Um Modelo de Rastreabilidade com suporte ao Gerenciamento de Mudanças e Análise de Impacto

Pablo Dall'Oglio, João Pablo Silva da Silva, Sérgio Crespo Coelho da Silva Pinto
UNISINOS – Universidade do Vale do Rio dos Sinos
{pablo.dalloglio, jpabloss}@gmail.com, crespo@unisinos.br

Resumo

É de senso comum que a gestão da mudança de requisitos exerce papel fundamental no processo de desenvolvimento de software, principalmente no que tange à qualidade. Para que se possa gerenciar a mudança de requisitos de forma efetiva e precisa, torna-se indispensável o uso de informações de rastreabilidade, que por sua vez, torna possível a realização da análise de impacto. Para que se possa efetivamente gerenciar as mudanças de software é necessário a existência de um modelo que permita representar com precisão a rastreabilidade de software, as mudanças ocorridas e os impactos gerados. Apesar destes assuntos serem de suma importância dentro da área da engenharia de requisitos, parece não haver um modelo que permita representar estes aspectos ao mesmo tempo de forma que supra os requisitos necessários identificados na literatura. Desta forma, a proposta deste trabalho é o desenvolvimento de um modelo de rastreabilidade que suporte o gerenciamento da mudança e análise de impacto, de forma que atenda aos requisitos identificados nos trabalhos já desenvolvidos na área.

1. Introdução

Para Leite [13], a sociedade tem passado a ver software como parte de suas vidas e isto tem gerado uma maior demanda por qualidade nos produtos. O desafio da engenharia de software está em prover tal qualidade com os limitados recursos disponíveis.

Muitas metodologias surgiram para organizar o desenvolvimento de software, a maioria delas baseada na previsibilidade dos requisitos [2]. Como nem sempre é possível obter todos requisitos antes da construção do software, é necessário saber o que fazer quando ocorrer uma mudança [2].

Neste contexto, ganha importância a engenharia de requisitos, mais especificamente a gestão da mudança de requisitos, uma vez que o sucesso do software vai depender de quanto ele se adequar às mudanças de ambiente [16].

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 de uma mudança [18]. Dentre as formas de se realizar a análise de impacto, a rastreabilidade é a principal [1]. 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 um modelo que permita representar com precisão a rastreabilidade entre requisitos, as mudanças ocorridas e os impactos gerados. Apesar de vários modelos já terem sido propostos com este objetivo, são percebidas diversas deficiências nestes, sendo que nenhum é tido como completo conforme os critérios adotados no presente artigo. Para a maioria dos autores, não há um consenso sobre qual tipo de informação deva integrar o modelo de rastreabilidade, bem como a semântica mais adequada para utilizar em seus relacionamentos.

Este trabalho propõe um novo modelo de rastreabilidade por meio da unificação e extensão de aspectos encontradas nos principais modelos já desenvolvidos. O objetivo deste modelo é atender ao maior número de requisitos catalogados possível, permitindo assim, a criação de aplicações que possam explorar todos as potencialidades inerentes ao gerenciamento de mudanças e a análise de impacto.

Para tal, na seção 2 serão abordados os principais tópicos relacionados à área em questão, na seção 3 serão abordados os principais trabalhos relacionados, na seção 4 será abordado o modelo proposto, bem como suas diversas facetas, na seção 5 serão apresentados critérios de comparação entre os trabalhos relacionados e o modelo proposto e na seção 6 serão apresentadas as conclusões.

Desta maneira, a partir dos modelos avaliados, bem como dos requisitos identificados, será elaborado um modelo de representação de requisitos que suporte as características essenciais identificadas e procure suprir os requisitos para os quais não foi encontrado suporte nos modelos estudados.

2. Contexto

Como visto, para que se possa gerenciar a mudança de requisitos, é indispensável o uso da rastreabilidade, que por sua vez, possibilita a realização da análise de impacto, conceitos que serão abordados neste capítulo.

2.1. Gestão da mudança

Para De Grande [5], os requisitos evoluem com o tempo, seja por erros detectados, seja pela evolução do conhecimento do cliente. O gerenciamento dos requisitos é necessário para que se possa controlar as mudanças e refletir estas mudanças no sistema e nos objetivos da organização, dando subsídios para as análises de custos e impactos.

Nurmuliani [15] identifica quatro fases que integram o processo de gerência da mudança de requisitos:

- **Requisição inicial**: um membro da equipe submete uma proposta de mudança;
- **Validação e Avaliação**: a mudança é validada em função da análise de impacto;
- **Implementação**: a mudança aceita e aprovada é implementada e passa a ser parte do sistema;
- **Verificação**: é verificado se a mudança foi implementada corretamente;

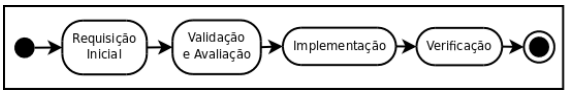

Figura 1. Fases da mudança de requisitos

2.2. Análise de impacto

A análise de impacto é atividade essencial da gestão da mudança de requisitos e identifica as entidades que são possivelmente afetadas por uma mudança proposta no sistema.

De acordo com Kotonya [11], são seis as atividades decorrentes da etapa de análise de impacto:

1. Verificar se a solicitação de mudança é válida;
2. Identificar os requisitos afetados diretamente;
3. Descobrir os requisitos dependentes;
4. Propor mudanças nos requisitos;
5. Estimar os custos das mudanças;
6. Avaliar se os custos são aceitáveis;

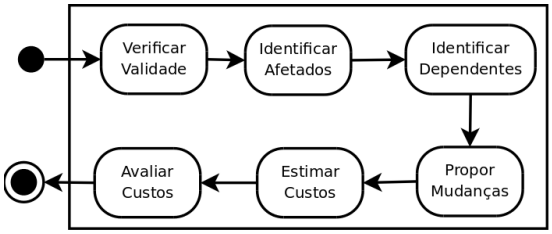

Figura 2. Etapas da análise de impacto

2.3. Rastreabilidade

A rastreabilidade de requisitos consiste em ligações entre as informações produzidas no desenvolvimento de software e é fator chave para que se possa desempenhar a análise de impacto. Para Pinheiro [17], tais ligações são essenciais no desenvolvimento de sistemas que servem tanto para propósitos técnicos quanto gerenciais.

Sayão [20] destaca diversas vantagens no uso da rastreabilidade, das quais pode-se citar:

- Verificação de alocação entre requisitos e sua implementação;
- Validação do sistema, verificando se o mesmo atende ao conjunto de requisitos proposto;
- Estimativas de custos de prazos quando da inserção de uma nova funcionalidade;
- Identificação de riscos que possam impactar os requisitos;
- Identificação de ligações entre código e documentos de análise que possam proporcionar futuro reaproveitamento de código;

A rastreabilidade pode ser horizontal ou vertical. A rastreabilidade vertical está relacionada com a capacidade de relacionar artefatos dependentes dentro de um modelo, enquanto que a rastreabilidade horizontal está relacionada à habilidade de relacionar artefatos entre diferentes modelos. Entre os artefatos que podem estar relacionados estão requisitos, artefatos de análise, de design, código-fonte, casos de teste, dentre outros.

Para De Lucia [6], a rastreabilidade vertical fornece apenas uma visão limitada sobre os artefatos afetados por uma mudança. Sistemas complexos necessitam de modelos com vários níveis de abstração, como por exemplo, o código-fonte. Desta forma, muitas abordagens acabam estendendo a rastreabilidade vertical com a horizontal, permitindo gerenciar dependências entre requisitos e artefatos de design, entre requisitos e o código-fonte, entre requisitos e casos de teste, e entre artefatos de design e o código-fonte. Na figura 3, é exibida uma representação da rastreabilidade horizontal.

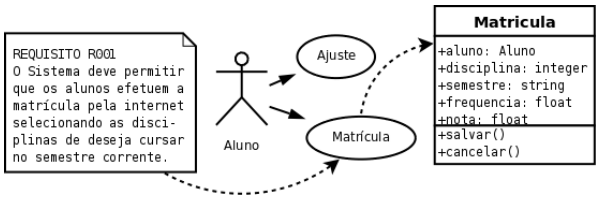

Figura 3. Rastreabilidade horizontal

3. Trabalhos relacionados

Nesta seção serão apresentados os principais trabalhos relacionados que propõem modelos de representação de requisitos. A partir destes trabalhos, será proposto um modelo único, com o objetivo de atender à todos os requisitos identificados. Alguns dos modelos apresentados aqui passaram por um processo de editoração eletrônica para que pudessem se adaptar ao espaço disponível.

3.1. Modelo de Ramesh

Um dos pioneiros a abordar a rastreabilidade de requisitos foi Ramesh [19]. Apesar de já terem decorridos muitos anos desde seu estudo, é relevante ressaltar a importânicia histórica de seu trabalho. Os artigos de Ramesh [19] introduziram questionamentos relevantes sobre a rastreabilidade e propuseram um dos primeiros modelos de processos desta área.

Conforme Ramesh, na época em que o estudo fora realizado não haviam indícios na literatura sobre um modelo que representasse explicitamente o tipo de informação a ser rastreada. Desta forma, o autor apresenta um estudo de caso sobre a aplicação de rastreabilidade e desenvolve um modelo que descreve a sua prática em uma organização. A figura 4 procura demonstrar o modelo criado por Ramesh. Este modelo identifica a informação de rastreabilidade, os requisitos, raciocínios (*rationale*), alocação de requisitos e de recursos.

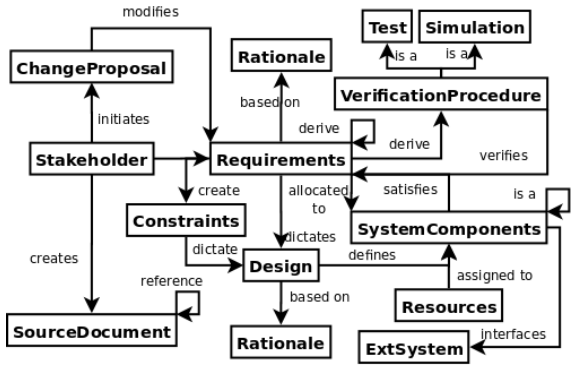

Figura 4. Modelo de Ramesh

Como características relevantes deste modelo destacam-se: o relacionamento entre requisitos (*requirements*) e suas derivações, entre estes com os documentos de origem (*rationale*), entre requisitos e documentos de testes (*test*) entre requisitos e componentes (*System Componentes*), entre requisitos e *stakeholders*, dentre outros. O modelo permite ainda representar as solicitações de mudanças propostas (*Change Proposal*). Entretanto, o modelo não comporta a representação dos impactos gerados por uma mudança e também não possibilita representar versionamento e configuração de granularidade.

3.2. Modelo de Lang

Lang [12] apresenta um protótipo chamado *RM-Tool*, uma ferramenta que gerencia e controla requisitos dentro de um processo multidisciplinar. Este protótipo possui um escopo limitado e não ambiciona cobrir todos tipos de requisitos que fazem parte do desenvolvimento de um sistema. Basicamente, a ferramenta proposta, bem como o modelo representado na figura 5, focam nas características que não são cobertas satisfatoriamente por ferramentas comerciais.

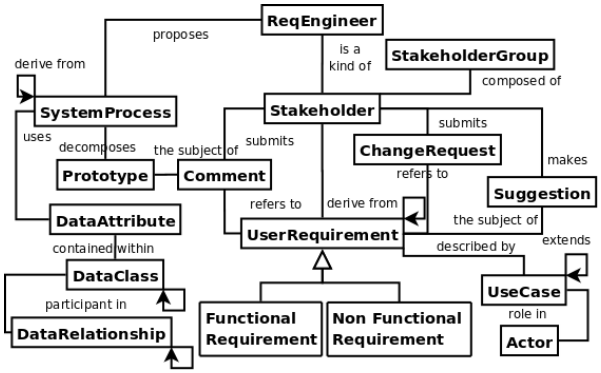

Figura 5. Modelo de Lang

Dentre as características do modelo proposto, encontram-se o suporte a rastreabilidade entre requisitos (vertical) e entre requisitos e outros artefatos como *use cases* (horizontal). Além de explicitamente representar as mudanças ocorridas (*Change Request*), permite ainda a segmentação de requisitos em funcionais e não-funcionais e o relacionamento com S*takeholders*. Da mesma maneira que o modelo de Ramesh, o modelo de Lang não permite representar impactos, configuração de granularidade e versionamento.

3.3. Modelo de Letelier

Para Letelier [14], a UML surge como um meio de estabelecer um modelo comum para representar a rastreabilidade entre requisitos. Desta forma, o autor apresenta um modelo para rastreabilidade de requisitos baseado na linguagem UML, que possibilita integrar desde especificações textuais até elementos de modelos UML. Assim, o autor espera obter uma representação em comum que possa ser utilizada para identificar quaisquer artefatos de software, bem como as relações entre eles.

Na figura 6, é apresentado um diagrama de classes contendo o modelo de rastreabilidade de requisitos proposto por Letelier. Neste modelo, as classes filhas de *TraceableSpecification* representam os tipos de artefatos que encontram representação no modelo e as associações (*modifies*, *responsibleOf*, *rationaleOf*, *validatedBy*, *verifiedBy* e *assignedTo*) representam os tipos de rastreabilidade.

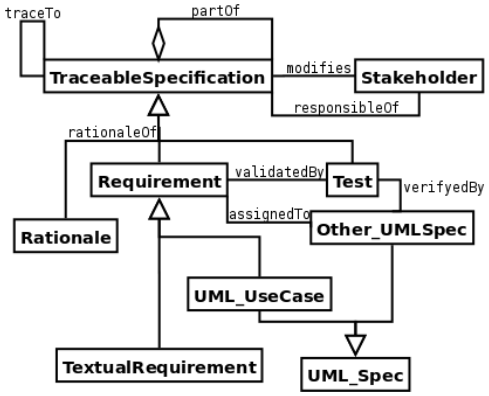

Figura 6. Modelo de Letelier

O modelo proposto por Letelier possui como características positivas a riqueza dos detalhes na representação dos links de rastreabilidade, provendo uma semântica com um nível de detalhamento grande entre os elementos que formam a especificação de software. Além disto, o modelo permite configurar o nível de rastreabilidade, uma vez que a classe *TraceableSpecification* permite compor uma estrutura de agregação com a própria classe, que suporta desde requisitos textuais até artefatos em UML e permite definir rastreabilidade vertical e horizontal. O modelo também permite ser extendido por meio da adição de novos esteriótipos UML. Entretanto, o modelo não considera código-fonte em sua definição, assim como não permite representar mudanças, impactos, tampouco versionamento de requisitos.

3.4. Modelo de Briand

Para Briand [3], a utilização de modelos em linguagem UML (*Unified Modeling Language*) acarreta a criação de uma série de diagramas inter-relacionados que podem ser impactados por mudanças de requisitos ou mudanças nestes próprios modelos. Para o autor, é objetivo da análise de impacto identificar o que pode ser afetado por uma mudança e, com base nesta informação, manter os diagramas atualizados e consistentes. Desta forma, pode-se identificar o custo e a complexidade das mudanças ocorridas e, desta forma, decidir implementar ou não tais modificações no sistema.

Para atingir os objetivos citados, permitindo melhorar o processo decisório e o planejamento de mudanças, Briand propõe um modelo UML que suporte o gerenciamento de mudanças e a análise de impacto, como pode ser visto na figura 7. Dentre os objetivos do modelo, destacam-se: verificação de consistência, representação de mudanças, análise e priorização de impactos.

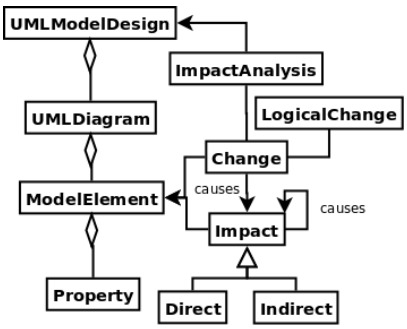

Figura 7. Modelo de Briand

O modelo permite representar mudanças (*Change*) e impactos (*Impact*) sobre elementos de um modelo (*Model Element*) em UML, que pode ser uma classe, mas não uma de suas partes (*Property*). Além disto, o modelo suporta versionamento de requisitos e permite classificar as mudanças e os impactos em diferentes tipos. Destaca-se ainda o foco prioritário do trabalho de Briand em modelos UML, não considerando requisitos textuais ou código-fonte. O autor cita em seu trabalho o relacionamento de modelos UML com código-fonte, porém este relacionamento não encontra correspondente no modelo proposto, uma vez que todas as classes giram em torno do conceito "*UML Diagram*". Pode-se notar que o modelo não suporta rastreabilidade horizontal e não permite relacionar artefatos com *Stakeholders*.

3.5. Modelo de De Grande

De Grande [5] apresenta uma ferramenta automatizada para gerenciar requisitos, chamada SIGERAR. Dentre os objetivos da ferramenta, podem ser citados o gerenciamento das mudanças de requisitos durante todo o ciclo de vida do software e a rastreabilidade entre os requisitos e documentos de especificação. A ferramenta permite coletar, armazenar e manter os requisitos, gerenciar suas mudanças e rastrear as relações entre requisitos e entre requisitos e outros documentos.

A modelagem da ferramenta se deu por meio do uso da UML (*Unified Modeling Language*), como pode ser visto na figura 8. O modelo permite representar os diferentes projetos gerenciados pela organização, controla glossários, termos, usuários e seus vínculos com a organização (*Departamento*), os

requisitos de cada projeto e sua dependência (*RequisitoDependente*), controle de mudanças (*VersaoRequisito*), que contém a descrição da alteração de mudança, seus motivos, solicitante, importância, o custo e o estado da solicitação.

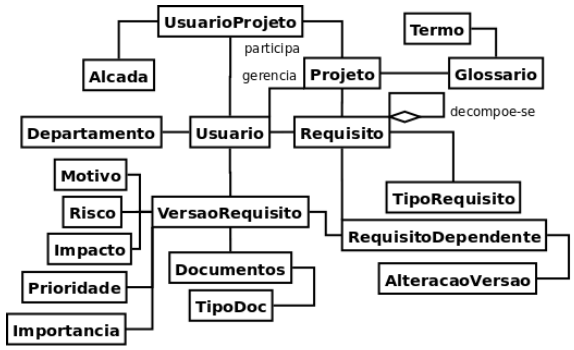

Figura 8. Modelo de De Grande

O modelo possui versionamento (*VersaoRequisito*), que possui dados de controle de versão e relação com documentos envolvidos no processo (*Documentos*), permitindo rastreabilidade. Possui também outras classes (*Risco*, *Impacto*, *Prioridade*) relativas ao requisito. Salienta-se que o modelo de De Grande foca somente em especificações textuais, não suportando a rastreabilidade horizontal, bem como a configuração de granularidade nos relacionamentos.

3.6. Modelo de Kassab

Para Kassab [10], muitas organizações concentram seus esforços na implementação da rastreabilidade para requisitos funcionais, entretanto a rastreabilidade de requisitos não-funcionais tem sido esquecida por completo. Para o autor, um dos motivos que leva à este esquecimento é a dificuldade de sua implementação, uma vez que os requisitos não-funcionais tendem a se distribuir ao longo de diversos módulos quando estes são mapeados no sistema.

Kassab propõe um modelo em linguagem UML para representar os conceitos relacionados aos requisitos não-funcionais, bem como requisitos funcionais e seus relacionamentos. Neste modelo, um requisito funcional está relacionado a um modelo, que está relacionado a uma fase do ciclo de vida do software. Como exemplo, pode-se citar uma classe, que está relacionada ao modelo de classes e este, à fase de *design*. Cada modelo pode ser composto de artefatos (*use case*, diagrama de classes) e o artefato em si pode ser composto de elementos como associações e heranças.

Neste modelo, que pode ser visto na figura 9, um requisito não-funcional como "*a tela do cadastro deve ser web*" pode estar associado à um ou diversos requisitos funcionais. Pode-se notar que o modelo suporta a configuração de granularidade, visto que um requisito pode estar relacionado por meio da classe *Association* com outro requisito ou mesmo com um elemento (*Element*) de outro requisito.

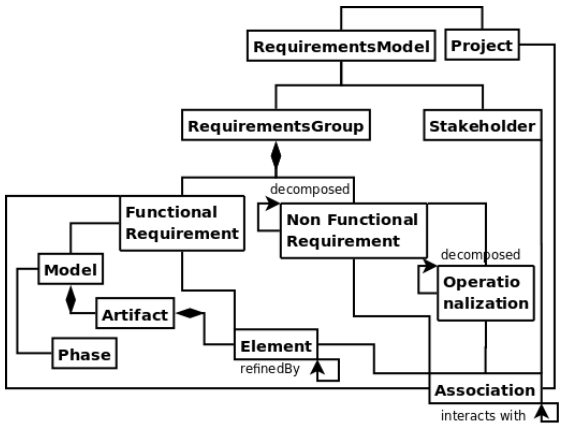

Figura 9. Modelo de Kassab

O modelo suporta a rastreabilidade vertical e horizontal, tendo em vista que permite a associação entre artefatos independentemente do modelo e fase a qual o artefato pertence. Também permite relacionar requisitos com documentos externos, que podem ser representados por requisitos não funcionais e suporta diferentes projetos, representados pela classe *Project*. O modelo não permite representar mudanças e impactos ocorridos. Além disto, o autor não comenta em momento algum a utilização de código-fonte como um artefato do sistema, bem como se trata ou não do versionamento de artefatos. Como o modelo não exibe tais conceitos, pressupõe-se sua ausência.

3.7. Modelo de Ghazi

Para Ghazi [8], a rastreabilidade de requisitos auxilia as instituições a gerenciarem a evolução dos mesmos. Entretanto, as técnicas atuais de rastreabilidade não estão preparadas para considerar sua natureza multi-facetada, que requer informações a partir de múltiplas perspectivas e está relacionada com diferentes disciplinas da engenharia de software.

Para Ghazi [8], é necessário construir um modelo que represente a informação de rastreabilidade, bem como os demais produtos derivados deste processo. Para tal, o autor propõe um método de gerenciamento de rastreabilidade com múltiplas perspectivas (*MV - TMM: A Multi View Traceability Management Method*).

O modelo proposto, que pode ser visto na figura 10 (construído em UML), permite gerenciar diversos tipos de rastreabilidade. O modelo permite representar atores e papéis envolvidos no projeto, bem como seu envolvimento com os artefatos. Além disto, permite

descrever a estrutura dos requisitos, bem como elementos derivados de sua aplicação. O modelo permite relacionar elementos (*TraceElement*) de diferentes fases do ciclo de desenvolvimento, suportando a rastreabilidade vertical e horizontal e classificando estes relacionamentos conforme uma taxonomia própria, que indica o tipo de link (contribuição, dependência, evolução, dentre outros).

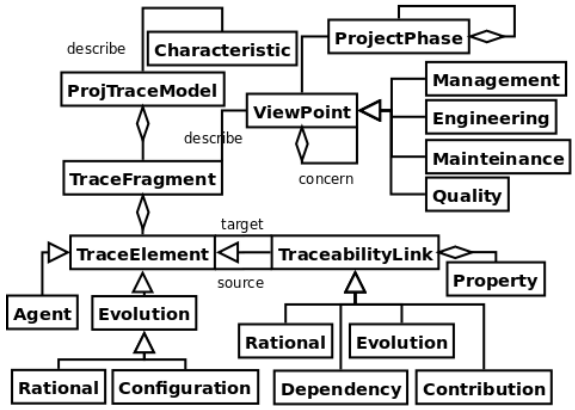

Figura 10. Modelo de Ghazi

Além disto, o modelo possibilita representar a evolução dos links de rastreabilidade de um elemento (*Evolution*) e também relacionar elementos com visões, que representam um determinado aspecto da rastreabilidade a ser capturado, como por exemplo: gerenciamento, engenharia, qualidade e manutenção.

Nota-se que o modelo de Ghazi não comporta a configuração de granularidade, representação de mudanças, de impactos e versionamento de requisitos.

4. Modelo proposto

Dos modelos avaliados, nenhum suporta a representação de artefatos de todo o ciclo de desenvolvimento de software e ao mesmo tempo permite representar a configuração da granularidade, o gerenciamento de mudanças, de impactos e o versionamento dos artefatos. Assim, qualquer tipo de extensão à um destes modelos, iria descaracterizá-los, além de adicionar dificuldades em sua implementação.

A construção de um novo modelo permite integrar todos estes critérios de forma harmoniosa, facilitando sua futura implementação por meio da aplicação de certos padrões (*Design Patterns*) de mapeamento. Desta forma, é proposto um novo modelo de rastreabilidade que suporta a configuração de granularidade, a gestão da mudança, a gestão de impactos e o versionamento dos artefatos.

Inicialmente, o modelo será criado por aspectos, sendo que, cada um destes aspectos será abordado separadamente para após serem consolidados em um modelo único. Na seção 5, o modelo proposto será comparado com os demais modelos estudados.

4.1. Versionamento

Inicialmente, o modelo deverá ser capaz de armazenar diferentes versões de um projeto, permitindo que se tenha o histórico de todas alterações que ocorreram no modelo ao longo do tempo.

Estruturas similares com este objetivo foram encontradas nos trabalhos de Briand [3] e De Grande [5]. Briand propõe uma abordagem para análise de impactos baseada em um modelo UML e representa o versionamento em seu modelo por meio da replicação de estruturas compostas. O trabalho de Briand focava principalmente em artefatos de design (artefatos UML). Além de Briand, De Grande também considera versionamento de requisitos em seu trabalho por meio de uma classe de versionamento. Salienta-se que tal abordagem simplificada funciona devido ao modelo de De Grande ser focado apenas em requisitos textuais.

Como forma de consolidar as idéias apresentadas, é apresentado o primeiro fragmento do modelo proposto, na figura 11.

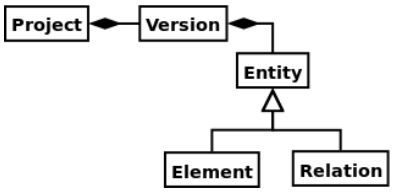

Figura 11. Versionamento

Neste modelo parcial, determinado projeto (*Project*) é composto de versões (*Version*). Uma versão do projeto é composta de entidades (*Entity*), termo genérico que indica um elemento (*Element*) do modelo (diagrama de classe, caso de uso, atributo, método, código-fonte) ou uma relação (*Relation*) entre dois elementos. Esta estrutura permite um fácil mapeamento para uma estrutura Entidade-Relacional e está aderente aos requisitos de versionamento e *baseline* levantados. Sempre que uma nova versão do sistema for detectada, uma nova instância deste objeto será agregada em *Project* e a estrutura composta é replicada, de maneira similar ao trabalho de Briand. Entretanto, a composição é elaborada de maneira a trabalhar com quaisquer artefatos de software (desde requisitos textuais até código-fonte).

4.2. Rastreabilidade

A segunda característica que deverá estar presente no modelo, é a capacidade de representar a rastreabilidade. Neste sentido, faz-se necessário que o modelo permita representar não somente a

rastreabilidade vertical (entre elementos de um mesmo modelo), mas também a rastreabilidade horizontal (entre elementos de modelos diferentes). Além disto, conforme visto em vários trabalhos, a configuração da granularidade da rastreabilidade ainda constitui um problema a ser resolvido.

Modelos de rastreabilidade com aspectos similares foram identificados nos trabalhos de Letelier [14], Briand [3] e Kassab [10]. Letelier propõe um modelo UML que permite registrar requisitos de software, tanto textuais quanto artefatos de design em uma estrutura de agregação recursiva. Entretanto, o modelo de Letelier permite um conjunto finito e pré-definido de relacionamentos entre as entidades do modelo. Já o trabalho de Briand é focado exclusivamente em artefatos UML e não suporta rastreabilidade horizontal, enquanto o de Kassab não considera código-fonte como artefato.

Neste sentido, procurou se desenvolver um modelo que permita o registro de relacionamentos entre requisitos de pouca granularidade (*Coarse-Grained)*, como um documento de especificação textual, um caso de uso ou uma classe UML e também relacionamentos entre requisitos de muita granularidade (*Fine-Grained*), como uma palavra do documento de especificação, um ator, um método de uma classe UML ou um método de uma classe no código-fonte. Desta forma, será possível indicar precisamente em qual parte do código-fonte, determinado método é implementado. Também poderá se representar em qual caso de uso, está determinado ator citado nos requisitos textuais. Para representar a rastreabilidade, é proposto o fragmento de modelo na figura 12.

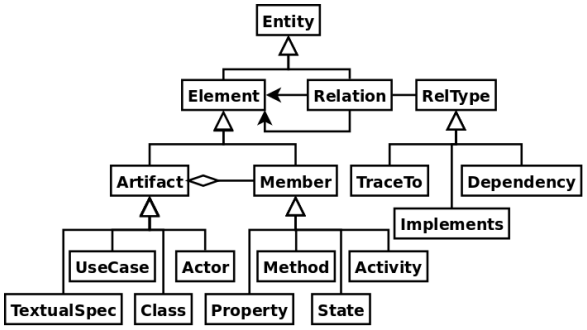

Figura 12. Rastreabilidade

Este modelo parcial inicia em uma entidade (*Entity*), que pode ser um elemento (*Element*) ou uma relação (*Relation*) entre dois elementos. Os dois relacionamentos entre as entidades *Relation* e *Element* indicam que um relacionamento sempre se dá entre dois elementos (origem e destino), como representado posteriormente pelo modelo consolidado. Uma relação poderá ter um tipo (*TraceTo*, *Implements*, *Dependency*, dentre outros aqui não representados em função do espaço). Já um elemento poderá ser um artefato (requisito textual, uma classe, um caso de uso, dentre outros) ou um membro de um artefato (propriedade, método, estado, atividade, dentre outros). Um artefato poderá agregar 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.

4.3. Gestão da mudança

A terceira característica que deve estar presente no modelo é o registro das mudanças ocorridas, suas origens, seus responsáveis e seus motivos. Modelos similares que apresentaram a mudança como um aspecto relevante, foram encontrados nos trabalhos de Ramesh [19], Lang [12], Briand [3] e De Grande [5]. Tanto o modelo de Briand, quanto o modelo de Lang permitem representar somente mudanças ocorridas em elementos de pouca granularidade (*Coarse-Grained*). Já os modelos de Ramesh e De Grande são focados exclusivamente em requisitos textuais. Nenhum dos modelos identifica as origens e razões das mudanças.

Para permitir um controle de mudanças que suporte tanto requisitos de pouca e de grande granularidade, bem como permitir uma classificação das mudanças, é proposto o seguinte fragmento de modelo na figura 13.

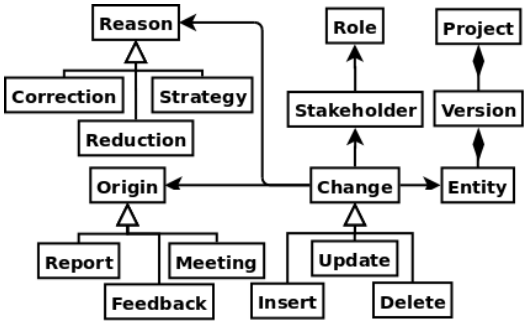

Figura 13. Gestão da mudança

Este modelo parcial é centrado no conceito de mudança (*Change*). Uma mudança ocorre em uma entidade (*Entity*). Como visto anteriormente, uma entidade pode ser um elemento do modelo (caso de uso, classe), partes de elementos (membros) ou um relacionamento entre elementos. Desta forma, uma mudança pode ocorrer em um elemento de pouca granularidade como também em um elemento de muita granularidade. Uma mudança é desencadeada por um colaborador (*stakeholder*), que por sua vez possui um papel (*Role*). Cada mudança possui uma razão (*Reason*) e uma origem (*Origin*). Conforme classificação proposta por Nurmuliani [15], são exemplos de razões: correção de defeito, requisito faltante, melhoria funcional, estratégia de produto, etc e são exemplos de origens: relatórios de erros,

observações da gerência do projeto, do grupo de marketing, análise do desenvolvedor, etc. Além disto, cada mudança pode ser do tipo: adição, exclusão ou uma atualização. No modelo, só foram representados alguns exemplos de razões e origens em função do espaço disponível.

4.4. Gestão de impactos

O próximo aspecto a ser modelado é a representação de impactos gerados a partir de mudanças. Modelos similares que apresentaram o impacto como aspecto relevante, foram encontrados nos trabalhos de Briand [3] e De Grande [5].

O modelo de Briand permite representar um impacto, relacionar este com uma mudança e indicar para cada impacto, apenas um elemento afetado. Desta forma, percebe-se que Briand considera cada elemento afetado por uma mudança no sistema como um impacto diferente. O modelo de Briand permite ainda representar impactos causados por impactos, como uma recursão da classe *Impact*. O modelo de Briand não permite relacionar impactos com *stakeholders* e está focado em artefatos de design (UML). Já o trabalho de De Grande permite relacionar um impacto à cada versão de requisito encontrada no modelo. Entretanto, o modelo não permite relacionar impactos com mudanças, tampouco com *stakeholders*. Desta forma, na figura 14 é proposto o seguinte fragmento de modelo para representar impactos.

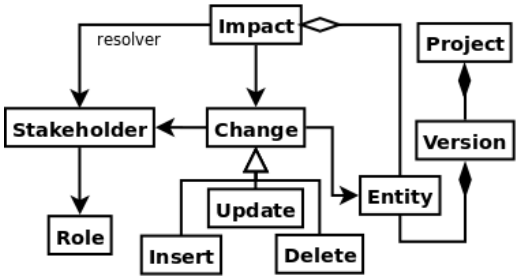

Figura 14. Gestão de impactos

Neste modelo parcial, um impacto (*Impact*) é uma classe que está relacionada diretamente com uma mudança (*Change*), uma vez que todo impacto será originado a partir de uma mudança. Entretanto, nem toda mudança irá gerar um impacto. Assim, a classe *Impact* irá indicar quando o impacto ocorreu, se está resolvido e também quem foi o responsável por sua resolução, além da solução adotada.

Enquanto a relação da 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.

4.5. Modelo consolidado

Na figura 15, é apresentado o modelo completo, consolidado a partir dos fragmentos de modelo apresentados até o momento: rastreabilidade, gestão da mudanças, gestão de impactos e versionamento. Cada aspecto é representado por uma região de cor diferente no modelo consolidado. A rastreabilidade é representada pela região em amarelo, a gestão da mudança pela região em verde, a gestão de impactos pela região laranja e o versionamento pela região azul. A classe Entidade (*Entity*) que representa qualquer artefato de software, desde um requisito textual, até um diagrama de classes, um método ou um relacionamento entre dois artefatos, é central no modelo pois é compartilhada por todos aspectos.

Para verificar se o modelo é completo o suficiente para atender os requisitos levantados no presente trabalho, na seção 5 o modelo proposto será comparado com os demais modelos avaliados.

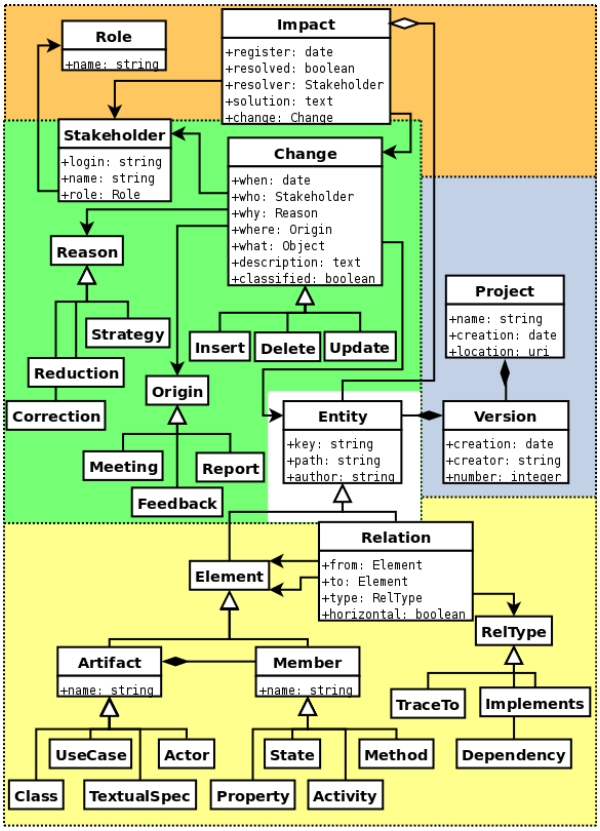

Figura 15. Modelo consolidado

Como visto, o modelo proposto especifica claramente quais os tipos de artefatos e relacionamentos que se fazem necessários para prover a rastreabilidade, ao mesmo tempo em que permite extensibilidade por meio da especialização e criação de novas subclasses daquelas já existentes.

4.6. Modelo entidade-relacionamento

A partir do modelo consolidado em UML proposto no item anterior, pode-se realizar seu mapeamento para uma estrutura relacional por meio da aplicação de certos padrões ORM (*Object-Relational Mapping*). Os padrões utilizados neste trabalho para o mapeamento objeto-relacional foram retirados do catálogo de *Design Patters* para uso em aplicações corporativas de Fowler [7].

Dentre os padrões utilizados neste trabalho para a criação do modelo relacional, pode-se citar o padrão *Class Table Inheritance*, utilizado para mapear os relacionamentos de herança, como os encontrados entre as classes *Entity*, *Element* e *Artifact*, o padrão *Foreign Key Mapping*, para mapear os relacionamentos de associação, como os encontrados entre as classes *Change* e *Stakeholder* e os relacionamentos de composição, como os encontrados entre as classes *Version* e *Entity*. Já para mapear relacionamentos de agregação, como o encontrado entre as classes *Impact* e *Entity*, foi utilizado o padrão *Association Table Mapping*.

Alguns relacionamentos de herança, como os encontrados a partir das classes *Artifact* e *Member* puderam ser mapeados por meio do padrão *Single Table Inheritance*, tendo uma tabela de apoio para representar o tipo de objeto armazenado na estrutura (*artifact_class* para *artifact* e *member_class* para *member*).

O modelo consolidado a partir dos fragmentos já apresentados é exibido na figura 16.

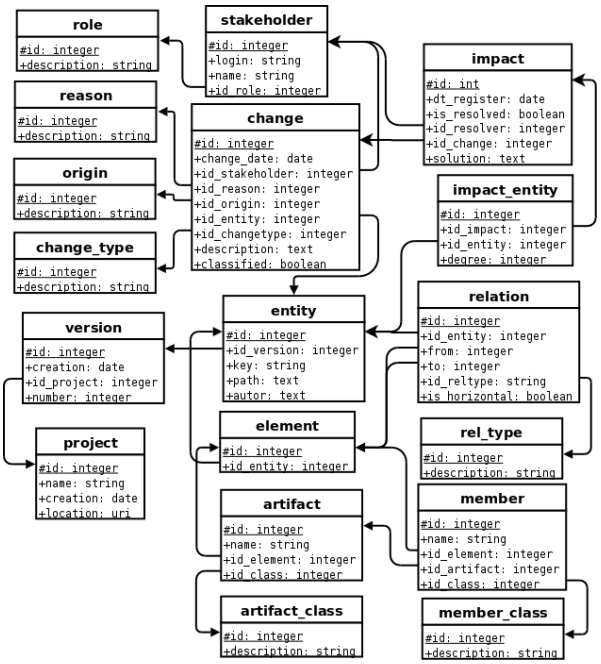

Figura 16. Modelo entidade-relacionamento

4.7. Consultas ao modelo em SQL

Para exemplificar a eficiência da versão entidade-relacionamento do modelo proposto, algumas consultas em linguagem SQL serão realizadas. Na figura 17, é apresentada uma listagem dos artefatos de pequena granularidade (*Coarse-Grained*) do projeto 1, cuja versão também é 1. São listados requisitos, atores e casos de uso.

```
SELECT artifact.id,
       artifact_class.description,
       artifact.name
FROM   artifact, element, entity, version,
       artifact_class
WHERE  artifact.id_element = element.id AND
       artifact.id_class   = artifact_class.id AND
       element.id_entity   = entity.id AND
       entity.id_version   = version.id AND
       version.id_project = 1 AND
       version.number = 1
ORDER BY 1 LIMIT 4;

 id | description |    name
----+-------------+------------
  1 | requirement | matricular
  2 | actor       | aluno
  3 | usecase     | matricular
  4 | usecase     | cadastrar
```

Figura 17. Exemplo de consulta SQL 1

Já na consulta da figura 18, é exibida uma listagem dos elementos do projeto de código 1 e versão também 1 de grande granularidade (*Fine-Grained*). São listados nesta consulta o método *get_nome_curso* da classe *Contrato*, o método *addTurma* da classe *Contrato*, o método *store* da classe *Matricula* e o método *get_nome_disciplina* da classe *Matricula*.

```
SELECT member.id as id,
       artifact.name as art_name,
       member_class.description ,
       member.name as member_name
FROM   member, element, entity, artifact,
       version, member_class
WHERE  member.id_element=element.id AND
       member.id_class=member_class.id AND
       member.id_artifact=artifact.id AND
       element.id_entity=entity.id AND
       entity.id_version=version.id AND
       version.id_project = 1 AND
       version.number = 1
ORDER BY 3 LIMIT 4;

 id | art_name  | description | member_name
----+-----------+-------------+--------------
 18 | Contrato  | method      | get_nome_curso
 19 | Contrato  | method      | addTurma
 20 | Matricula | method      | store
 21 | Matricula | method      | get_nome_disciplina
```

Figura 18. Exemplo de consulta SQL 2

Na figura 19, são exibidos algumas relações entre entidades do sistema. Neste caso, é demonstrado que a classe *Contrato* possui uma associação com a classe *Curso* e a classe *Pessoa* possui uma associação com a classe *Cidade*. Devido ao espaço, relacionamentos *Fine-Grained* foram suprimidos.

```
SELECT t1.key as from_key,
       rt.description as relation,
       t2.key as to_key
FROM   relation r, element e1, element e2,
       entity t1, entity t2, entity t3, rel_type rt,
version
WHERE r.from_element=e1.id and
      r.to_element=e2.id and
      r.id_entity=t3.id and
      t3.id_version=version.id and
      t1.id_version=t2.id_version and
      t2.id_version=t3.id_version and
      e1.id_entity=t1.id and e2.id_entity=t2.id and
      r.id_reltype=rt.id and version.id_project=1 and
      version.number = 1
LIMIT 4;

     from_key      |  relation   |       to_key
-------------------+-------------+-----------------
class('Contrato')  | association |class('Curso')
class('Pessoa')    | association |class('Cidade')
class('Turma')     | association |class('Disciplina')
class('Matricula') | association |class('Turma')
```

Figura 19. Exemplo de consulta SQL 3

Na figura 20, são exibidas estatísticas de mudanças do projeto 1 conforme sua origem. Como exemplo, ocorreram 2 mudanças devido à relatórios de erros e 3 mudanças devido à chamados da engenharia.

```
SELECT change.id_origin,
       origin.description,
       count(*)
FROM change, entity, version, origin
WHERE change.id_origin=origin.id and
      change.id_entity=entity.id and
      entity.id_version=version.id and
      change.classified='1' and
      version.id_project=1
GROUP BY 1,2

 id_origin |       description        | count
-----------+--------------------------+-------
         1 | Relatorios de erros      |     2
         2 | Chamado da Engenharia    |     3
         3 | Gerenciamendo do Projeto |     5
         5 | Análise do Desenvolvedor |     1
```

Figura 20. Exemplo de consulta SQL 4

4.8. Aplicações e trabalhos futuros

O modelo proposto no presente trabalho está sendo utilizado no desenvolvimento de uma aplicação colaborativa que auxilia na gestão da mudança de requisitos de software por meio da aplicação de um processo auxiliado por agentes de software. Esta aplicação já se encontra em estágios finais de validação. A versão Entidade-Relacionamento do modelo, bem como as consultas demonstradas aqui fazem parte desta ferramenta.

O desenvolvimento desta aplicação, bem como seu uso, permitem concluir que o modelo adotado provou ser coeso, consistente, robusto e completo. Além disto, seu mapeamento objeto-relacional permitiu um rápido desenvolvimento por meio da utilização de modernos *frameworks*.

Com base no modelo proposto, espera-se o surgimento de novos métodos, processos e ferramentas que o utilizem como base para o armazenamento das informações do desenvolvimento de software e venham melhorar as técnicas existentes para a gestão da mudança de requisitos e dos impactos gerados.

5. Tabela comparativa

Como forma de comparar os modelos avaliados frente ao modelo proposto, buscou-se na literatura alguns trabalhos cujas propostas objetivam a definição de critérios para avaliar ferramentas de gestão de requisitos. Dentre estes trabalhos destacam-se Hoffmann [9], Lang [12] e Cleland-Huang [4].

Para Hoffmann [9], a utilização de sistemas para gerenciamento de requisitos auxiliam em manter a especificação consistente, atualizada e acessível. Para o autor, existem muitos requisitos que devem ser considerados no desenvolvimento de um sistema para o gerenciamento de requisitos. Desta forma, Hoffmann [9] apresenta um catálogo de requisitos para o desenvolvimento de ferramentas para o gerenciamento de requisitos. Dentre os requisitos identificados por Hoffmann, destacam-se:

- Suportar diferentes modelos de requisitos;
- Suportar requisições de mudanças;
- Suportar versões dos artefatos (*baseline)*;
- Suportar rastreabilidade entre requisitos;
- Suportar interfaces com outras ferramentas;
- Permitir trabalho colaborativo;
- Prover funções de análise de progresso do projeto;

Além de Hoffmann, também Lang [12] elaborou um pequeno catálogo de requisitos para ferramentas de gerência de requisitos com o foco em colaboração. Dentre os requisitos, destacam-se:

- Manter descrições únicas de todos os requisitos;
- Classificar requisitos em grupos lógicos;
- Especificar requisitos de forma textual e gráfica;
- Definir relacionamentos rastreáveis entre os requisitos;
- Verificar atribuições de requisitos à especificações de *design* feitas pelo usuário;
- Manter arquivo das versões de mudanças;
- Permitir mecanismo para autenticação e aprovação de mudanças;
- Suportar trabalho cooperativo multidisciplinar;
- Suportar trabalho distribuído;
- Suportar sistemas padrão de modelagem;
- Manter repositório compartilhado de componentes e requisitos;
- Suportar protocolos de comunicação com outras ferramentas;

Cleland-Huang [4] sugere um conjunto de "boas-práticas" que deveriam estar presentes em ferramentas de gestão de requisitos, especialmente aquelas que suportam rastreabilidade:

- Para cada *link*, o usuário deverá localizar a origem, o destino e o tipo do requisito;
- Os colaboradores deverão decidir qual o nível de granularidade da rastreabilidade;
- Prover rastreabilidade para artefatos construídos e armazenados em seus ambientes nativos;

5.1. Critérios para comparação

Com base nos critérios apontados por Hoffmann, Lang e Cleland-Huang, definiu-se um grupo de 12 critérios a serem utilizados a fim de se comparar os modelos avaliados frente ao modelo proposto. A seguir constam os critérios, bem como uma explicação detalhada de cada um deles.

A) Suporta relação com documentos externos: Permitir relacionar artefatos ou requisitos com documentos externos, tais como textos ou planilhas;

B) Suporta requisitos textuais: Suporta a representação de especificação textual de requisitos;

C) Suporta documentos UML: Suporta artefatos de design, como digrama de classes ou use cases;

D) Suporta código-fonte: Suporta a representação de arquivos contendo código-fonte, classes e métodos;

E) Suporta rastreabilidade vertical: Permite rastreabilidade entre requisitos de um mesmo modelo;

F) Suporta rastreabilidade horizontal: Permite rastreabilidade entre requisitos de modelos diferentes;

G) Suporta diferentes projetos: Permite representar requisitos de projetos distintos;

H) Suporta configuração de granularidade: Permite configurar o nível de granularidade da rastreabilidade;

I) Suporta representação de mudanças: Permite representar as mudanças ocorridas;

J) Suporta representação de impactos: Permite representar os impactos causados pelas mudanças;

K) Suporta versionamento (*baseline*): Permite representar as versões dos requisitos;

L) Permite relacionar *stakeholders* e artefatos: Permite relacionar artefatos com *stakeholders*;

A partir dos critérios selecionados para comparação, são definidos os modelos a serem comparados:

M1) Modelo de Ramesh;
M2) Modelo de Lang;
M3) Modelo de Letelier;
M4) Modelo de Briand;
M5) Modelo de De Grande;
M6) Modelo de Kassab;
M7) Modelo de Ghazi;
MP) Modelo Proposto;

5.2. Comparação entre modelos

Desta forma, se estabelece, conforme pode ser visto pela tabela 1, a comparação dos modelos conforme os critérios selecionados. Cada ponto de interseção representa um critério atendido. Nas linhas tem-se os modelos e nas colunas os critérios (A-L).

Tabela 1. Comparação entre Modelos

	A	B	C	D	E	F	G	H	I	J	K	L
M1	x	x			x				x			x
M2		x	x		x	x			x			x
M3		x	x		x	x		x				x
M4			x		x			x	x	x	x	
M5	x	x			x		x		x	x	x	x
M6	x	x	x		x	x	x	x				x
M7	x	x	x	x	x	x	x					x
MP	x	x	x	x	x	x	x	x	x	x	x	x

6. Considerações finais

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 à 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.

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

O desenvolvimento do modelo de rastreabilidade proposto no presente trabalho possibilita a criação de uma série de aplicações para gestão da mudança de requisitos, que venham a implementar as funcionalidades aqui descritas.

7. Referências

[1] Arnold, R. S. and Bohner, S. A. 1993. “Impact Analysis - Towards a Framework for Comparison”. In Proceedings of the Conference on Software Maintenance D. N. Card, Ed. IEEE Computer Society, Washington, DC, 292-301.

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

[3] Briand, L. C., Labiche, Y., and O'Sullivan, L. 2003. Impact Analysis and Change Management of UML Models. In Proceedings of the international Conference on Software Maintenance (September 22 - 26, 2003). ICSM. IEEE Computer Society, Washington, DC, 256.

[4] 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.

[5] De Grande, José Inácio ; MARTINS, L. E. G. . SIGERAR: uma Ferramenta para Gerenciamento de Requisitos. In: IX Workshop on Requirements Engineering, 2006, Rio de Janeiro. p. 75-83.

[6] De Lucia, A., Fasano, F., Oliveto, R. "Traceability Management for Impact Analysis", Frontiers of Software Maintenance, Beijing, China, IEEE, 2008, pp. 21-30.

[7] Fowler, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

[8] Ghazi, H. E. 2008. MV - TMM: A Multi View Traceability Management Method. In Proceedings of the 2008 32nd Annual IEEE international Computer Software and Applications Conference (July 28 - August 01, 2008). COMPSAC. IEEE Computer Society, Washington, DC, 247-254.

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

[10] Kassab, M., Ormandjieva, O., and Daneva, M. 2008. A Traceability Metamodel for Change Management of Non-functional Requirements. In Proceedings of the 2008 Sixth international Conference on Software Engineering Research, Management and Applications (August 20 - 22, 2008). SERA. IEEE Computer Society, Washington, DC, 245-254.

[11] Kotonya & Sommerville. Requirements Engineering: Processes and Techniques. John Willey & Sons Ltd, 1998.

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

[13] Leite, Júlio Cesar Sampaio do Prado. Extreme Requirements. In: Jornadas de Ingeniería de Requisitos Aplicada, 2001, Sevilha. Jornadas de Ingeniería de Requisitos Aplicada, 2001. p. 1-13.

[14] Letelier, P., "A Framework for Requirements Traceability in UML-based Projects", Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TFFSE 2002), September 2002

[15] 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.

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

[17] Pinheiro, Francisco. Requirements Traceability. Capítuo 5: Julio Cesar S. P. Leite, Jorge H. Doorn. (Org.). Perspectives on Software Requirements. Boston: Kluwer Academic Publishers, 2004, v. 1, p. 91-110.

[18] Queille, J., Voidrot, J., Wilde, N., and Munro, M. 1994. The Impact Analysis Task in Software Maintenance: A Model and a Case Study. In Proceedings of the international Conference on Software Maintenance H. A. Müller and M. Georges, Eds. IEEE Computer Society, Washington, DC, 234-242.

[19] Ramesh, B., Powers, T., Stubbs, C., and Edwards, M. 1995. Implementing requirements traceability: a case study. In Proceedings of the Second IEEE international Symposium on Requirements Engineering (March 27 - 29, 1995). RE. IEEE Computer Society, Washington, DC, 89.

[20] Sayão, Miriam. Leite, Julio Cesar. Rastreabilidade de Requisitos. Revista de Informática Teórica e Aplicada, v. XIII, p. 57-86, 2006.