ChangeMan: Um Sistema Multi-Agente para Gestão da Mudança de Requisitos com suporte à Rastreabilidade e Análise de Impacto

Pablo Dall'Oglio, João Pablo S. da Silva, Sérgio Crespo C. S. Pinto

Programa Interdisciplinar de Computação Aplicada
Universidade do Vale do Rio dos Sinos (UNISINOS) – São Leopoldo, RS - Brasil

{pablo.dalloglio, jpabloss}@gmail.com, crespo@unisinos.br

Abstract. *It's common sense that the requirement's change management has a fundamental role in the quality of the software development process. In order to manage the change precisely, the use of the traceability information is indispensable. The traceability, by its turn, makes possible the implementation of the impact analysis process. Despite the importance of these subjects, research papers demonstrate a lack of methods and tools to fill the requirements identified in the literature. This way, the current article introduces a requirement's change management system with support to the traceability and impact analysis that implements a set of requirements identified in the main works in this area.*

Resumo. *É de senso comum que a gestão da mudança de requisitos exerce papel fundamental na qualidade do processo de desenvolvimento de software. Para que se possa gerenciar a mudança de forma precisa, o uso de informações de rastreabilidade torna-se indispensável. A rastreabilidade, por sua vez, torna possível a realização da análise de impacto. Apesar da importância destes assuntos, trabalhos de pesquisa revelam uma falta de métodos e ferramentas que supram os requisitos identificados na literatura. Desta forma, este trabalho apresenta um sistema para gestão da mudança de requisitos com suporte à rastreabilidade e análise de impactos que atende a um conjunto de requisitos identificados pelos principais trabalhos da área.*

1. Introdução

Muitas metodologias surgiram para organizar o desenvolvimento de software, a maioria delas baseada na previsibilidade dos requisitos. Como nem sempre é possível obter todos os requisitos antes da construção do software, é necessário saber o que fazer quando ocorrer uma mudança [Beck 2004]. Neste contexto, ganha importância a gestão da mudança de requisitos, uma vez que o sucesso do software vai depender de quanto ele se adequar às mudanças de ambiente [Nuseibeh 2000].

Dentro da gestão da mudança de requisitos, a análise de impacto é a atividade que identifica as entidades que podem ser afetadas por uma mudança proposta no sistema. O objetivo principal da análise de impacto é minimizar os efeitos colaterais de uma mudança. Dentre algumas formas de se realizar a análise de impacto, a rastreabilidade é uma das mais eficientes [Mader 2009]. A rastreabilidade de requisitos consiste em ligações entre as informações produzidas no desenvolvimento de software.

Para que se possa efetivamente gerenciar as mudanças de software, é necessário uma ferramenta que permita controlar com precisão a rastreabilidade entre requisitos, as mudanças ocorridas e os impactos gerados. Apesar de vários métodos já terem sido propostos com este objetivo, são percebidas algumas deficiências nestes, sendo que nenhum é tido como completo conforme os critérios adotados no presente artigo. Este trabalho propõe um método para gestão da mudança de requisitos com suporte à rastreabilidade e análise de impacto, de forma que atenda os requisitos identificados nos trabalhos já desenvolvidos na área, conforme os critérios analisados na seção 4.

A partir do estudo dos trabalhos relacionados, foi desenvolvido um novo método, com o objetivo de atender a um maior número de requisitos funcionais identificados. Para tal, na seção 2 serão abordados os principais trabalhos relacionados, na seção 3 será abordado o método proposto, bem como suas diversas facetas, na seção 4 serão apresentados critérios de comparação (requisitos) entre os trabalhos relacionados e o método proposto e na seção 5 serão apresentadas as conclusões. O protótipo gerado a partir dos conceitos propostos foi preliminarmente testado em um ambiente real para produção de software, tendo mostrado resultados muito consistentes.

2. Trabalhos Relacionados

Nesta seção serão apresentados os principais trabalhos relacionados que propõem métodos para gestão de requisitos.

2.1. ADAMS

ADAMS (*ADvanced Artefact Management System*) é uma ferramenta *web* proposta por De Lucia [De Lucia 2005], que integra características de gerenciamento de projetos, alocação de recursos, gerenciamento de artefatos, coordenação e cooperação dos colaboradores, versionamento de artefatos e rastreabilidade. ADAMS trata diferentes tipos de documentos (requisitos textuais, modelos de *design* UML, código-fonte), sendo que o usuário pode criar um artefato composto, onde cada uma de suas partes (nodos) pode ser um outro artefato [De Lucia 2005]. Além disto, ADAMS permite que seu usuário registre a rastreabilidade entre os artefatos da mesma hierarquia (rastreabilidade vertical) ou entre artefatos de hierarquias diferentes (rastreabilidade horizontal).

A ferramenta ADAMS conta ainda com um módulo de recuperação de rastreabilidade horizontal, que identifica *links* de rastreabilidade pela similaridade entre termos. ADAMS permite que os colaboradores possam ser notificados sobre eventos que ocorrem sobre os artefatos e utiliza os *links* de rastreabilidade para a análise de impacto e a propagação da notificação dos colaboradores envolvidos em uma mudança. ADAMS não permite um gerenciamento proativo do processo, necessitando a ação do usuário para obtenção de resultados, não fornece estatísticas gerenciais, não fornece mecanismos de acompanhamento e decisão do processo via *workflow* e não suporta mecanismos de extensibilidade.

2.2. Event Based Traceability

Cleland-Huang [Cleland-Huang 2003] propõe um método de rastreabilidade baseado em notificação de eventos, chamado *Event-Based Traceability* (EBT). Neste método, os artefatos são relacionados por meio de uma estrutura de publicação/assinatura, onde os artefatos alterados assumem o papel de publicadores (*publishers*) e os artefatos dependentes assumem o papel de assinantes (*subscribers*). Desta forma, sempre que

uma mudança ocorre, eventos são publicados em um "servidor de eventos" e notificações são enviadas aos desenvolvedores dos artefatos dependentes (*subscribers*).

Para Cleland-Huang [Cleland-Huang 2003], a principal vantagem na implementação da abordagem baseada em eventos (EBT) está em melhorar a coordenação entre os membros responsáveis pelas alterações nos artefatos. Cabe salientar que o método EBT foi projetado para integrar-se com ferramentas de gestão de requisitos textuais como *DOORS* ou *RequisitePro*, não suportando modelos de *design* UML e código-fonte. Verificou-se também a ausência de configuração de projeto, da configuração de granularidade da rastreabilidade, versionamento dos requisitos, de mecanismos para detecção da rastreabilidade, de extensibilidade e de estatísticas.

2.3. iACMTool

Briand [Briand 2003] propõe uma ferramenta de análise de impacto chamada iACMTool. Esta ferramenta realiza a leitura de duas versões de um modelo UML representado em XMI (*XML Metadata Interchange*) e realiza um relatório de impacto baseado nas diferenças entre estas versões. As mudanças são detectadas e classificadas conforme taxonomia própria. A ferramenta verifica a consistência de uma determinada versão do modelo, realiza a análise de impacto e gera relatórios de impacto.

Dentre as principais qualidades da ferramenta iACMTool, destacam-se a leitura da estrutura de documents XMI, permitindo a obtenção de informações de artefatos e relacionamentos com maior grau de fidelidade. Além disto, suporta a rastreabilidade, provê gerenciamento das mudanças e análise de impacto, melhorando a comunicação e coordenação da equipe. Como a ferramenta é focada em documentos UML, não possui suporte de integração com requisitos textuais, tampouco com artefatos de código-fonte, não oferecendo recursos de rastreabilidade horizontal. Além disto, não suporta uso on-line, configuração de granularidade, proatividade na execução das atividades, mecanismos para configuração de projeto e extensibilidade.

2.4. Chianti

Ren [Ren 2004] propõe um método de análise de impacto baseado em testes de unidade. Seu método inicia com a análise do código-fonte, para obter as mudanças ocorridas a partir da comparação entre duas versões do sistema. A partir de então, constrói um grafo semântico de execução, que demonstra as chamadas para cada teste de unidade (*unit test*). Este grafo contém um nodo para cada método e uma aresta para cada relacionamento de chamada. Neste grafo, pode-se perceber quais testes de unidade dependem de quais classes ou métodos, baseado em suas chamadas de execução. A partir deste grafo, o método detecta quais foram os testes afetados por uma mudança.

A ferramenta Chianti, proposta por Ren, trata-se de um protótipo de ferramenta para análise de impacto, cujo funcionamento se dá por meio da leitura do repositório de um projeto de desenvolvimento (CSV) em intervalos regulares e armazena as informações das mudanças e das dependências em arquivos XML para posterior análise. A ferramenta possui gráficos gerenciais que indicam as mudanças e os tipos de mudanças ocorridos em um determinado período. Como o foco da ferramenta é o código-fonte, não possui suporte para requisitos textuais ou modelos UML, não oferecendo recursos de rastreabilidade horizontal. Além disto, não permite utilização *on-line*, versionamento dos requisitos, notificação sobre alterações e não suporta mecanismos de proatividade, *workflow*, configuração de projeto e extensibilidade.

3. Trabalho proposto

3.1. Visão e objetivos

O método proposto no presente trabalho consiste em um sistema composto por agentes de software que auxiliam na gestão da mudança de requisitos por meio da utilização das informações de rastreabilidade e da realização da análise de impacto. A figura a seguir procura demonstrar uma visão geral do sistema, que é composto por três dimensões: ambiente, agentes e modelo, que serão detalhadas a seguir.

A dimensão *ambiente* representa os artefatos de um software, tais como requisitos textuais, modelos UML ou código-fonte, armazenados em um repositório centralizado. O presente trabalho assume que os artefatos de um projeto estejam armazenados em um repositório *subversion*, com o qual oferece meios de integração.

A dimensão *agentes* representa os agentes de software responsáveis pela interpretação dos diferentes modelos de artefatos, pela detecção de relacionamentos de rastreabilidade, pelo registro deste conhecimento na base de dados, pela detecção de mudanças no modelo e pela notificação dos *stakeholders* envolvidos no projeto.

A dimensão *modelo* representa os fatos conhecidos sobre o ambiente, ou seja, quais são os artefatos de software (requisitos, classes, métodos) e quais são seus relacionamentos. O conhecimento é descoberto por agentes de software e armazenado em uma base de dados cujo modelo será detalhado no tópico 3.3.

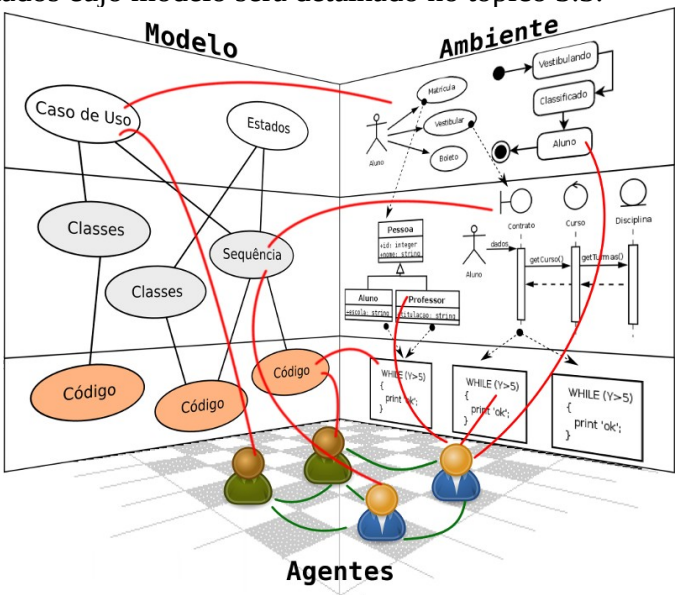

Figura 1. Visão geral da aplicação

O objetivo principal da aplicação é gerenciar a mudança de requisitos. Para tal, este objetivo será desmembrado em quatro capacidades necessárias para o sucesso da aplicação do método proposto, que são descritos a seguir.

- **Manter um modelo**: Consiste na interpretação dos artefatos e de seus relacionamentos, na análise de sua estrutura, detecção e registro das mudanças ocorridas e gerenciamento de diferentes versões do modelo;
- **Gerenciar o processo**: Conhecimento sobre o processo de software, suas etapas e papéis. Compreende a notificação de *stakeholders* sobre mudanças, inconsistências no modelo e a solicitação de informações;

- **Gerenciar a rastreabilidade**: Compreende a descoberta da rastreabilidade, a verificação de sua consistência e o registro destas informações na base de dados;
- **Realizar análise de impacto**: Compreende a identificação dos artefatos afetados por uma determinada mudança e a identificação dos *stakeholders* envolvidos.

3.2. Arquitetura

A arquitetura do sistema é modular e composta por quatro "fronteiras". No centro da figura a seguir, são representados quatro agentes de software que serão detalhados adiante. Cada agente é responsável por realizar um dos objetivos traçados no tópico 3.2.

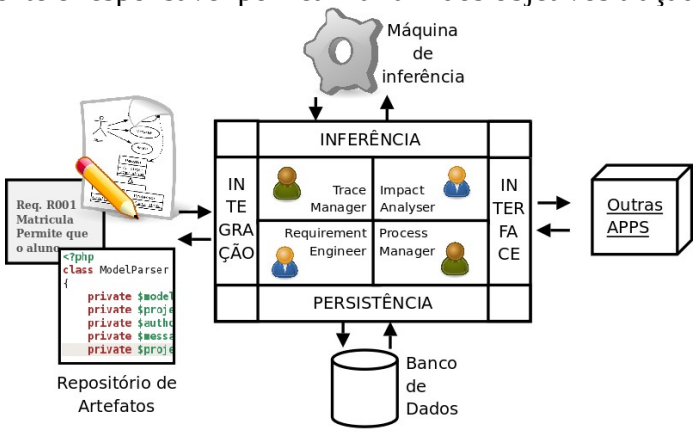

Figura 2. Arquitetura do sistema

Os agentes da aplicação (representados ao centro) são implementados na forma de *Web Services*. Assim, outras aplicações poderão ser construídas sobre eles, o que é representado pela fronteira "interface". O sistema, por meio de seus agentes, possui integração com diferentes tipos de artefatos como requisitos textuais, modelos UML e código-fonte, o que é representado pela camada "integração". O conhecimento obtido pelos agentes a partir dos artefatos do sistema alimenta o modelo apresentado no tópico 3.3. Para realizar a análise de impacto, os agentes se comunicam com uma máquina de inferência externa, no caso deste trabalho o software SWI-Prolog.

3.3. Modelo

Para registrar relações de rastreabilidade entre artefatos de um software, um modelo de rastreabilidade se faz necessário. Desta forma, é proposto um modelo de rastreabilidade com suporte à configuração de granularidade, à gestão da mudança, gestão de impactos e versionamento, conforme pode ser visto a seguir.

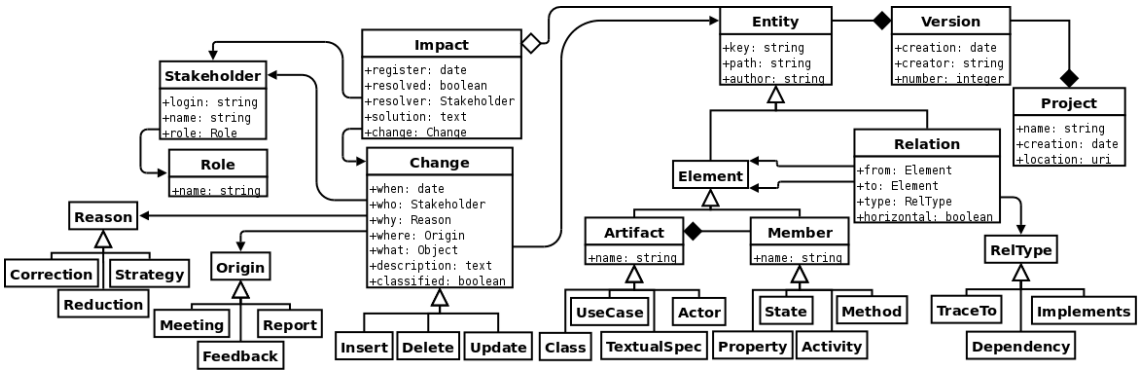

Figura 3. Modelo do sistema

O modelo é capaz de armazenar diferentes versões de um projeto, permitindo que se tenha o histórico de alterações que ocorreram ao longo do tempo. Assim, determinado projeto (*Project*) é composto de versões (*Version*). Uma versão é composta de entidades (*Entity*), que indica um elemento (*Element*) do modelo (caso de uso, classe, atributo, método, código-fonte) ou uma relação (*Relation*) entre dois elementos.

O modelo também é capaz de representar a rastreabilidade vertical e horizontal. Além disto, suporta relacionamentos entre elementos *Coarse-Grained*, como uma classe UML ou um código-fonte e relacionamentos entre elementos *Fine-Grained*, como um método ou um atributo de uma classe UML. Para tal, o modelo representa uma entidade (*Entity*), que pode ser um elemento (*Element*) ou uma relação entre dois elementos (*Relation*). Uma relação poderá ter um tipo (*TraceTo*, *Implements*, *Dependency*, dentre outros). Já um elemento poderá ser um artefato (requisito textual, classe, caso de uso) ou um membro de um artefato (propriedade, método, estado). Um artefato poderá ser composto por diversos membros. A configuração da granularidade se dará no relacionamento entre elementos, uma vez que a superclasse *Element* é uma generalização tanto de *Artifact*, quanto de *Member*. Desta forma, poderá existir um relacionamento entre artefatos, entre membros e entre artefatos e membros.

O modelo também permite o registro das mudanças ocorridas, suas origens, seus responsáveis e motivos. Uma mudança (*Change*) ocorre em uma entidade (*Entity*), que pode ser elemento de *coase-grained* ou *fine-grained*. Uma mudança é realizada por um colaborador (*stakeholder*), que por sua vez possui um papel (*Role*). Cada mudança possui uma razão (*Reason*), uma origem (*Origin*) e um tipo: adição, exclusão ou mudança, conforme classificação proposta por Nurmuliani [Nurmuliani 2004].

O modelo também permite representar os impactos gerados (*Impact*), que estarão relacionados com as mudanças que os originaram (*Change*). Enquanto a relação de mudança (*Change*) com o *Stakeholder* indica o causador da mudança, a relação de *Impact* com *Stakeholder* indica quem resolveu o impacto. Cada impacto poderá afetar um conjunto de entidades (*Entity*), que poderão ser artefatos ou relacionamentos representadas por meio de uma estrutura de agregação.

3.4. Modelagem dos Agentes

Como exibido na figura a seguir, o sistema possui quatro agentes: *RequirementEngineer (a)*, *TraceabilityManager (b)*, *ImpactAnalyser (c) e ProcessManager (d)* .

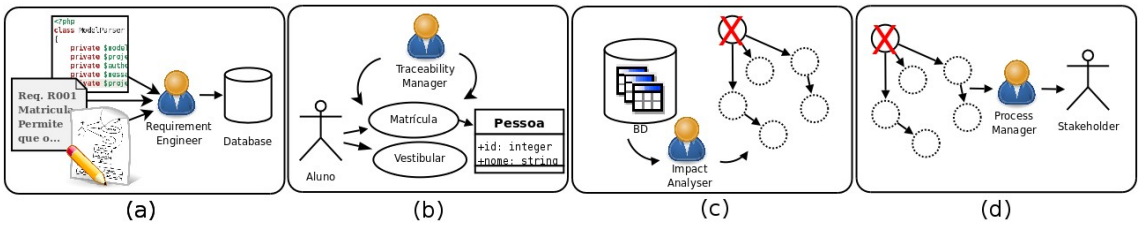

Figura 4. Papéis dos agentes

RequirementEngineer é responsável por manter o modelo de requisitos, interpretar os requisitos e seus relacionamentos, detectar e registrar as mudanças ocorridas e controlar diferentes versões do modelo. Este agente é apoiado por uma estrutura de classes responsáveis pela interpretação de diversos tipos de artefatos tais como: documentos de requisitos armazenados em arquivos padrão ODF (*Open Document Format*), modelos de *design* UML armazenados em arquivos XMI (*XML Metadata Interchange*) e códigos-fonte de aplicações PHP ou Java.

O agente *TraceabilityManager* é responsável pela verificação da consistência da rastreabilidade em um modelo, pelo registro da rastreabilidade vertical e pela descoberta da rastreabilidade horizontal, por meio da detecção de similaridade entre termos de diferentes modelos de artefatos.

O agente *ImpactAnalyser* é responsável por identificar o conjunto de requisitos afetados por uma mudança e dimensionar este impacto por meio da aplicação de determinadas métricas. A análise de impacto é realizada por meio da exportação das informações de rastreabilidade contidas no modelo para um conjunto de axiomas em *Prolog* contendo toda a estrutura de interdependência do projeto. A partir de um conjunto de funções em lógica descritiva, este agente é capaz de identificar, a partir de uma mudança, todo o conjunto de artefatos afetados no projeto.

O agente *ProcessManager* é responsável por conhecer o processo, bem como as informações sobre o projeto em desenvolvimento, tais como: *stakeholders*, atribuições e alocações. A partir do conhecimento que possui sobre o projeto, o agente *ProcessManager* notifica os *stakeholders* sobre mudanças e impactos ocorridos.

3.5. Método

Na figura seguir, demonstra-se como os agentes iteragem durante as atividades. Em primeiro lugar, o *stakeholder* modifica um artefato. O agente *RequirementEngineer*, que monitora frequentemente o repositório de artefatos, realiza a interpretação destes e registra uma nova versão do modelo, mantendo o histórico da evolução dos requisitos (*baseline*). Após uma nova versão registrada, o agente *RequirementEngineer* detecta as mudanças ocorridas. Para cada mudança, ele aciona o agente *ProcessManage*r, que irá notificar os *stakeholders* envolvidos na modificação, solicitando que os mesmos classifiquem as mudanças. Após esta classificação, o agente *ImpactAnalyser* registra as mudanças na base de dados.

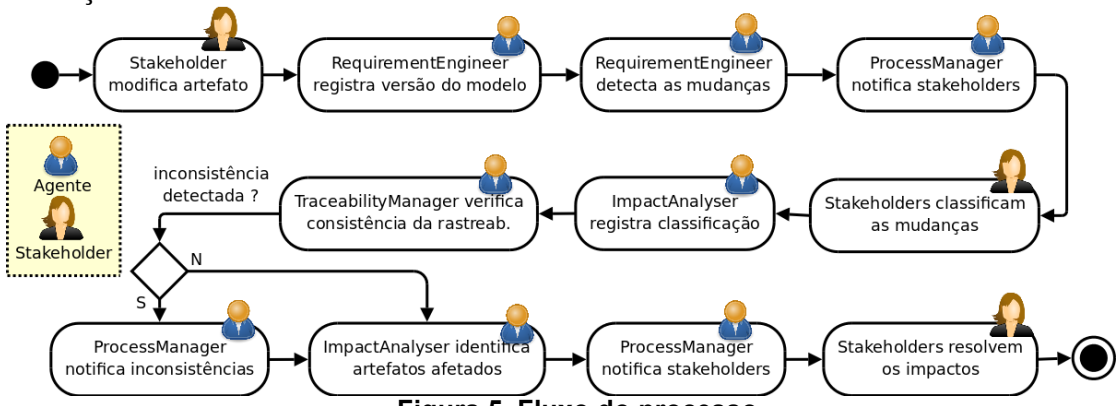

Figura 5. Fluxo do processo

A partir do registro das mudanças, o agente *TraceabilityManager* procura no modelo por inconsistências na rastreabilidade. Caso encontre alguma, notifica ao gerente do projeto. Após, o agente *ImpactAnalyser* irá identificar os requisitos afetados pelas mudanças de forma recursiva, por meio da rastreabilidade. A partir dos impactos identificados, *ProcessManager* irá notificar os *stakeholders* envolvidos. A última etapa consiste na resolução de impactos, que é realizada pelos *stakeholders* no sistema.

Como visto, a partir das mudanças registradas, o agente *ImpactAnalyser* realiza a análise de impacto, que se dá pela transposição do conhecimento armazenado na base de dados para uma estrutura de axiomas em *Prolog*. A partir destes axiomas, a aplicação

comunica-se com uma máquina de inferência (neste caso *SWI-Prolog*) e realiza questionamentos para aferir dependências.

No exemplo a seguir, são declarados: o requisito *Realizar Matricula*, o ator *Aluno*, o caso de uso *Matricular* e as classes *Aluno* e *Matricula*, além de membros de artefatos, como a palavra *aluno* do requisito *Realizar Matricula*, e os métodos *Matricular* e *Trancar* da classe *Aluno*. Após são definidas relações de rastreabilidade horizontal e vertical por meio do operador *traceto*. Então, o agente *ImpactAnalyser* pode descobrir os artefatos dependentes por meio das funções específicas *dependof, findall e write_table*, para registrar os impactos e notificar os *stakeholders* responsáveis.

Tabela 1. Representação em Prolog

Definição de artefatos e relacionamentos

`requirement('Realizar Matrícula').`
`actor('Aluno').`
`usecase('Matricular').`
`class('Aluno').`
`class('Matricula').`
`member(requirement('Realizar Matrícula'), word('aluno')).`
`member(class('Aluno'), method('Matricular')).`
`member(class('Aluno'), method('Trancar')).`
`partof( requirement('Realizar Matrícula'),`
`member(requirement('Realizar Matrícula'), word('aluno'))).`
`partof( class('Aluno'), member(class('Aluno'), method('Matricular'))).`
`partof( class('Aluno'), member(class('Aluno'), method('Trancar'))).`
`traceto(actor('Aluno'), 'association', usecase('Matricular')).`
`traceto(class('Aluno'), 'composition', class('Matricula')).`
`traceto(usecase('Matricular'), 'association', requirement('Realizar Matrícula')).`
`traceto(member(class('Aluno'), method('Matricular')), 'association',`
`usecase('Matricular')).`

Descoberta de artefatos dependentes
`findall(X,dependof(X, requirement('Realizar Matrícula')),List), write_table(List).`

`member(requirement(Realizar Matrícula), word(aluno))`
`usecase(Matricular)`
`actor(Aluno)`
`member(class(Aluno), method(Matricular))`

3.6. Aplicação

Sobre a infra-estrutura de *Web Services* criada para representar os agentes, foi criado um protótipo para explorar todas as funcionalidades destes. Este protótipo implementa o método proposto e disponibiliza para o usuário uma interface *web* para gerenciar o projeto, os *stakeholders*, as mudanças, os impactos gerados, emitir relatórios gerenciais dentre outros. A figura a seguir mostra a arquitetura deste protótipo, onde é possível visualizar a aplicação com seus componentes comunicando-se por meio do protocolo SOAP com os agentes disponibilizados por *Web Services*.

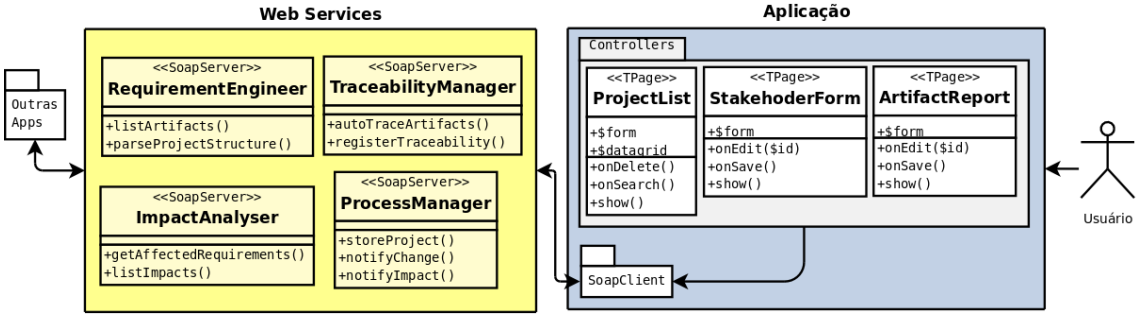

Figura 6. Arquitetura do protótipo

O protótipo fornece um *dashboard*, onde é possível visualizar as mudanças detectadas, não classificadas, os impactos gerados e ainda não resolvidos pelos *stakeholders*. O *dashboard* permite ao usuário saltar para uma etapa do processo, como classificar mudanças ou resolver impactos, clicando sobre a etapa. A figura a seguir exibe o *dashboard* (7a) e a interface de classificação de mudanças (7b), que pode ser acionada mediante o clique na etapa correspondente, na qual o usuário seleciona os artefatos que tiveram algum tipo de modificação e indica qual foi a razão e a origem da modificação conforme taxonomia de Nurmuliani [Nurmuliani 2004]. Além disto, sempre que uma mudança ocorre, o *stakeholder* causador recebe um e-mail contendo uma mensagem com um atalho que permite que ele acesse diretamente esta interface.

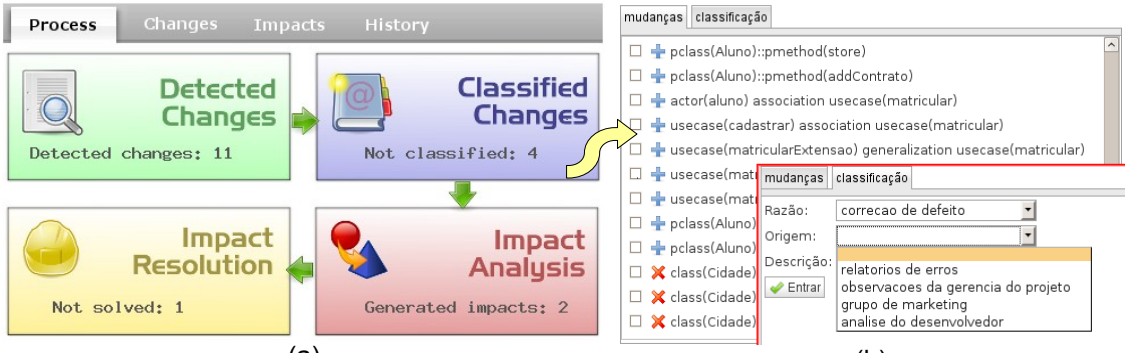

Figura 7. Dashboard e classificação de mudanças

Após classificar as mudanças, a ferramenta realiza a análise de impacto e notifica os *stakeholders* afetados pelos impactos causados. Para realizar a análise de impacto, a aplicação utiliza a informação de rastreabilidade alimentada pelos agentes a partir da leitura e interpretação dos artefatos e armazenada em sua base de dados.

A figura a seguir (8a), gerada pela ferramenta com o uso da biblioteca *GraphViz*, procura demonstrar as relações de rastreabilidade armazenadas pela ferramenta com alto nível de granularidade, entre requisitos, modelos de *design* e código-fonte.

Todas as operações alimentam métricas demonstradas por gráficos gerenciais que exibem indicadores. Na figura a seguir (8b) são exibidos os impactos ocorridos a partir das mudanças classificadas pelo usuário conforme a classificaçao de Nurmuliani [Nurmuliani 2004], ou seja, por razão e por origem. Além da quantidade de impactos, também é exibida a quantidade de artefatos impactados.

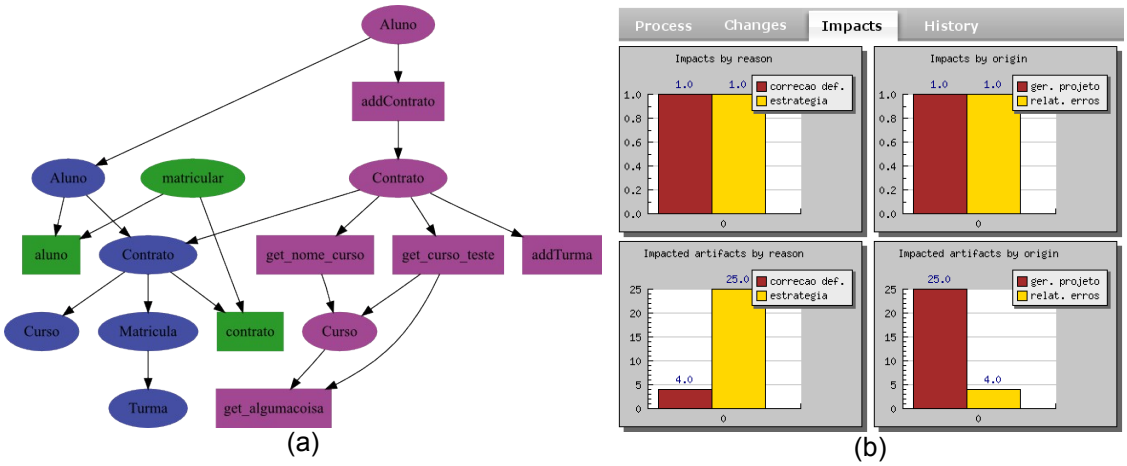

Figura 8. Grafo de rastreabilidade e estatísticas de impactos

4. Comparação entre métodos

Como forma de comparar os métodos estudados frente ao proposto, foram pesquisados na literatura alguns trabalhos que buscam a definição de critérios para avaliar ferramentas de gestão de requisitos. Dentre estes trabalhos destacam-se Hoffmann [Hoffmann 2004], Lang [Lang 2001], Cleland-Huang [Cleland-Huang 2007] e Fasano [Fasano 2007].

Hoffmann [Hoffmann 2004], desenvolveu um catálogo de requisitos para o desenvolvimento de ferramentas deste tipo, dentre os quais destacam-se: permitir requisições de mudanças; suportar versões dos artefatos (*baseline)*; permitir rastreabilidade; prover interfaces com outras ferramentas; permitir trabalho colaborativo e funções de análise de progresso do projeto. Além de Hoffmann, Lang [Lang 2001] elaborou um pequeno catálogo de requisitos com o foco em colaboração. Dentre os requisitos, destacam-se: classificar requisitos em grupos lógicos; definir relacionamentos rastreáveis entre os requisitos; verificar alocação de requisitos à especificações de *design*; manter arquivo das versões de mudanças; permitir mecanismo para aprovação de mudanças; suportar trabalho distribuído; manter repositório compartilhado de artefatos e suportar comunicação com outras ferramentas.

Já Cleland-Huang [Cleland-Huang 2007] sugere um conjunto de "boas-práticas" para ferramentas de gestão de requisitos, especialmente aquelas que suportam rastreabilidade, dentre as quais merecem destaque: permitir ao usuário localizar a origem e o destino de cada *link* de rastreabilidade; permitir aos colaboradores decidir qual o nível de granularidade da rastreabilidade e prover rastreabilidade para artefatos construídos e armazenados em seus ambientes nativos. Fasano [Fasano 2007] adiciona mais alguns critérios de comparação em seu trabalho, dentre os quais destacam-se: prover suporte ao gerenciamento do projeto; suportar mecanismo automático ou semi-automático para recuperação da rastreabilidade; permitir notificação de eventos disparados quando algum artefato é alterado e permitir a propagação de eventos, ou seja, a propagação da notificação conforme o grafo de rastreabilidade.

4.1. Critérios para comparação

Com base nos critérios apontados por Hoffmann, Lang, Cleland-Huang e Fasano, definiu-se um grupo de 20 critérios a serem utilizados a fim de se comparar os métodos avaliados frente ao método proposto. A seguir, consta uma tabela com a codificação utilizada mais adiante na tabela de comparação.

Tabela 2. Critérios de comparação

ID	Descrição	ID	Descrição
C01	Suporta o gerenciamento do projeto	C11	Suporta requisitos textuais
C02	Suporta o gerenciamento de mudanças	C12	Suporta modelos de design UML
C03	Suporta rastreabilidade entre artefatos	C13	Suporta código-fonte
C04	Suporta análise de impactos	C14	Recuperação da rastreab. vertical
C05	Suporta notificação de eventos	C15	Recuperação da rastreab. horizontal
C06	Suporta propagação de eventos	C16	Parsing das estruturas dos document.
C07	Suporta utilização on-line	C17	Gerenciamento proativo
C08	Suporta integração com outras ferramentas	C18	Acompanhamento processual (workflow)
C09	Suporta rastreabilidade fine-grained	C19	Fornece estatísticas gerenciais
C10	Suporta versionamento dos artefatos	C20	Permite extensibilidade

4.2. Tabela comparativa

Desta forma, se estabelece, conforme pode ser visto pela tabela a seguir, a comparação dos métodos conforme os critérios selecionados. Cada ponto de interseção representa um critério atendido. Nas linhas são exibidos os critérios e nas colunas os métodos. A verificação quanto a aceitação dos critérios por parte do método proposto foi realizada juntamente com a aplicação do protótipo em um ambiente real de desenvolvimento.

Tabela 3. Comparação entre métodos

	ADAMS	EBT	iACMTool	Chianti	Proposta		ADAMS	EBT	iACMTool	Chianti	Proposta
C01	x				x	C11	x	x			x
C02	x	x	x	x	x	C12	x		x		x
C03	x	x	x	x	x	C13	x			x	x
C04	x	x	x	x	x	C14	x		x	x	x
C05	x	x	x		x	C15	x				x
C06	x	x	x	x	x	C16	x	x	x	x	x
C07	x	x			x	C17		x			x
C08	x	x	x	x	x	C18		x	x		x
C09	x			x	x	C19			x	x	x
C10	x		x		x	C20					x

5. Considerações finais e trabalhos futuros

A gestão da mudança de requisitos é uma atividade que exerce papel fundamental em relação à qualidade no processo de desenvolvimento de software. A gestão da mudança está diretamente relacionada com a análise de impactos, que permite dimensionar os custos de uma mudança. A análise de impactos, por sua vez, está relacionada com a informação de rastreabilidade, que provê subsídios para o correto dimensionamento dos impactos gerados por uma mudança.

Para a criação de um ambiente que possibilite a implementação de uma efetiva gestão de mudanças, é necessário o desenvolvimento de um modelo que contemple todos os aspectos relacionados à mudança de requisitos, como o suporte de artefatos de qualquer fase do processo de desenvolvimento, o suporte à rastreabilidade (horizontal e vertical), a configuração da granularidade, a gestão dos impactos, o versionamento dos requisitos, dentre outros critérios. A criação de um modelo que suporte os critérios apontados permite o desenvolvimento de um método que explore todo o potencial da gerência da mudança de requisitos.

Neste sentido, a maior contribuição deste trabalho está em desenvolver um método de gestão da mudança com suporte à rastreabilidade e à análise de impactos, suprindo um conjunto de requisitos não encontrados em sua totalidade em nenhum outro método dentre os trabalhos avaliados.

O desenvolvimento do método proposto no presente trabalho possibilita a criação de outras aplicações para gestão da mudança de requisitos, que venham a implementar funcionalidades adicionais para esta aplicação, visto que tal método destaca-se por permitir adaptabilidade e extensibilidade por meio de *Web Services* que expõem todas as funcionalidades descritas no presente trabalho. Desta forma, trabalhos futuros podem explorar a criação de novas funcionalidades sem a necessidade de reescrever os recursos já existentes na ferramenta, bastando para isto, a compatibilidade com os protocolos utilizados pelos *Web Services*.

Referências

Beck, Kent. ANDRES, Cynthia. Extreme Programming Explained : Embrace Change (2nd Edition). 224 pages. Publisher: Addison-Wesley Professional; 2 edition (November 16, 2004).

Briand, L., Labiche, Y., O'Sullivan, L. "Impact Analysis and Change Management of UML Models". Software Maintenance, IEEE International Conference on, vol. 0, no. 0, pp. 256, 19th IEEE International Conference on Software Maintenance (ICSM'03), 2003.

Cleland-Huang, J., Chang, C. K., and Christensen, M. 2003. Event-Based Traceability for Managing Evolutionary Change. IEEE Trans. Softw. Eng. 29, 9 (Sep. 2003), 796-810.

Cleland-Huang, J.; Settimi, R.; Romanova, E.; Berenach, B., and Clark, S., Best practices for automated traceability, IEEE Computer Magazine. Vol. 40, no. 6, p. 27-35-35. 2007.

De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. 2005. ADAMS Re-Trace: A Traceability Recovery Tool. InProceedings of the Ninth European Conference on Software Maintenance and Reengineering (March 21 - 23, 2005). CSMR. IEEE Computer Society, Washington, DC, 32-41.

Fasano, Fausto. Fine-grained management of software artefacts. In Software Maintenance, 2007. ICSM 2007. IEEE International Conf., 2007, pp. 507-508.

Hoffmann, M., Kuhn, N., Weber, M., and Bittner, M. 2004. Requirements for Requirements Management Tools. In Proceedings of the Requirements Engineering Conference, 12th IEEE international (September 06 - 10, 2004). RE. IEEE Computer Society, Washington, DC, 301-308.

Lang, M.. and Duggan, J. 2001. A Tool to Support Collaborative Software Requirements Management. Requirements Engineering Journal. Vol. 6. No. 3.

Mader, P., Gotel, O., and Philippow, I. 2009. Getting back to basics: Promoting the use of a traceability information model in practice. In Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering (May 18 - 18, 2009). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 21-25.

Nurmuliani, Zowghi D., and Fowell S., Analysis of Requirements Volatility during Software Development Life Cycle, proceedings of the Australian Software Engineering Conference (ASWEC), April 13-16, Melbourne, Australia, 2004.

Nuseibeh, B. Easterbrook, S. Requirements Engineering: A Roadmap. Proceedings of International Conference on Software Engineering (ICSE-2000), 4-11 June 2000, Limerick, Ireland.

Ren, X., Shah, F., Tip, F., Ryder, B. G., and Chesley, O. 2004. Chianti: a tool for change impact analysis of java programs. InProceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (Vancouver, BC, Canada, October 24 - 28, 2004). OOPSLA '04. ACM, New York, NY, 432-448.