processo de Solução de Problemas; Análise e Especificação de Requisitos de Software; Técnicas de elicitação de requisitos; Projeto (design) de Software; Projeto de Banco de Dados; Padrões de Projeto usados; Arquitetura do Sistema; Interação do sistema com seu ambiente; Gerência de Projeto; Planos de Projeto; Engenharia de Sistemas; Avaliação e Melhoria de Processo de Software; Modelagem de Processo; Ferramentas usadas; Linguagens e técnicas de programação; Implementação da aplicação; Treinamento; Operação de Sistema e Suporte a Usuários; Reutilização de Itens de Software; Gerência de Configuração; Gerência de Riscos; Processo de Fornecimento de Produto de Software.

Conhecimento sobre Manutenção	Manutenção de Software; Detalhes sobre a requisição de modificação; Opções identificadas para implementação da modificação; Negociação com relação ao prazo para realização da manutenção; Estimativa de esforço para realização da modificação; Documentos que foram modificados; Componentes do Software modificados; Inter-relacionamento entre sistemas; Inconsistências entre análise de requisitos e projeto; Oportunidades de re-reengenharia do software; Rastreabilidade entre artefatos; Documentação do processo de manutenção e de apoio usados; Acompanhamento da modificação; Processo de Manutenção.
Outros Conhecimentos	Processo de Software da Organização; Roteiros de Documentos com Exemplos de Uso; Itens de Software Úteis para Desenvolvimentos Futuros; Avaliação e Áreas para Melhoria do Processo de Software da Organização; Tipos de Software desenvolvidos na Organização; Respostas para as Perguntas mais Frequentes entre os Desenvolvedores; Negociação entre área do setor de tecnologia; Estimativa de Custos; Ajuda (manuais); Ferramentas adicionais (linguagens estrangeiras); Análises e interpretações sobre estratégias e planos.

APÊNDICE F – MATRIZ DE ASSOCIAÇÃO DOS ARTEFATOS ÁGEIS AO TIPO DE CONHECIMENTO GERADO

Metodologia: *Extreme Programming* (XP)

	Tipo de Conhecimento	Processos Organizacionais Cliente	Distribuição de Competências Org.	Restrições, Def. e Potenciais Org.	Diretrizes e Normas Org.	Gerência de Projeto	Gerência de Riscos	Restrições, Def. e Potenciais PT	Domínio da Aplicação	Codificação	Testes	Documentação de Software	Gerência de Qualidade	Análise e Especificação de Requisitos	Arquitetura do Sistema	Operação de Sist e Suporte a Usuários	Manutenção de Software	Detalhes sobre a req de modificação	Padrões de Projeto usados	Implementação da aplicação	Ferramentas usadas	Processo de Fornecimento de Produto	Técnicas de Testes utilizadas	Projeto (design) de Software	Projeto de Banco de Dados	Ajuda (manuais)	Gerência de Configuração	Linguagens e técn. de programação	Planos de Projeto	Estimativa de Custos	Negociação entre área do setor de	Avaliação e Melhoria de Proces. SW
Tipo de Artefato																															
Plano de release						X																X					X		X		
Release						X																X									
Instruções sobre release						X						X										X									
Plano de Iteração						X									X		X	X										X			
Tarefas			X			X								X						X				X							
Tarefas técnicas						X																		X							
Estimativas das tarefas						X																				X		X			
Relatórios e notas sobre reuniões		X				X				X		X		X	X				X	X											X
Atas de reuniões						X																									
Anotações de conversas						X								X																	
Documentação adicional a partir de conversas						X								X																	
Restrições				X		X		X																							
Plano global de orçamento						X	X															X							X		
Relatórios sobre progressos						X																									X
Registro de tempo de trabalho na tarefa						X																									X
Outras métricas						X																									X
Monitoramento de resultados						X																									X
Estória do Usuário (User Stories)						X			X					X														X	X	X	
Estimativas das estórias						X								X												X		X	X		
Novas versões das estórias						X								X																	
Estórias compartilhadas (shared stories)						X								X																	
Metáfora		X							X					X	X									X							
Descrição arquitetural										X		X			X									X							
Protótipo de arquitetura															X				X		X							X			
Protótipo (spike)															X																
Cartão CRC															X									X							
Esboço UML															X									X							
Projeto – CRC, UML (esboços)																								X							
Diagrama de Classe UML																								X							
Modelo de Dados (ER)																									X						
Descrição do banco de dados																									X						
Descrição de construção										X														X							
Descrição sobre formatos de arquivo interno										X														X							
Documentos de projeto (produzidos ao final)																								X							
Documentação de Suporte																X								X							
Defeitos e dados associados															X			X													
Código/comentários no código										X		X								X											
Código de unidade										X										X								X			
Código de sistema										X										X								X			
Descrição de estórias de testes											X												X								
Plano de testes de aceitação											X												X								
Testes de aceitação											X										X										
Testes de unidade										X	X																				
Dados de teste											X												X								
Código de teste										X	X																				
Código de casos de teste										X	X																				
Resultados de teste											X			X				X													
Padrões de codificação										X									X												
Regras próprias					X														X												

Workspace (desenvolvimento e facilidades)																	X		X										
Ferramentas de framework de testes										X							X		X										
Ferramenta de gerenciamento de código					X												X		X						X				
Instruções sobre ferramentas																	X		X										
Javadoc									X								X					X							
Manual do Usuário														X										X					
Tutorial														X										X					
Ajuda on-line																								X					

Metodologia: *Scrum*

	Tipo de Conhecimento	Processos Organizacionais Cliente	Distribuição de Competências Org.	Restrições, Def. e Potenciais Org.	Gerência de Projeto	Restrições, Def. e Potenciais PT	Domínio da Aplicação	Codificação	Testes	Documentação de Software	Processo de Solução de Problemas	Análise e Especificação de Requisitos	Arquitetura do Sistema	Padrões de Projeto usados	Implementação da aplicação	Ferramentas usadas	Processo de Fornecimento de Produto	Técnicas de Testes utilizadas	Projeto (design) de Software	Linguagens e técn. de programação	Planos de Projeto	Estimativa de Custos	Processo de Software da Organização
Tipo de Artefato																							
Dados do projeto					X					X													
Product backlog					X		X					X					X				X		
Sprint backlog			X		X										X							X	
Plano de iteração					X																		
Project increment															X		X						
Tarefas do projeto					X																	X	
Burndown chart					X																		
Relatório diário de status					X					X	X												
Project reporting					X					X													
Workshops de reflexão					X					X	X												
Impediment list				X	X	X					X												
Requisitos de usabilidade		X										X											
Requisitos de workflow												X							X				
Requisitos de interface de usuário												X			X								
Documentos de especificação										X		X											X
Casos de teste									X	X					X			X					
Esboços do projeto																			X				
Protótipos em papel																			X				
Imagens de representações																			X				
Documentos										X			X										
Modelos										X			X						X				
Programas					X					X						X			X				
Código/comentários no código								X		X				X	X				X	X			

APÊNDICE G – TABELA DE ARTEFATOS DAS METODOLOGIAS ÁGEIS

Metodologia	Categoria*	Propósito*	Tipo
XP	Documento	Produto do Trabalho	Estórias (user stories)
			Estimativas das estórias
			Novas versões das estórias
			Estórias compartilhadas (shared stories)
			Metáfora
			Projeto – CRC, UML (esboços)
			Cartão CRC
			Esboço UML
			Diagrama de classes (UML)
			Descrição de construção
			Modelo de dados (ER)
			Descrição do banco de dados
			Descrição sobre formatos de arquivo interno
			Documentos de projeto (produzidos ao final)
			Descrição arquitetural
			Protótipo de arquitetura
			Protótipo (spike)
		Documento do Processo	Estimativas das tarefas
			Tarefas
			Tarefas técnicas
			Restrições
			Plano de release
			Instruções sobre release
			Plano de iteração
			Plano global de orçamento
			Relatórios sobre progressos
			Registro de tempo de trabalho na tarefa
			Outras métricas
			Monitoramento de resultados
			Relatórios e notas sobre reuniões
			Atas de reuniões
			Anotações de conversas
			Documentação adicional a partir de conversas
			Descrição de estórias de testes
			Plano de testes de aceitação
			Testes de aceitação
			Testes de unidade
			Dados de teste
			Resultados de teste
			Defeitos e dados associados
		Documento de Apoio	Documentação de Suporte
			Padrões de codificação
			Regras próprias
			Workspace (desenvolvimento e facilidades)
			Instruções sobre ferramentas
			Javadoc
			Ajuda on-line
			Tutorial
			Manual do Usuário
	Componente	Componente de Implantação	Releases
		Componente de Produto de Trabalho	Código de teste
			Código de casos de teste
			Código de unidade
			Código de sistema
			Código/comentários no código

		Componente de Execução	Ferramentas de framework de testes
			Ferramenta de gerenciamento de código
Scrum	Documento	Produto do Trabalho	Requisitos de usabilidade
			Requisitos de workflow
			Requisitos de interface de usuário
			Documentos de especificação
			Esboços do projeto
			Protótipos em papel
			Imagens de representações
			Documentos
			Modelos
		Documento do Processo	Dados do projeto
			Product backlog
			Sprint backlog
			Plano de iteração
			Project increment
			Tarefas do projeto
			Burndown chart
			Relatório diário de status
			Project reporting
			Workshops de reflexão
			Impediment list
			Casos de teste
	Componente	Componente de Produto de Trabalho	Código/comentários no código
		Componente de Execução	Programas

*Fonte: Extraído de [ANQ07]

APÊNDICE H – FERRAMENTAS PARA DESENVOLVIMENTO DE SOFTWARE COM METODOLOGIAS ÁGEIS

Ferramenta	Metodologia	Licença	Linguagem	Plataforma	Referência
AccuRev	Ágil	Comercial	Java	Desktop/Web	http://www.accurev.com/
Agile PRM	Ágil	Comercial	Java	Web	http://pmr.sourceforge.net/
AgileLog	Ágil	Comercial	Java	Web	http://www.agilelog.net/
AgileTrack	Ágil	Comercial	Java	Desktop	http://agiletrack.net/
ExtremePlanner	Ágil	Comercial	Java	Web	http://www.extremeplanner.com/
Mingle	Ágil	Comercial	-	Web	http://www.thoughtworks.com/mingle
OutSystemsEE	Ágil	Comercial	.NET	Desktop	http://www.outsystems.com/agile/
Planning Poker	Ágil	Comercial	JavaScript	Online	http://www.planningpoker.com/
Project Locker	Ágil	Comercial	-	Online	http://www.projectlocker.com/
ProjectEngine	Ágil	Comercial	Java	Desktop/Web	http://www.projectengine.nu/
RADvolution	Ágil	Comercial	.NET	Desktop	http://www.developguidance.com/
Rally	Ágil	Comercial	-	Web	http://www.rallydev.com/
Silver Catalyst	Ágil	Comercial	-	Desktop/Web	http://www.silverstripesoftware.com/
SpiraPlan	Ágil	Comercial	ASP	Web	http://www.inflectra.com/SpiraPlan/
TargetProcess	Ágil	Comercial	.NET	Web	http://www.targetprocess.com/
VersionOne	Ágil	Comercial	.NET	Web	http://www.versionone.com/
UnitMetrics	Ágil	CPL	Java	Desktop	http://unitmetrics.sourceforge.net/
Agile Grid	Ágil	EclipsePL	Java	Desktop	http://agilegrid.sourceforge.net/
Agile RCP	Ágil	EclipsePL	Java	Framework	http://agilercp.metafinanz.de/
JRequisite	Ágil	EclipsePL	Java	Desktop	http://jrequisite.sourceforge.net/
jAPS	Ágil	GPL	Java	Web	http://www.japsportal.org/
Open PM	Ágil	GPL	Java	Web	http://openprojectmanager.com
Point-Agile	Ágil	GPL	Java	Web	http://sourceforge.net/projects/pointagile/
PPTS	Ágil	GPL	PHP	Web	http://ses-ppts.sourceforge.net/
PrjPlanner	Ágil	GPL	Python	Desktop	http://prjplanner.sourceforge.net/
Project Dune	Ágil	GPL	Java	Web	http://projectdune.org
Yoxel	Ágil	GPL	PHP	Web	http://yoxel.com/
BPMspace	Ágil	LGPL	Java	Web	http://www.bpmspace.org/
Floranta	Ágil	LGPL	Java	Web	http://www.floranta.com/
TrackIt	Ágil	LGPL	Java	Web	http://trackit.sourceforge.net/
Agilefant	Ágil	MITL	Java	Web	http://www.agilefant.org/
FDDPMA	FDD	GPL	Java	Web	http://www.fddpma.net/
BananaScrum	Scrum	Comercial	RubyOnRails	Online	http://www.bananascrum.com/
ScrumDesk	Scrum	Comercial	.NET	Desktop/Web	http://www.scrumdesk.com/
ScrumWorks	Scrum	Comercial	-	Desktop/Web	http://danube.com/scrumworks
VirtualScrumBoard	Scrum	Comercial	.NET	Desktop	http://www.virtualscrumboard.com/
Agilo	Scrum	ApacheSL	Python	Web	http://www.agile42.com/
IceScrum	Scrum	ApacheSL	Java	Web	http://www.icescrum.org/
Scrum Time	Scrum	ApacheSL	Groovy	Web	http://scrumtime.sourceforge.net/
Scrinch	Scrum	GPL	Java	Desktop	http://scrinch.sourceforge.net/
Scrum Vision	Scrum	GPL	Java	Desktop	http://scrumvision.sourceforge.net/
Teamwork	Scrum	GPL	Java	Web	http://www.twproject.com/
BagLock	Scrum	MITL	RubyOnRails	Web	http://baglock.wiki.sourceforge.net/
ScrumPlanPoker	Scrum	MPL	ASP	Web	http://scrumpoker.sourceforge.net/
NBehave	TDD	BSD	.NET	Desktop	http://nbehave.org/
DevPlanner	XP	Comercial	-	Desktop	http://devplanner.com/
XPmanager	XP	Comercial	Delphi	Desktop	http://www.naphta.com.br/xpmanager.html
XTremeTeam	XP	Comercial	ASP	Online	http://www.xscalable.com/
XP Studio	XP	EclipsePL	Java	Desktop	http://xpstudio.sourceforge.net/
Baba XP	XP	GPL	Java	Web	http://babaxp.dev.java.net/
Story Server	XP	GPL	Java	Web	http://storyserver.sourceforge.net/

User Story.NET	XP	GPL	ASP	Web	http://userstorynet.sourceforge.net/
US-FeatureTracker	XP	GPL	VB	Desktop	http://featuretracker.sourceforge.net/
XP4IDE	XP	GPL	Java	Desktop	http://xp4ide.sourceforge.net/
XPCGI	XP	GPL	Perl	Web	http://xpcgi.sourceforge.net/
XPMT	XP	GPL	PHP	Web	http://www.tegonal.com/de/os/xpmt/
XPWeb	XP	GPL	PHP	Web	http://xpweb.sourceforge.net/
XPlanner	XP	LGPL	Java	Web	http://xplanner.org/
XP Roundup	XP	PythonSL	Python	Desktop	http://xproundup.sourceforge.net/

Fontes pesquisadas:
- SourceForge: http://sourceforge.net
- XProgramming: http://www.xprogramming.com/software.htm
- Software Development Tools Directory: http://www.softdevtools.com
- Agile Tools: http://www.agile-tools.net/
- IEEE Xplore: http://ieeexplore.ieee.org
- ACM Digital Library: http://portal.acm.org/dl.cfm
- ScienceDirect: http://www.sciencedirect.com
- SpringerLink: http://springerlink.metapress.com

APÊNDICE I – TREINAMENTO PARA OS PARTICIPANTES DO EXPERIMENTO

Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação
Curso de Mestrado
Intelligent Systems Engineering Group – ISEG

Estudo Experimental

Representação de Conhecimento em Metodologias Ágeis

Aluno: Fernando Selleri Silva – fernando.selleri@pucrs.br
Orientador: Prof. Dr. Marcelo Blois Ribeiro

Maio de 2009

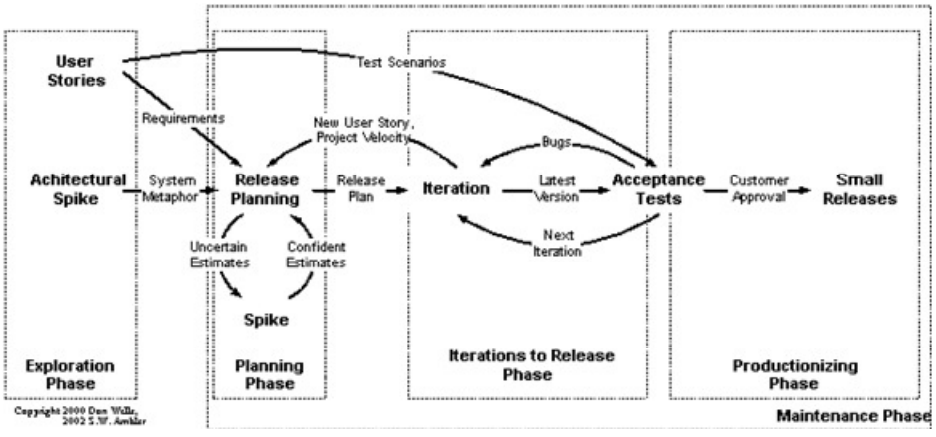
Agenda

- Metodologias ágeis
  - *Extreme Programming*
- Estudo Experimental
  - Definição
  - Planejamento
  - Execução

Metodologias Ágeis

- Manifesto Ágil (Valores)
  - **Indivíduos e interações** acima de processos e ferramentas
  - **Software funcionando** acima de documentação abrangente
  - **Colaboração com o cliente** acima de negociação de contratos
  - **Resposta a mudanças** acima de obediência a um plano
- Exemplos
  - *Extreme Programming* (XP); *Scrum; Crystal; Feature-Driven Development* (FDD); *Lean Software Development* (LSD); *Dynamic Systems Development Methodology* (DSDM); *Agile Modeling* (AM); *Adaptive Software Development* (ASD); *Agile Unified Process* (AUP); *Test-Driven Development* (TDD); *Personal Software Planning* (PSP); *Team Software Planning* (TSP); *Iconix.*

Extreme Programming (XP)



Estudo Experimental - Definição

- Objetivo Global
  - Comparar no *Extreme Programming* o esforço de produção (representação) e a precisão da compreensão (recuperação) da documentação utilizando a representação de conhecimento com relação ao processo atual de documentação da metodologia.

Estudo Experimental - Definição

- Objetivo do Estudo
  - ***Comparar*** no *Extreme Programming* a compreensão (recuperação) da documentação com a representação de conhecimento com a compreensão (recuperação) sem a representação,
  - ***Com o propósito*** de caracterizar o tempo gasto no uso e os elementos recuperados por cada abordagem,
  - ***Com foco*** no esforço e na precisão,
  - ***Sob o ponto de vista*** do arquiteto de software,
  - ***No contexto*** de manutenção de um sistema de informação desenvolvido por estudantes (*toy example*) no domínio de um gerenciador de projetos de estágio e monografia de uma universidade.

Estudo Experimental - Definição

- Objetivo da Medição
  - Qual o *esforço* (medido em minutos por participante) necessário para indexação (representação) dos artefatos de documentação com e sem o uso da representação de conhecimento?
  - Qual a *precisão* definida por meio da corretude e completude dos artefatos recuperados com e sem o uso da representação de conhecimento, quanto aos relacionamentos no sistema?

Estudo Experimental - Definição

- Métricas
  - Tempo gasto em minutos por cada participante para indexar (representar) a documentação em cada abordagem;
  - Precisão na compreensão (recuperação) da documentação em cada abordagem para identificar os artefatos a serem impactados.
    - Precisão: razão entre o conjunto correto de artefatos recuperados e o conjunto total de artefatos relacionados.

Estudo Experimental - Planejamento

- Participantes
  - Alunos do último período da graduação na UNEMAT
- Seleção dos indivíduos para cada abordagem de forma aleatória

Estudo Experimental - Planejamento

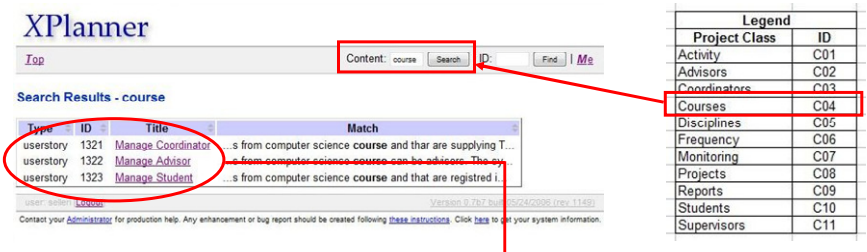
- Instrumentação
  - Objetos
    - Modelo de Domínio de um sistema
    - Plano de Iteração, estórias e tarefas
    - Matriz de rastreabilidade (MS Excel)
    - Ferramenta Ágil (XPlannerKnOWLedge)
    - Ferramenta CASE (ArgoUML+ONTrace)
  - Guias
    - Treinamento
    - Tutorial sobre os procedimentos para o experimento
  - Métricas
    - Por meio de formulário preenchido pelos participantes

Estudo Experimental - Execução

- Treinamento para os participantes que utilizarão a abordagem de indexação sem a proposta de representação de conhecimento

Estudo Experimental - Execução

- Etapa 1 – Representação (associar classes às *user stories*)



Legend	
Project Class	ID
Activity	C01
Advisors	C02
Coordinators	C03
Courses	C04
Disciplines	C05
Frequency	C06
Monitoring	C07
Projects	C08
Reports	C09
Students	C10
Supervisors	C11

Start Time:
End Time:

User Story	Dependency	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11
ManageCoordinator	UE01				X							
ManageAdvisor	UE02				X							
ManageStudent	UE03				X							
ManageSupervisor	UE04											
ElaborateTraineeshipProject	UE05											
ElaborateMonographProject	UE06											
GenerateMonitoringReport	UE07											
GenerateActivityReport	UE08											
GenerateFrequencyReport	UE09											

Estudo Experimental - Execução

- Etapa 2 – Recuperação (identificar *user story* e classes associadas)

Start Time:
End Time:

User Story	Dependency	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11
ManageCoordinator	UE01			X	X	X						
ManageAdvisor	UE02		X	X	X							
ManageStudent	UE03			X	X	X					X	
ManageSupervisor	UE04		X	X								X
ElaborateTraineeshipProject	UE05		X	X					X		X	
ElaborateMonographProject	UE06		X	X					X		X	
GenerateMonitoringReport	UE07		X	X				X	X	X	X	
GenerateActivityReport	UE08	X	X	X					X	X	X	X
GenerateFrequencyReport	UE09		X	X			X		X	X	X	

Estudo Experimental - Execução

- Etapa 2 – Recuperação (para cada classe, listar as *user stories* associadas – Classe 02)

User Story	Dependency	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11
ManageAdvisor	UE02		X	X	X							
ManageSupervisor	UE04		X	X								X
ElaborateTraineeshipProject	UE05		X	X					X		X	
ElaborateMonographProject	UE06		X	X					X		X	
GenerateMonitoringReport	UE07		X	X				X	X	X	X	
GenerateActivityReport	UE08	X	X	X					X	X	X	X
GenerateFrequencyReport	UE09		X	X			X		X	X	X	

Associations Recovery							
Artifact Element	Related Artifact						
US02 - ManageAdvisor	UE04	UE05	UE06	UE07	UE08	UE09	

Estudo Experimental - Execução

- Etapa 2 – Recuperação (para cada classe, listar as *user stories* associadas – Classe 03)

User Story	Dependency	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11
ManageCoordinator	UE01			X	X	X						
ManageAdvisor	UE02		X	X	X							
ManageStudent	UE03			X	X	X					X	
ManageSupervisor	UE04		X	X								X
ElaborateTraineeshipProject	UE05		X	X					X		X	
ElaborateMonographProject	UE06		X	X					X		X	
GenerateMonitoringReport	UE07		X	X				X	X	X	X	
GenerateActivityReport	UE08	X	X	X					X	X	X	X
GenerateFrequencyReport	UE09		X	X			X		X	X	X	

Associations Recovery								
Artifact Element	Related Artifact							
US02 - ManageAdvisor	UE04	UE05	UE06	UE07	UE08	UE09	UE01	UE03

15

Estudo Experimental - Execução

- Etapa 2 – Recuperação (para cada classe, listar as *user stories* associadas – Classe 04)

User Story	Dependency	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11
ManageCoordinator	UE01			X	X	X						
ManageAdvisor	UE02		X	X	X							
ManageStudent	UE03			X	X	X					X	

Associations Recovery								
Artifact Element	Related Artifact							
US02 - ManageAdvisor	UE04	UE05	UE06	UE07	UE08	UE09	UE01	UE03

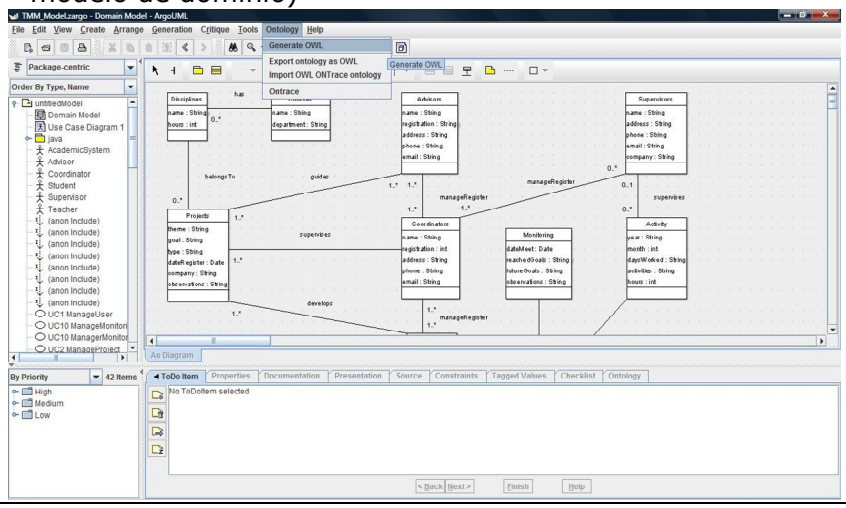
16

Estudo Experimental - Execução

- Treinamento para os participantes que utilizarão a abordagem de indexação com a proposta de representação de conhecimento

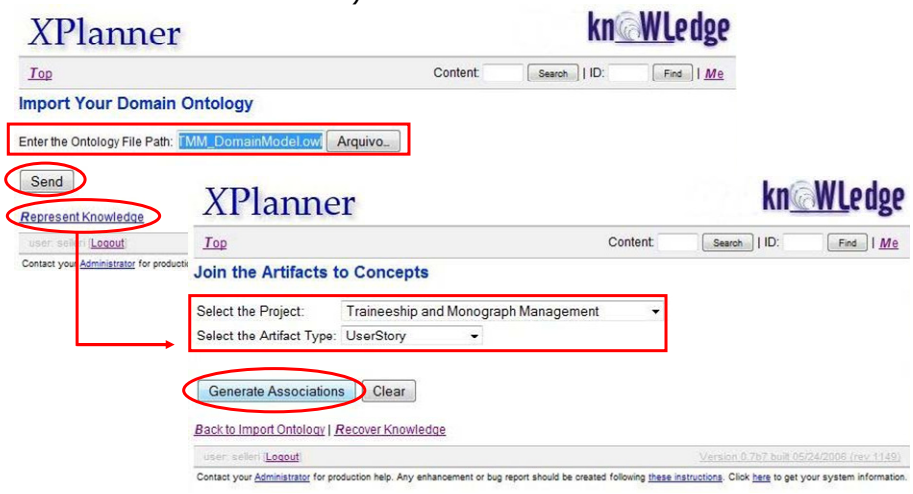
Estudo Experimental - Execução

- Etapa 1 – Representação (gerar a ontologia a partir do modelo de domínio)



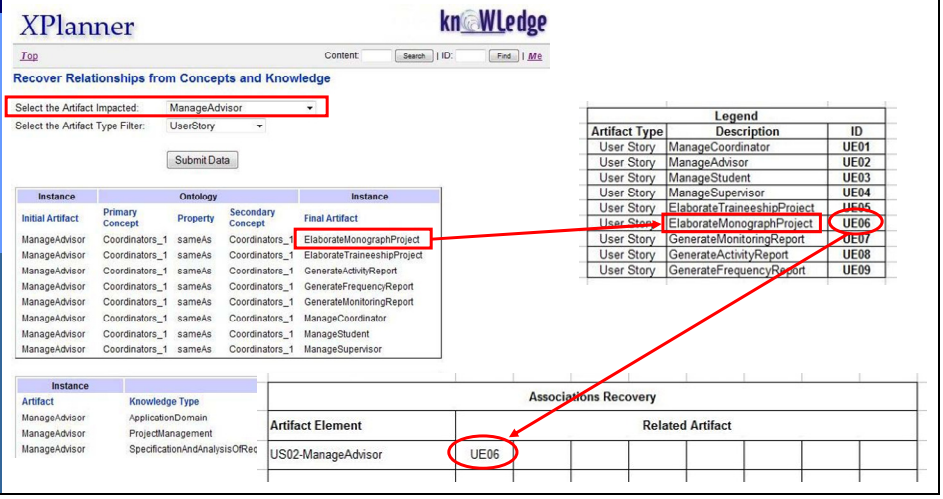
Estudo Experimental - Execução

- Etapa 1 – Representação (associar *user stories* aos conceitos do domínio)



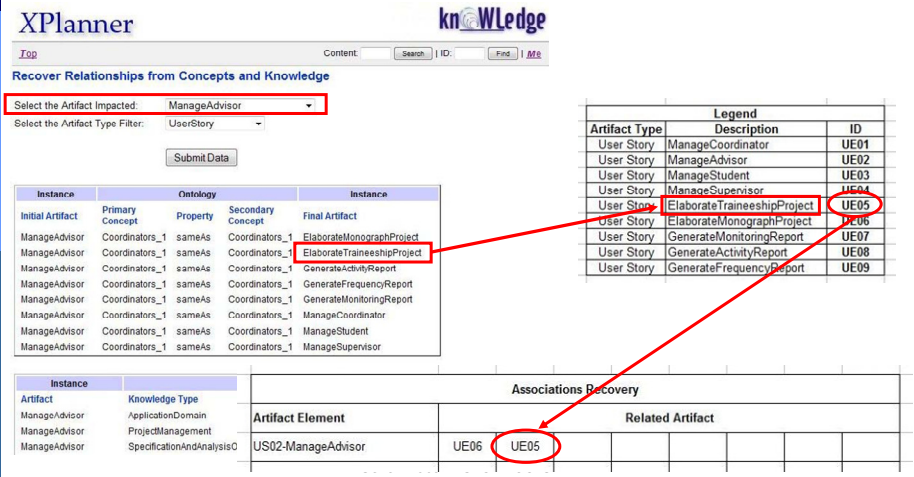
Estudo Experimental - Execução

- Etapa 2 – Recuperação (selecionar *user story* e recuperar artefatos relacionados)



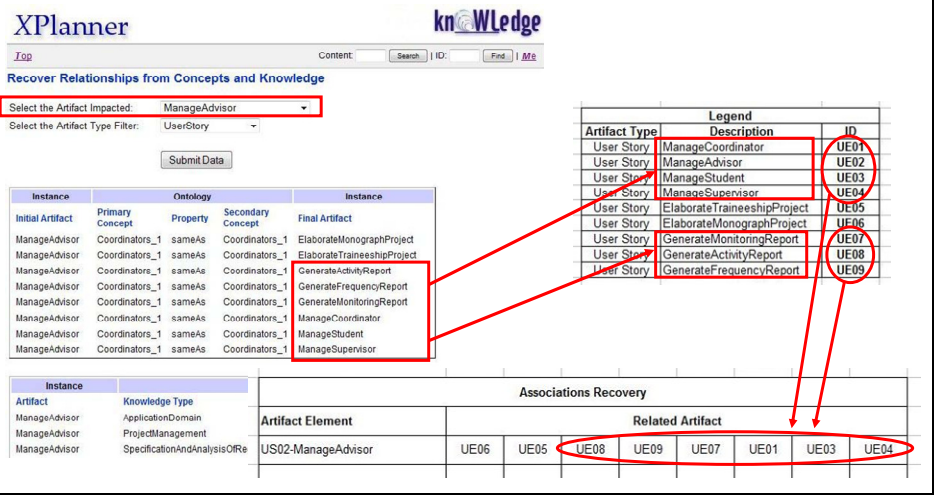
Estudo Experimental - Execução

- Etapa 2 – Recuperação (selecionar *user story* e recuperar artefatos relacionados)



Estudo Experimental - Execução

- Etapa 2 – Recuperação (selecionar *user story* e recuperar artefatos relacionados)



Estudo Experimental - Execução

- Instruções Gerais
  - Fazer registro do horário de início e término de cada etapa;
  - Manter os celulares desligados;
  - Evitar interação com o ambiente externo;
  - Não é possível a interação entre os grupos;
  - Em caso de dúvidas quanto aos procedimentos, perguntar somente para os responsáveis pelo experimento.

Estudo Experimental - Execução

- Observações
  - O resultado demanda atenção e qualidade, e não agilidade;
  - Na documentação do experimento será mantido o anonimato dos participantes.
- Dúvidas / Sugestões

Obrigado!

APÊNDICE J – TREINAMENTO PARA INDEXAÇÃO DA DOCUMENTAÇÃO COM REPRESENTAÇÃO DE CONHECIMENTO

ESTUDO EXPERIMENTAL

ABORDAGEM COM REPRESENTAÇÃO DE CONHECIMENTO

Representação de Conhecimento em Metodologias Ágeis
Mestrando: Fernando Selleri Silva

Neste documento são apresentadas as instruções necessárias para condução do experimento, utilizando o ArgoUML e o XPlanner, com suas novas funcionalidades, para a abordagem de indexação com a proposta de representação de conhecimento.

Atividade 01: Associação dos elementos de artefatos aos conceitos do domínio

1. **Registrar o horário de início da atividade**
2. Abra o projeto "TMM_Model.zargo" do ArgoUML, contendo o Modelo de Domínio.
3. Para **gerar a ontologia**, clique em **Ontology** no menu superior e selecione a opção **Generate OWL**. Selecione então o Modelo de Domínio a partir do qual será gerada a ontologia.
4. Depois de gerada a ontologia, clique em **Ontology** no menu superior e selecione a opção **Export ontology as OWL**. Salve o arquivo na área de trabalho do computador.
5. Na ferramenta XPlanner, clique em **AgileKnOWLedge** no menu inferior e na página que se abre clique no botão **Arquivo**, selecione o arquivo referente a ontologia gerada e clique no botão **Send**. Depois de carregado o arquivo, clique em **Represent Knowledge** no menu inferior e na página que se abre selecione o projeto "Traineeship and Monograph Management", o tipo de artefato "UserStory" e clique no botão **Generate Associations**.
6. Após o carregamento da página, **registrar o horário de término da atividade.**

Atividade 02: Recuperação dos elementos de artefatos relacionados.

1. Para recuperar as associações criadas, clique em **Recover Knowledge** no menu inferior e na página que se abre selecione o artefato a ser impactado, o tipo de filtro por "UserStory" e clique no botão **SubmitData**.
2. Os resultados compreendem os elementos de artefato que se relacionam diretamente ou indiretamente com o elemento selecionado, por meio dos conceitos da ontologia.
3. Para cada elemento de artefato definido na planilha "RecoverKnowledge" do arquivo "KnowledgeRepresentation_With.xls", completar a lista de elementos associados, com base na legenda apresentada na planilha "Legend".

APÊNDICE K – TREINAMENTO PARA INDEXAÇÃO DA DOCUMENTAÇÃO SEM REPRESENTAÇÃO DE CONHECIMENTO

ESTUDO EXPERIMENTAL

ABORDAGEM SEM REPRESENTAÇÃO DE CONHECIMENTO

Representação de Conhecimento em Metodologias Ágeis
Mestrando: Fernando Selleri Silva

Neste documento são apresentadas as instruções necessárias para condução do experimento, utilizando a abordagem de indexação sem a proposta de representação de conhecimento.

Atividade 01: Associação dos elementos de artefatos às classes de projeto

1. **Registrar o horário de início da atividade**
2. Na ferramenta XPlanner, para cada classe de projeto faça uma busca utilizando o campo **Content** e o botão **Search** para identificar as *user stories* que estão relacionadas.
3. A relação das *user stories* definidas encontram-se na planilha “RepresentKnowledge” do arquivo “KnowledgeRepresentation_Without.xls” e a relação de classes de projeto estão na planilha “Legend”. Para cada classe definida nesta matriz, identificar quais as *user stories* relacionadas.
4. Repetir as atividades acima para cada classe.
5. **Registrar o horário de término da atividade.**

Atividade 02: Recuperação dos elementos de artefatos relacionados.

1. Após relacionar as classes as *user stories*, é necessário recuperar as associações.
2. Para cada *user story* definida na planilha “RepresentKnowledge”, identificar quais as classes que estão relacionadas a mesma.
3. Para cada classe identificada, recuperar todas as *user stories* que ela relaciona utilizando a opção de filtro na tabela.
4. Preencher a planilha “RecoverKnowledge” com as *user stories* associadas a classe.
5. Repetir o procedimento para todas as *user stories* definidas nesta planilha.

APÊNDICE L – FORMULÁRIO PARA PREENCHIMENTO DOS RESULTADOS DO EXPERIMENTO

Associations Recovery								
Artifact Element	**Related Artifact**							
US01-ManageCoordinator								
US04-ManageSupervisor								
US06-ElaborateMonographProject								
US08-GenerateActivityReport								

APÊNDICE M – AVALIAÇÃO QUALITATIVA DAS ABORDAGENS DE INDEXAÇÃO

ESTUDO EXPERIMENTAL

Representação de Conhecimento em Metodologias Ágeis
Mestrando: Fernando Selleri Silva

Avaliação Qualitativa da Abordagem de Indexação

Questão 1: Como você considera a **usabilidade** desta abordagem de indexação?
( ) 1. Muito Ruim ( ) 2. Ruim ( ) 3. Regular ( ) 4. Boa ( ) 5. Muita Boa

Questão 2: Como você considera a **utilidade** desta abordagem de indexação?
( ) 1. Muito Ruim ( ) 2. Ruim ( ) 3. Regular ( ) 4. Boa ( ) 5. Muita Boa

Questão 3: Como você considera o **esforço para aprendizagem** da abordagem?
( ) 1. Muito trabalhoso ( ) 3. Regular ( ) 5. Muito Cômodo
( ) 2. Pouco trabalhoso ( ) 4. Cômodo

Questão 4: Como você considera o **esforço para a associação** dos elementos do modelo utilizando esta abordagem?
( ) 1. Muito trabalhoso ( ) 3. Regular ( ) 5. Muito Cômodo
( ) 2. Pouco trabalhoso ( ) 4. Cômodo

Questão 5: Como você considera o **esforço para recuperação** de elementos relacionados utilizando esta abordagem?
( ) 1. Muito trabalhoso ( ) 3. Regular ( ) 5. Muito Cômodo
( ) 2. Pouco trabalhoso ( ) 4. Cômodo

Questão 6: Você **usaria na prática** novamente esta abordagem de indexação?
( ) 1.Discordo Plenamente ( ) 3.Não discordo nem concordo ( ) 5.Concordo Plenamente
( ) 2.Discordo ( ) 4.Concordo

Questão 7: A abordagem de indexação utilizada **atende o que se propôs**?
( ) 1.Discordo Plenamente ( ) 3.Não discordo nem concordo ( ) 5.Concordo Plenamente
( ) 2.Discordo ( ) 4.Concordo

APÊNDICE N – AVALIAÇÃO QUALITATIVA DA EXPERIÊNCIA DOS PARTICIPANTES

ESTUDO EXPERIMENTAL

Representação de Conhecimento em Metodologias Ágeis
Mestrando: Fernando Selleri Silva

Avaliação Qualitativa da Experiência dos Participantes

Questão 1: Qual a sua experiência em **desenvolvimento** de sistemas?
( ) Até 1 ano ( ) 1 a 2 anos ( ) 2 a 3 anos ( ) 3 a 4 anos ( ) Mais de 4 anos

Questão 2: Qual a sua experiência em **manutenção** de sistemas?
( ) Até 1 ano ( ) 1 a 2 anos ( ) 2 a 3 anos ( ) 3 a 4 anos ( ) Mais de 4 anos

Questão 3: Com que frequência você atua no **desenvolvimento** de sistemas em seu trabalho?
( ) Nunca
( ) Raramente - 1 a 4 vezes ao ano
( ) Eventualmente - 1 a 3 vezes ao mês
( ) Frequentemente - 1 a 3 vezes na semana
( ) Sempre - Diariamente

Questão 4: Com que frequência você atua na **manutenção** de sistemas em seu trabalho?
( ) Nunca
( ) Raramente - 1 a 4 vezes ao ano
( ) Eventualmente - 1 a 3 vezes ao mês
( ) Frequentemente - 1 a 3 vezes na semana
( ) Sempre - Diariamente

Questão 5: Com que frequência você utiliza **metodologias ágeis** no desenvolvimento de sistemas?
( ) Nunca
( ) Raramente - 1 a 4 vezes ao ano
( ) Eventualmente - 1 a 3 vezes ao mês
( ) Frequentemente - 1 a 3 vezes na semana
( ) Sempre - Diariamente

Questão 6: Nas atividades de desenvolvimento de software, você se preocupa em manter a **rastreabilidade** entre os artefatos?
( ) Não Plenamente ( ) Não ( ) Regular ( ) Sim ( ) Sim Plenamente

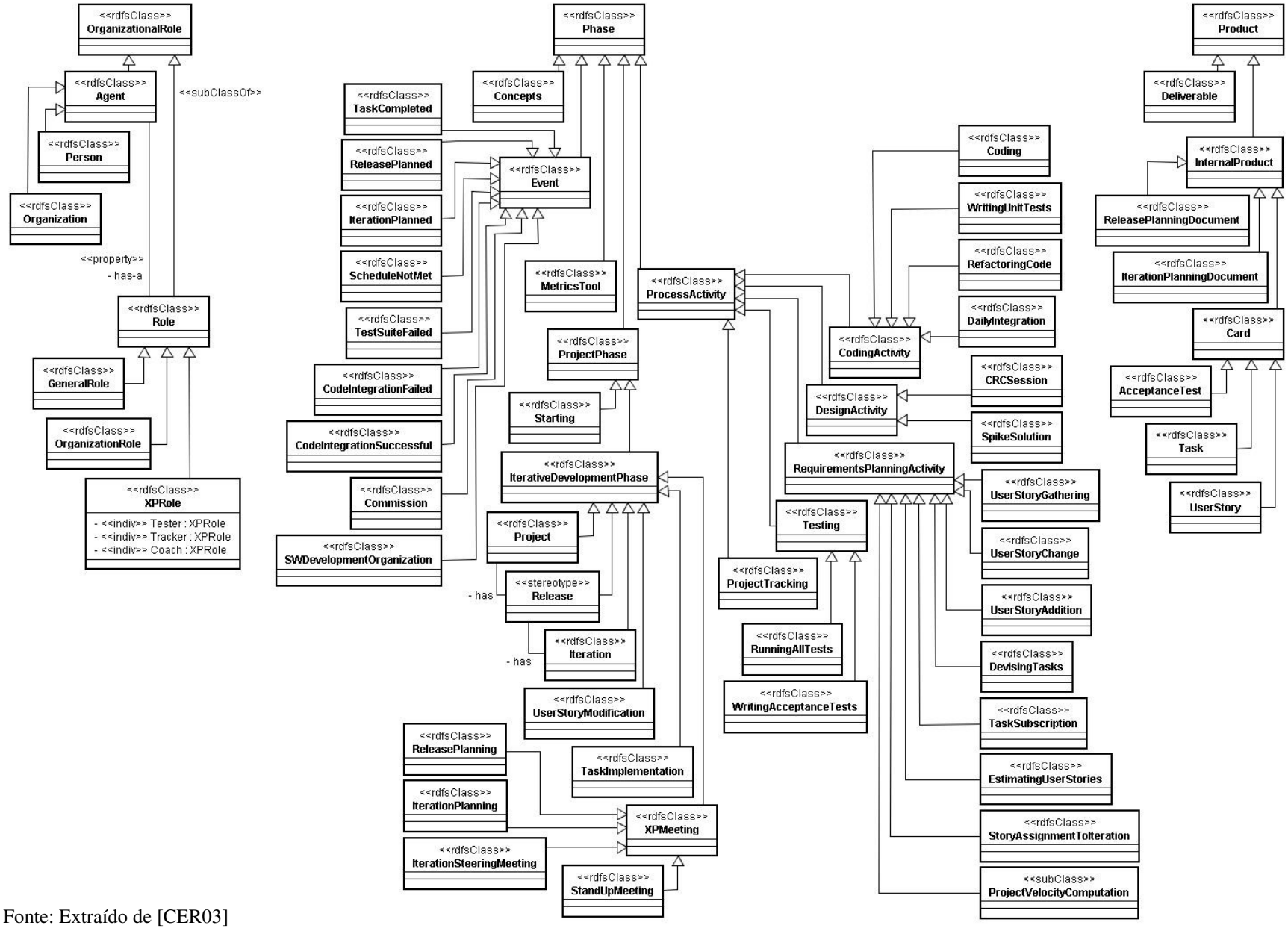
Questão 7: Você considera importante manter a **rastreabilidade** entre os artefatos desenvolvidos?
( ) Não Plenamente ( ) Não ( ) Regular ( ) Sim ( ) Sim Plenamente

Questão 8: Como você considera seu conhecimento sobre **desenvolvimento** de sistemas?
( ) Muito Pouco ( ) Pouco ( ) Regular ( ) Bom ( ) Muito Bom

Questão 9: Como você considera seu conhecimento sobre **manutenção** de sistemas?
( ) Muito Pouco ( ) Pouco ( ) Regular ( ) Bom ( ) Muito Bom

Questão 10: Como você considera seu conhecimento sobre **metodologias ágeis**?
( ) Muito Pouco ( ) Pouco ( ) Regular ( ) Bom ( ) Muito Bom

ANEXO A – *EXTREME PROGRAMMING ONTOLOGY* (XPO)



Fonte: Extraído de [CER03]