UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE SISTEMAS DE INFORMAÇÃO

RICHARD RAMBOR

ESPECIFICAÇÃO COLABORATIVA DE SOFTWARE UTILIZANDO EXEMPLOS:
um estudo de caso aplicado à manutenção de um sistema ERP

São Leopoldo
2015

Richard Rambor

ESPECIFICAÇÃO COLABORATIVA DE SOFTWARE UTILIZANDO EXEMPLOS:
um estudo de caso aplicado à manutenção de um sistema ERP

Artigo apresentado como requisito parcial para obtenção do título de Bacharel pelo Curso de Sistemas de Informação da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Prof. Dr. Kleinner Silva Farias de Oliveira

São Leopoldo

2015

ESPECIFICAÇÃO COLABORATIVA DE SOFTWARE UTILIZANDO EXEMPLOS: um estudo de caso aplicado à manutenção de um sistema ERP

Richard Rambor
rrambor@gmail.com

Prof. Dr. Kleinner Silva Farias de Oliveira
kleinnerfarias@unisinos.br

RESUMO

Manter um software que é utilizado globalmente apresenta vários desafios. Clientes, usuários e equipes de desenvolvimento em diferentes localizações são apenas alguns deles. Este tipo de software está suscetível às leis e regulamentações de cada país e suas regiões. Isso faz com que a manutenção do software seja constante e a introdução de novos requisitos seja frequente. Diferentes metodologias têm sido criadas para especificar, modelar, validar e gerenciar requisitos ao longo de um projeto. Porém, especificações de software tradicionais feitas por equipes de desenvolvimento muitas vezes não são claras, completas e detalhadas o suficiente para que possam ser de fato compreendidas pelos clientes. Ainda pior, regras de negócio complexas e pertinentes ao domínio de negócios dos clientes também não são compreendidas na totalidade e profundidade pelas equipes de desenvolvimento. Logo, prazos, custos e qualidade são frequentemente afetados. Este trabalho, portanto, focou na melhoria da especificação de software utilizando exemplos com o objetivo de reduzir problemas comuns relacionados aos requisitos de software em fase de manutenção. Para isso, uma ferramenta web foi criada e aplicada em projetos de manutenção de um software ERP que é usado globalmente. Os resultados, embora preliminares, são bastante promissores. O número de requisitos levantados com a ferramenta foi, em média, 77% maior do que o número de requisitos levantados sem a ferramenta. Além disso, a ferramenta teve boa aceitação por parte dos *stakeholders* e ajudou na melhoria da qualidade dos requisitos. Ou seja, foram observados benefícios quantitativos e qualitativos.

Palavras-chave: Engenharia de software. Engenharia de requisitos. Especificação de software. Especificação por exemplos. Manutenção de software ERP.

1 INTRODUÇÃO

Uma quantidade imensurável de informações passa por sistemas computacionais, que as estruturam, classificam, interpretam e as disponibilizam aos indivíduos e organizações em forma de conhecimento. O conhecimento enquanto ativo intangível passa a ser mais importante para as organizações do que os ativos tangíveis. (MARTELETO e SILVA, 2004, p. 47)

A constante evolução do software e do hardware possibilita o processamento de uma quantidade cada vez maior de informações em um menor espaço de tempo. Por isso, é constante também, a evolução da Engenharia de Software - área de conhecimento que estuda o desenvolvimento de software. Novas técnicas, metodologias e conceitos surgem a todo o momento com foco, principalmente, na eficiência e na agilidade dos processos de desenvolvimento.

Segundo Pressman (2011, p. 38) “[...] os requisitos de tecnologia de informação demandados por indivíduos, empresas e órgãos governamentais estão se tornando cada vez mais complexos a cada ano”.

Tudo se relaciona e está interconectado e a experiência do usuário de software passa a ser o centro das atenções. Por isso, não basta desenvolver software de forma eficiente (da forma certa). Fazer o software da forma certa e fazer o software certo são coisas diferentes, e precisamos de ambos para obter sucesso. (ADZIC, 2011, p. 3)

Mesmo com a aplicação das melhores práticas da Engenharia de Software, é muito frequente que o software não atenda às necessidades e expectativas dos clientes. O problema se agrava quando focamos em manutenção de software, pois ela é reativa e por isso mais caótica do que o desenvolvimento. (DIAS, ANQUETIL e OLIVEIRA, 2003, p. 642)

Softwares em manutenção não conseguem se adaptar às novas tecnologias com a mesma rapidez que um software novo. A manutenção depende de conhecimento pré-existente sobre softwares criados, muitas vezes há décadas, com linguagens obsoletas, algoritmos ineficientes (para a capacidade computacional da época) e por pessoas que não estão mais na ativa. A inserção de novos requisitos resulta na injeção de efeitos colaterais, nem sempre fáceis de evitar ou contornar. (DIAS, ANQUETIL e OLIVEIRA, 2003, p. 642)

Este trabalho, portanto, propõe uma abordagem para tratar requisitos de software que são complexos, interconectados e interdependentes, em sistemas em fase de manutenção. Essa abordagem inclui a criação de uma ferramenta web projetada para dar suporte às fases de elicitação e validação de requisitos de software em manutenção. A ferramenta organiza e classifica elementos do escopo do software existente e gera exemplos a partir da combinação desses elementos, que são validados ou invalidados pelos *stakeholders* de forma assíncrona e colaborativa.

Resultados preliminares identificam que a ferramenta trouxe benefícios quantitativos e qualitativos. No quesito quantitativo, a aplicação da ferramenta foi responsável por um acréscimo de mais de 70% (em média) no número de requisitos levantados, em relação ao mesmo projeto sem o uso da ferramenta. Esses requisitos adicionais estão relacionados com o escopo já existente do software. Isso significa que se não fossem levantados, provocariam efeitos colaterais no software já existente. Já no quesito qualitativo, foi possível observar que os requisitos foram melhor entendidos e validados pelos *stakeholders*, o que se deve principalmente à utilização de exemplos. A ferramenta foi aplicada em dois projetos de manutenção de software de pequeno porte, com duração entre três e dez dias de desenvolvimento.

O artigo está organizado da seguinte forma. A seção 2 apresenta a base teórica para o assunto, enquanto que a seção 3 caracteriza o problema relacionado à manutenção de software. A seção 4 apresenta a proposta da ferramenta que foi desenvolvida com o objetivo de melhorar o processo de especificação de software de um sistema ERP em fase de manutenção. A seção 5 apresenta os resultados parciais obtidos com o uso dessa ferramenta. Por fim, a seção 6 apresenta as conclusões e ideias de melhoria futuras.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta uma visão geral sobre Engenharia de Requisitos.

2.1 Engenharia de Requisitos

A Engenharia de Requisitos é uma subárea da Engenharia de Software, e antes de abordar detalhes da Engenharia de Requisitos, é necessário definir o que é um requisito. Segundo IEEE (1990) um requisito é:

> (1) Uma condição ou capacidade necessária ao usuário para resolver um problema ou atingir um objetivo;
> (2) Uma condição ou capacidade que um sistema ou componente precisa possuir para satisfazer um contrato, padrão, especificação ou outro documento formal imposto;
> (3) Uma representação documentada de uma condição ou capacidade com em (1) ou (2).

Um requisito pode ser visto como um acordo ou contrato que é feito entre quem desenvolve e quem necessita do software. A literatura define dois principais tipos de requisitos: requisitos funcionais e requisitos não funcionais. Segundo Damke e Moraes (2008, p. 45) requisitos funcionais e não funcionais são definidos como:

> ✓ Requisitos Funcionais: são aqueles que expressam funções ou serviços que um software deve ou pode ser capaz de executar ou fornecer. As funções ou serviços são, em geral, processos que utilizam entradas para produzir saídas (Cysneiros 2001).
> ✓ Requisitos Não Funcionais: são requisitos que colocam restrições sobre o produto em desenvolvimento, sobre o processo de desenvolvimento, e também, especificam restrições externas ao produto [...]. Normalmente são determinados pela natureza e tamanho do sistema no qual o software está inserido.

Segundo Falbo (2012, p. 1) “A Engenharia de Requisitos é o processo pelo qual os requisitos de um produto de software são coletados, analisados, documentados e gerenciados ao longo de todo o ciclo de vida do software (AURUM; WOHLIN, 2005). ”.

Para Audy e Prikladnicki (2007, p. 124):

> Um processo de engenharia de requisitos é um conjunto estruturado de atividades que são seguidas para derivar, validar e manter um documento de requisitos de um sistema. A descrição completa de um processo deve incluir as atividades que devem ser conduzidas, a estrutura ou agenda dessas atividades, as entradas e saídas de cada atividade e as ferramentas utilizadas para suportar a engenharia de requisitos (Sommerville e Sawyer, 1997).

Além do processo de engenharia de requisitos, a literatura também define o processo de gerência de requisitos. Ambos são vistos em mais detalhe a seguir.

2.2 Processo de engenharia de requisitos

A Engenharia de Requisitos define atividades específicas para tratar os requisitos de um projeto de software. O modelo de processos genérico para a Engenharia de Requisitos define as seguintes atividades: a) Elicitação dos requisitos; b) Análise e Negociação dos requisitos; c) Documentação dos requisitos; d) Validação dos requisitos. (ESPINDOLA, MAJDENBAUM e AUDY, 2004)

Segundo Espindola, Majdenbaum e Audy (2004), na fase de elicitação de requisitos são identificadas as necessidades e expectativas dos *stakeholders* com relação ao software. Esses requisitos servem de entrada para a análise e negociação, onde cada requisito é classificado, relacionado com os demais requisitos e priorizado de acordo com as necessidades dos *stakeholders*. Para Idem (2004) os requisitos são negociados e aceitos ou rejeitados buscando-se consenso. Posteriormente cada requisito é documentado de forma que os *stakeholders* possam entendê-los, e um documento de especificação de requisitos é criado juntamente com uma especificação do sistema a ser desenvolvido.

Por fim, na fase de validação, os requisitos são analisados a fim de que não existam erros, ambiguidades ou omissão de informações. Na prática, essas atividades se sobrepõe e interagem entre si. (ESPINDOLA, MAJDENBAUM e AUDY, 2004)

2.3 Processo de gerência de requisitos

Mudanças nos requisitos são comuns, por isso devemos gerenciar os requisitos de forma eficiente. O modelo de processo genérico da Gerência de Requisitos define as seguintes atividades: a) Identificação e armazenamento dos requisitos; b) Gerência de mudança dos requisitos; c) Rastreamento dos requisitos. (ESPINDOLA, MAJDENBAUM e AUDY, 2004)

Cada requisito deve ser classificado e receber um identificador único de forma que seja possível gerenciá-los. Separar cada requisito em um documento torna mais fácil o versionamento dos documentos, mas dificulta o rastreamento das dependências entre eles. Por outro lado, agrupar requisitos em um único documento torna mais claras as dependências, mas dificulta a manutenção e a rastreabilidade referente aos projetos. (Idem, 2004)

As atividades acima foram mencionadas porque estão diretamente relacionadas com o foco deste trabalho. Essas atividades exigem muita disciplina e se tornam ainda mais complicadas quando o software em questão tem um ciclo de vida muito longo. Não é raro que esses requisitos tenham que ser mantidos por décadas, e à medida que o escopo do software aumenta, a dificuldade de se manter as interdependências e o rastreamento aumenta proporcionalmente.

3 O PROBLEMA ESTUDADO

O Quadro 1 apresenta os 10 principais desafios relacionados a projetos de software segundo o estudo realizado por *The Standish Group International*, com 365 empresas de diversos portes e segmentos, e que representa um universo de 8.380 aplicações desenvolvidas.

Quadro 1 - Fatores desafiadores em projetos de software

Fatores desafiadores de projetos	% de respostas	Relação
Falta de envolvimento dos usuários	12,8%	Requisitos
Requisitos e especificações incompletas	12,3%	Requisitos
Requisitos e especificações desafiadoras	11,8%	Requisitos
Falta de suporte executivo	7,5%	Gerência
Incompetência tecnológica	7,0%	Capacitação
Falta de recursos	6,4%	Planejamento
Expectativas não realistas	5,9%	Requisitos
Objetivos não claros	5,3%	Requisitos
Prazos não realistas	4,3%	Planejamento
Novas tecnologias	3,7%	Capacitação
Outros	23%	Outros

Fonte: The Standish Group International[6]

É possível concluir que quase a metade dos fatores desafiadores em um projeto (48,1%) tem relação com os requisitos. O mesmo estudo ainda apresenta os 10 principais motivos pelos quais projetos de software foram cancelados:

[6] Tabela adaptada pelo próprio autor

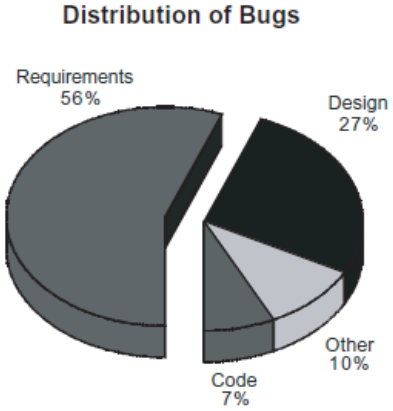
Quadro 2 - Fatores que ocasionaram o cancelamento de projetos de software

Fatores que ocasionaram o cancelamento de projetos	% de respostas	Relação
Requisitos incompletos	13,1%	Requisitos
Falta de envolvimento dos usuários	12,4%	Requisitos
Falta de recursos	10,6%	Planejamento
Expectativas não realistas	9,9%	Requisitos
Falta de suporte executivo	9,3%	Gerência
Requisitos e especificações que mudam	8,7%	Requisitos
Falta de planejamento	8,1%	Planejamento
Necessidade modificada	7,5%	Requisitos
Falta de gerencia de TI	6,2%	Gerência
Analfabetismo tecnológico	4,3%	Capacitação
Outros	9,9%	Outros

Fonte: Fonte: The Standish Group International[7]

Novamente pode-se identificar que mais da metade (51,6%) dos fatores que causam o cancelamento de um projeto estão relacionados com os requisitos. Outros estudos, como o proposto por Mogyorodi (2003) demonstram também, que mais da metade dos defeitos em projetos de software (56%) são provenientes de requisitos, conforme a Figura 2.

Figura 2 - Distribuição de defeitos em projetos de software

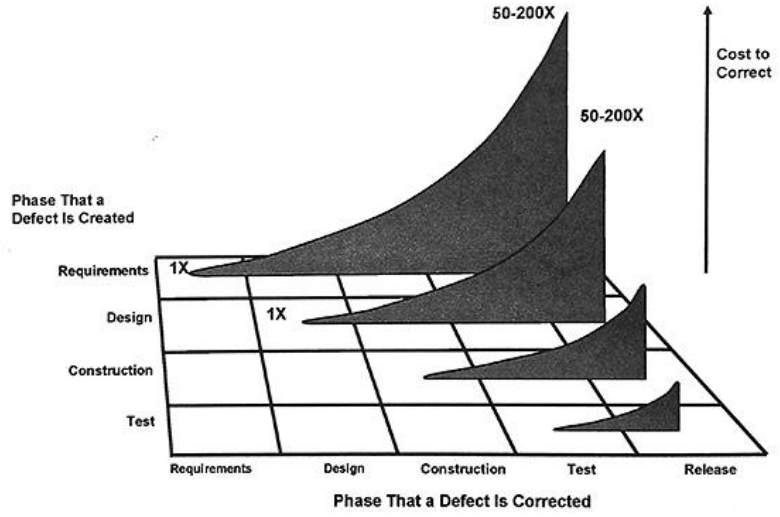


Fonte: Mogyorodi (2003)

Por fim, observando a questão dos custos de um projeto nota-se que os defeitos relacionados aos requisitos influenciam fortemente os custos de um projeto. A Figura 3 apresenta os custos de requisitos pobres ao longo das fases de um projeto:

[7] Tabela adaptada pelo próprio autor

Figura 3 - Custo de requisitos pobres



Fonte: McConnel (1998)

Quanto mais tarde um defeito é descoberto, maior será o custo para sua correção, e os defeitos provenientes de requisitos pobres são os que têm maior custo para correção quando descobertos tardiamente. A correção desses erros podem custar até duzentas vezes mais do que custariam se fossem descobertos na fase de elicitação e análise de requisitos, que faz parte das fases iniciais de um projeto.

A situação torna-se ainda mais complicada quando o software está em fase de manutenção. A manutenção de software é reativa e por isso mais complicada do que o desenvolvimento. A manutenção resulta de uma necessidade de adaptar o software, e na maioria dos casos essas mudanças não podem ser evitadas ou postergadas já que tem relação com leis e regulamentações. (DIAS, ANQUETIL e OLIVEIRA, 2003, p. 642)

A literatura sobre Engenharia de Software propõe soluções para todas as fases e atividades do processo de desenvolvimento de software. O quadro abaixo apresenta um resumo das áreas de conhecimento dentro da Engenharia de Software, algumas definições, modelos, ferramentas, metodologias, e seus respectivos objetivos:

Quadro 3 - Áreas da Engenharia de Software e Objetivos

Área Eng.SW	Definição	Modelos, Metodologias, Ferramentas	Objetivos

Processos de desenvolvimento de software	Representações abstratas do processo de desenvolvimento de software, também chamados de modelos de ciclo de vida do software	• Modelo em cascata
		• Evolucionários
		  ○ Concorrente
		  ○ Espiral
		  ○ Iterativo-incremental
		• Modelos formais
		• Orientados a reuso
		• Processo Unificado
		• *Rational Unified Process* (RUP)
		• Modelos Ágeis
		  ○ *Agile Modeling*
		  ○ *Agile Software Development*
		  ○ *Agile Unified Process*
		  ○ *Scrum*
		  ○ *Extreme programming (XP)*
		  ○ *Crystal Clear*
		  ○ *Dynamic Systems Development*
		  ○ *Feature Driven Development*
		  ○ *Lean Software Development*
			• Definir uma estratégia para o desenvolvimento de software, englobando processos, métodos e ferramentas;
			• Definir fases para o desenvolvimento;
			• Responder rapidamente a mudanças no decorrer do projeto de software (modelos Ágeis).

Melhoria de processos e de qualidade de software	Normas e modelos de apoio à definição e à melhoria dos processos de software nas organizações	• ISO/IEC 12207 • ISO/IEC 15504 • CMMI • MPS.BR	• Melhorar a qualidade; • Aumentar a produtividade; • Reduzir custos; • Apoiar a definição e a melhoria dos processos de software nas organizações.
Mapeamento e modelagem de processos	Conjunto de atividades realizadas para se definir, padronizar e documentar processos organizacionais (composto de técnicas e ferramentas)	• *Business Process Improvement* (BPI) • *Joint Application Design* (JAD) • *Unified Modeling Language* – UML • *Business Process Management Notation* (BPMN)	• Padronização; • Análise de custos; • Implantação de sistemas ERP; • Certificação em sistema de qualidade, entre outros.

Fonte: Próprio Autor[8]

É possível identificar que os objetivos da tabela acima têm relação com a eficiência e a qualidade dos processos de desenvolvimento de software. Porém, pouco tem sido criado nas últimas décadas com foco na manutenção de software. (DIAS, ANQUETIL E OLIVEIRA, 2003, p. 642)

A manutenção de software é muito mais complexa, pois tem que lidar com sistemas desenvolvidos há anos - muitas vezes décadas - com linguagens e

[8] Tabela adaptada de KNEWITZ, 2011, passim e SOUZA, 2011, passim.

processos obsoletos e para computadores limitados, e por isso com algoritmos adaptados ou fora dos padrões atuais. (DIAS, ANQUETIL e OLIVEIRA, 2003, p. 642)

As pessoas que mantém o software precisam de conhecimento específico do domínio da aplicação, do escopo existente, da organização que utiliza o software, das práticas de engenharia de software do presente e do passado, de diferentes linguagens de programação e suas diferentes versões, além é claro de habilidades técnicas avançadas. (Idem, 2003, p. 642)

Segundo Dias, Anquetil e Oliveira (2003, apud Pfleeger 2001 e Pigoski 1996), estudos demonstram que de 40% a 60% do esforço da manutenção de software é dedicado apenas para entender o sistema.

A manutenção acontece de forma relativamente desorganizada, o que leva a deterioração das estruturas do software. Quanto mais deteriorada estiver a estrutura do software, menos conhecimento sobre manutenção as equipes terão, e quanto menos conhecimento, mais deterioração será gerada, gerando assim uma espécie de círculo vicioso. (DIAS, ANQUETIL E OLIVEIRA, 2003, p. 642)

Como o software está constantemente se adaptando às necessidades de mercado, leis e usuários, a introdução de novos requisitos é frequente e inevitável. Ainda que os modelos e processos como os citados anteriormente possam ser utilizados e adaptados à manutenção de software, eles não focam ou dão pouco suporte ao processo de manutenção, especialmente no que diz respeito aos requisitos.

4 FERRAMENTA PROPOSTA

Com base na pesquisa bibliográfica e nos problemas encontrados, este estudo propôs uma abordagem para reduzir problemas relacionados aos requisitos na fase de manutenção de software. A proposta incluiu a criação de uma ferramenta de apoio que teve o objetivo de tornar o processo de especificação colaborativo, documentado, não ambíguo, e principalmente consciente sobre o escopo do software pré-existente.

As atividades que fizeram parte do desenvolvimento dessa ferramenta foram: a) a construção de um metamodelo de requisito; b) a construção de um modelo de variabilidade de requisitos; c) a construção de um metamodelo de projeto; d) a pré-configuração do sistema (que compreende aspectos relacionados ao escopo já

existente); e) um cadastro de projetos; f) a construção de exemplos baseados no modelo de variabilidade; g) a submissão dos exemplos aos *stakeholders;* h) a validação dos exemplos e/ou a adição de novos exemplos; i) a consolidação dos resultados e a resolução de conflitos.

Visando atingir os objetivos, as seguintes tecnologias foram utilizadas para o desenvolvimento da ferramenta: Apache[11]; PHP 5[12]; e MySQL[13].

A ferramenta desenvolvida teve como principais características: 1) arquitetura web - através de uma arquitetura web pretendeu-se atingir a colaboração de forma assíncrona uma vez que os *stakeholders* muitas vezes estão localizados em várias partes do mundo; 2) utilização de exemplos - a utilização de exemplos tem os seguintes objetivos: a) evidenciar os impactos da introdução de um requisito no escopo existente; b) facilitar a troca de conhecimento específico de domínio; c) remover a ambiguidade; d) agilizar o processo de validação dos requisitos; e 3) Customização - a ferramenta é flexível e permite que cada equipe faça o *setup* de acordo com suas necessidades.

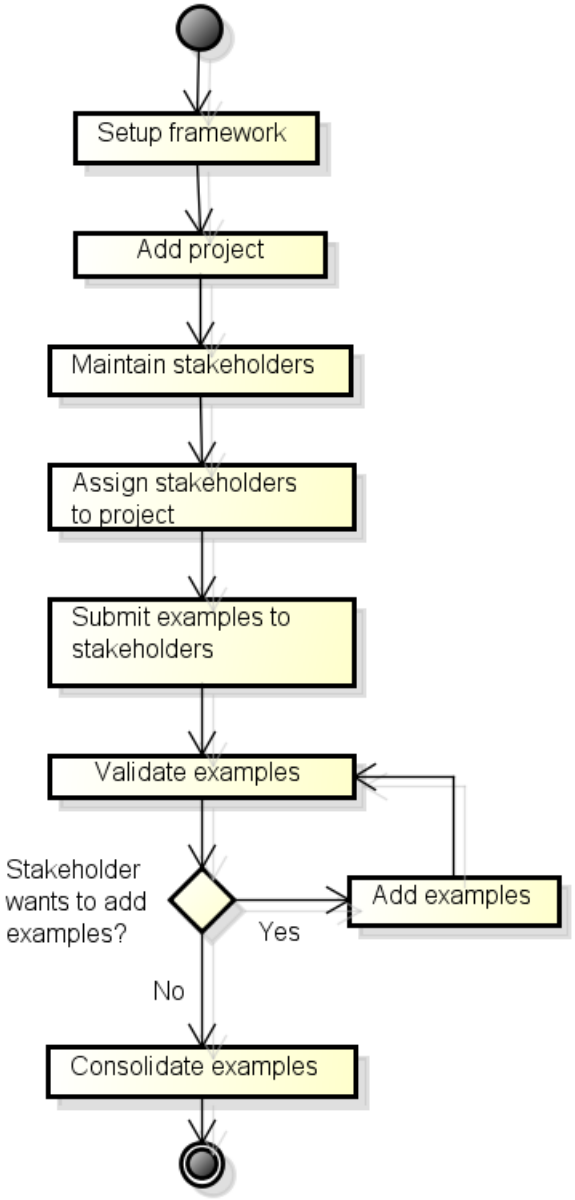
Os diagramas e as figuras que definem a modelagem do sistema encontram-se nas seções seguintes.

4.1 Diagramas de atividades e sequencia da ferramenta

O diagrama de atividades abaixo representa as tarefas que serão desempenhadas no sistema.

[11] o Apache é um servidor de páginas web largamente utilizado e reconhecido
[12] o PHP é uma linguagem de programação desenvolvida para a criação de aplicações web;
[13] o MySQL é um banco de dados relacional amplamente utilizado que segue o padrão SQL.

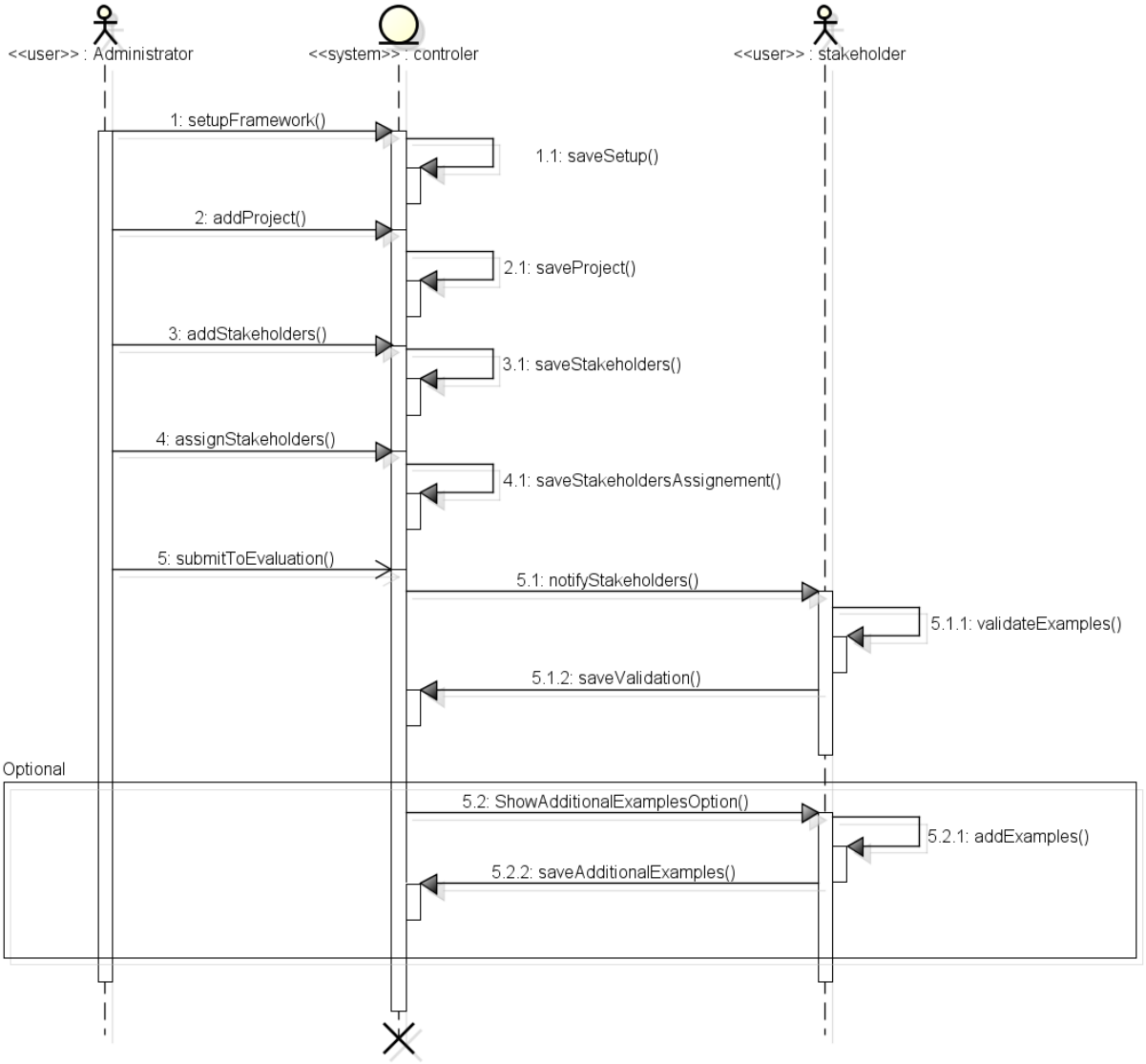
Figura 4 - Diagrama de atividades da ferramenta



Fonte: Próprio Autor

O diagrama abaixo apresenta a sequência das atividades.

Figura 5 - Diagrama de sequência da ferramenta



Fonte: Próprio Autor

A seguir são descritas as atividades do sistema em detalhes.

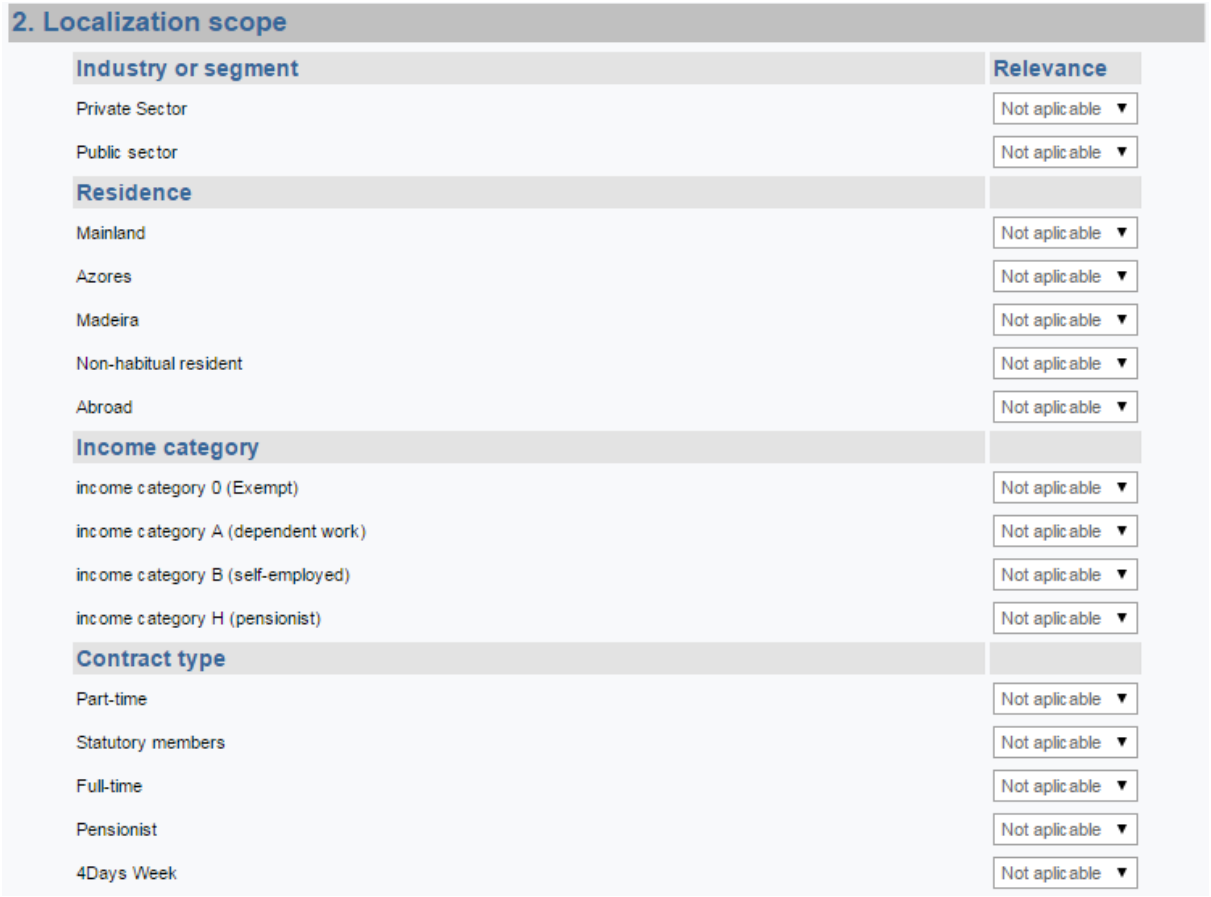
4.1.1 *Setup*

Nesta atividade são identificados os itens existentes que pertencem ao escopo da solução existente (em manutenção) e também os itens que são relevantes para validação (elaboração de exemplos). Esta atividade pretende fornecer aos desenvolvedores e *stakeholders* um guia para pensar nos efeitos colaterais de manutenções feitas no software. O objetivo principal do *setup* é guiar os desenvolvedores e *stakeholders* através de itens relevantes para a construção e validação de exemplos. Para o *setup* do sistema, os usuários mais experientes da

equipe de desenvolvimento devem ser consultados, a fim de identificar o máximo possível de itens que pertencem ao escopo.

Neste estudo, os itens de escopo foram divididos em duas categorias: escopo da localização (referente ao país[14] onde o software roda) e escopo genérico (comum a todos os países). A Figura 6 apresenta os itens selecionados do escopo da localização e a Figura 7 apresenta os itens selecionados do escopo genérico.

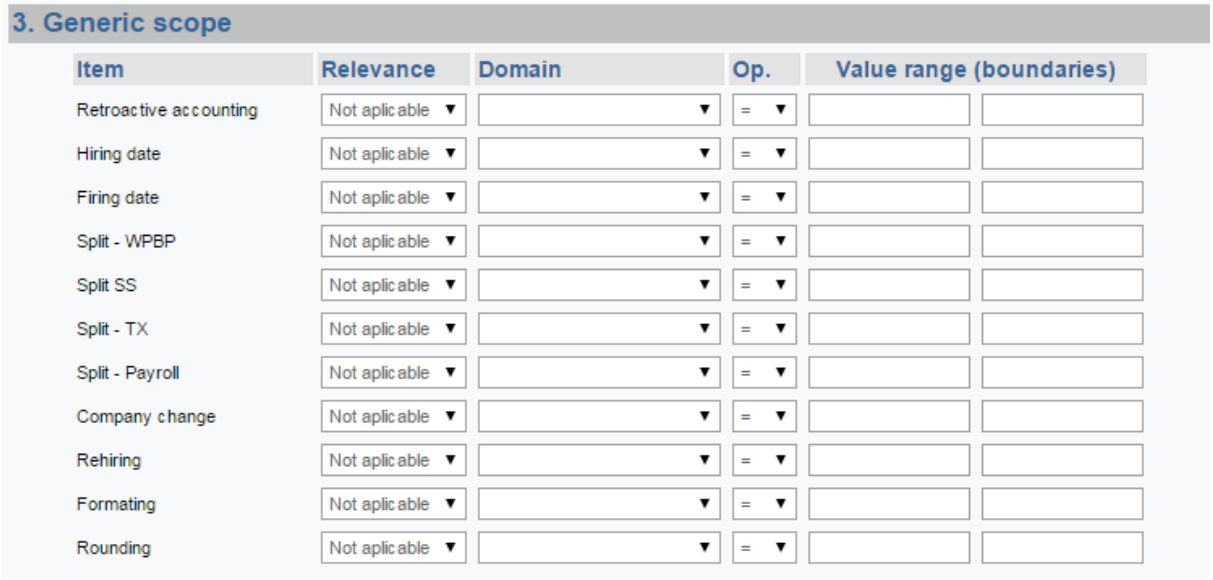
Figura 6 - Itens de escopo da localização



Fonte: Próprio Autor

[14] O exemplo utiliza dados de localização do software para o país Portugal.

Figura 7 - Itens de escopo genérico



3. Generic scope

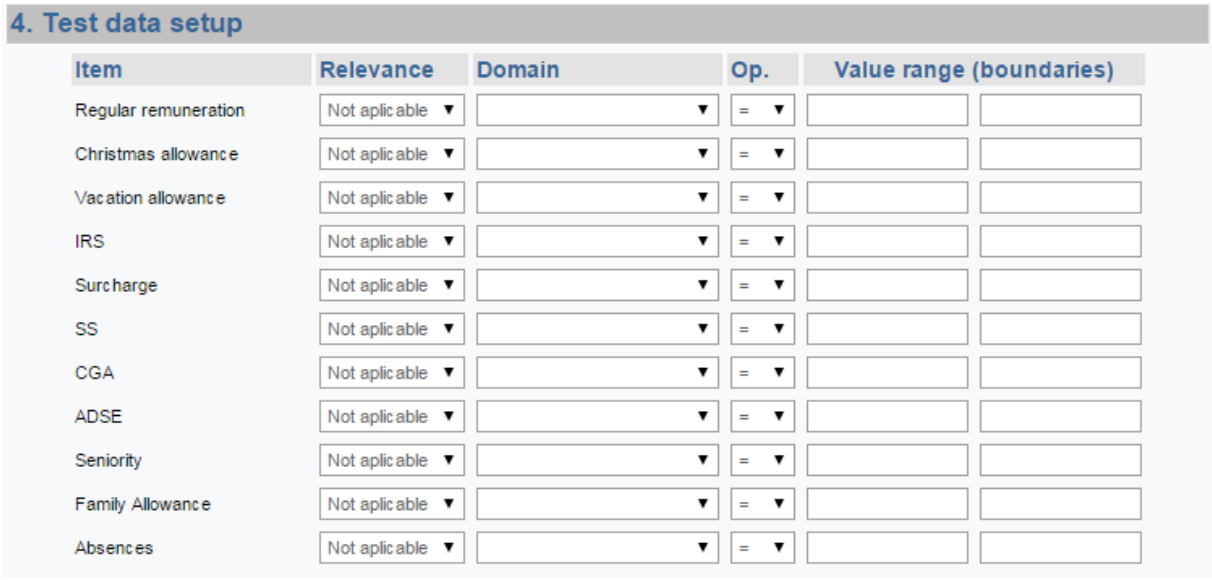
Item	Relevance	Domain	Op.	Value range (boundaries)	
Retroactive accounting	Not aplicable		=		
Hiring date	Not aplicable		=		
Firing date	Not aplicable		=		
Split - WPBP	Not aplicable		=		
Split SS	Not aplicable		=		
Split - TX	Not aplicable		=		
Split - Payroll	Not aplicable		=		
Company change	Not aplicable		=		
Rehiring	Not aplicable		=		
Formating	Not aplicable		=		
Rounding	Not aplicable		=		

Fonte: Próprio Autor

Os itens *Relevance*, *Domain*, *Op*. e *Value range (boundaries)* das Figuras 6 e 7 serão abordados mais adiante no item 4.1.2 - Cadastro de projetos.

Após definir os itens do escopo do software em manutenção, é necessário definir os itens de teste. Os itens de teste devem ter relação com o cenário identificado no cadastro de projeto. Esses itens são combinados com os itens do escopo para a construção dos exemplos (mais detalhes no item 4.1.2 - Cadastro de projetos). Os itens da Figura 8 foram identificados como relevantes para este estudo.

Figura 8 - Itens de teste



4. Test data setup

Item	Relevance	Domain	Op.	Value range (boundaries)	
Regular remuneration	Not aplicable		=		
Christmas allowance	Not aplicable		=		
Vacation allowance	Not aplicable		=		
IRS	Not aplicable		=		
Surcharge	Not aplicable		=		
SS	Not aplicable		=		
CGA	Not aplicable		=		
ADSE	Not aplicable		=		
Seniority	Not aplicable		=		
Family Allowance	Not aplicable		=		
Absences	Not aplicable		=		

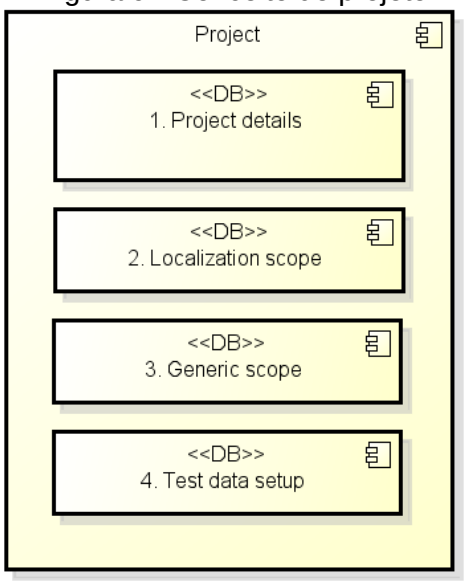
Fonte: Próprio Autor

Uma vez que os itens do escopo e de teste foram identificados e inseridos na ferramenta, pode-se então inserir projetos.

4.1.2 Cadastro de projeto

Um projeto corresponde à uma alteração no sistema. Normalmente uma alteração está associada a vários requisitos. Para que seja possível criar exemplos a partir de uma manutenção, é necessário identificar as possíveis relações que essa mudança tem com o escopo já existente e também quais são as validações desejadas para essa mudança. Para atingir esse objetivo a ferramenta propõe a criação de projetos segundo o conceito de projeto abaixo, ilustrado através de um diagrama de componentes.

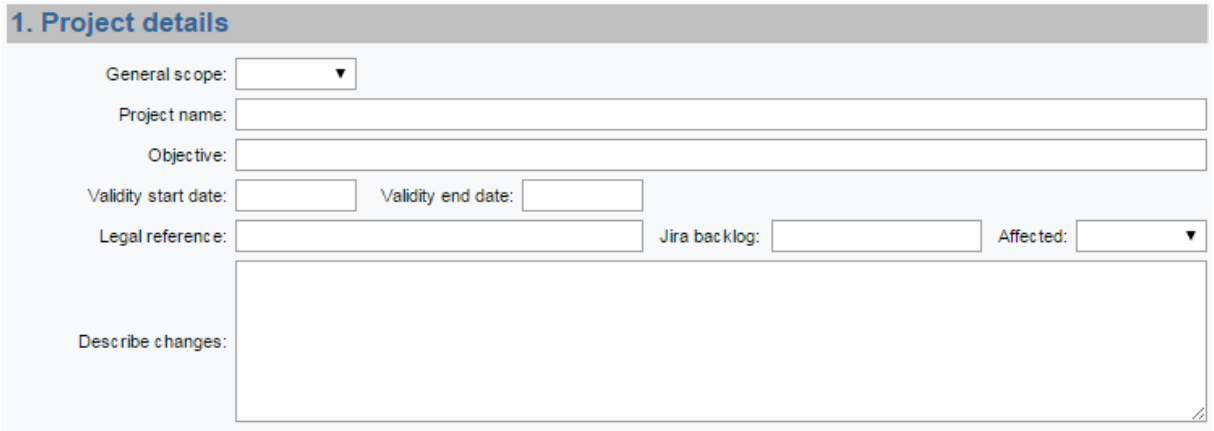
Figura 9 - Conceito de projeto



Fonte: Próprio Autor

O conceito da Figura 9 define que um projeto é formado por informações de identificação (*Project details*), itens de escopo da localização, itens de escopo genérico e dados de teste. A Figura 10 apresenta os dados utilizados para a identificação do projeto.

Figura 10 - Identificação de projeto



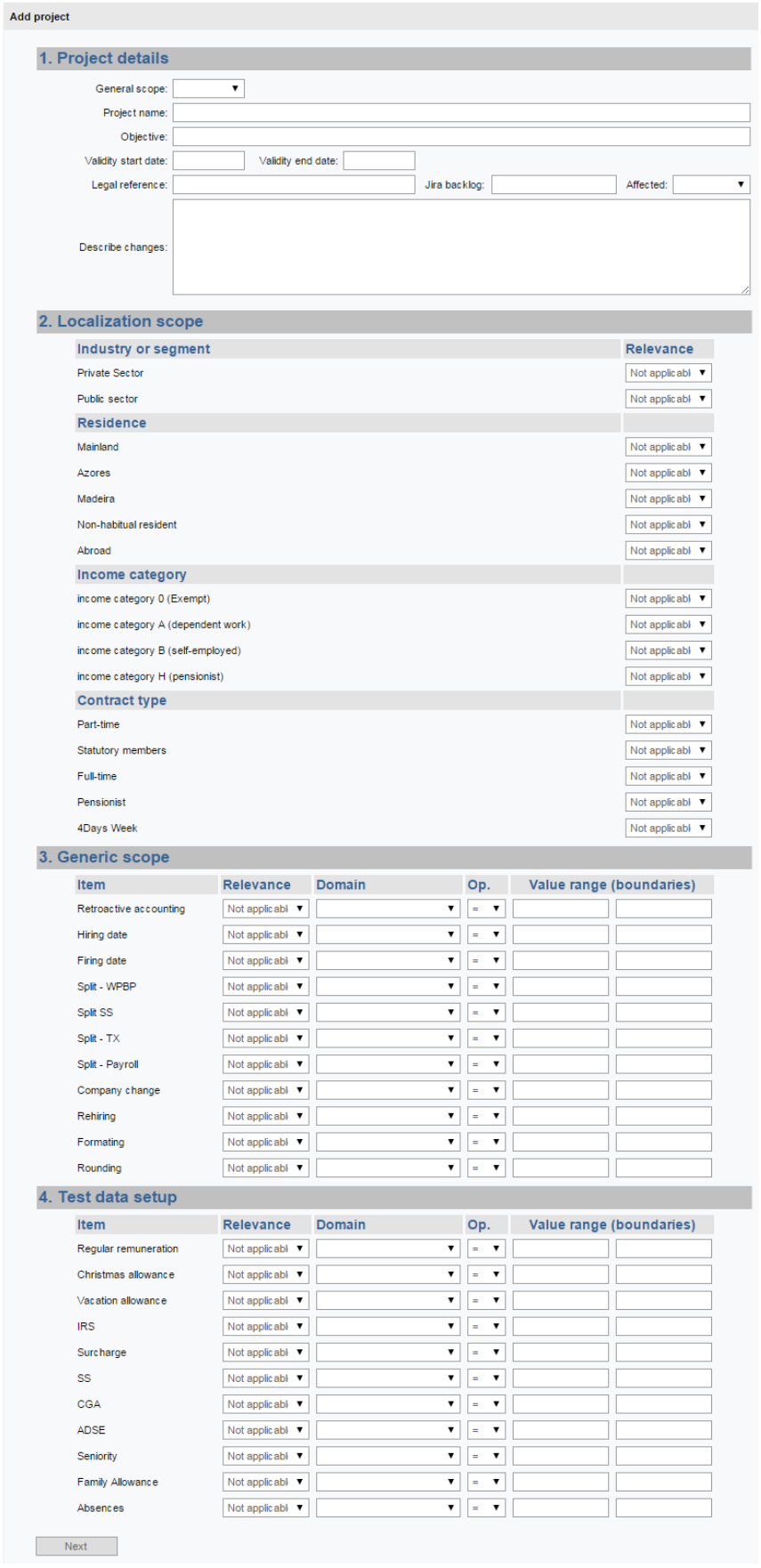
Fonte: Próprio Autor

Os itens do escopo da localização, os itens genéricos, e os dados de teste, foram definidos no *setup* da ferramenta. Durante o cadastro de um projeto, os itens considerados relevantes são selecionados utilizando-se a opção *Relevance*, que tem as seguintes opções: *Applicable*; e *Not applicable*. Além disso, é necessário definir o comportamento e as características dos itens de escopo (caso sejam relevantes) e dos dados de teste para a construção de exemplos. Os itens do escopo da localização apresentam apenas valores fixos, enquanto que os itens do escopo genérico e os dados de teste podem apresentar variações (Figuras 7 e 8). Para definir as variações desses itens em um projeto, os seguintes campos foram criados:

- *Domain: Number, Amount, Rate, Date, In period, For period, Year, Month, Day;*
- *Operator (Op.): = (equal), > (greater than), < (lower than), BT (Between);*
- *Value range (boundaries): value(1), value(2).*

Conceitualmente, dois valores (*value* 1 e *value* 2) são apenas utilizados em conjunto com o operador *between* (BT), definindo assim um intervalo de possíveis valores que uma variável pode assumir. A Figura 11 apresenta a tela de cadastro de projeto completa, com todos os itens apresentados anteriormente.

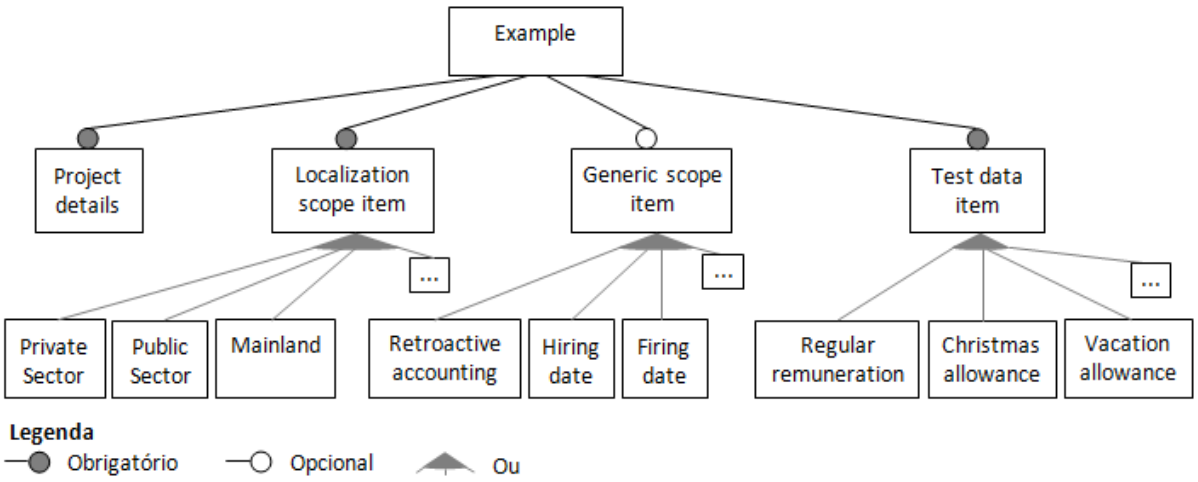
Figura 11 - Cadastro de projeto



Fonte: Próprio Autor

No próximo passo após o cadastro de um projeto a ferramenta apresenta a geração automática dos exemplos. Cada exemplo é construído a partir de uma combinação dos dados de teste, itens de escopo genéricos e itens do escopo da localização. As possíveis combinações são definidas segundo o diagrama de variabilidade abaixo.

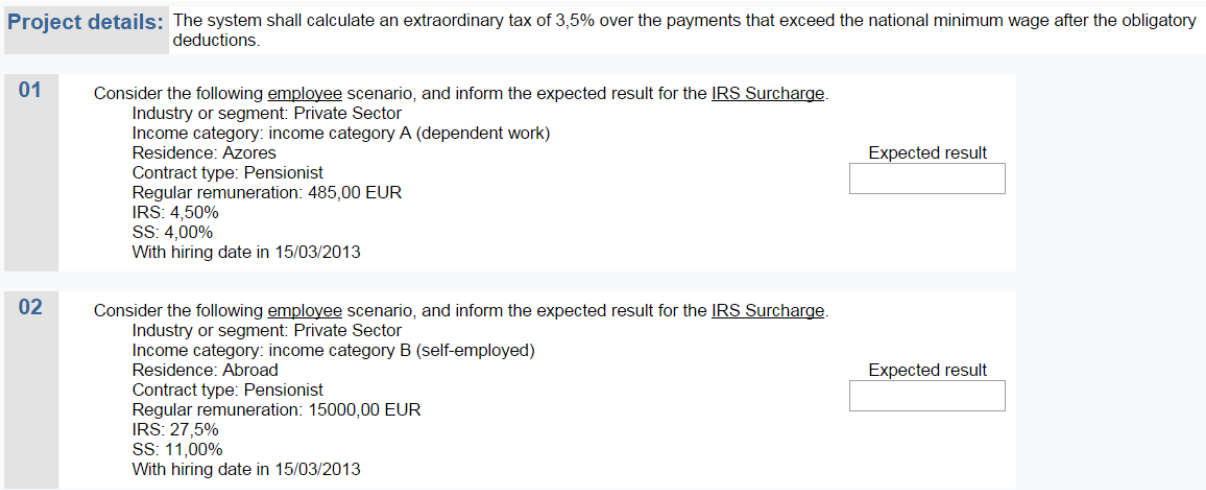
Figura 12 - Diagrama de Variabilidade



Fonte: Próprio Autor

A Figura 13 apresenta dois modelos de exemplos gerados automaticamente pela ferramenta seguindo o diagrama de variabilidade acima.

Figura 13 - Modelo de exemplo gerado pela ferramenta



Fonte: Próprio Autor

A variabilidade dos exemplos gerados depende da quantidade de combinações possíveis entre os itens de escopo, itens genéricos e dados de teste.

4.1.3 Cadastro de *stakeholders*

Nesta atividade, são cadastrados os *stakeholders* que irão fazer a validação dos exemplos gerados automaticamente pela ferramenta. A Figura 14 apresenta a tela de cadastro de *stakeholders*.

Figura 14 - Cadastro de *stakeholder*

Fonte: Próprio Autor

Os *stakeholders* deste estudo foram: desenvolvedores, consultores externos, clientes e usuários.

4.1.4 Vinculação de *stakeholders*

Nessa atividade o administrador do sistema vincula os *stakeholders* ao projeto. O número de *stakeholders* associados a um projeto fica a critério do administrador do sistema. A figura 15 apresenta o modelo de vinculação utilizado.

Figura 15 - Vinculação *stakeholders*

Fonte: Próprio Autor

Ao final dessa atividade, os *stakeholders* associados ao projeto recebem um e-mail, convidando-os a participar da validação dos exemplos. O e-mail contém um

link para uma página web onde os exemplos são apresentados, e também instruções para o processo de validação.

4.1.5 Validação de exemplos

Após acessarem o sistema, os *stakeholders* são guiados através dos exemplos gerados automaticamente. Para cada exemplo, o *stakeholder* deve informar o resultado esperado. O *stakeholder* também pode invalidar o exemplo, neste caso fornecendo uma justificativa por escrito. Ao final do processo de validação cada *stakeholder* pode contribuir com novos exemplos ou mesmo variações de exemplos anteriores. Esses novos exemplos são descritos textualmente e são gravados separadamente dos exemplos gerados automaticamente. Posteriormente o administrador do sistema pode julgar se os exemplos são relevantes e se há dados suficientes que caracterizam um exemplo válido.

4.1.6 Consolidação de exemplos

Nesta atividade o administrador do sistema analisa os resultados obtidos a partir das respostas. Se alguma contradição é identificada (*stakeholder* 1 respondeu A enquanto *stakeholder* 2 respondeu B) o administrador do sistema contata os *stakeholders* envolvidos para esclarecer este ponto específico. Esse é um processo administrativo, feito fora da ferramenta de apoio. Quando não existirem mais contradições entre as respostas o administrador pode documentar os requisitos segundo os resultados obtidos.

5 AVALIAÇÃO

Esta seção visa descrever como a ferramenta proposta foi avaliada e discutir quais foram os resultados obtidos. Para isso, a seção 5.1 apresenta o estudo de caso realizado e a seção 5.2 discute os resultados obtidos.

5.1 Estudo de caso

O estudo de caso deste trabalho foi realizado em uma grande empresa de desenvolvimento de software, aqui denominada Empresa A. A equipe de desenvolvimento que aplicou a abordagem proposta é responsável pela manutenção do módulo de recursos humanos, onde a folha de pagamento está inserida e tem sua localização específica para cada país. A localização da folha de pagamento compreende as adaptações necessárias a cada país e suas regiões, no que diz respeitos aos relatórios legais, cálculo de impostos, segurança social, benefícios e muitos outros itens.

Grandes empresas que mantém suas atividades em diversos países utilizam a folha de pagamento da Empresa A. Essas empresas confiam no sistema porque sabem que o software está em conformidade com a legislação vigente e que seus relatórios legais, não irão gerar penalidades frente aos órgãos regulatórios e governamentais.

O sistema começou a ser desenvolvido na década de 70, e desde então, módulos como o de recursos humanos estão em constante manutenção para se adequar às leis e às necessidades de mercado. Sempre que governos publicam novas leis, impostos, ajustes e medidas, novos requisitos são gerados e devem ser imediatamente inseridos no escopo da folha de pagamento. Por serem derivados de leis, esses requisitos são geralmente vagos, ambíguos e até conflitantes com outras leis. O entendimento correto de cada requisito é fundamental e por vezes isso requer tempo. Porém, os prazos e o escopo são determinados pelas leis, e não podem ser negociados. Leis muitas vezes são publicadas com data efetiva retroativa, o que significa que o projeto já está atrasado antes mesmo de começar. Por isso, o tempo que pode ser investido na elicitação e validação dos requisitos é ainda menor, o que aumenta muito a probabilidade de problemas nas fases seguintes.

Dentre as nove áreas de conhecimento do PMI[15], as áreas de tempo, custo, escopo e qualidade são as consideradas como críticas por Cardoso, Davies e Veronez (2012, p. 6).

Uma vez que não há possibilidade de alterar prazo ou escopo – que são definidos por leis -, temos que buscar alternativas relacionadas à qualidade ou então aos custos.

[15] Project Management Institute

Segundo Martins (2007, p. 272), “[...] custos geralmente estão associados ao esforço de análise e programação. Na construção de um sistema, o homem/hora é o principal recurso consumido [...]”. Em outras palavras, os custos geralmente estão associados aos recursos. Contudo, adicionar recursos (homens/hora) a um projeto não resolve os problemas de prazo e qualidade, e por vezes, acaba criando ainda mais problemas para o gerenciamento. Analogamente, uma casa construída por um único homem em mil dias, não poderá ser construída por 1000 homens em um dia.

Por fim, só nos resta abordar a questão da qualidade. Os requisitos foram selecionados por dois motivos: a) é onde a maior parte dos problemas acontece, e b) é uma área onde há liberdade para se trabalhar no contexto da Empresa A. Nessa empresa, os *stakeholders* estão localizados em diferentes países, com diferentes fusos horários, falam diferentes idiomas, tem diferentes culturas e diferentes conhecimentos. Assim, a colaboração por si só, já era um desafio.

5.2 Resultados

A ferramenta foi utilizada em dois projetos de manutenção de um sistema ERP em um período de três meses, denominados aqui de projetos A e B. Durante esse período as solicitações de mudança foram tratadas de duas formas: inicialmente da forma tradicional, e posteriormente com a utilização da nova ferramenta, para fins de comparação.

O processo tradicional envolvia reuniões e discussões com os *stakeholders* para definir os requisitos. Posteriormente os requisitos eram documentados e davam origem a um documento de requisitos e uma especificação de software, que em seguida eram validados novamente com os *stakeholders*. Depois de validados os requisitos, o projeto seguia para o desenvolvimento, testes e implantação. Um problema comum nesse modelo, é que outros requisitos não aparentes eram esquecidos. Esses requisitos se referiam principalmente aos efeitos colaterais da alteração em questão, ou seja, como a funcionalidade sendo inserida afetada as funcionalidades já existentes.

No novo processo, o contexto pré-existente do software foi cadastrado na ferramenta de apoio. Em seguida um novo projeto foi criado para a construção automática de exemplos. Esses exemplos foram enviados automaticamente para o e-mail dos *stakeholders* previamente cadastrados na ferramenta. Os *stakeholders*,

ao clicar no link enviado por e-mail, eram direcionados para uma página web do projeto contendo todos os exemplos gerados automaticamente pela ferramenta. Esses exemplos eram exibidos um de cada vez, e para cada exemplo, o *stakeholder* devia validar ou invalidar, e também informar o resultado esperado para o cenário descrito pelo exemplo caso esse fosse um exemplo válido. Ao final, todos os dados eram salvos na base de dados da ferramenta. As respostas de todos os *stakeholders* foram comparadas através da ferramenta, que identificou os cenários com discordâncias. Essas discordâncias foram em seguida esclarecidas por e-mail ou reunião extraordinária e depois atualizadas novamente na ferramenta.

Os resultados obtidos por cada forma foram comparados quantitativamente, em relação ao número de requisitos, conforme tabela abaixo.

Tabela 1 – Quantidade de requisitos elicitados e validados vs processos de elicitação e validação

	TIPO DE PROCESSO	
	Tradicionais	Com uso da ferramenta
Projeto A	9	17
Projeto B	3	5

Fonte: Próprio Autor

Apesar de utilizar uma amostra muito pequena, o estudo demonstra que a utilização da ferramenta de apoio traz benefícios, tanto quantitativos quanto qualitativos. Os projetos que fizeram uso da ferramenta tiveram um número de requisitos 88,88% e 66,66% maiores, respectivamente. Esses requisitos adicionais estavam relacionados principalmente às dependências com o escopo já existente. Isto significa que, se esses requisitos fossem esquecidos, certamente provocariam efeitos colaterais em outras partes do software já existente. Além disso houve boa aceitação por parte dos *stakeholders.* Eles sentiram-se mais confortáveis com o processo de validação, tanto por não sofrer influência de outros participantes (o que acontece nas reuniões do processo tradicional), quanto pela questão dos horários flexíveis para a validação de requisitos.

Abaixo são apresentados os prós e contras do uso da ferramenta, citados pelos *stakeholders*.

As principais vantagens apontadas foram: (1) A ferramenta nos faz refletir sobre efeitos colaterais que normalmente não estariam claros; (2) A ferramenta torna

o processo mais flexível quando comparado às reuniões de especificação tradicional, principalmente porque não depende de um horário comum entre os *stakeholders*; (3) A ferramenta torna o processo mais agradável; (4) A ferramenta torna o processo mais ágil; (5) A ferramenta diminui problemas de ambiguidade com a geração de exemplos; (6) A ferramenta testa valores que normalmente não são testados (*boundaries*).

As principais desvantagens identificadas são: (1) O número de exemplos gerados é muito grande; (2) Alguns exemplos não fazem sentido; (3) Não há uma discussão coletiva sobre cada requisito.

6 CONCLUSÃO

A manutenção de software é desafiadora em muitos sentidos, especialmente no que se refere à introdução de novos requisitos. Este estudo reforça a criticidade dos requisitos para o sucesso de um projeto de desenvolvimento de software. Reforça também, que a utilização de metodologias, processos e ferramentas é fundamental para o sucesso de qualquer projeto de software e esses devem evoluir constantemente.

A ferramenta propõe uma abordagem diferente, que põe o *stakeholder* no centro do processo, tenta eliminar ambiguidades, auxiliar na transferência de conhecimento específica de domínio e automatizar alguns processos que normalmente seriam feitos manualmente.

A abordagem que utilizou a ferramenta de apoio mostra dados parciais promissores, mas ainda carece de mais aplicações práticas e melhorias. Os pontos negativos citados pelos *stakeholders* devem ser melhorados, principalmente no aspecto da colaboração. Uma ideia futura é a introdução de ferramentas de comunicação síncrona para que os *stakeholders* possam trocar informações e discutir os requisitos em tempo de validação.

O maior benefício citado pelos *stakeholders* certamente se refere à conscientização de possíveis efeitos colaterais no escopo já existente. Uma vez que a ferramenta expõe o escopo de forma organizada e cria exemplos a partir deles, os *stakeholder* são guiados a pensar sobre cada item de escopo e como eles se relacionam entre si. Esse ponto contribui para um aspecto importante que é a

rastreabilidade dos requisitos, porém sem precisar de documentos guardados em diferentes locais e por longo tempo.

A ferramenta também constitui uma forma de documentação dos requisitos. Cada requisito validado pode ser considerado um contrato entre quem necessita, e quem desenvolvem o software. Este é um aspecto que pode ser melhor e mais explorado em melhorias futuras.

Assim ficam as seguintes ideias para melhorias futuras nesta ferramenta:

1. Geração automática de documentação;
2. Geração de código de teste unitário automático baseado nas validações dos exemplos;
3. Introdução de ferramentas de comunicação síncrona;
4. Limitação mais eficiente do número de exemplos gerados em um projeto; e
5. Distribuição aleatória de exemplos para *stakeholders.*

REFERÊNCIAS

ADZIC, Gojko. **Specification by example**. 1ª. ed. Manning Publications Co., New York, 2011.

AUDY, Jorge. PRIKLADNICKI, Rafael. **Desenvolvimento distribuído de software**. Rio de Janeiro: Elsevier. 2008.

CARDOSO, Patrícia A. DAVIES, Yasmin M. VERONEZ, Larissa H. **Identificação de um Sistema de Medição de Desempenho para Gestão de Projetos em Redes de Colaboração**. In: SIMPÓSIO INTERNACIONAL DE GESTÃO DE PROJETOS, 1., 2012. São Paulo. Anais... São Paulo. 2012. p. 1-20.

DAMKE, Daniele E. MORAES, Patrícia F. de. **Aplicação da abordagem GQM para a definição de um processo de Engenharia de Requisitos de Software Embarcado.** Brasília: Trabalho de Pós-Graduação (Curso de Engenharia de Software) - Universidade Católica de Brasília. Brasília. 2008. Disponível em <https://claudiameloprof.files.wordpress.com/2010/08/aplicac3a7c3a3o-da-abordagem-gqm-para-a-definic3a7c3a3o-de-um-processo-de-engenharia-de-requisitos-de-software-embarcado.pdf>. Acesso em 24 mar 2015.

DIAS, Márcio G. B. ANQUETIL, Nicolas. OLIVEIRA, Káthia M. **Organizing the Knowledge Used in Software Maintenance.** Journal Of Universal Computer Science, S.l. v. 9. n. 7, p. 641-658. 28/07/2003.

ESPINDOLA, Rodrigo Santos de. MAJDENBAUM, Azriel. AUDY, Jorge Luiz Nicolas. **Uma Análise Crítica dos Desafios para Engenharia de Requisitos em**

Manutenção de Software. Programa de Pós-Graduação em Ciência da Computação
Faculdade de Informática - Pontifícia Universidade Católica do Rio Grande do Sul. In: WER, 2004. Disponível em <http://www.inf.pucrs.br/~munddos/docs/EspindolaMajdenbaumAudyWER2004Final.pdf>. Acesso em 22 out 2015.

FALBO, Ricardo de A. **Engenharia de Requisitos**. Espírito Santos: Notas de Aula – Universidade Federal Espirito Santos. Espirito Santos, 2012. Disponível em <http://www.inf.ufes.br/~falbo/files/Notas_Aula_Engenharia_Software.pdf>. Acesso em 12 mar 2015.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Standard Glossary of Software Engineering Terminology**. New York. 1990.

KNEWITZ, Marcos A. **Desenvolvimento de Software –** 1. São Leopoldo, RS: Ed. UNISINOS, 2011

MARTELETO, Regina M. SILVA, Antonio B. de O. e. **Redes e capital social: o enfoque da informação para o desenvolvimento local**. Brasília: Ciência da Informação, v. 33, n. 3, p.41-49, set./dez. 2004. Disponível em <http://www.scielo.br/pdf/ci/v33n3/a06v33n3>. Acesso em 07 mar 2015.

MARTINS, José C. C. Gerenciando **projetos de desenvolvimento de Software com PMI, RUP e UML.** 4 ed. Rio de Janeiro: Brasport, 2007.

MAZZOLA, Vitório B. **Engenharia de Software: Conceitos Básicos**. S.l. S.d. Apostila

MCCONNEL, Steve. **Software Project Survival Guide.** Redmond, Washington, US, 1998.

MOGYORODI, Gary E. **What is requirements-based Testing?.** Crosstalk, 2003. Disponível em <http://www.crosstalkonline.org/storage/issue-archives/2003/200303/200303-Mogyorodi.pdf>. Acesso em 24 abr 2015

PRESSMAN, Roger. S. **Engenharia de software: Uma Abordagem profissional**. 7. ed. Porto Alegre: McGraw-Hill, 2011.

REZENDE, Denis A. **Engenharia de Software: e Sistemas de Informações**. 3 ed. Rio de Janeiro: Brasport. 2005.

SOUZA, Vinícius. **Definindo o Processo de Software**. São Leopoldo: Unisinos, 2011.

THE STANDISH GROUP INTERNATIONAL (Ed.). **The Chaos Report**. [s.l]: [s.n.], 1995. Disponível em <http://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf>. Acesso em 26 abr 2015.