UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

JOÃO VITOR ALMEIDA SOARES

Uma proposta para a inserção da equipe de desenvolvimento na gestão de requisitos por meio de ferramenta de controle de tarefas

RECIFE
2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

JOÃO VITOR ALMEIDA SOARES

Uma proposta para a inserção da equipe de desenvolvimento na gestão de requisitos por meio de ferramenta de controle de tarefas

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Engenharia da Computação, orientada pelo professor Márcio Lopes Cornélio e co-orientada pelo professor Gabriel Ramos Falconieri Freitas.

RECIFE
2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

JOÃO VITOR ALMEIDA SOARES

Uma proposta para a inserção da equipe de desenvolvimento na gestão de requisitos por meio de ferramenta de controle de tarefas

Monografia submetida ao corpo docente da Universidade Federal de Pernambuco, defendida e aprovada em 04 de julho de 2018.

Banca Examinadora:

Orientador

Márcio Lopes Cornélio

Doutor

Examinadora

Carla Taciana Lima Lourenço Silva Schuenemann

Doutora

Dedico este trabalho à minha avó e ao legado que ela deixou.

AGRADECIMENTOS

Agradeço aos meus pais e aos meus irmãos pelo carinho, dedicação e apoio em todos os momentos que vivi nessa jornada. Sou muito grato e feliz por tê-los como família.

Agradeço aos meus orientadores, Márcio e Gabriel, que trouxeram tranquilidade para as preocupações e incertezas que surgiram ao longo desse desafio. Obrigado pela confiança e pelo tempo dedicado.

À minha tia Nilde, pelas correções relâmpago e a disponibilidade, quase que integral, para me ajudar.

A seu Edson, pelas revisões valiosas de gramática e pela disposição em me ajudar, mesmo não entendendo sobre o tema.

Aos meus amigos e amigas, pela compreensão e os momentos divertidos que me proporcionaram. Em especial à Renata, Jéssica e Aline, que, mesmo distantes, são as figuras mais presentes na minha vida.

Aos meus colegas e amigos de trabalho, Rodrigo, Ana, Ihago e os demais, cujo companheirismo e experiência me inspiraram a sempre dar o meu melhor.

“A minha alucinação é suportar o dia a dia, e meu delírio é a experiência com coisas reais.”
- Belchior

RESUMO

O processo de gerência dos requisitos é fundamental no desenvolvimento de *Software*, no qual a volatilidade dos requisitos é uma característica intrínseca aos projetos. Manter controle das mudanças e, sobretudo, garantir que a implementação esteja alinhada com a documentação são tarefas que demandam atenção para dois atores: o cliente e o desenvolvedor. Quando a interação com a equipe de desenvolvimento está fracamente estabelecida, o entendimento dos requisitos por parte dos desenvolvedores é afetado e defasagens na implementação podem surgir. Além disso, mudanças nos requisitos podem advir da equipe de desenvolvimento e, para esses casos, a integração dela com os processos de gerência dos requisitos passa a ser um aspecto determinante na qualidade do projeto. Baseado na experiência profissional em uma empresa de desenvolvimento de sistemas industriais, este trabalho propõe uma solução tecnológica para promover a integração da equipe de desenvolvimento nas tarefas de gerência dos requisitos, fechando a lacuna existente entre a ferramenta de controle de tarefas Agilo For Trac e a plataforma de modelagem de sistemas Enterprise Architect, ambas utilizadas nos processos de desenvolvimento da empresa.

Palavras-chave: Gerência de requisitos; Agilo For Trac; integração; Enterprise Architect.

ABSTRACT

The requirements management process is fundamental in Software development, considering that the volatility of the requirements is an intrinsic characteristic of projects. Keeping control of changes and, especially, ensuring that the implementation is synchronized with the specification are tasks that require attention for two actors: the client and the developer. When the development team interaction with the requirements analysts is poorly established, developers' understanding of the requirements model is affected and discrepancies may arise. In addition, changes in requirements may come from the development team, and in such cases, its integration with requirements management process becomes a determining factor in project quality. Based on professional experience in an industrial systems development company, this paper proposes a technological solution to encourage the integration of the development team in the requirements management tasks, closing the gap between the task control tool Agilo For Trac and the systems modeling platform Enterprise Architect, both used in the company's development processes.

Keywords: Requirements management, Agilo for Trac, integration, Enterprise Architect.

Sumário

Lista de Figuras

Lista De Tabelas

Tabela De Siglas

Sigla	Significado	Página
XMI	XML Metadata Interchange	20
XML	eXtensible Markup Language	20
HTTP	Hypertext Transfer Protocol	32
EA	Enterprise Architect	17
UC	Use Case	40
UML	Unified Modeling Language	8

1. Introdução

Apesar de todos os avanços recentes na Engenharia de Software, mudanças de requisitos continuam sendo recorrentes durante todo o ciclo de vida de um projeto de desenvolvimento. Sommerville afirma que "durante o processo de validação dos requisitos, você raramente encontrará todos os problemas de requisitos", e conclui que mudanças na documentação são inevitáveis no desenvolvimento de Software [1].

Os impactos destas mudanças na especificação de um projeto abrangem custo, cronograma e até a qualidade do mesmo [2]. Minimizar a frequência e acelerar o processo de gerência dessas mudanças são ações a serem tomadas para mitigar os danos listados. Assim, é preciso abraçar mudanças como parte da rotina do processo da engenharia de requisitos.

Dentre os diversos fatores que geram mudanças nos requisitos, o entendimento detalhado do sistema por parte dos desenvolvedores se estabelece como uma das principais causas de *change requests* na especificação [3]. Essas mudanças são desencadeadas pela compreensão mais minuciosa do sistema quando o desenvolvimento está, de fato, acontecendo.

1.1 Motivação

Nesse contexto, a harmonização entre os desenvolvedores e os analistas de requisitos durante a fase de implementação deve ser não somente garantida como também preservada por meio da documentação existente. Isto é, o entendimento da documentação por parte de quem está desenvolvendo deve ser constantemente validado por quem a elaborou. Para que esse processo aconteça, duas premissas devem ser consideradas:

- **Premissa 1**. Os desenvolvedores devem ter acesso direto e contínuo à documentação atualizada.
- **Premissa 2**. Os analistas de requisitos devem possuir meios consistentes de validar o entendimento da equipe de desenvolvimento.

A premissa 1 permite que os desenvolvedores possam, a qualquer momento, checar e analisar todos os requisitos levantados para a funcionalidade que está sendo implementada. Isto inclui acompanhar as mudanças feitas nos mesmos e, também, questionar o que está especificado.

A premissa 2 garante que o entendimento e a análise da equipe de desenvolvimento sobre os requisitos sejam documentados. Essa premissa é importante no processo de identificação e gerência de mudanças na especificação existente.

Baseado em experiência profissional, quando a sincronização entre as duas equipes é falha ou debilitada, duas situações podem surgir como consequência: a primeira decorre de documentações dúbias ou omissas, em que há espaço para interpretações incorretas a respeito do que se está sendo especificado. Nesses casos, o entendimento dos requisitos por parte dos desenvolvedores é deficiente, levando-os a implementar as funcionalidades do sistema de forma incorreta.

A segunda situação advém de mudanças do sistema não refletidas na especificação de forma completa, i.e., quando a análise de impacto dessas modificações não abrange todos os artefatos afetados e, portanto, não são executadas tarefas para atualização completa da documentação. Esse tipo de inconsistência é comum em ambientes carentes de políticas de rastreabilidade de requisitos.

Em ambas as situações, a consistência da especificação é perdida, gerando uma defasagem entre a implementação e a documentação do sistema.

1.2. Objetivo

O objetivo central desse trabalho é propor e implementar uma solução tecnológica para a lacuna existente entre a equipe de desenvolvimento e os analistas de requisitos em uma empresa de desenvolvimento real. A solução será dada por meio da customização da ferramenta de controle de tarefas Agilo For Trac[1], que será integrada com a funcionalidade de modelagem de requisitos da plataforma Enterprise Architect[2].

A solução descrita se baseia na experiência de estágio curricular em uma empresa multinacional de desenvolvimento de sistemas industriais, ocorrido durante o período de outubro a dezembro de 2017. A proposta procura minimizar problemas relacionados ao

[1] http://www.agilofortrac.com
[2] http://sparxsystems.com/products/ea/

entendimento e visualização de requisitos durante a fase de implementação dos projetos da equipe.

1.3. Estrutura do documento

Este documento está estruturado da seguinte maneira: No Capítulo 2, os conceitos básicos para o entendimento do trabalho são apresentados; no Capítulo 3 é descrito o processo de modelagem e implementação da customização; já no Capítulo 4, uma prova de conceito para a solução é elaborada; por fim, no Capítulo 5, apresentamos as contribuições do trabalho e a conclusão.

2. Fundamentação Teórica

Este capítulo descreve os conceitos teóricos usados como base para a proposta deste trabalho. Na Seção 2.1, é explicado o que são requisitos de software e como podem ser classificados. A Seção 2.2 descreve a engenharia de requisitos, listando suas principais etapas e destacando sua importância na engenharia de software. A Seção 2.3 apresenta a ferramenta Enterprise Architect, descrevendo alguma de suas funcionalidades. Na Seção 2.4 é explicado o padrão XMI, usado na implementação da ferramenta proposta. A Seção 2.5 explica a técnica de desenvolvimento iterativa incremental, relevante para a contextualização. E, por fim, a Seção 2.6 faz uma breve introdução a ferramenta Agilo For Trac.

2.1. Requisitos de Software

O termo requisito aparece de maneira extensiva em artigos e em ambientes de desenvolvimento de software e sistemas. Na literatura, existem inúmeras definições do termo, contudo, traduzindo do glossário padrão de Engenharia de Software e Sistemas [4]:

> *3.3430*
> *Requisito.*
> (1) Declaração que traduz ou expressa uma necessidade e as condições e restrições associadas a ela.
> (2) Condição ou capacidade que precisa ser atingida ou possuída por um sistema, componente de sistema, produto ou serviço para satisfazer um acordo, padrão, especificação ou outros documentos impostos formalmente.
> [ISO/IEC/IEEE 24765:2017(E)]

Em outras palavras, requisito é qualquer tipo de informação ou artefato que seja relevante para a identificação e análise de um Software, definindo seu comportamento e restrições. É esperado que todo projeto de Software tenha seu início na fase de identificação dos requisitos, onde é feito o processo de imersão da equipe que irá desenvolver o sistema

com os diferentes *stakeholders*. Nesse contexto, é importante a elaboração de documentos em vários níveis de detalhamento, permitindo o uso e compreensão de diferentes leitores. Um gestor de projeto, por exemplo, demanda uma visão diferente do sistema do que os desenvolvedores.

Sommerville divide requisitos em dois grupos quanto ao seu nível de descrição [1]:

1. **Requisitos de usuário**, onde se encaixam as frases em linguagem natural, diagramas, serviços oferecidos e restrições operacionais do sistema. Exemplo: normas e procedimentos de uma fábrica que devem ser obedecidos pelo sistema.
2. **Requisitos de sistema**, onde os detalhes técnicos e comportamentais são definidos. A **especificação funcional** é o documento elaborado para contemplar os requisitos de sistema e descreve, com exatidão, o que deve ser codificado.

Ainda segundo Sommerville, requisitos também podem ser classificados como:

1. **Funcionais** – que descrevem o que o sistema deve fazer. Eles detalham o comportamento do sistema para entradas específicas e o que deve ser esperado como saída da sua execução.
2. **Não funcionais** – que descrevem restrições dos serviços do sistema a ser desenvolvido. Restrições de tecnologia, qualidade e processo são exemplos de requisitos nessa categoria.

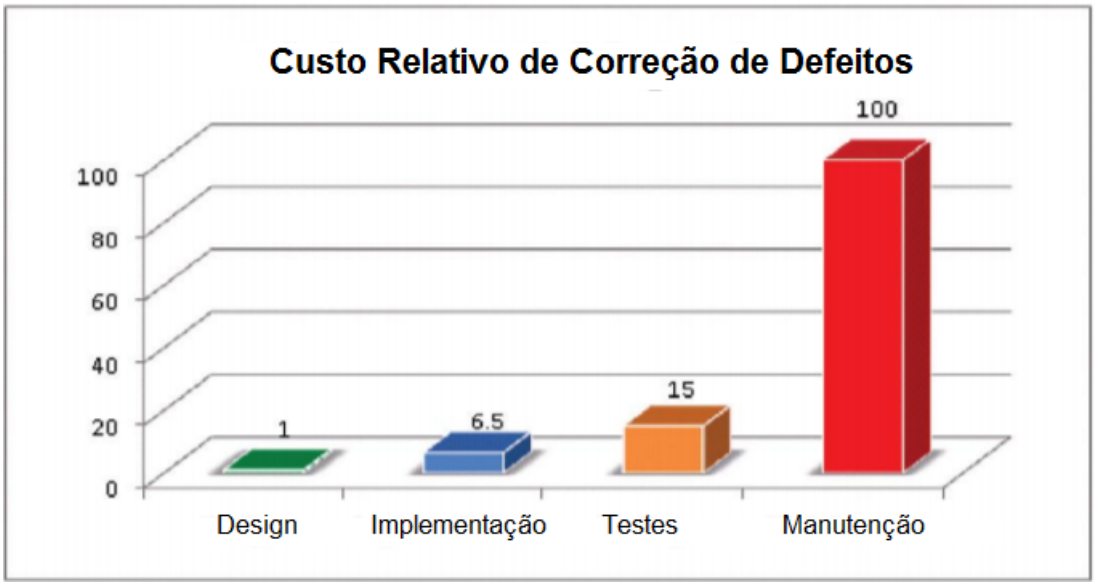
2.2 Engenharia de Requisitos

Duas ações são necessárias para o sucesso de um projeto de desenvolvimento de Software: identificar seus requisitos e documentá-los da maneira apropriada [5]. Essa condição permite que os responsáveis do projeto possam entender e formalizar o comportamento do sistema a ser desenvolvido e, assim, tomar decisões que prezem pelo bom andamento das atividades. Apesar dos diversos motivos que levam à falha de um sistema, projetos que fizeram pouco ou nenhum uso da engenharia de requisitos são mais prováveis de falharem em algum ponto de sua execução [2]. De fato, segundo o Chaos Report de 2006

[6], 13.1% dos executivos de TI entrevistados acreditam que requisitos incompletos é a principal causa de problemas e cancelamentos de projetos de software. Na pesquisa, essa foi a causa número 1 (um) na lista de respostas.

Um dos principais fatores que justificam a importância da engenharia de requisitos é o custo de correções para esta etapa. A Figura 2.1 traz um comparativo de custo para a correção de defeitos em diferentes fases da engenharia de software. Nota-se que quanto mais se prolonga a detecção e reparação dos defeitos, mais agressivo é o custo para realizar essas correções. Defeitos corrigidos na fase de implementação, por exemplo, chegam a custar seis vezes mais que aqueles detectados e mitigados durante a fase de levantamento de requisitos. Portanto, processos que auxiliem a detecção de problemas nas etapas iniciais do desenvolvimento podem evitar excedentes de custo e corroboram com o sucesso do projeto.

Figura 2.1 - Custo relativo para correção de defeitos em diferentes fases da engenharia de software. Adaptado de [7].



O conjunto de técnicas e métodos usados para especificar e gerenciar requisitos é chamado de engenharia de requisitos. Pressman afirma que a engenharia de requisitos é uma das principais etapas da engenharia de software, tendo início nas atividades de concepção e se estendendo às atividades de modelagem e manutenção dos requisitos [8]. Em poucas palavras,

o objetivo principal da engenharia de requisitos é traduzir as necessidades do cliente em artefatos que possam ser usados para a correta e completa elaboração do produto. Pohl [5] divide a engenharia de requisitos em quatro atividades. São elas:

- **Elicitação**, onde são coletados os requisitos dos *stakeholders* e outras fontes. As informações coletadas nesta fase são detalhadas e organizadas durante a documentação;
- **Documentação**, onde é feita a descrição e formalização dos requisitos. Pode ser feito o uso de linguagem natural ou outros métodos conceituais de especificação.
- **Validação e negociação**, onde é feita a definição dos critérios de qualidade e condições a serem atingidas pelo sistema por meio da validação e negociação com o cliente.
- **Gerenciamento**, onde é feito o acompanhamento de mudanças e a garantia de consistência dos requisitos.

É relevante destacar que essas fases, apesar de estarem dispostas de maneira sequencial, acontecem de maneira cíclica, com cada fase sendo revisitada à medida que se faz necessário, até que se obtenha uma especificação funcional aprovada e que satisfaça um limiar de qualidade preestabelecido.

2.2.1 Elicitação dos requisitos

Durante a elicitação dos requisitos, o papel da comunicação é fundamental. Nessa fase, cliente e usuário são questionados a respeito das suas necessidades, e o processo de visualização do sistema a ser modelado tem início. É preciso entender como o sistema será usado, como ele se encaixa no negócio dos usuários, o que ele deve contemplar etc. Contudo, muitas vezes, não está claro ou bem definido para o cliente como será o funcionamento do sistema ou, ainda mais comum, a falta de conhecimento dos *stakeholders* sobre sistemas computacionais os levam a propor requisitos inconsistentes ou infactíveis. O fato é que cada *stakeholder* possui uma interpretação diferente do sistema e, portanto, inconsistências são inevitáveis na grande maioria dos casos [1].

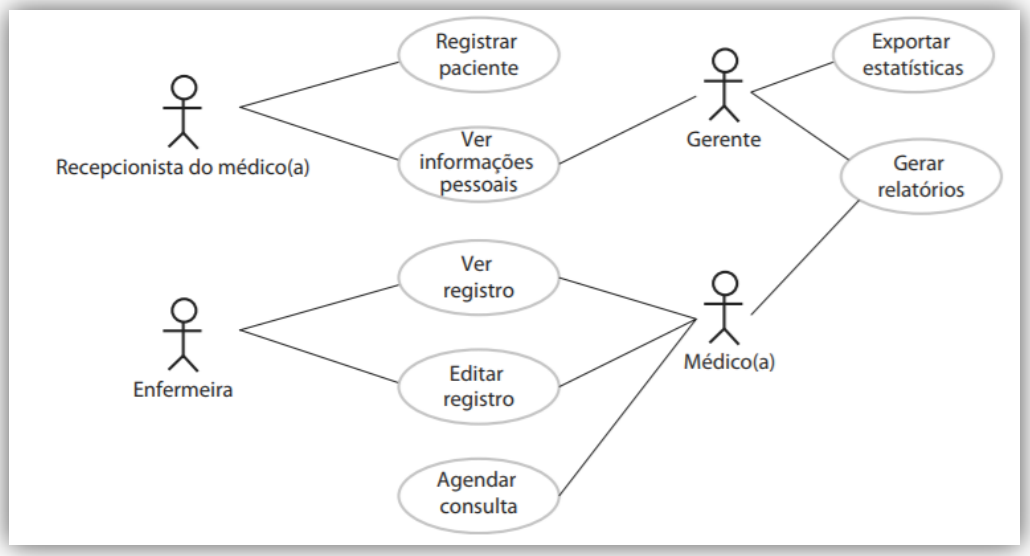
Christel e Kang [9] definem três tipos de problemas que podem surgir na elicitação dos requisitos:

- **Problemas de escopo**: os limites de escopo do sistema estão fracamente definidos, i.e., a área de abrangência não é clara. Isso pode levar à omissão de requisitos relevantes para o sistema ou até o levantamento de requisitos desnecessários.
- **Problemas de entendimento**: o entendimento do cliente é escasso, tanto a respeito do problema a ser sanado quanto aos detalhes do funcionamento do sistema. Os analistas de requisitos também podem ter pouco domínio da área de conhecimento do problema, gerando demoras no processo e inferências erradas.
- **Problemas de volatilidade**: Os requisitos inerentemente mudam com o tempo.

A quantidade de informações levantadas ao fim desta etapa geralmente demanda uma análise criteriosa para separar os requisitos relevantes dos que não serão essenciais para o desenvolvimento do sistema. Além disso, as demais etapas da engenharia de requisitos farão uso das informações coletadas na elicitação e, portanto, é importante certo nível de organização [8]. Em auxílio ao processo de descoberta dos requisitos, algumas técnicas são comumente usadas pelos engenheiros de requisitos, dentre as quais, destaca-se: entrevistas, etnografia e elaboração de cenários e casos de uso. No contexto do trabalho, as técnicas de cenário e casos de uso são especialmente relevantes, pois são adotadas pela empresa do estudo.

Na abordagem dessas técnicas, um caso de uso modela as interações dentro do sistema por meio do detalhamento do comportamento e dos atores envolvidos. Parte fundamental da linguagem de modelagem unificada (UML, *unified modeling language*) [10], os casos de uso podem ser descritos em mais detalhes com o auxílio de diagramas de sequência e fluxos de uso. Para cada iteração de desenvolvimento, os casos de uso são dispostos em um diagrama de casos de uso (Figura 2.2).

Figura 2.2 - Exemplo de diagrama de casos de uso. Retirado de [1]



2.2.2 Documentação

Após a elicitação, é preciso definir e formalizar o que foi entendido pelos analistas de requisitos. O documento de requisitos (SRS, do inglês *Software Requirements Specification*) contém exatamente o que será implementado pelos desenvolvedores. Nessa etapa, os requisitos estão em um nível maior de detalhes, tanto a respeito da sua definição quanto às interações que eles possuem dentro do sistema ou com interfaces subjacentes. São incluídos no documento tanto os requisitos de usuário quanto os requisitos de sistema, explicitando as condições em que os mesmo são satisfeitos.

Na prática, a forma que este documento tomará depende do projeto. Uma SRS pode ser um documento em texto, um conjunto de diagramas, descrições de casos de uso ou uma combinação destes. Não existe um padrão do que deve ser produzido nessa fase e essa flexibilidade é importante para a diversidade de projetos que existe no mercado [8]. Com as metodologias ágeis, contudo, a elaboração de especificações de requisitos passou a ser iterativa ou até descartada em muitos ambientes de desenvolvimento. Na *Extreme*

Programming, ou XP, por exemplo, as atividades de engenharia de requisitos acontecem em paralelo com as demais etapas da engenharia de software [11]. Assim, os requisitos são coletados e definidos de maneira efêmera, usualmente em *user stories*. Para casos onde o desenvolvimento é feito por um contratante externo, a elaboração da especificação funcional é fundamental, uma vez que ela poderá ser usada como contrato entre ambas as partes.

Um dos aspectos mais importantes na elaboração de uma especificação de software é seu nível de rastreabilidade de requisitos. A rastreabilidade pode acontecer de duas formas: horizontal e vertical. Na primeira, o *tracing* ocorre entre os artefatos de requisitos, e.g., rastrear a definição de uma regra de negócio citada na descrição de um caso de uso. Na rastreabilidade vertical, o *tracing* ocorre entre artefatos de diferentes fases do desenvolvimento. A rastreabilidade de requisitos é tratada em mais detalhes na seção 2.2.4

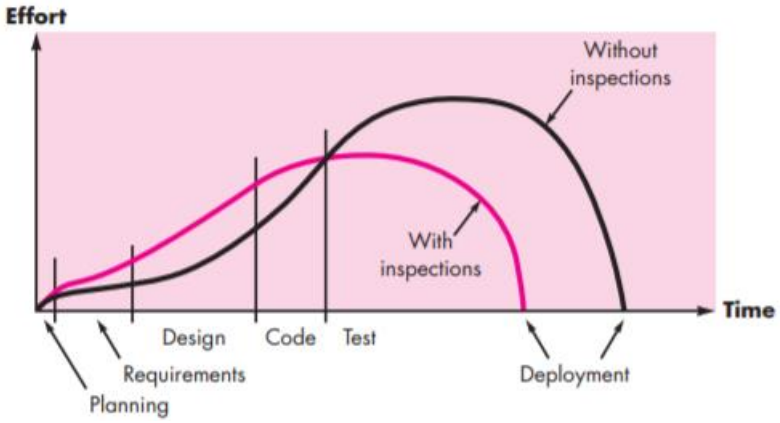
Uma vez produzida a documentação com as características descritas, o acompanhamento de execução e mudança dos requisitos é melhorado, gerando menos retrabalho e maior controle dos gastos e esforço envolvido em cada requisito. Ferramentas são usadas para simplificar o trabalho dos analistas, fornecendo métodos que gerem os aspectos organizacionais e técnicos dos requisitos. O Enterprise Architect (detalhado no capítulo 2.3) é um exemplo de ferramenta usada para modelagem e especificação dos requisitos e é a ferramenta de referência para o desenvolvimento da solução deste trabalho.

2.2.3 Validação e negociação

Na fase de validação e negociação, é verificado se os requisitos atendem às expectativas e necessidades dos *stakeholders*. O foco dos revisores é encontrar erros ou inconsistências que possam comprometer a especificação funcional do sistema. Esse processo é particularmente importante, pois é onde acontece a detecção prévia de problemas antes dos próximos estágios do projeto: implementação, testes e assim por diante. Fagan relata em sua pesquisa que inspeções na especificação foram responsáveis pela detecção de até 93% de defeitos dos projetos do estudo, levando a uma redução de 25% em recursos de

desenvolvimento em alguns casos [12]. A Figura 2.3 traz um comparativo do esforço (e, por fim, custo) demandado com e sem o uso do processo de inspeção adotado.

Figura 2.3 - Esforço demandado em diferentes fases do desenvolvimento de software com (rosa) e sem (preto) a adoção de inspeções de especificação. Retirado de [8].



Várias verificações são feitas para que uma especificação possa ser aprovada na fase de validação e negociação. Algumas delas são listadas abaixo [1]:

- Validade: um usuário pode acreditar que certas funções são responsabilidade do sistema, mas, após reflexões, tais funções podem ser reclassificadas como desnecessárias ou redundantes. Diferentes *stakeholders* possuem diferentes demandas e necessidades, portanto, é preciso que estas estejam alinhadas para que se tenha uma clara definição do escopo do sistema. É nesse ponto que a negociação de requisitos tem papel mais importante;
- Consistência: requisitos devem coexistir em harmonia dentro de uma especificação. Descrições ou informações conflituosas e contraditórias são alvo dessa verificação;
- Completude: os requisitos especificados devem satisfazer todas as funcionalidades e restrições demandas pelo usuário para aquele escopo de análise;
- Realismo: os requisitos precisam ser factíveis. Nessa checagem, são considerados os limitantes tecnológicos e logísticos (cronograma, custo);

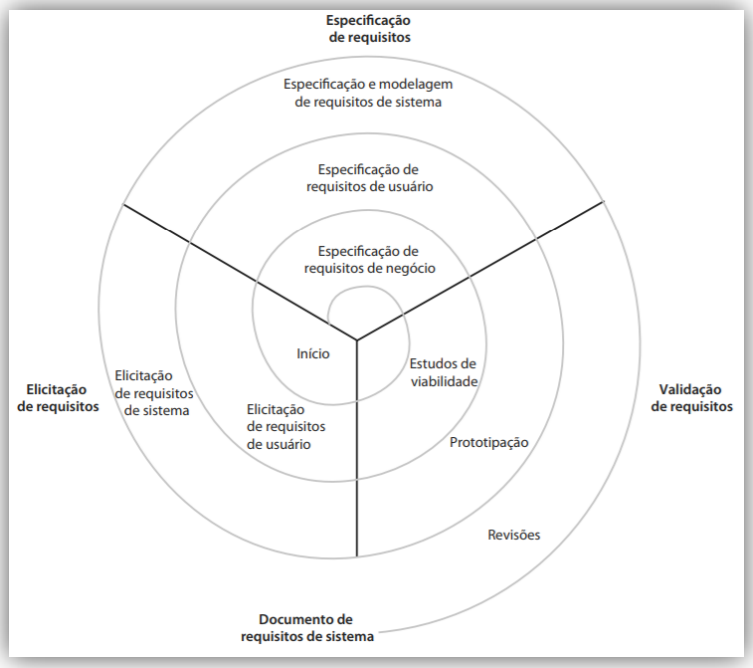
- Verificabilidade: deve ser possível demonstrar que um requisito foi verificado, i.e., deve ser possível escrever testes que demonstrem a satisfatibilidade dos requisitos.

A fase de validação não pode ser subestimada. Imaginar como o sistema irá funcionar, as situações que podem ocorrer e como ele irá interagir com outras interfaces não é uma tarefa simples. Até analistas mais experientes possuem dificuldades nessa atividade e raramente conseguem identificar todos os problemas de uma especificação [1]. Frente a essa questão, técnicas de validação são propostas pela literatura e adotadas em ambientes reais de desenvolvimento. Para obter resultados satisfatórios, geralmente mais de uma técnica é usada para validação. Algumas delas são descritas a seguir:

- Geração de casos de teste: os requisitos devem ser testáveis, i.e., deve ser possível elaborar cenários e casos de teste que possam confirmar que um requisito foi satisfeito. Segundo Sommerville, um requisito que seja difícil ou impossível de ser testado deve ser reconsiderado na especificação, pois pode levar a dificuldades na fase de implementação [1].
- Prototipação: consiste na elaboração de um modelo executável do sistema. O nível de detalhamento de um protótipo depende de em qual fase ele está sendo elaborado. Um protótipo usado na fase de elicitação, por exemplo, deve abrir mão de complexidades para que seja passível de mudanças. Protótipos de validação de especificação devem contemplar os principais (ou todos) requisitos e usado para confirmar a execução das funcionalidades de interesse.
- Revisões: uma equipe de revisores analisa a especificação em busca de problemas. É a técnica mais utilizada e geralmente inclui algum de tipo de controle de defeitos na especificação, e.g., quadro de revisões.

No fim da fase de validação, o nível de detalhamento dos requisitos do sistema já pode ser considerado razoável e suficiente para a entrega da especificação. Se houver algum problema que impeça a especificação de ser aprovada, é iniciado um novo ciclo do processo aqui descrito e, caso necessário, qualquer atividade da engenharia de requisitos pode ser revisitada até que se obtenha um documento aprovado. A Figura 2.4 ilustra como acontece o ciclo de atividades até se obter uma versão definitiva do documento de requisitos.

Figura 2.4 - Ciclo de fases para obtenção do documento de requisitos. Retirado de [1]



2.2.4 Gerenciamento

Requisitos de software são entidades dinâmicas e, portanto, estão em constante evolução em um sistema. A atividade de gerenciamento dos requisitos visa identificar, controlar e rastrear as mudanças que podem ocorrer em uma especificação. Mudanças podem ocorrer em qualquer fase do desenvolvimento de um sistema e, geralmente, vem acompanhadas de ajustes na codificação ou no sistema como um todo. Por isso, é importante para os engenheiros de requisitos entenderem como cada requisito afeta o sistema e, a partir dessa análise, estimar o custo que mudanças podem trazer.

Para que uma gerência de mudanças seja efetiva, é preciso antes estabelecer um planejamento de gerência para os requisitos do sistema. Dentro desse planejamento, algumas das políticas precisam ser definidas são: identificação dos requisitos, rastreabilidade e

ferramentas de apoio [1]. Esses pontos geralmente são decididos durante o planejamento do projeto e procuram se adequar às demandas de cada equipe. Em casos onde o cliente possui processos de engenharia de software implantados, algumas dessas decisões são impostas pelo contratante como, por exemplo, qual ferramenta de apoio será usada.

A **identificação dos requisitos** define a forma como cada requisito será nomeado e caracterizado dentro da especificação funcional. É importante que essa identificação seja única e escalável, facilitando os processos de rastreabilidade e verificação adotados.

As **políticas de rastreabilidade** definem e registram os relacionamentos requisito-requisito (rastreabilidade horizontal) e requisito-artefatos do sistema (rastreabilidade vertical). Esse processo é importante e muitas vezes subestimado por muitas equipes de desenvolvimento. Manter o controle de mudanças é uma tarefa complexa, mas também essencial em qualquer projeto de software. Executá-la sem uma política de rastreabilidade bem estabelecida pode trazer atrasos e custos adicionais para o projeto. O *Capability Maturity Model Integration*® (CMMI℠) para Engenharia de Software lista manter rastreabilidade bidirecional dos requisitos como uma boa prática no desenvolvimento de sistemas e destaca que "para analisar efetivamente o impacto de mudanças, é necessário que a fonte de cada requisito seja conhecida e a razão para cada mudança seja documentada" [13].

Digamos, por exemplo, que o cliente sugira uma mudança de uma regra de negócio de um sistema que esteja em ambiente de produção, i.e., em uso efetivo pelos usuários finais. Fazer a análise de impacto e, consequentemente, planejar as tarefas de alteração pode ser tornar uma atividade custosa caso o analista não possua uma estrutura de requisitos que dê suporte à rastreabilidade. Quais funcionalidades serão afetadas pela mudança? O que deverá ser testado novamente?

O exemplo acima também pode se aplicar para a ponta oposta da cadeia de desenvolvimento. A mudança pode surgir em um dos produtos do projeto (uma tela do sistema, por exemplo) e não necessariamente na especificação funcional. Nesse caso, é preciso identificar quais os requisitos são afetados pela mudança e, assim, atualizar a documentação apropriadamente.

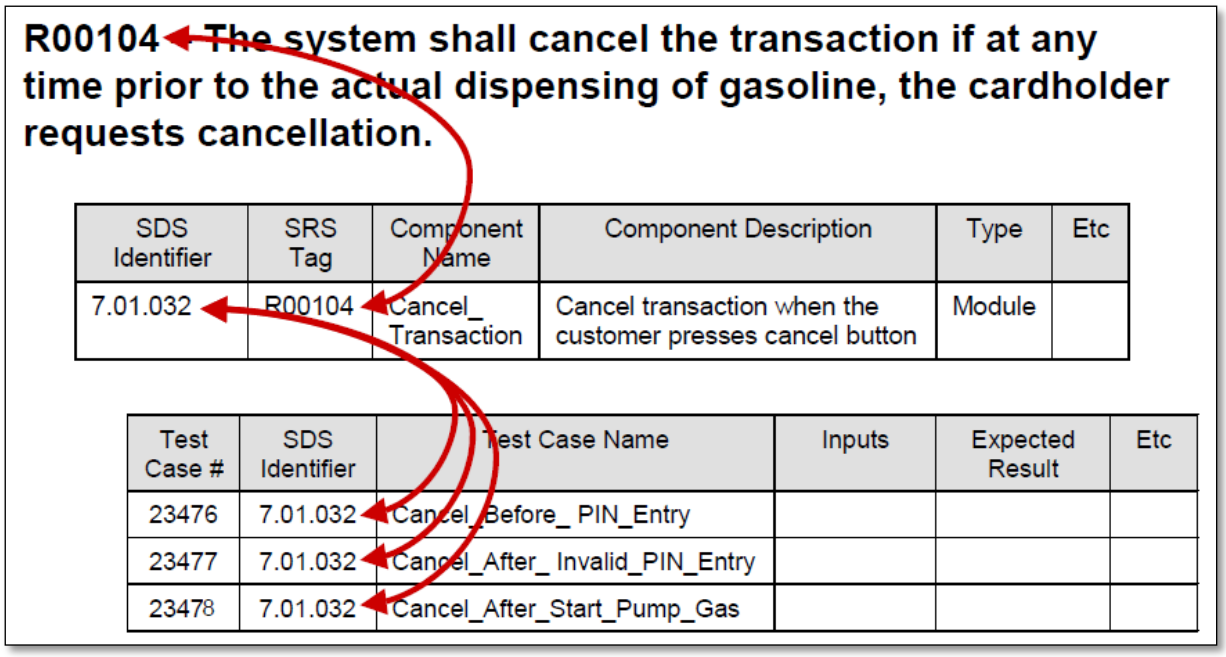
A maneira mais clássica de ser obter rastreabilidade dos requisitos é através da construção de uma matriz de rastreabilidade. Nela, requisitos e os diversos artefatos do projeto são relacionados através das linhas e colunas, sendo possível identificar os relacionamentos de maneira rápida e centralizada. A matriz de rastreabilidade é capaz de armazenar ambas as direções de rastreio (*forward* e *backward*) de todos os produtos em um único local [14], estabelecendo-se como uma técnica poderosa de rastreabilidade. Um exemplo de matriz de rastreabilidade vertical pode ser observado na Figura 2.5.

Figura 2.5 - Exemplo de matriz de rastreabilidade vertical. Retirada de [14]

Requirement Source	Product Requirements	HLD Section #	LLD Section #	Code Unit	UTS Case #	STS Case #	User Manual
Business Rule #1	R00120 Credit Card Types	4.1 Parse Mag Strip	4.1.1 Read Card Type	Read_Card _Type.c Read_Card _Type.h	UT 4.1.032 UT 4.1.033 UT 4.1.038 UT 4.1.043	ST 120.020 ST 120.021 ST 120.022	Section 12
			4.1.2 Verify Card Type	Ver_Card _Type.c Ver_Card _Type.h Ver_Card _Types. dat	UT 4.2.012 UT 4.2.013 UT 4.2.016 UT 4.2.031 UT 4.2.045	ST 120.035 ST 120.036 ST 120.037 ST 120.037	Section 12
Use Case #132 step 6	R00230 Read Gas Flow	7.2.2 Gas Flow Meter Interface	7.2.2 Read Gas Flow Indicator	Read_Gas _Flow.c	UT 7.2.043 UT 7.2.044	ST 230.002 ST 230.003	Section 21.1.2
	R00231 Calculate Gas Price	7.3 Calculate Gas price	7.3 Calculate Gas price	Cal_Gas_ Price.c	UT 7.3.005 UT 7.3.006 UT 7.3.007	ST 231.001 ST 231.002 ST 231.003	Section 21.1.3

Outra técnica usada para garantir a rastreabilidade dos requisitos é a *trace tagging* (rastreamento de marcação, traduzindo para o português). Nessa técnica, os requisitos e os artefatos que os implementam possuem identificadores únicos e são marcados em cada artefato subsequente. Por exemplo, durante a elaboração de um caso de teste, deve-se informar quais *tags* se relacionam com aquele documento (requisitos ou outras especificações, por exemplo) e assim por diante. Exemplo na Figura 2.6.

Figura 2.6 - Ilustração da técnica *trace tagging*. Adaptado de [14]



Ambas as técnicas descritas demandam um comprometimento dos responsáveis para manter o modelo de rastreabilidade atualizado e correto. Assim, é preciso um processo bem estabelecido desde o início das atividades de gerência de requisitos e a garantia que sejam mantidas atividades de atualização em sincronia com as tarefas de elaboração dos artefatos.

Para dar suporte às atividades citadas anteriormente, as **ferramentas de apoio** são indispensáveis na gerência de sistemas com grande quantidade de informação. Nessa categoria se encaixam desde simples planilhas até sistemas especializados na gerência de requisitos.

2.3 Enterprise Architect

O avanço tecnológico de ferramentas CASE (**C**omputer-**A**ided **S**oftware **E**ngineering) tem auxiliado a padronização e consequente implantação de processos em ambientes corporativos de desenvolvimento de software. Esse efeito é notável, sobretudo, em empresas de pequeno a médio porte, onde o barateamento dessas ferramentas possibilitou a adoção e gerência de processos em infraestruturas menos robustas. Dentro desse âmbito, o surgimento do CMMI[SM] (*Capability Maturity Model Integration*) teve grande contribuição na melhoria de processos da engenharia de software.

Desenvolvido pelo Instituto de Engenharia de Software (SEI, *Software Engineering Institute*) da Universidade Carnegie Mellon, o CMMI[SM] é um modelo de referência, listando práticas a serem adotadas por organizações a fim de se obter melhoras no desempenho da empresa e amadurecimento dentro no mercado [15]. Estruturalmente, o CMMI[SM] define Áreas de Processo (*Process Areas*) e práticas recomendadas para cada um desses grupos. É importante ressaltar que o foco do modelo não é determinar metodologias específicas de cada área, mas listar as práticas que devem ser implantadas para se obter os resultados esperados [16].

O **Enterprise Architect** (EA) é uma ferramenta CASE de modelagem baseada no padrão UML (*Unified Modeling Language*) e mantido pela Sparx Systems[3]. O EA oferece suporte para todas as atividades do ciclo de vida de um sistema, desde a modelagem dos requisitos até a fase de testes e manutenção. Dentre as funcionalidades suportadas pela plataforma, destacam-se: diagramação UML, modelagem de dados, controle de versão, gerenciamento de requisitos e engenharia de código [17]. Por meio dessas ferramentas, usuários podem customizar diversos processos relevantes no desenvolvimento de software, adequando-os ao ambiente e aos recursos do projeto.

[3] http://sparxsystems.com/

Hunnicutt e Millar [18] analisam como as diversas práticas descritas no CMMI[SM] são satisfeitas dentro do Enterprise Architect. Para realizar o comparativo, foi considerado como alvo a implementação dos *Work Products* (ou artefatos de processo) idealizados para cada Área de Processo dentro do CMMI[SM]. *Work Products* podem ser definidos como o produto resultante da aplicação de um processo, e.g., relatórios de avaliação oriundos de um processo de garantia de qualidade. O comparativo para o Processo de Gerenciamento de Requisitos (REQM) pode ser conferido na Tabela 2.1.

Tabela 2.1 - Mapeamento entre Work Products (artefatos de processo) do CMMI e as funcionalidades do Enterprise Architect. Adaptado de [18].

Work Products (CMMI[SM])	Implementação No Enterprise Architect
Histórico de mudanças	Base de dados integrada de requisitos
Rastreabilidade bidirecional	Diagramas de hierarquia
Matriz de rastreabilidade	Diagramas de hierarquia
Impactos do requisito	Propriedades dos requisitos
Repositório de requisitos	Oferece suporte para oito tipos de banco de dados
Acompanhamento de requisitos	Geração de relatórios situacionais
Compromissos documentados	Registrados em narrativas

Alguns exemplos das funcionalidades oferecidas no EA podem ser conferidas na Figura 2.7 e na Figura 2.8. A Figura 2.7 destaca a tela de modelagem de requisitos, onde são definidas propriedades dos requisitos. Por meio dessa ferramenta, o impacto de um requisito no sistema pode ser estipulado e, também, acompanhado. Na Figura 2.8 é ilustrada a matriz de rastreabilidade gerada pelo software. Nessa tela, é possível realizar diversas análises para diferentes aspectos do sistema, abrangendo os dois tipos de rastreabilidade (horizontal e vertical) e os tipos de relacionamento entre as entidades.

Outras funcionalidades presentes na Tabela 2.1 são demonstradas em mais detalhes no Capítulo 4, onde é feita a modelagem para uma iteração de um sistema de exemplo, adotando o processo de modelagem utilizado pela empresa foco do estudo.

Figura 2.7 - Janela de modelagem de requisitos e elementos do EA.

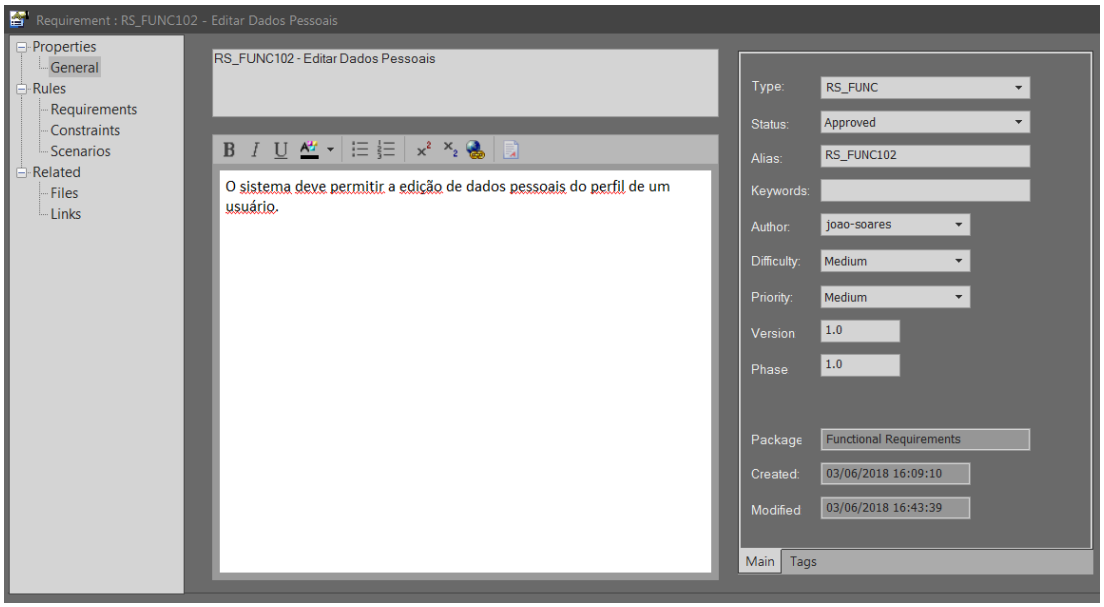
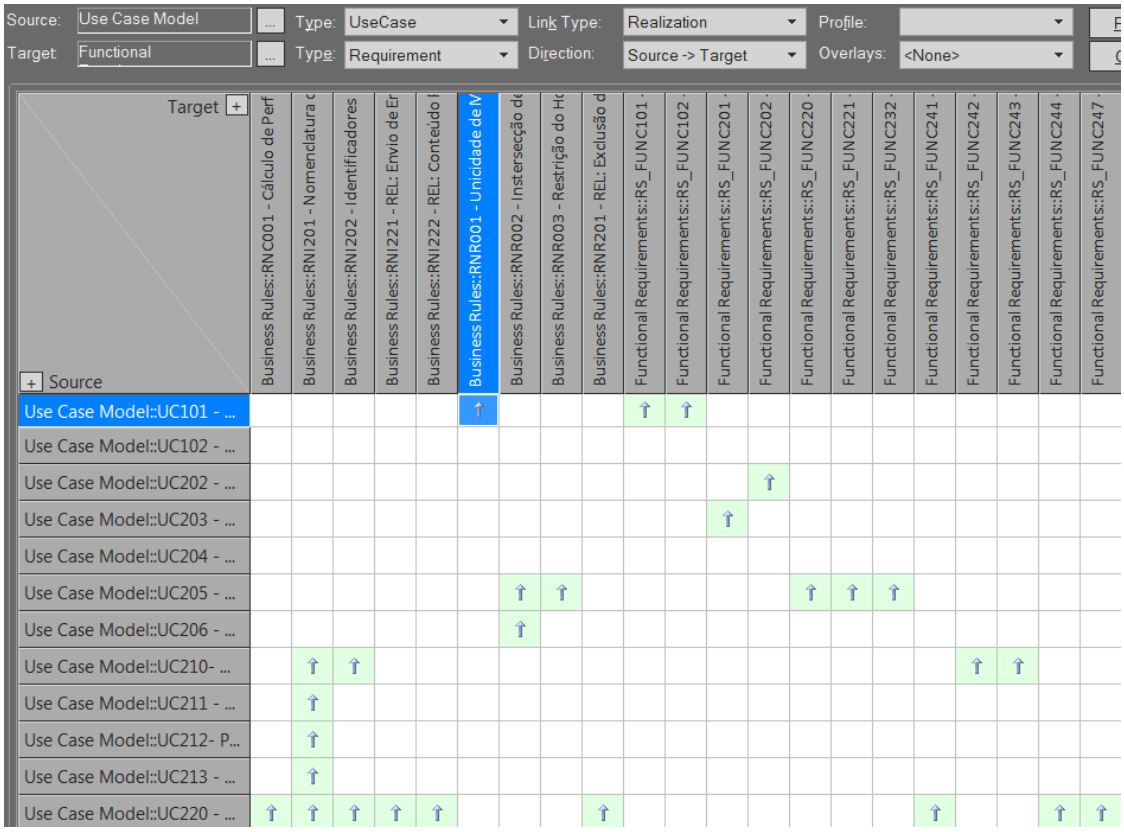


Figura 2.8 – Matriz de rastreabilidade horizontal no EA.



2.4 Modelo XMI

Uma preocupação importante dentro do design de ferramentas CASE é a portabilidade do modelo para outras aplicações ou até mesmo entre membros da equipe que também usam a ferramenta. Para isso, é importante que haja um padrão de manipulação e armazenamento dos elementos, permitindo a integração do modelo com outras interfaces sem perda de informações. Para as ferramentas baseadas em UML, é utilizada a linguagem MOF (*Meta Object Facility*) [19] como padrão para o intercâmbio de modelos e meta-modelos. A MOF define como as informações do modelo UML serão tipificadas e as regras associadas a esses tipos.

Contudo, também é preciso definir o formato no qual o meta-modelo será encapsulado. Para esse papel, surge o formato XMI (*XML Metadata Interchange*) [20], adotado como padrão pela especificação da MOF [19]. XMI é baseado no padrão XML (*eXtensible Markup Language*), definido pela W3C [4] (*World Wide Web Consortium*), organização responsável pelo padrão HTML. Isso significa que a sintaxe e ambientação do XML são usadas pelo XMI para a troca de informação de aplicações de design orientado a objeto, desde diagramas até modelos de dados. Portanto, os arquivos XMI possibilitam a integração de modelos UML entre diferentes plataformas; um dos principais interesses deste trabalho.

2.4.1 Estrutura XMI

A linguagem XMI é dividida em duas entidades: os documentos e os DTDs (*Document Type Declaration*). A primeira organiza as informações como um conjunto de rótulos, enquanto os DTDs definem regras para a manipulação desses rótulos. Isto é, os documentos representam os dados de domínio e os DTDs ditam a maneira como esses dados

[4] https://www.w3.org/

devem ser interpretados. Ambas as entidades precisam existir para que qualquer informação útil seja extraída do arquivo.

Um conjunto de documento de DTD é disposto na Figura 2.9 e na Figura 2.10, respectivamente. Os rótulos (também chamados de elementos) possuem uma cláusula de início e fim, nas quais o conteúdo do rótulo é declarado. No DTD da Figura 2.10 é definido que o rótulo "Auto" pode conter os rótulos "Make", "Model", "Year", "Color" e "Price".

Figura 2.9 – Exemplo de um documento XMI para modelar um automóvel. Retirado de [21]

```
<Auto>
    <Make>   Ford          </Make>
    <Model>  Mustang  </Model>
    <Year>   99       </Year>
    <Color>  blue          </Color>
    <Price>  25000    </Price>
</Auto>
```

Figura 2.10 – Exemplo de DTD para um automóvel. Retirado de [21]

```
<!ELEMENT Auto (Make, Model, Year, Color, Price)>
```

A conversão do modelo UML para o formato XMI segue dois procedimentos principais. Primeiro, é gerado o DTD, usado para o intercâmbio das meta-informações do modelo. Segundo, é gerado o arquivo XMI que fará uso do DTD gerado, contendo os dados dos rótulos. Dada a classe "Auto", do exemplo anterior, as seguintes regras são obedecidas na etapa de geração do DTD [21]:

- Existe apenas um elemento para cada classe e atributo;
- Cada classe ou atributo é unicamente identificada dentro do modelo XMI;
- Cada classe e cada atributo contém sua própria declaração do tipo de dado a ser inserido;

Para nosso exemplo, o DTD resultante da conversão está na Figura 2.11. Os tipos declarados para o elementos classe-atributo podem conter valores de texto ("#PCDATA") ou mesmo tipos não suportados pelo padrão XML (*XMI.reference*). A geração do DTD permite que o intercâmbio do modelo UML seja possível para diferentes ferramentas que dão suporte ao modelo XMI. Exemplos dessas ferramentas são IDEs e plataformas de design.

Figura 2.11 – DTD XMI originado da conversão da classe "Auto" para o modelo XMI. Retirado de [21]

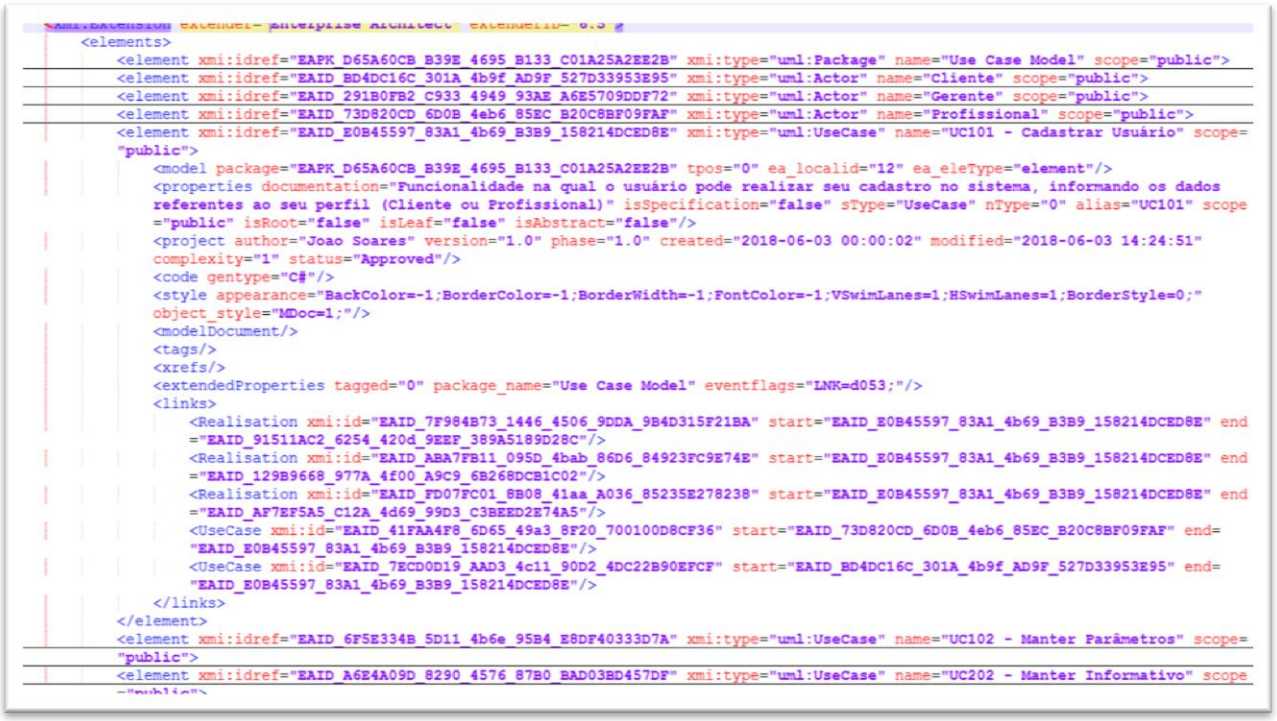
```
<!ELEMENT Auto (Auto.Make, Auto.Model, Auto.Year,
                Auto.Color, Auto.Price,
                XMI.extension*)? >
<!ATTLIST Auto %XMI.element.att; %XMI.link.att;>

<!ELEMENT Auto.Make  (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Model (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Year  (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Color (#PCDATA | XMI.reference)* >
<!ELEMENT Auto.Price (#PCDATA | XMI.reference)* >
```

Todo modelo a ser descrito contém elementos estruturais definidos pelo padrão XMI. Na hierarquia XML, essas informações ficam armazenadas dentro do elemento *XMI element*. Quatro subelementos são definidos para o *XMI element*, são eles:

- *Header*: contém informações de versionamento e onde é possível documentar o elemento XMI em questão;
- *Content*: onde os dados de domínio a serem transferidos são inseridos;
- *Differences*: onde é disposta a diferença entre duas versões de um modelo XMI. Útil para evidenciar pequenas mudanças em modelos XMI muito longos (grande quantidade de elementos);
- *Extensions*: onde é possível realizar a transferência de informações específicas de cada ferramenta. Essa estrutura é importante para intercâmbios recorrentes, onde informações que dependem da estrutura interna das ferramentas são relevantes.

A Figura 2.12 traz um exemplo de arquivo XMI exportado para um pacote de casos de uso do Enterprise Architect. Notar como a estrutura dentro da diretiva *extensions* segue padrão específico da ferramenta. Para o desenvolvimento que discutiremos nesta monografia, foi feito um estudo do modelo XMI para se analisar como as informações do modelo de requisitos (casos de uso, requisitos, cenários, dentre outros) serão extraídas e sincronizadas com o Agilo.

Figura 2.12 – Trecho do modelo XMI gerado pelo Enterprise Architect.



2.5 Desenvolvimento Iterativo Incremental

Uma prática comum em ambientes de desenvolvimento é a divisão do escopo do projeto em pequenos incrementos. As iterações são janelas de tempo de duração pré-estabelecida onde esses incrementos do projeto são encaixados [8]. Dentro das iterações, todas as fases do desenvolvimento podem ser visitadas: planejamento, análise, codificação e testes. No fim de cada iteração, um produto é demonstrado ao cliente e o *feedback* registrado. A esse processo é dado o nome de **iterativo incremental**, ou apenas desenvolvimento

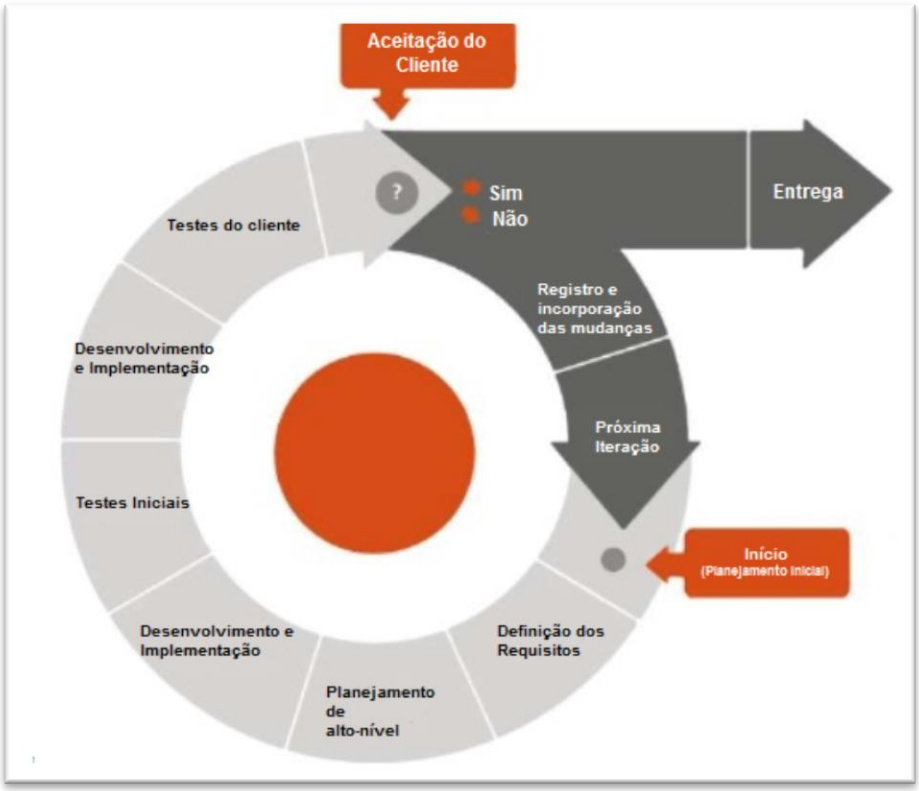
iterativo. A Figura 2.13 ilustra o ciclo de atividades do método. Importante destacar que as mudanças de uma iteração são incorporadas nas próximas, trazendo uma evolução gradativa da implementação.

A origem desse método data dos anos 30, quando um especialista da Bell Labs propôs o uso de ciclos "planejar-fazer-estudar-agir" (do inglês, *plan-do-study-act*) para melhorar a qualidade dos projetos da empresa [22]. Considerando a volatilidade dos projetos de software, a abordagem do desenvolvimento iterativo surge como uma maneira de mitigar os problemas decorrentes de mudanças de requisitos.

Ao quebrar o escopo em iterações, os analistas de requisitos tem um ambiente mais propício para executar os processos da engenharia de requisitos, onde a complexidade e quantidade de informações são fracionadas. A gerência de riscos também é beneficiada, pois o *feedback* recorrente do cliente permite um melhor entendimento do problema ao longo do processo de desenvolvimento [23]. Basili e Turner [23] sugerem diretrizes para o uso do processo de desenvolvimento iterativo. Algumas delas estão abaixo:

- Qualquer dificuldade no design, codificação ou depuração de uma modificação deve indicar a necessidade de reestruturar os componentes já existentes;
- Modificações devem ser endereçadas para módulos isolados e de fácil localização. Qualquer dificuldade nesse processo indica falhas no planejamento e o *redesign* é recomendado;
- A opinião do usuário deve ser sempre solicitada e analisada para indicação de deficiências na implementação existente;
- O sistema deve ser constantemente analisado para se determinar o quanto ele está satisfazendo os objetivos do projeto.

Figura 2.13 – Ilustração do ciclo de atividades do método iterativo incremental. Adaptado de [24]



2.6 Agilo For Trac

O Agilo for Trac é uma ferramenta *web-based* usada para controle e manutenção de projetos. Com uma estrutura flexível e minimalista, o Agilo auxilia a execução das tarefas rotineiras dos desenvolvedores sem impor processos ou políticas específicas para os usuários [25]. Apesar de ser uma modificação da ferramenta Trac para dar suporte à metodologia *Scrum*, o Agilo oferece extensibilidade de novas funcionalidades e políticas por meio de plug-ins gratuitos [26].

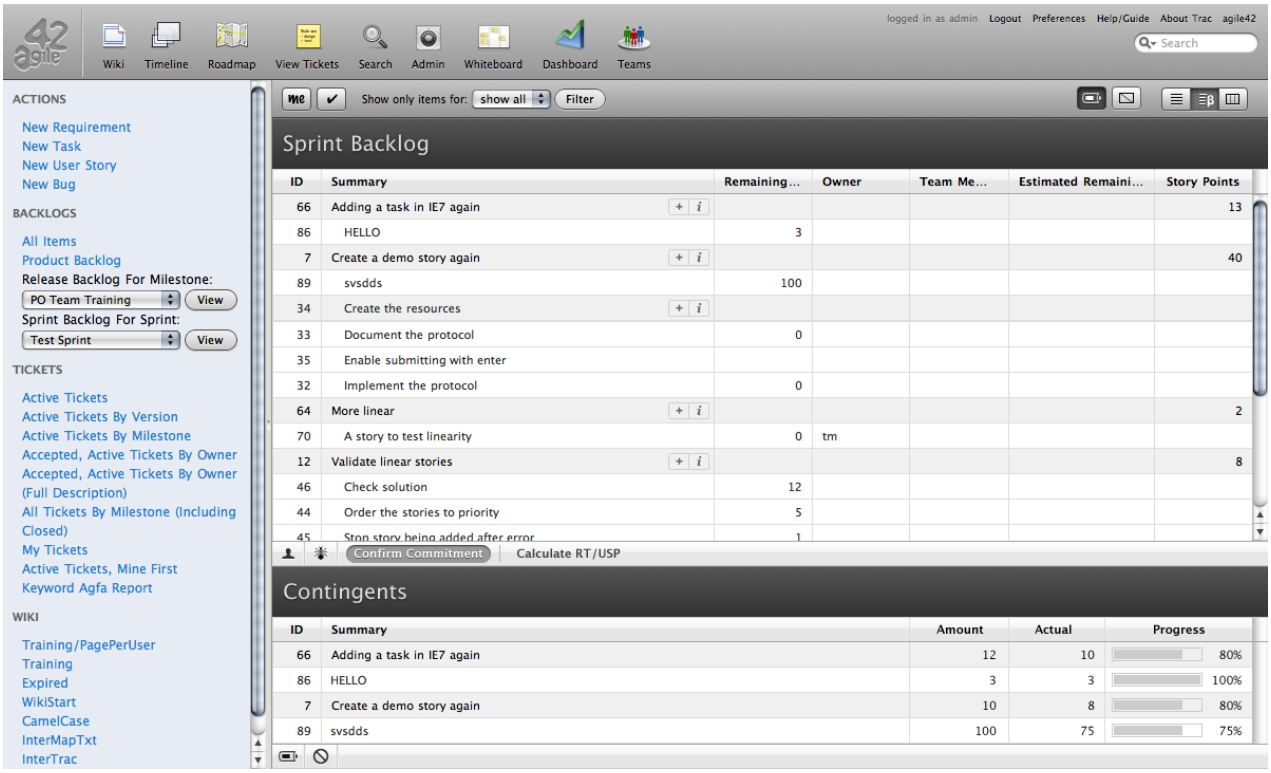
Componentes centrais na estrutura do Agilo, os *tickets* podem ser usados para representar tarefas, *bugs*, solicitações de mudanças ou outros elementos do desenvolvimento de software. Tipos e fluxos de tíquetes são parâmetros configuráveis na plataforma,

permitindo a adequação de processos específicos de cada usuário no uso da ferramenta. Tíquetes podem ser editados, comentados, atribuídos, priorizados e discutidos dentro da ferramenta, de acordo com as políticas de acesso configuradas para perfil. Tíquetes podem referenciar ou serem referenciados outros tíquetes. Por meio dessa funcionalidade, é possível realizar políticas de rastreabilidade e hierarquia de tíquetes dentro da ferramenta. Além disso, os campos que compõe cada tipo de tíquete podem ser modificados, acrescendo ainda mais a capacidade de customização do Agilo.

O Agilo também proporciona visualização e acompanhamento das atividades através das visões de *backlog* dos tíquetes. O *backlog* funciona como um repositório dos tíquetes de alguns tipos pertencentes a uma *milestones* (ou iteração). A Figura 2.14 ilustra a tela de visualização de *backlogs*.

O Agilo For Trac tem sido usado por anos em muitos dos processos adotados na empresa foco deste estudo. Tarefas como apontamento de horas, gerência de configuração e controle de versionamento foram integradas dentro da ferramenta, tornando o Agilo elemento essencial na rotina e gerência interna da equipe.

Figura 2.14 – Visualização de Backlogs no Agilo. Retirado de [26]



3 Desenvolvimento da solução

Neste capítulo, são apresentados os detalhes envolvidos no desenvolvimento da customização proposta, explicando sua estrutura, funcionalidades e adaptações necessárias. Ao longo das seções, os pontos relevantes para a resolução dos problemas levantados são destacados e analisados, bem como informações importantes e desafios encontrados ao longo do desenvolvimento teórico e técnico da solução.

3.1 Visão Geral

A solução proposta foi desenvolvida como um *plugin* de extensão da ferramenta Agilo For Trac. De maneira geral, a customização procura trazer a estrutura de requisitos e casos de uso especificados na plataforma de modelagem Enterprise Architect (EA) para a plataforma Agilo, adaptando a estrutura de *tickets* da ferramenta para a estruturação dos elementos importados.

Para a extração do modelo de requisitos, foi usado o documento XMI da ferramenta de modelagem. Informações como definição de elementos e relacionamento entre eles são extraídas do modelo e adaptadas para uso dentro do Agilo For Trac. Para cada elemento extraído, é criado um ticket dentro do ambiente do Agilo For Trac. O relacionamento entre as entidades é feito através de *links* entre os *tickets*, definindo origem e destino do relacionamento.

Uma vez que os *tickets* são criados dentro do ambiente do Agilo For Trac, a estrutura de *tickets* pode ser aplicada na gerência dos requisitos. Isto é, histórico de mudanças e discussões são centralizados nos *tickets* correspondentes, dando ao desenvolvedor mais objetividade e participação no processo de gerenciamento da especificação.

É importante ressaltar que a integração executada pelo *plugin* não tem como objetivo trazer o processo de manutenção dos requisitos para o Agilo. Mesmo com o uso da customização, as tarefas de modelagem e especificação do projeto continuam sendo realizadas através do EA, sendo as informações de requisitos inalteráveis pelos desenvolvedores dentro Agilo.

A estruturação dos elementos especificados no modelo de requisitos através de *tickets* permite que algumas *features*, ou funcionalidades, sejam incluídas no processo de gerência de

requisitos do time de analistas de requisitos da empresa. A lista de funcionalidades rastreadas é exibida na Tabela 3.1. Na Seção 3.4, essas funcionalidades são explicadas em detalhes, trazendo elementos da solução para evidenciar a forma como as mesmas são implementadas.

Tabela 3.1 - Lista de funcionalidades obtidas pelo uso da solução.

FUNCIONALIDADE	RESUMO
Controle de responsabilidade dos requisitos	Com o sistema de responsabilidade dos tickets, os desenvolvedores podem endereçar suas dúvidas de maneira direta ao analista de requisitos responsável pela especificação daquele requisito.
Versionamento de mudanças	O Agilo For Trac versiona as mudanças realizadas nos *tickets* dentro da própria estrutura do ticket, sendo o histórico de mudanças sempre acessível aos usuários.
Centralização de dúvidas e solicitações de mudanças (*change requests*)	O processo de solicitação de mudanças de um requisito e dúvidas sobre ele pode ser acompanhado por meio do ticket que o representa. Desse modo, desenvolvedores podem sugerir e engatilhar mudanças dentro da plataforma Agilo For Trac, que manterá o histórico de discussões para análise por parte dos engenheiros de requisitos.
Visualização centralizada dos requisitos (pelos desenvolvedores)	A visualização dos requisitos é centralizada na ferramenta de controle de tarefas, evitando que os desenvolvedores precisem ter acesso ao Software utilizado para a modelagem.

3.2 Adaptações no Agilo

Apenas trazer as informações da ferramenta EA para dentro do Agilo não é suficiente para atender a todas as funcionalidades citadas na seção anterior. É preciso que os *tickets* usados na integração se comportem de maneira que seja possível para os desenvolvedores compreenderem a dinâmica dos requisitos. No Agilo For Trac, o sistema de *tickets* permite

adaptações variadas na estrutura e fluxo de vida dessas entidades, sendo possível a customização para diferentes necessidades do time. As mudanças feitas para a solução se resumem em duas: criação de novos tipos de *tickets* e alteração das propriedades individuais desses tipos.

Dois novos tipos de tickets foram criados para integração com o EA. Essa medida foi necessária para personalizar o tratamento dos *tickets* gerados pelo *plugin* de integração e, também, evitar o acoplamento com a estrutura de tíquetes usada para delegação de tarefas. Dessa forma, é possível gerir o modelo gerado pela integração com EA de maneira individual, como, por exemplo, construir visões de *backlog* restritas para o modelo de requisitos.

Seguindo o processo de modelagem utilizado pela equipe, foram criados tipos de *ticket* para lidar com as duas entidades mais recorrentes na modelagem dos projetos: os casos de uso e os requisitos. O tipo *Requirement* se refere aos requisitos criados para o sistema, desde regras de negócio a mensagens usadas no *Software*. Os casos de uso são descritos no tipo *Use Case*, onde o fluxo de execução é descrito e o relacionamento com os requisitos é feito através da funcionalidade Agilo *Links*, nativa da plataforma Agilo.

Para cada novo tipo de *ticket* criado, as propriedades dos mesmos foram revisadas, tendo como base as informações dispostas no EA. Dessa forma, novos campos foram criados para os tipos *Use Case* e *Requirement* para que as informações contidas no EA pudessem ser aproveitadas no processo de integração, dentre eles, pode-se citar os campos "*Version",* usado no versionamento de um requisito e "*Alias",* ou apelido, usado pelos analistas de requisitos para armazenar os identificadores dos elementos.

3.3 Arquitetura do Software

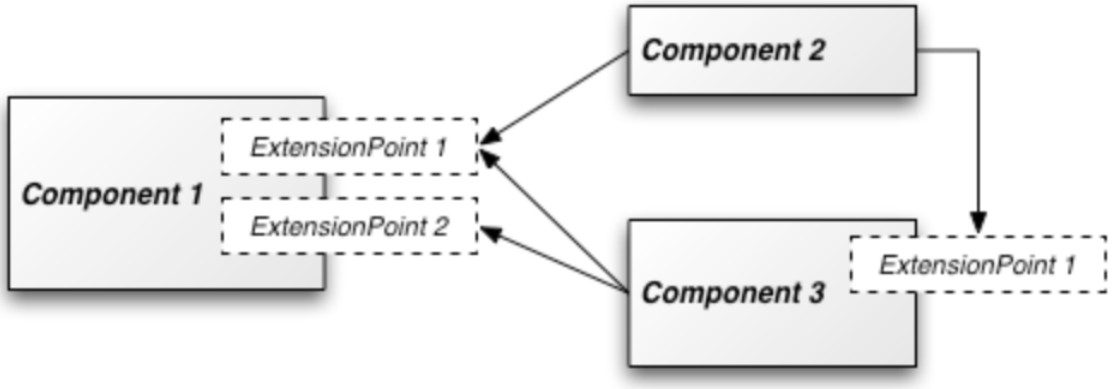
A ferramenta Agilo For Trac permite a adição de funcionalidades extras por meio do uso de *plugins*. Os *plugins* podem oferecer diversos serviços dentro da plataforma, desde a manipulação especial de *tickets* até a integração com sistemas externos. A estrutura dos *plugins* é mantida pelo núcleo de componentes do Agilo, onde são implementadas todas as funcionalidades básicas que tornam a ferramenta extensível.

3.3.1 Arquitetura de Componentes do Agilo

Na arquitetura do Agilo, um componente é definido como um objeto que provê algum tipo de serviço para a plataforma. De maneira geral, cada componente representa um subsistema funcional e, portanto, é único no contexto da aplicação.

Todo componente pode declarar pontos de extensão, ou *extension points*, onde algumas ou todas as suas funcionalidades são expostas para outros componentes. Assim, cada componente pode estender outro componente e ser extensível simplesmente expondo e acessando *extension points.* Esse tipo de estrutura permite que o Agilo seja facilmente customizável. A Figura 3.1 ilustra como a interação entre componentes é definida.

Figura 3.1 – Arquitetura de componentes do Agilo For Trac. Imagem retirada de [25].



3.3.2 Estrutura do *plugin*

Para adicionar qualquer tipo de funcionalidade personalizada no Agilo, é preciso implementar um componente. Portanto, todo *plugin* é, para o núcleo da ferramenta, um componente. E, como explicado na Seção 3.3.1, componentes se comunicam entre si através dos pontos de extensão. A Figura 3.2 exibe trecho do código de um *plugin* que estende (ou implementa) *extension points*. A classe do *HelloWorldPlugin* herda a classe *Component*, usada pelo Agilo para a implementação de *plugins*. Então, a implementação dos pontos de extensão *INavigationContributor* e *IRequestHandler* é declarada pelo método *implements*.

Figura 3.2 – Trecho de um *plugin* do Agilo For Trac. Imagem retirada de [25].

```
from trac.core import Component, implements
from trac.util.html import html
from trac.web import IRequestHandler
from trac.web.chrome import INavigationContributor

class HelloWorldPlugin(Component):

    implements(INavigationContributor, IRequestHandler)
```

Na implementação da solução, foram estudados e selecionados pontos de extensão a serem implementados pelo *plugin*. Para essa análise, é preciso considerar quais as entidades e funcionalidades já existentes no ambiente do Agilo vão ser afetadas pela customização. A Tabela 3.2 lista os *extension points* usados no código da implementação junto a suas descrições. O código do componente do *plugin* pode ser conferido no Apêndice A - Código: Componente do *plugin*.

Tabela 3.2 – Pontos de extensão implementados pelo *plugin*. Imagem retirada de [25].

Ponto de Extensão	Descrição
INavigationContributor	Permite que *plugins* personalizem o menu de navegação da interface web
ITemplateProvider	Permite que *plugins* usem seus próprios *templates* e recursos estáticos (css, js, etc.)
IRequestHandler	Permite ao componente processar requisições HTTP da interface web
IRequestFilter	Permite ao componente pré- e pós-processar requisições HTTP da interface web
IEnvironmentSetupParticipant	Necessário para plugins que possuem seu modelo de dados próprio, pois permite a manipulação da estrutura da base de dados durante a criação e atualização do ambiente

3.3.3 Leitura do modelo XMI

Todas as informações do modelo presente no Enterprise Architect são extraídas de arquivos XMI gerados pela ferramenta [17]. Dois modelos do EA são usados no processo de importação: *Requirements Model* e *Use Case Model*. A escolha desses dois modelos foi baseada no processo de levantamento de requisitos usado pela empresa de estudo. No processo, os analistas modelam as funcionalidades por meio de casos de uso, com os devidos requisitos sendo referenciados. Assim, a principal fonte de informação para os desenvolvedores está nos casos de uso e requisitos presentes no modelo.

Quando a fase de implementação de uma iteração se inicia, os casos de uso são as entidades levadas aos desenvolvedores para os estágios de codificação e teste. Isso implica que as atividades cadastradas no Agilo e a delegação de responsabilidades das mesmas ocorrem em nível de casos de uso. No processo atual, cada caso de uso é cadastrado na *milestone* da iteração correspondente como um *ticket* do tipo Tarefa e, por meio de ferramenta interna da equipe, *tickets* das sub-tarefas de implementação e teste são criados automaticamente para cada caso de uso.

Contudo, no procedimento, nenhuma informação adicional do EA é trazida para dentro do Agilo. Dessa forma, as entidades do sistema (i.e., casos de uso e requisitos) permanecem isoladas na ferramenta EA. Em vista dessa situação, justifica-se a escolha dos modelos supracitados como os elementos a serem importados pela ferramenta.

Dentro desses dois modelos escolhidos para serem importados pela customização, algumas informações se mostraram críticas para o escopo da rotina da equipe, são elas: descrição dos requisitos e dos casos de uso, fluxos de execução dos cenários e relacionamentos entre as entidades. Essas três informações foram classificadas como prioritárias para o processamento do modelo XMI e, portanto, tiveram mais peso durante a modelagem do modelo de dados elaborado para o *plugin*.

Um componente foi implementado para o processamento dos arquivos XMI e, na sua estrutura, a extração das informações do modelo foram modularizadas considerando as entidades: Casos de Uso, Cenários e seus fluxos, Requisitos e Referências. Um trecho da implementação desse componente é exibido na Figura 3.3. No trecho, é definida a classe *Xmi*, responsável pela leitura e aquisição das informações do modelo. Essa classe lê arquivos XMI

e realiza a extração dos elementos de interesse (casos de uso, requisitos, cenários e referências). O código completo do módulo XMI pode ser visualizado no Apêndice B.

Figura 3.3 - Trecho de código do componente de processamento dos arquivos XMI.

```
class xmi(object):
    ns = {'xmi': "http://schema.omg.org/spec/XMI/2.1",
          'uml': "http://schema.omg.org/spec/UML/2.1"}
    elementos = []
    referencias = []
    filtroUC = ["UseCase"]
    filtroRE = ["Requirement"]
    def __init__(self):
    def obterUseCases(self):
    def obterScenarios(self, ucXml, elemento):
    def obterRequirements(self):
    def obterReferencias(self):
    def listarReferencias(self):
    def listarUseCases(self):
    def listarScenarios(self, elemento)
```

3.4 Funcionalidades

Com a estrutura do *plugin* relatada nas seções anteriores, é relevante destacar as funcionalidades agregadas pelo uso da customização para a equipe em estudo neste trabalho. Nesta seção, as funcionalidades resumidas na Tabela 3.1 serão exploradas em detalhes, evidenciando como a solução provê recursos para a implementação das mesmas.

3.4.1 Controle de responsabilidade dos requisitos

A tarefa de levantamento e especificação de requisitos nem sempre é responsabilidade de apenas um analista de requisitos. Dentro de um mesmo projeto, é possível que a documentação tenha sido elaborada por vários analistas, seja de forma contínua (diferentes

analistas em diferentes iterações) ou paralela (vários analistas dividem as atividades em uma mesma iteração).

Na fase de ambientação dos desenvolvedores com a documentação da iteração corrente, é comum que os analistas sejam consultados para solução de dúvidas e discussões sobre o modelo de requisitos especificado. Também, é importante que os desenvolvedores responsáveis por implementar os requisitos e casos de uso estejam em constante contato e tenham acesso aos analistas que especificaram os elementos em questão.

Ao trazer o modelo presente no EA para dentro do Agilo, os desenvolvedores e analistas de requisitos podem ter um controle de responsabilidades e, com isso, serem notificados de alterações nos elementos de interesse. Isso é possível através do sistema de notificações do Agilo, projetado para alertar os membros cadastrados como dono e responsável daquele *ticket*. Caso mais membros precisem ser notificados de mudanças na estrutura do *ticket* de um elemento do modelo de requisitos, o campo de cópia (*cc*) pode ser usado para adicionar recipientes extras a serem notificados.

3.4.2 Versionamento de mudanças

Um ponto importante na dinâmica do desenvolvimento de *Software* é o controle do que foi modificado na documentação do sistema. Esse ponto é relevante não apenas para os analistas, mas também para os desenvolvedores do sistema. Possuir um histórico de onde e quando a documentação foi alterada pode auxiliar na avaliação do trabalho a ser feito.

No sistema de *tickets* do Agilo, informações como descrição e versão do ticket podem ser alteradas por um usuário com as devidas permissões. Os dados anteriores, contudo, não são descartados, mas salvos na própria estrutura do ticket (*log* de alterações). Dessa forma, duas informações ficam disponíveis para os usuários: quando houve a alteração e o que foi alterado.

Para o contexto considerado neste trabalho, essa funcionalidade agora é aplicada também para o modelo de requisitos do sistema. Quando a ferramenta EA e o Agilo são sincronizados, as atualizações são devidamente versionadas nos *tickets* correspondentes e os

desenvolvedores passam a ter acesso ao histórico de mudanças das entidades por meio dos *tickets* que as representam.

Na Figura 3.4, a descrição de um ticket do tipo *Use Case* é exibida. Ao lado do nome do campo, é possível notar a mensagem indicando a modificação e seu autor. Ao clicar no *hyperlink*, o usuário é direcionado para a página de diferenciação, exibida na Figura 3.5. As mudanças que foram executadas no EA são destacadas dentro do *ticket* do caso de uso no Agilo.

Figura 3.4 - Disposição do campo Descrição de um ticket do tipo Use Case e o indicativo de mudanças.

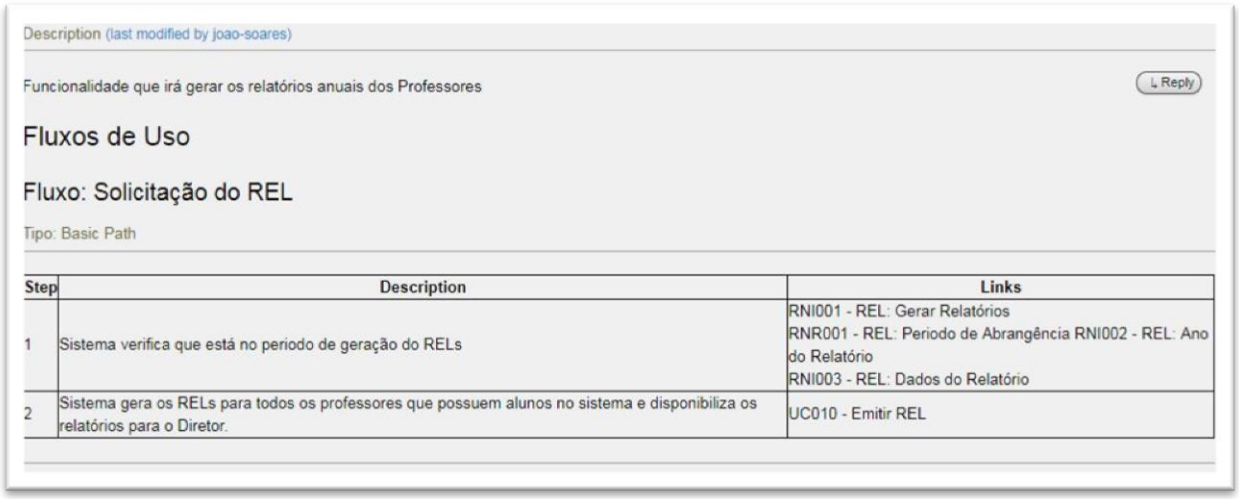
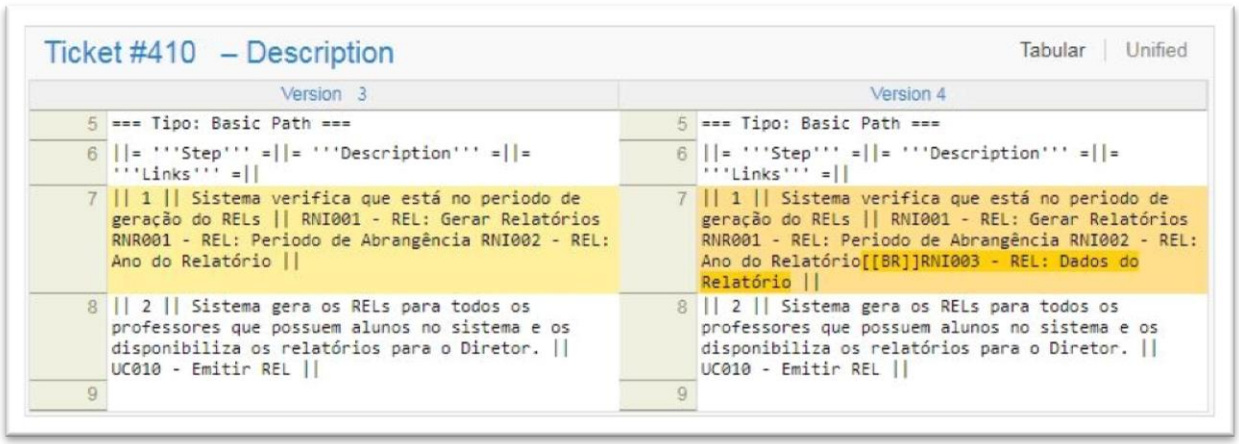


Figura 3.5 - Diferenciador de versões de *tickets* do Agilo.



Todo o histórico de modificações é mantido dentro do próprio ticket por meio de anotações na seção de comentários. Assim, é possível acessar o diferenciador para qualquer mudança que tenha sido executada no *ticket*. A Figura 3.6 ilustra um exemplo de como as alterações são salvas na estrutura do *ticket* (o diferenciador é acessível por meio do *hyperlink* "diff").

Figura 3.6 - Histórico de mudanças persistido por meio de comentários no Agilo.

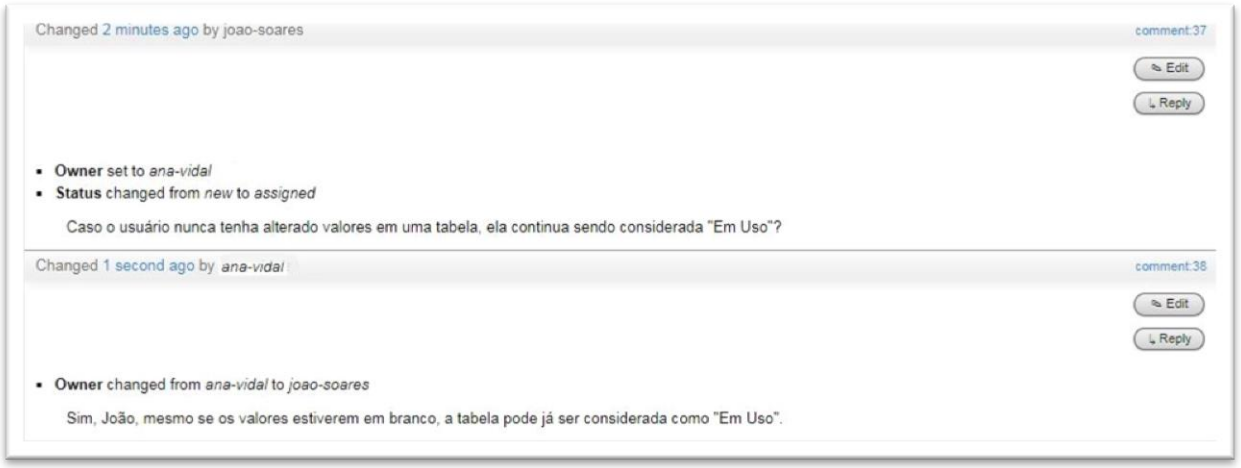
3.4.3 Centralização de dúvidas e solicitações de mudanças

Um ponto relevante na abordagem proposta diz respeito às dúvidas que surgem durante a ambientação do time de desenvolvimento com a documentação. Na rotina da equipe da empresa, o desenvolvedor que tivesse dúvidas sobre a documentação dirigia-se a qualquer membro da equipe de analistas para tentar endereçar seu questionamento. Como não havia ferramenta oficial para o registro de dúvidas, os canais de comunicação interna da equipe eram usados para esse fim e toda a troca de informação era individualizada entre os dois membros.

Com o uso da customização na rotina do time, as dúvidas podem ser registradas diretamente nos *tickets* dos elementos do sistema, e o responsável é então notificado pela plataforma do Agilo For Trac. Ainda que haja a troca de informação de maneira verbal, os pontos discutidos relevantes podem ser cadastrados no *ticket*. Esse histórico de interação agrega valor ao processo de gerência dos requisitos, pois expõe as discussões para os demais

membros e facilita a análise de mudanças pelos analistas. A Figura 3.7 ilustra a seção de comentários do Agilo.

Figura 3.7 - Seção de comentários do Agilo. Reproduzida pelo Autor.



No mesmo contexto, é possível usar a ferramenta para uso no processo de gerência de mudanças nos requisitos, isto é, manter o histórico de solicitações de mudanças associado ao requisito no próprio Agilo For Trac. No processo atual do time, quando surge uma solicitação de mudança, é criado um *ticket* para o cadastro das tarefas associadas a ela. Essa ação, contudo, é sempre engatilhada por solicitações do cliente e, por isso, permanece isolada dentro do contexto da equipe de desenvolvimento, ou seja, não possui interação com o restante das tarefas dentro ferramenta.

Ao integrar o modelo de requisitos por meio da customização, passa a ser possível gerir todo o processo de mudanças de requisitos dentro do Agilo. Agora, um *ticket* do tipo *change request* está associado a um *ticket* do tipo requisito, que, por sua vez, possui seu próprio histórico de discussões e mudanças persistido e de fácil acesso ao analista de requisitos. Além disso, o processo de solicitações de mudanças desencadeadas das análises do time de desenvolvimento é otimizado, uma vez que as discussões estão centralizadas no Agilo.

4 Prova de Conceito

Este capítulo descreve uma prova de conceito para validar o uso da ferramenta proposta. Processos e elementos usados na rotina da equipe em estudo são considerados na elaboração dos cenários, visando fundamentar a pertinência da solução no ambiente da empresa. De maneira geral, a validação proposta consiste na modelagem de um sistema fictício junto à simulação de cenários possíveis em equipes de desenvolvimento de Software, relatando os casos em que o uso da customização implementada simplifica ou aperfeiçoa as tarefas dos executores do projeto.

Para a modelagem de requisitos do sistema, foram utilizados os módulos *Use Case Model* e *Requirements Model* do Enterprise Architect (EA) com aplicação da técnica de análise de casos de uso. A instância do Agilo For Trac usada na hospedagem da customização foi configurada dentro do ambiente da empresa de estudo, incluindo-se nela *plugins* e *add-ons* usados nos diversos processos estabelecidos pelo setor de qualidade interno. Para a divisão de tarefas do projeto, foi considerado o modelo de desenvolvimento iterativo e incremental, descrito em detalhes na Seção 2.

As seções a seguir apresentam a modelagem do sistema dentro do EA, evidenciando como as regras e casos de uso são visualizados e mantidos pelos analistas da empresa. Traçando um paralelo, será exibido como essas informações são dispostas dentro do Agilo após o uso da customização. E, por fim, serão analisados dois cenários pertinentes para o problema em questão: o de mudanças na especificação e a fase de alinhamento dos requisitos com desenvolvedores.

4.1 WebSPA

O sistema modelado compreende um gerenciador para clínicas de estética, auxiliando no agendamento de visitas e controle das atividades internas do estabelecimento. Dentre as funcionalidades modeladas, destaca-se a emissão de relatórios de desempenho dos profissionais contratados e a solicitação de agendamentos de visita. Funcionalidades básicas, como *login* e sistema de permissões, foram propositalmente omitidas a fim de se obter objetividade nessa prova de conceito. Três atores foram definidos para o sistema: cliente, profissional e gerente, e os casos de uso foram divididos em duas iterações, com a segunda iteração sendo o foco da análise deste trabalho.

A Figura 4.1 exibe o diagrama de casos de uso para a primeira iteração. Dois UCs (*use cases*) foram criados na primeira iteração para ilustrar como a divisão de iterações é tratada dentro do ambiente do Agilo. Para a segunda iteração, foram designados os UCs das funcionalidades de emissão de relatório e agendamento de visita. O diagrama de casos de uso da segunda iteração é ilustrado no diagrama da Figura 4.2. Foram elaborados UCs para os três atores; os casos de uso da funcionalidade de agendamento foram focados no cliente, enquanto que a modelagem de relatórios se restringiu ao profissional e gerente.

Dentro da ferramenta EA, os casos de uso são modelados no mesmo pacote, sendo a divisão de iterações controlada de duas formas: através do diagrama de casos de uso de cada iteração e da numeração dos identificadores de cada UC. Na Figura 4.3 é ilustrado como a visualização dos casos de uso é modelada no EA.

Figura 4.1 - Diagrama de casos de uso da primeira iteração.

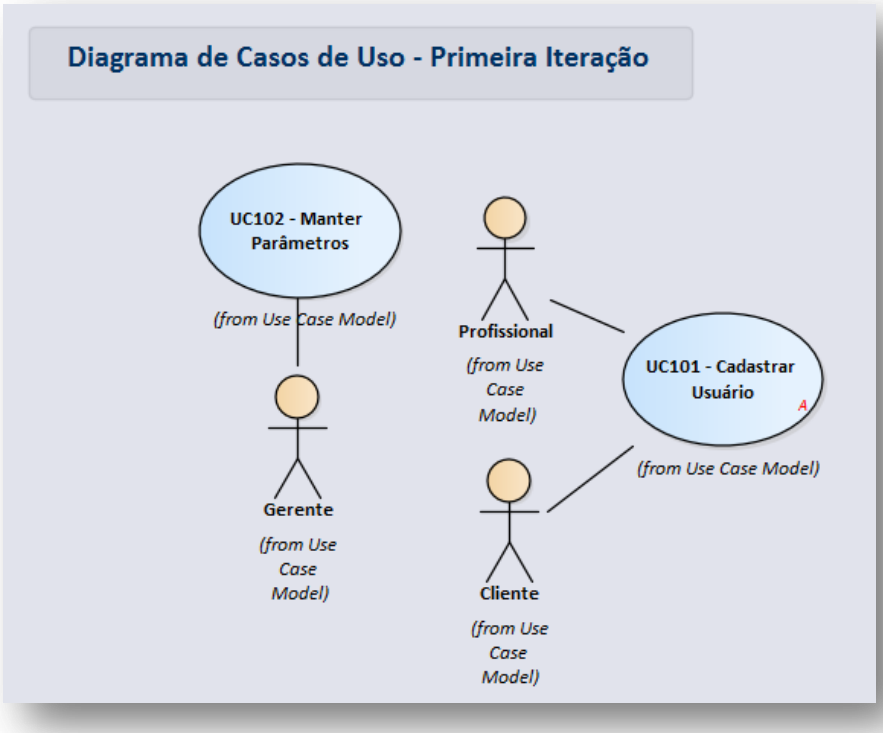


Figura 4.2 - Diagrama de casos de uso da segunda iteração

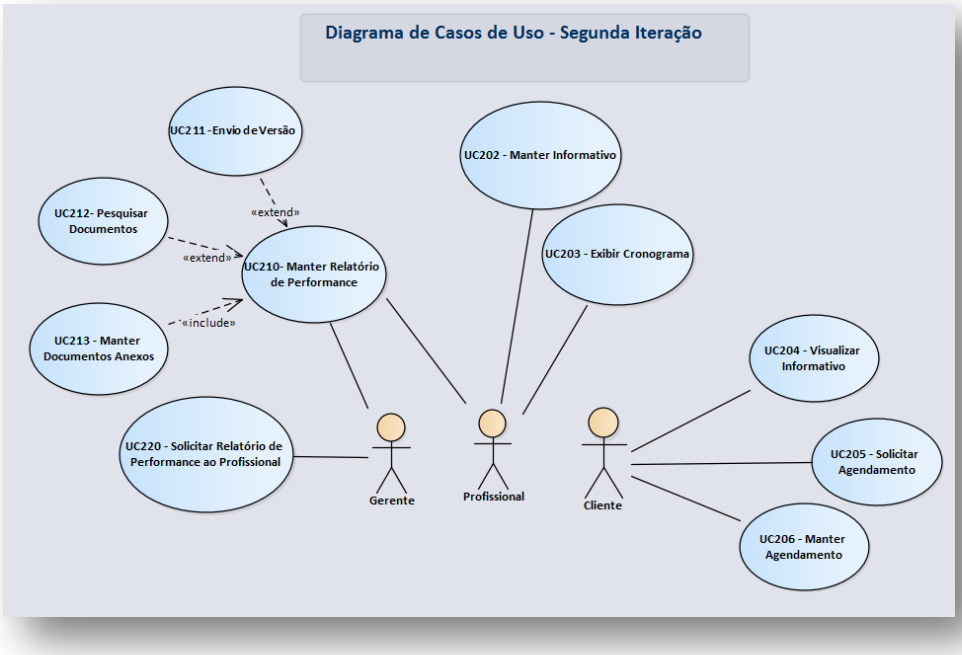
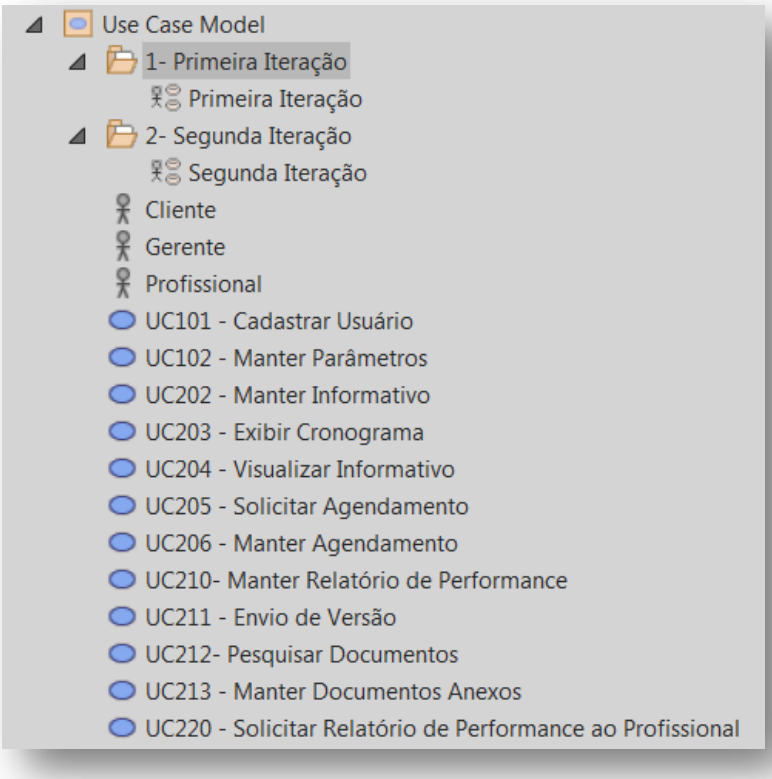


Figura 4.3 - Organização do Use Case Model no EA

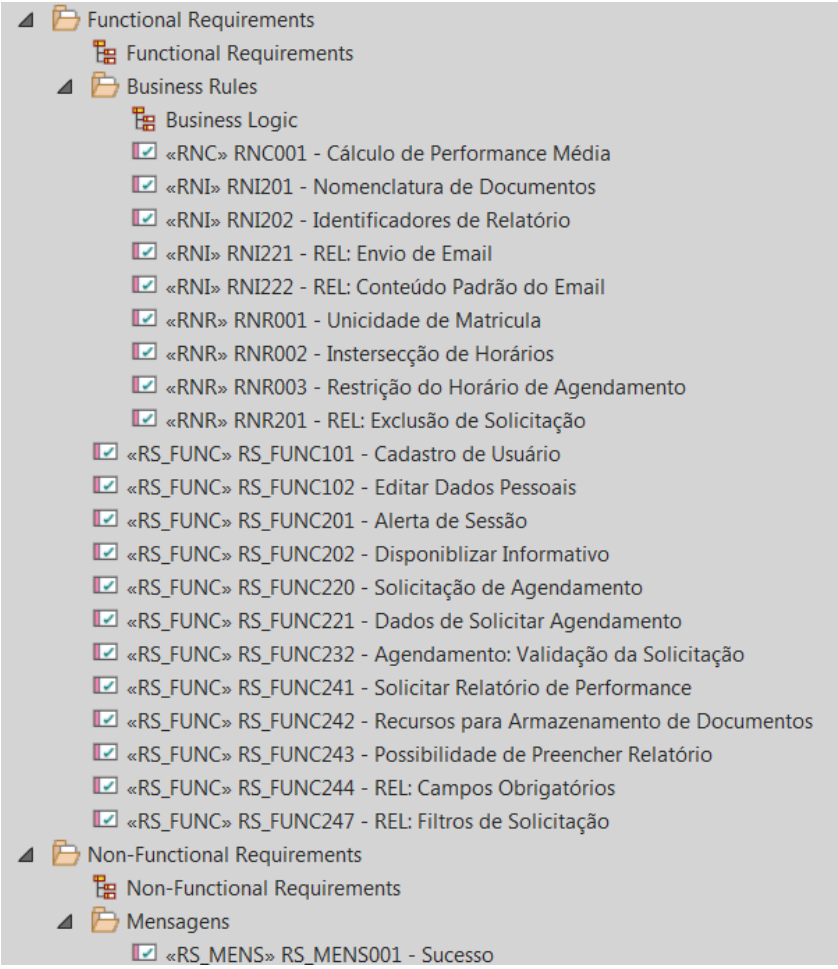


Para a modelagem dos requisitos, foi usado o módulo *Requirements Model*, onde os requisitos funcionais e não funcionais são especificados. Dentro desse grupo, definem-se também as regras de negócio e os requisitos do tipo mensagem. Para nomenclatura e classificação, foi usada a recomendação adotada pela equipe de analistas de requisitos da empresa, explicada na Tabela 4.1. Na Figura 4.4 é ilustrado o Requirements Model e suas subestruturas.

Tabela 4.1 - Tipos de regras de negócio e suas siglas.

Regras de Negócio Formais	**Sigla**
De cálculo	RNC
De restrição	RNR
De inferência	RNI
Habilitadoras de ação	RNH

Figura 4.4 - Organização dos requisitos dentro do Enterprise Architect (EA)

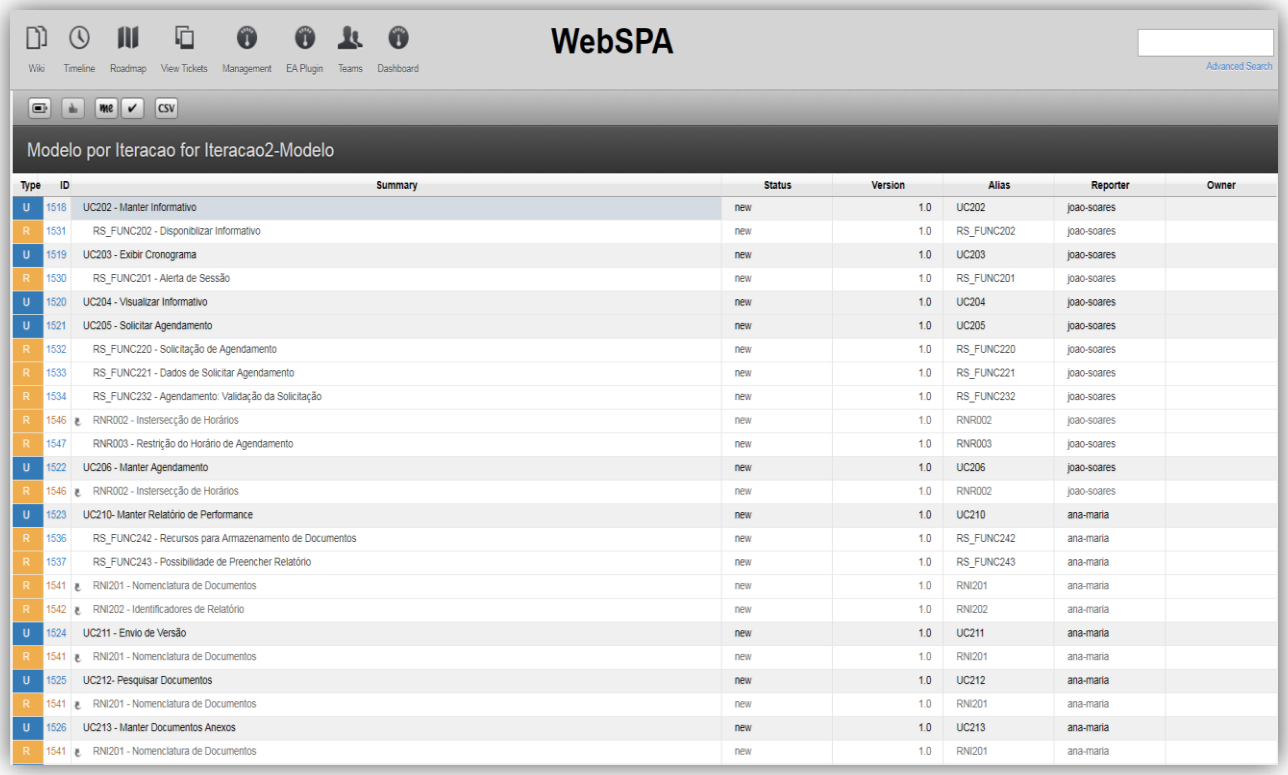


Uma vez modelados e validados os requisitos para a segunda iteração, as tarefas de implementação entram em planejamento e a equipe de desenvolvimento passa a ter conhecimento da modelagem e sua especificação. É nesse ponto que as informações do EA podem ser sincronizadas com o Agilo, adaptando o modelo de requisitos à estrutura de *tickets* da ferramenta de controle de tarefas. Usando as informações contidas no modelo XMI do EA, o *plugin* de integração manterá o modelo de requisitos atualizado no Agilo, criando novos elementos ou atualizando informações de elementos já existentes. É importante ressaltar que o *plugin* é de uso exclusivo dos analistas de requisitos dentro do Agilo. Essa condição é garantida pelo sistema de permissões da ferramenta, personalizado pelo *plugin* de integração.

No ambiente do Agilo, o analista de requisitos é responsável pela criação da *milestone* que identificará os *tickets* do modelo da iteração a ser importada. Para este caso, foi criada a *milestone* **Iteracao2-Modelo**. Usando a funcionalidade de *backlogs* do Agilo, é possível criar

visões personalizadas da estrutura dos *tickets* dentro da ferramenta, simplificando a visualização por parte dos usuários. Esse painel pode ser configurado para filtrar tipos específicos de *tickets* e exibir campos de interesse. A visualização final do modelo criado para a segunda iteração pode ser conferida na Figura 4.5.

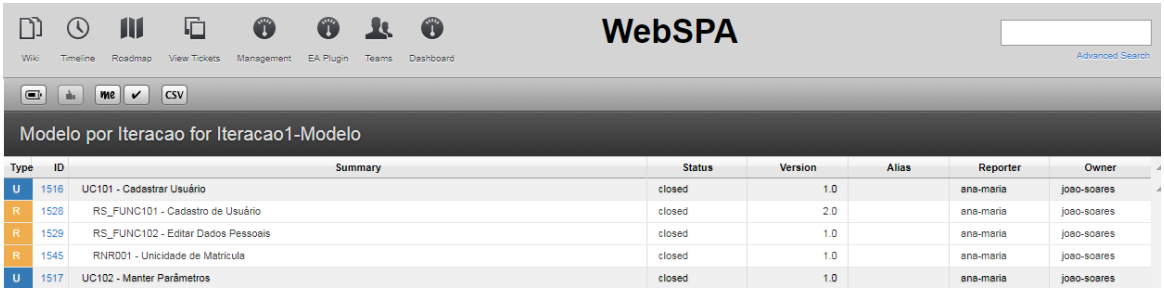
Figura 4.5 - Visualização personalizada dos tickets criados na importação do modelo.



Type	ID	Summary	Status	Version	Alias	Reporter	Owner
U	1518	UC202 - Manter Informativo	new	1.0	UC202	joao-soares	
R	1531	RS_FUNC202 - Disponibilizar Informativo	new	1.0	RS_FUNC202	joao-soares	
U	1519	UC203 - Exibir Cronograma	new	1.0	UC203	joao-soares	
R	1530	RS_FUNC201 - Alerta de Sessão	new	1.0	RS_FUNC201	joao-soares	
U	1520	UC204 - Visualizar Informativo	new	1.0	UC204	joao-soares	
U	1521	UC205 - Solicitar Agendamento	new	1.0	UC205	joao-soares	
R	1532	RS_FUNC220 - Solicitação de Agendamento	new	1.0	RS_FUNC220	joao-soares	
R	1533	RS_FUNC221 - Dados de Solicitar Agendamento	new	1.0	RS_FUNC221	joao-soares	
R	1534	RS_FUNC232 - Agendamento: Validação da Solicitação	new	1.0	RS_FUNC232	joao-soares	
R	1546	RNR002 - Instersecção de Horários	new	1.0	RNR002	joao-soares	
R	1547	RNR003 - Restrição do Horário de Agendamento	new	1.0	RNR003	joao-soares	
U	1522	UC206 - Manter Agendamento	new	1.0	UC206	joao-soares	
R	1546	RNR002 - Instersecção de Horários	new	1.0	RNR002	joao-soares	
U	1523	UC210- Manter Relatório de Performance	new	1.0	UC210	ana-maria	
R	1536	RS_FUNC242 - Recursos para Armazenamento de Documentos	new	1.0	RS_FUNC242	ana-maria	
R	1537	RS_FUNC243 - Possibilidade de Preencher Relatório	new	1.0	RS_FUNC243	ana-maria	
R	1541	RNI201 - Nomenclatura de Documentos	new	1.0	RNI201	ana-maria	
R	1542	RNI202 - Identificadores de Relatório	new	1.0	RNI202	ana-maria	
U	1524	UC211 - Envio de Versão	new	1.0	UC211	ana-maria	
R	1541	RNI201 - Nomenclatura de Documentos	new	1.0	RNI201	ana-maria	
U	1525	UC212- Pesquisar Documentos	new	1.0	UC212	ana-maria	
R	1541	RNI201 - Nomenclatura de Documentos	new	1.0	RNI201	ana-maria	
U	1526	UC213 - Manter Documentos Anexos	new	1.0	UC213	ana-maria	
R	1541	RNI201 - Nomenclatura de Documentos	new	1.0	RNI201	ana-maria	

Nota-se que a hierarquia dos tickets é evidenciada por meio de recuo de texto, e tanto a versão quanto o relator dos elementos podem ser acompanhados por esse painel. Também, é possível identificar os requisitos usados por mais um caso de uso, funcionalidade útil para os desenvolvedores rastrearem os efeitos de mudanças em requisitos já implementados. Na organização do Agilo, os painéis são divididos por *milestones* e, portanto, informações dos modelos de outras iterações podem ser acompanhadas de forma idêntica. A Figura 4.6 contém a visão *backlog* da primeira iteração do sistema WebSPA.

Figura 4.6 - Visualização dos tickets do modelo da primeira iteração.



Type	ID	Summary	Status	Version	Alias	Reporter	Owner
U	1516	UC101 - Cadastrar Usuário	closed	1.0		ana-maria	joao-soares
R	1528	RS_FUNC101 - Cadastro de Usuário	closed	2.0		ana-maria	joao-soares
R	1529	RS_FUNC102 - Editar Dados Pessoais	closed	1.0		ana-maria	joao-soares
R	1545	RNR001 - Unicidade de Matricula	closed	1.0		ana-maria	joao-soares
U	1517	UC102 - Manter Parâmetros	closed	1.0		ana-maria	joao-soares

4.2 Analisando um caso de uso

Para uma análise no nível de casos de uso, foi escolhido o UC220 como caso de uso de estudo desta seção. O UC220, nomeado de "Solicitar Relatório de Performance ao Profissional", especifica funcionalidade que irá realizar, consultar e cancelar solicitações de preenchimento do Relatório de Performance para um profissional de escolha do gerente. Nessa abordagem, os relatórios de performance reúnem informações fornecidas pelos profissionais e métricas retiradas dos dados registrados na base do WebSPA.

No sistema, as solicitações são gerenciadas em tela específica, onde é possível fazer as operações descritas no caso de uso, com cada uma dessas operações sendo especificada como um fluxo de uso do caso de uso. A Figura 4.7 e a Figura 4.8 ilustram como as informações do caso de uso são dispostas no EA. É relevante ressaltar que o relacionamento entre caso de uso e requisito é feito nos fluxos de execução, onde cada passo referencia os requisitos que modelam seu comportamento. Portanto, é importante para o desenvolvedor que essa estrutura seja respeitada.

Figura 4.7 - Informações gerais do UC220 dispostas no EA.

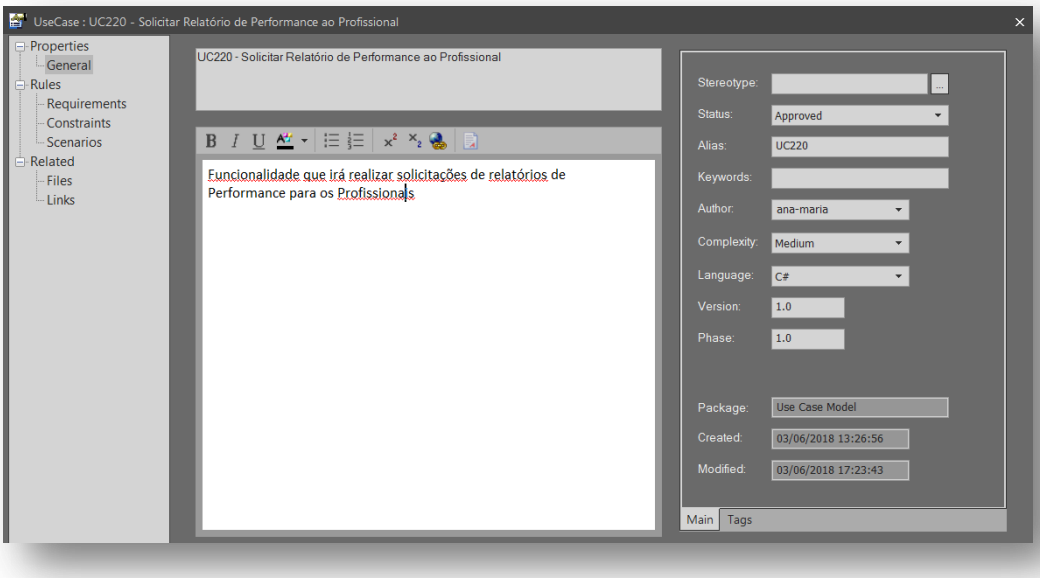
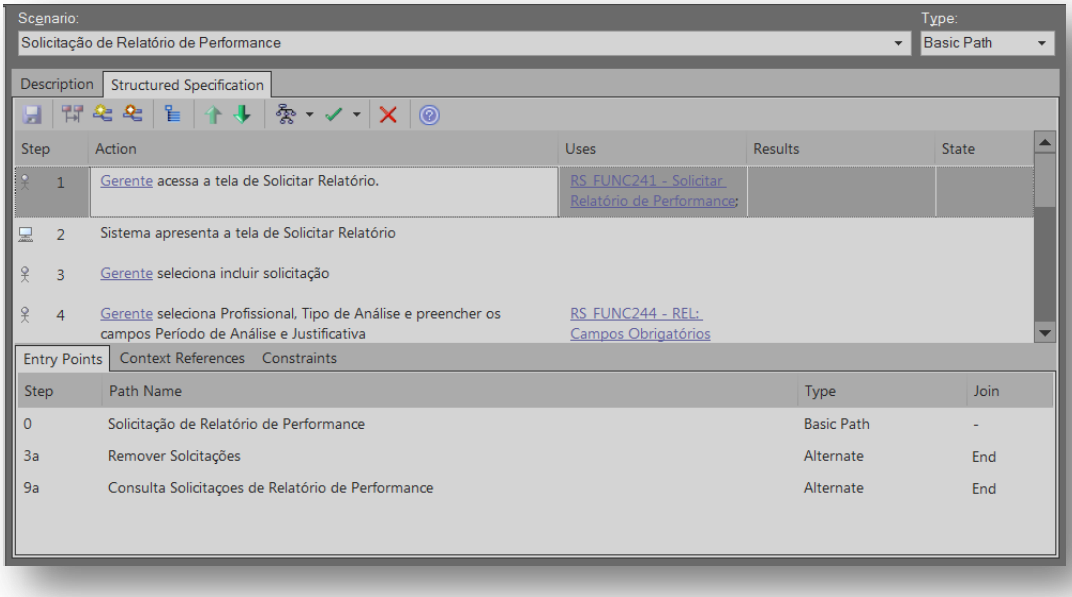
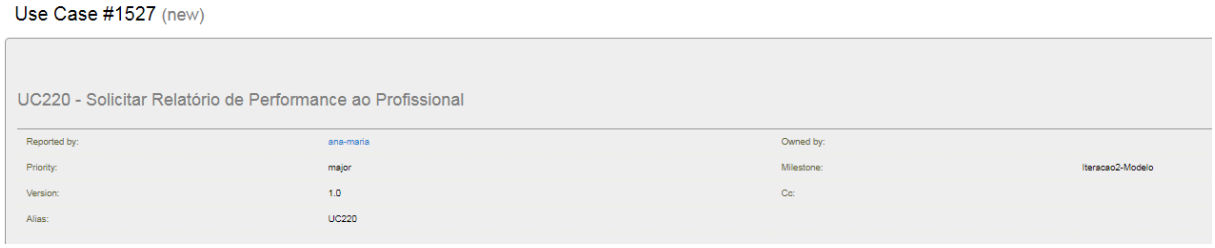


Figura 4.8 - Estrutura de fluxos do UC220 no EA. Na imagem, é detalhado apenas o fluxo básico de execução.



Após a migração, as informações relevantes para o desenvolvimento são organizadas dentro da estrutura do *ticket* de destino, dando ao desenvolvedor total independência do EA para as atividades básicas de desenvolvimento. A Figura 4.9 ilustra como as informações são dispostas no *ticket* do Agilo.

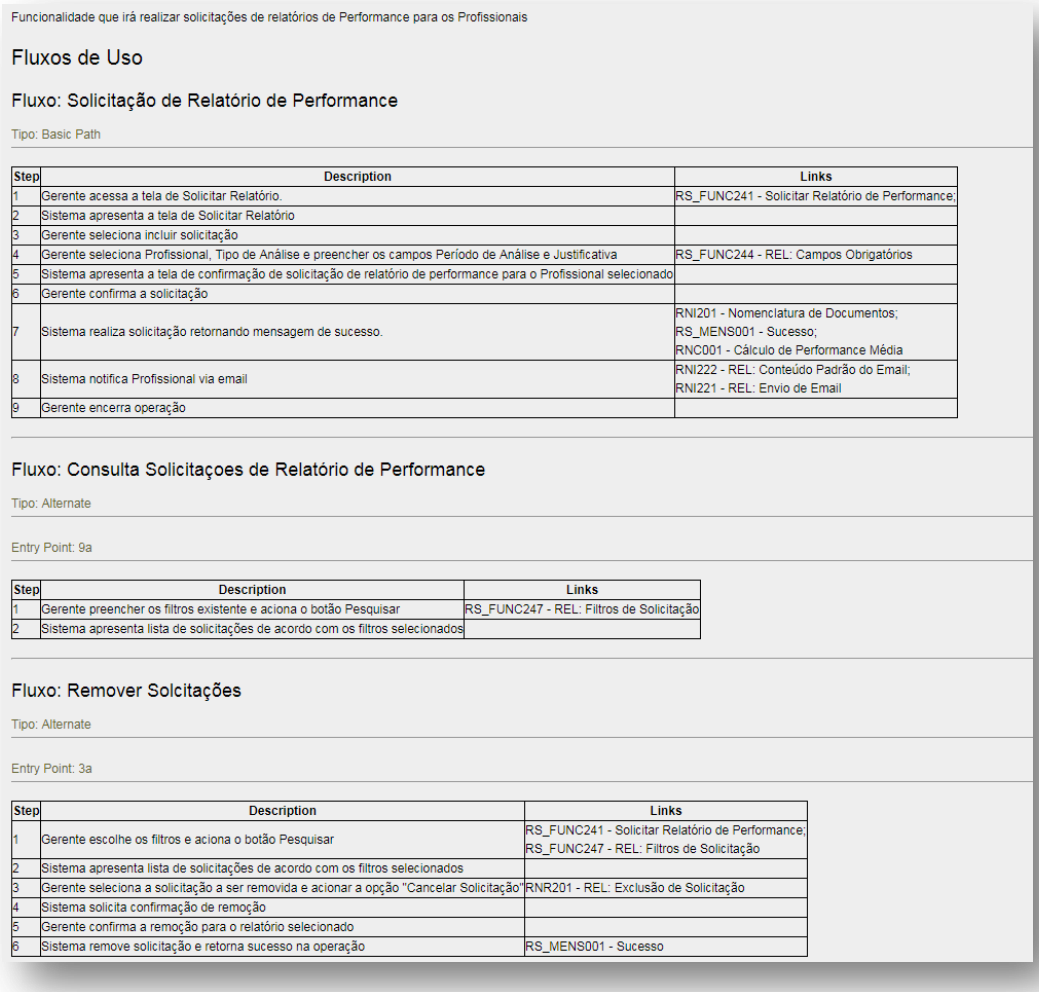
Figura 4.9 - Informações básicas no ticket de Caso de Uso criado no Agilo.



As informações comportamentais do caso de uso (definição e fluxos de uso) são expostas no campo *description* do *ticket*, onde é possível construir estrutura semelhante à existente no EA. A estrutura completa do campo pode ser conferida na Figura 4.10, onde os três fluxos e os respectivos *links* são especificados.

Uma vez que os links são estruturados no passo-a-passo de cada fluxo, as informações detalhadas de cada requisito referenciado podem ser acessadas por meio da seção de referências do *ticket* (Figura 4.11). Nesta seção, os tickets referenciados pelo ticket de caso de uso são listados junto ao *alias* de cada um, permitindo ao desenvolvedor acesso simplificado às informações necessárias para entendimento dos fluxos de uso.

Figura 4.10 - Estrutura de fluxos de uso construídas no ticket do UC220.



Funcionalidade que irá realizar solicitações de relatórios de Performance para os Profissionais

Fluxos de Uso

Fluxo: Solicitação de Relatório de Performance

Tipo: Basic Path

Step	Description	Links
1	Gerente acessa a tela de Solicitar Relatório.	RS_FUNC241 - Solicitar Relatório de Performance;
2	Sistema apresenta a tela de Solicitar Relatório	
3	Gerente seleciona incluir solicitação	
4	Gerente seleciona Profissional, Tipo de Análise e preencher os campos Período de Análise e Justificativa	RS_FUNC244 - REL: Campos Obrigatórios
5	Sistema apresenta a tela de confirmação de solicitação de relatório de performance para o Profissional selecionado	
6	Gerente confirma a solicitação	
7	Sistema realiza solicitação retornando mensagem de sucesso.	RNI201 - Nomenclatura de Documentos; RS_MENS001 - Sucesso; RNC001 - Cálculo de Performance Média
8	Sistema notifica Profissional via email	RNI222 - REL: Conteúdo Padrão do Email; RNI221 - REL: Envio de Email
9	Gerente encerra operação	

Fluxo: Consulta Solicitaçoes de Relatório de Performance

Tipo: Alternate

Entry Point: 9a

Step	Description	Links
1	Gerente preencher os filtros existente e aciona o botão Pesquisar	RS_FUNC247 - REL: Filtros de Solicitação
2	Sistema apresenta lista de solicitações de acordo com os filtros selecionados	

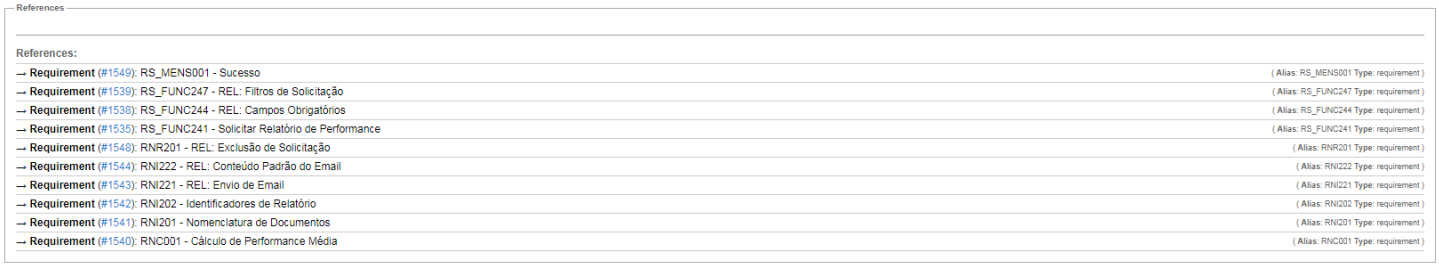
Fluxo: Remover Solcitações

Tipo: Alternate

Entry Point: 3a

Step	Description	Links
1	Gerente escolhe os filtros e aciona o botão Pesquisar	RS_FUNC241 - Solicitar Relatório de Performance; RS_FUNC247 - REL: Filtros de Solicitação
2	Sistema apresenta lista de solicitações de acordo com os filtros selecionados	
3	Gerente seleciona a solicitação a ser removida e acionar a opção "Cancelar Solicitação"	RNR201 - REL: Exclusão de Solicitação
4	Sistema solicita confirmação de remoção	
5	Gerente confirma a remoção para o relatório selecionado	
6	Sistema remove solicitação e retorna sucesso na operação	RS_MENS001 - Sucesso

Figura 4.11 – Seção de referências do *ticket* do UC220.



References

References:

→ **Requirement** (#1549): RS_MENS001 - Sucesso (Alias: RS_MENS001 Type: requirement)
→ **Requirement** (#1539): RS_FUNC247 - REL: Filtros de Solicitação (Alias: RS_FUNC247 Type: requirement)
→ **Requirement** (#1538): RS_FUNC244 - REL: Campos Obrigatórios (Alias: RS_FUNC244 Type: requirement)
→ **Requirement** (#1535): RS_FUNC241 - Solicitar Relatório de Performance (Alias: RS_FUNC241 Type: requirement)
→ **Requirement** (#1548): RNR201 - REL: Exclusão de Solicitação (Alias: RNR201 Type: requirement)
→ **Requirement** (#1544): RNI222 - REL: Conteúdo Padrão do Email (Alias: RNI222 Type: requirement)
→ **Requirement** (#1543): RNI221 - REL: Envio de Email (Alias: RNI221 Type: requirement)
→ **Requirement** (#1542): RNI202 - Identificadores de Relatório (Alias: RNI202 Type: requirement)
→ **Requirement** (#1541): RNI201 - Nomenclatura de Documentos (Alias: RNI201 Type: requirement)
→ **Requirement** (#1540): RNC001 - Cálculo de Performance Média (Alias: RNC001 Type: requirement)

4.3 Cenário 01: Mudanças no modelo EA

Com a ambientação descrita e definida nas seções anteriores, é hora de simular alguns cenários de uso da ferramenta proposta neste trabalho. Essa seção seguirá a ordem da Seção 4: descrição da ação seguida de imagens que ilustram as alterações dentro da plataforma do Agilo. A situação descrita nesta seção considera as seguintes premissas:

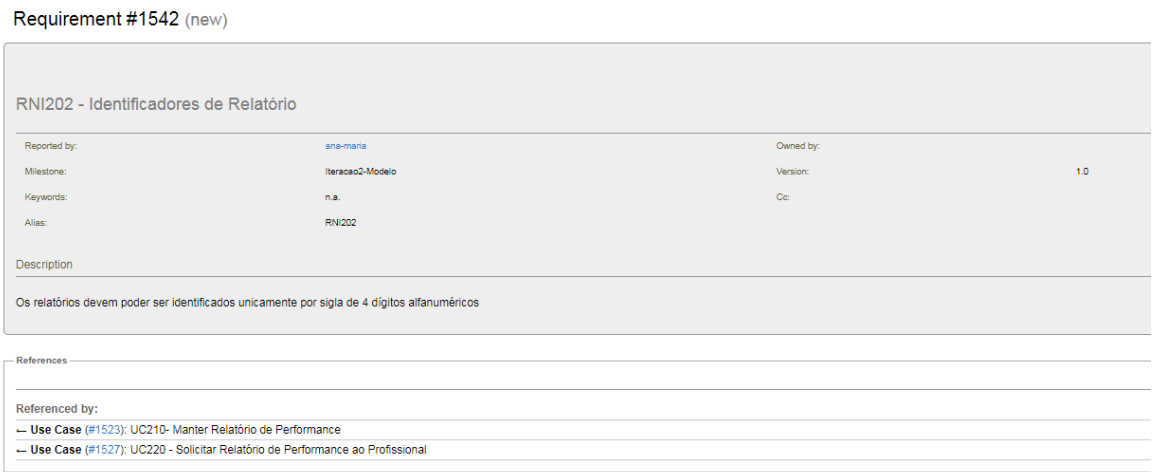
i. O modelo em fase de desenvolvimento é o da segunda iteração;
ii. A sincronização inicial com o Agilo já foi feita e, portanto, os desenvolvedores possuem acesso aos *tickets* do modelo;
iii. O caso de uso analisado será o UC220 - Solicitar Relatório de Performance ao Profissional.

Para a simulação, o time será dividido da seguinte maneira:

- Analistas de Requisitos: Ana e João;
- Desenvolvedor: Rodrigo;
- Gerente do Projeto: Eduardo.

No cenário 01, Rodrigo é o responsável pela codificação do UC220 no sistema WebSPA. Durante a fase de entendimento da especificação, Rodrigo notou problemas com a estruturação do caso de uso. A RNI202 é referenciada no ticket do UC220, porém não há nenhuma associação dela nos passos dos fluxos de uso. A informações da RNI 202 estão ilustradas na Figura 4.12.

Figura 4.12 - Ticket da RNI 202.



Com a informação do analista de requisitos responsável pela especificação do UC220 (ana-maria), Rodrigo alertou a Ana sobre a falha através de comentário no *ticket*. Ana, notificada pelo Agilo via email, tomou conhecimento da situação e pôde endereçar o problema de forma apropriada. Por estar impossibilitada durante aquele período, Ana delegou a tarefa de correção do modelo para João (joao-soares) ao atribuir a propriedade do ticket a ele. Também, ao adicionar o usuário de Eduardo em cópia (*cc*), gerente do projeto, ela o manteve atualizado acerca dos ajustes na especificação.

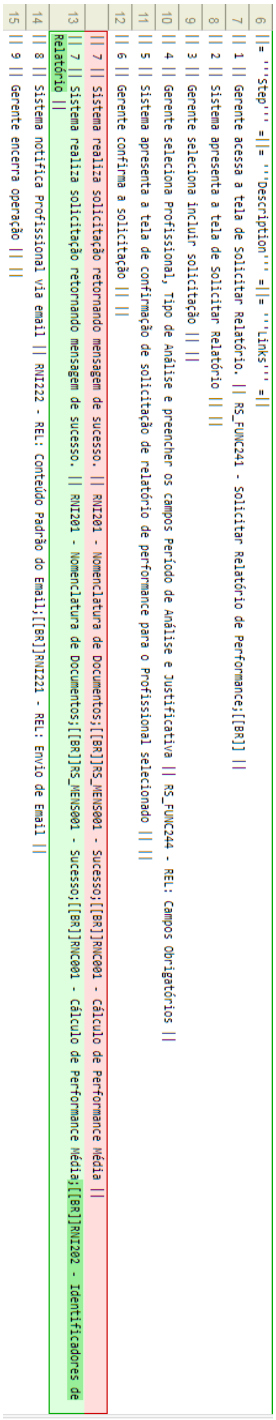
Com a tarefa delegada, João realizou os devidos ajustes no Enterprise Architect (EA) e efetuou a sincronização com o Agilo, com todos os passos registrados no histórico do *ticket* (Figura 4.13).

Figura 4.13 - Histórico de mudanças do ticket do UC220 logo após sincronização.

Rodrigo é, então, notificado pelo Agilo das alterações no ticket e pode retomar o processo de entendimento do caso de uso. Contudo, uma vez que as mudanças foram efetivadas, é importante saber **o que** foi modificado na estrutura do caso de uso. Para isso, Rodrigo acessa a funcionalidade de diferenciação do Agilo para o campo de descrição (*hyperlink* "diff" na Figura 4.13). Mesmo sem Ana ou João explicitarem onde a RNI202 foi

referenciada nos fluxos de uso, Rodrigo possui acesso a essa informação (Figura 4.14) e, por conseguinte, pode dar início à codificação do UC220.

Figura 4.14 – Diferenciação do campo *description* evidenciando as mudanças realizadas pelo analista João no UC220.



4.4 Cenário 02: Colaboração no entendimento do modelo

O segundo cenário desta prova de conceito considera, inicialmente, as mesmas premissas e personagens descritos na Seção 4.3. Nesse cenário, será ilustrado como o controle de comentários de *tickets* do Agilo pode auxiliar desenvolvedor e analista de requisitos no alinhamento da especificação em implementação. A RNC001 e seu relacionamento com o UC220 serão o foco desta etapa, elucidando a situação onde existem dúvidas a respeito de detalhes da codificação do comportamento.

Após início da codificação do caso de uso, o desenvolvedor Rodrigo se depara com uma dúvida que impede o andamento da implementação. A RNC001 define como deve ser calculada a variável **performance média**, presente no Relatório de Performance de um profissional. A descrição da RNC001 pode ser conferida na Figura 4.15. No momento da implementação da regra, contudo, Rodrigo não soube inferir como o total de horas trabalhadas de um profissional seria obtido da base.

Seguindo o processo estabelecido pela equipe, o desenvolvedor registrou sua dúvida no *ticket* referente à RNC001, notificando o relator (João). Agora ciente da dúvida, João repassa o questionamento para Ana, uma vez que ela foi responsável pela modelagem do modelo de dados do sistema. Após a notificação, Ana acessa o Agilo e responde o questionamento feito por Rodrigo. Com a informação, Rodrigo finaliza a implementação da regra e fecha o *ticket* do modelo. O histórico de interações é mantido no *ticket* e pode ser visualizado na Figura 4.16.

Figura 4.15 - Descrição do ticket da RNC001.

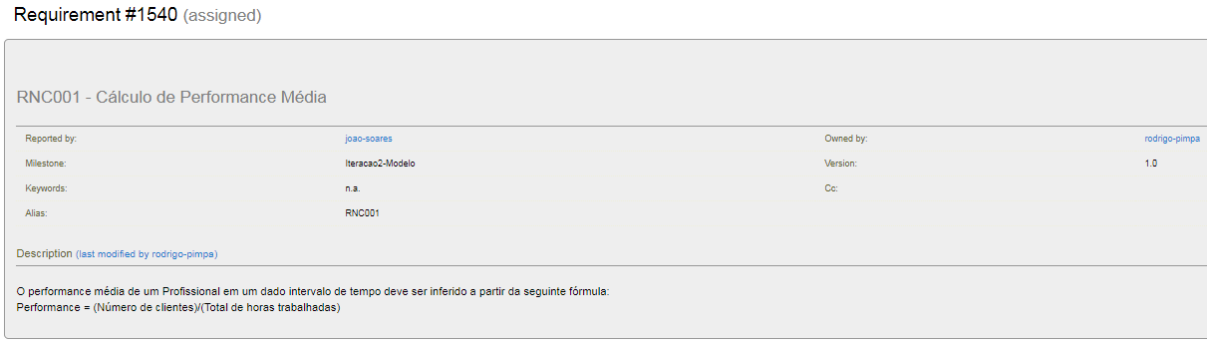
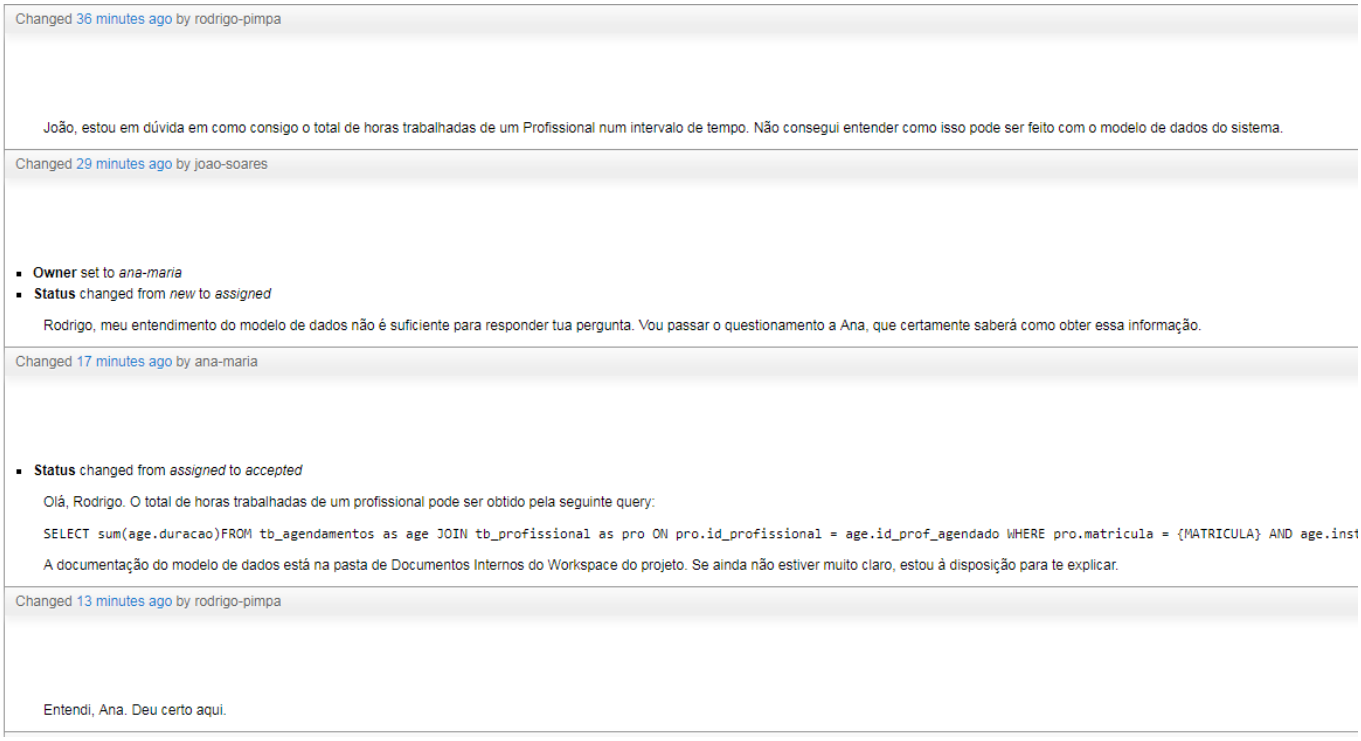


Figura 4.16 - Histórico de interação do ticket da RNC001.



5 Conclusões e Trabalhos Futuros

Neste trabalho, foi proposta uma solução tecnológica para promover a integração da equipe de desenvolvimento no processo gerência de requisitos de uma empresa de desenvolvimento de sistemas industriais. Por meio da customização da ferramenta de controle de tarefas Agilo for Trac, foi implementado um *plugin* de integração para uso interno da empresa com a ferramenta de modelagem Enterprise Architect. Elementos do modelo de requisitos, como casos de uso e regras funcionais foram adaptados à estrutura de *tickets* do Agilo, possibilitando a participação dos desenvolvedores no processo de gerência de requisitos da empresa.

5.1 Contribuições

A customização implementada neste trabalho pode ser de grande relevância para a gerência de requisitos utilizada na empresa, visto que pretende reduzir a possibilidade de problemas de harmonização entre os desenvolvedores e o modelo de requisitos. Como consequência, as chances de haver defasagem entre a especificação e a implementação do sistema podem ser minimizadas, evitando custos de correções para o projeto. Além disso, o processo de gerência dos requisitos da empresa pode ser refinado por meio do uso da ferramenta Agilo for Trac.

5.2 Trabalhos Futuros

Quanto às evoluções da ferramenta implementada, alguns aspectos são passíveis de melhoria. Dentre eles, a concepção de novos fluxos de uso específicos para os tipos de *tickets* criados para a solução (*Use Case* e *Requirement*), incorporando outros processos de manutenção dos requisitos utilizando o Agilo for Trac. Outra melhoria possível é a automatização do processo de sincronização da ferramenta, atualmente manual. Isto é, implementar a detecção de mudanças e sincronização automática do modelo, sem exigir a participação direta do analista de requisitos, o que pode agregar ainda mais benefícios ao uso do *plugin*.

Quanto à avaliação dos impactos da proposta, é cabível um estudo empírico no ambiente de desenvolvimento da empresa, incorporando a ferramenta em projetos reais da

equipe e realizando a coleta de métricas e indicadores que ajudem a evidenciar e quantificar as melhorias propostas neste trabalho.

Bibliografia

[1] I. Sommerville, Software Engineering, Ninth Edition, Pearson, 2011.

[2] A. Hussain, F. Kamal e E. Mkpojiogu, “The Role of Requirements in the Success or Failure of Software,” *EconJournals,* 2016.

[3] N. Nurmuliani, D. Zowghi e S. Fowell, “Analysis of Requirements Volatility during Software Development Life Cycle,” *IEEE,* 2004.

[4] IEEE Xplore, “Systems and software engineering — Vocabulary. ISO/IEC/IEEE 24765:2017(E).,” *IEEE,* p. 536, 2017.

[5] K. Pohl e C. Rupp, Requirements Engineering Fundamentals, 2nd ed., Santa Barbara, CA: Rocky Nook Inc., 2015.

[6] The Standish Group, "Chaos Report," The Standish Group, 2006.

[7] M. Dawson e E. Rahim, “Integrating Software Assurance into the Software Development Life Cycle (SDLC),” *Journal of Information Systems Technology and Planning,* pp. 49-53, 2010.

[8] R. S. Pressman, Software Engineering Practitioner's Approach, 7th ed., New York, NY: McGraw-Hill, 2010.

[9] M. G. Christel e K. C. Kang, “Issues in Requirements Elicitation,” 1992.

[10] OMG, “Unified Modeling Language Specification, v. 2.5.1,” 2017.

[11] K. Beck, “Embracing Change with Extreme Programming,” IEEE, 1999.

[12] M. Fagan, “Advances in Software Inspections,” *IEEE Trans. Software Engineering,* vol. 12, n. 6, Julho 1986.

[13] Software Engineering Institute, "CMMI for Systems Engineering/Software Engineering," CMMI Staged Representation, 2000.

[14] L. Westfall, “Bidirectional Requirements Traceability,” *The Westfall Team,* 2006.

[15] Carnegie Mellon University, “CMMI for Development, Version 1.3,” 2010.

[16] J. Dalton, “What is CMMI?,” 2017. [Online]. Available: https://broadswordsolutions.com/what-is-cmmi/. [Acesso em 3 Junho 2018].

[17] Sparx Systems, "Enterprise Architect 14 User Guide," 2018. [Online]. Available: http://www.sparxsystems.com/enterprise_architect_user_guide/14.0/index. [Accessed 03 Maio 2018].

[18] J. Hunnicutt e R. Millar, “Realizing CMMI using Enterprise Architect and UML for Process Improvement,” [Online]. Available: http://www.sparxsystems.com/resources/whitepapers/.

[19] OMG, “Meta Object Facility Core Specification, v. 2.5.1,” 2016.

[20] OMG, “XML Metadata Interchange (XMI) Specification, v. 2.5.1,” 2015.

[21] S. Brodsky, “XMI Opens Application Interchange,” IBM, 1999.

[22] C. Larman e V. Basili, “Iterative and Incremental Development: A Brief History,” 2003.

[23] V. R. Basili e A. J. Turner, “Iterative Enhancement: A Pratical Technique for Software Development,” *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili (2005),* p. p. 28, 1979.

[24] “Globalteckz ERP & eCommerce Experts,” globalteckz, [Online]. Available: https://www.globalteckz.com/. [Acesso em 10 06 2018].

[25] Edgewall Software, "The Trac User and Administration Guide," [Online]. Available: https://trac.edgewall.org/wiki/TracGuide. [Accessed 20 Maio 2018].

[26] “Agilo for trac - Documentation,” [Online]. Available: http://www.agilofortrac.com/documentation/. [Acesso em 13 Maio 2018].

Apêndice A - Código: Componente do *plugin*

```
from trac.core import *
from trac.web.chrome import INavigationContributor, ITemplateProvider, add_stylesheet,
IEnvironmentSetupParticipant
from trac.web.main import IRequestHandler
from trac.util.html import html
from trac.db import DatabaseManager
from pkg_resources import resource_filename

from process_xmi import *
from model_generation import *
from db_model import EAElementRepository, EALinkRepository

PLUGIN_NAME = 'TracEA'
PLUGIN_VERSION = 0.1
PLUGIN_SCHEMA = EAElementRepository.getSchema() + EALinkRepository.getSchema()

class EAPlugin(Component):
    implements(INavigationContributor, IRequestHandler,
ITemplateProvider,IEnvironmentSetupParticipant)

    # INavigationContributor methods
    def get_active_navigation_item(self, req):
        return 'ea'

    def get_navigation_items(self, req):
        yield 'mainnav', 'ea', html.a("EA Plugin", href = (req.base_path + "/ea"))

       # ITemplateProvider methods
    def get_templates_dirs(self):
        return [resource_filename(__name__, "templates")]

    def get_htdocs_dirs(self):
        return [("ea", resource_filename(__name__, "htdocs"))]

    # IRequestHandler methods
    def match_request(self, req):
        return req.path_info == '/ea'

    def pre_process_request(self, req, handler):
        return handler

    def process_request(self, req):
        add_stylesheet(req, "ea/css/icon.css")
        self.filtrarRequest(req)
        listaElementos = []
        reader = Xmi()
        reader.obterUseCases()
        reader.obterRequirements()
        reader.obterReferencias()
        self.reader = reader
        for e in reader.elementos:
            elem = {
                "id": e.id
            }
            listaElementos.append(elem)
        response = {"listaElementos": listaElementos}
        return 'ea.html',response,None
```

```
# IEnvironmentSetupParticipant
def quote(self, identifier):
    return "`%s`" % identifier.replace('`', '``')

def environment_created(self, db):
    self.upgrade_environment()

def environment_needs_upgrade(self, db):
    connector, _ = DatabaseManager(self.env)._get_connector( )
    cursor = db.cursor( )
    for table in PLUGIN_SCHEMA:
        cursor.execute("""
               SELECT COUNT(*) FROM information_schema.tables
               WHERE table_name=%s
               """, [table.name])
        for row in cursor:
            if row[0] == 0:
                return True
    return False

def upgrade_environment(self, db):
    connector, _ = DatabaseManager(self.env)._get_connector( )
    cursor = db.cursor( )

    for table in PLUGIN_SCHEMA:
        cursor.execute("DROP TABLE IF EXISTS " + db.quote(table.name))
        for stmt in connector.to_sql(table):
            cursor.execute(stmt)

    db.commit( )

# Internal Methods

def filtrarRequest(self, req):
    action = req.args.get("action", "view")
    if req.method == "POST":
        if action == "execute":
            gerarModeloTrac(self.env, self.reader)
```

Apêndice B - Código: Módulo XMI

```
class Xmi(object):
    ns = {'xmi': "http://schema.omg.org/spec/XMI/2.1",
          'uml': "http://schema.omg.org/spec/UML/2.1"}
    elementos = []
    referencias = []
    filtroUC = ["UseCase"]
    filtroRE = ["Requirement"]
    def __init__(self):
        self.UCTree = ET.parse(getUCXMI())
        self.BusinessTree = ET.parse(getReqXMI())
        UCRoot = self.UCTree.getroot()
        BusinessRoot = self.BusinessTree.getroot()

        uc_extension = UCRoot.find('xmi:Extension', self.ns)
        self.xmi_uc_elements = uc_extension.find('elements').findall('element')
        self.xmi_uc_connectors = uc_extension.find('connectors').findall('connector')

        re_extension = BusinessRoot.find('xmi:Extension', self.ns)
        self.xmi_re_elements = re_extension.find('elements').findall('element')
    def obterUseCases(self):
        for obj in self.xmi_uc_elements:
            objType = obj.find("properties").get("sType")
            if objType in self.filtroUC:
                elemento = Elemento()
                elemento.type = objType
                elemento.id = obj.get("{%s}idref" % self.ns['xmi'])
                elemento.name = obj.get("name")
                elemento = self.obterScenarios(obj, elemento)
                elemento.documentation = obj.find("properties").get("documentation")
                elemento.author = obj.find("project").get("author")
                elemento.version = obj.find("project").get("version")
                self.elementos.append(elemento)
    def obterScenarios(self, ucXml, elemento):
        if ucXml.find("EAModel.scenario") is None:
            return elemento
        allScenarios = ucXml.find("EAModel.scenario").findall("EAScenario")
        for scenario in allScenarios:
            cenario = Cenario()
            cenario.id = scenario.get("{%s}id" % self.ns['xmi'])
            cenario.name = scenario.get("name")
            cenario.type = scenario.get("type")
            if scenario.find("EAScenarioContent") is None:
                continue
            allSteps = scenario.find("EAScenarioContent").find("path").findall("step")
            for step in allSteps:
                passo = Step()
                passo.name = step.get("name")
                passo.level = step.get("level")
                allBranches = step.findall("extension")
                for branch in allBranches:
                    ext = Extension()
                    ext.level = branch.get("level")
                    ext.dest = "EAID_" + branch.get("guid")
                                               .replace("-", "_")
                                               .upper()[1:-1]
                    ext.join = branch.get("join")
                    passo.extensions.append(ext)
                for use in step.get("uses").split(';'):
```

```
                passo.uses.append(use.strip().replace("\r", " ").replace("\n", " "))
            cenario.steps.append(passo)
        elemento.scenarios.append(cenario)
    return elemento
def obterRequirements(self):
    for obj in self.xmi_re_elements:
        objType = obj.find("properties").get("sType")
        if objType in self.filtroRE:
            elemento = Elemento()
            elemento.type = objType
            elemento.id = obj.get("{%s}idref" % self.ns['xmi'])
            elemento.name = obj.get("name")
            elemento.documentation = obj.find("properties").get("documentation")
            elemento.author = obj.find("project").get("author")
            elemento.version = obj.find("project").get("version")
            self.elementos.append(elemento)
def obterReferencias(self):
    for con in self.xmi_uc_connectors:
        xmiSource = con.find("source")
        tipoSource = xmiSource.find("model").get("type")
        xmiTarget = con.find("target")
        tipoTarget = xmiTarget.find("model").get("type")
        filtroTipo = list(set().union(self.filtroUC, self.filtroRE))
        if (tipoSource in filtroTipo) and (tipoTarget in filtroTipo):
            referencia = Referencia()
            referencia.id = con.get("{%s}idref" % self.ns['xmi'])
            referencia.type = con.find("properties").get("ea_type")
            idSource = xmiSource.get("{%s}idref" % self.ns['xmi'])
            elementoSource = next((e for e in self.elementos
                                   if e.id == idSource), None)
            idTarget = xmiTarget.get("{%s}idref" % self.ns['xmi'])
            elementoTarget = next((e for e in self.elementos
                                   if e.id == idTarget), None)
            if elementoSource is None:
                elementoSource = Elemento()
                elementoSource.id = idSource
                elementoSource.name = "[Ext] " + xmiSource.find("model").get("name")
                elementoSource.type = tipoSource
                self.elementos.append(elementoSource)
            referencia.source = elementoSource

            if elementoTarget is None:
                elementoTarget = Elemento()
                elementoTarget.id = idSource
                elementoTarget.name = "[Ext] " + xmiTarget.find("model").get("name")
                elementoTarget.type = tipoSource
                self.elementos.append(elementoTarget)
            referencia.target = elementoTarget

            self.referencias.append(referencia)
            elementoSource.referencias.append(referencia)
            elementoTarget.referencias.append(referencia)
```

Apêndice C - Código: Repositório das classes

```
from trac.db import Table, Column, DatabaseManager
from ea_model import *

class EAElementRepository(object):
    """Represents a table."""
    schemaName = "ea_element"

    Id = "id_element"
    Ticket = "ticketid"
    Name= "nom_element"
    Status = "nom_status"
    Type = "nom_type"
    Documentation = "dsc_documentation"
    Version = "num_version"
    Author = "nom_author"

    tbSchema = [
        Table(schemaName, key = Id)[
            Column(Id),
            Column(Ticket),
            Column(Name),
            Column(Status),
            Column(Type),
            Column(Documentation),
            Column(Version, type = "decimal"),
            Column(Author),
        ]
    ]
    def getSchema(cls):
        return cls.tbSchema
    getSchema = classmethod(getSchema)

    def Execute(cls, db, env, command, parameters):
        if not db:
            db = env.get_db_cnx()
        cursor = db.cursor()
        cursor.execute(command, parameters)
        row = cursor.fetchone()
        db.commit()
        return row
    Execute = classmethod(Execute)

    def FindByKey(cls, env, id, db=None):
        command = ""
        if id is not None:
            command = """SELECT * FROM ea_element WHERE %s = """ % (cls.Id)
            command = command + """ %s """
            parameters = (id)
        params = cls.Execute(db, env, command, parameters)
        if params is None:
            return None
        result = Elemento()
        result.id = params[0]
        result.ticketid = params[1]
        result.name = params[2]
        result.status = params[3]
        result.type = params[4]
        result.documentation = params[5]
```

```
        result.version = params[6]
        result.author = params[7]
        return result
    FindByKey = classmethod(FindByKey)

    def Insert(cls, env, elemento, db=None):
        command = ""
        if elemento.id is not None:
            command = """INSERT INTO ea_element (%s, %s, %s, %s, %s, %s, %s, %s) """ %
(cls.Id, cls.Ticket, cls.Name, cls.Status, cls.Type, cls.Documentation, cls.Version,
cls.Author)
            command = command + """VALUES (%s, %s, %s, %s, %s, %s, %s, %s) """
            parameters = (elemento.id, elemento.ticketid, elemento.name, elemento.status,
elemento.type, elemento.documentation, elemento.version, elemento.author)
        result = cls.Execute(db, env, command, parameters)
        return result is not None
    Insert = classmethod(Insert)



class EALinkRepository(object):
    """Represents a table."""
    schemaName = "ea_link"

    Id = "id_link"
    Type = "nom_type"
    Source = "id_source"
    Target = "id_target"

    tbSchema = [
        Table(schemaName, key = Id)[
            Column(Id),
            Column(Type),
            Column(Source),
            Column(Target),
        ]
    ]

    def getSchema(cls):
        return cls.tbSchema
    getSchema = classmethod(getSchema)

    def Execute(cls, db, env, command, parameters):
        if not db:
            db = env.get_db_cnx()
        cursor = db.cursor()
        cursor.execute(command, parameters)
        row = cursor.fetchone()
        db.commit()
        return row
    Execute = classmethod(Execute)

    def FindByKey(cls, env, id, db=None):
        command = ""
        if id is not None:
            command = """SELECT * FROM ea_link WHERE %s = """ % (cls.Id)
            command = command + """ %s """
            parameters = (id)
```

```
        params = cls.Execute(db, env, command, parameters)
        if params is None:
                  return False
        return True
    FindByKey = classmethod(FindByKey)

    def Insert(cls, env, ref, db=None):
        command = ""
        if ref.id is not None:
            command = """INSERT INTO ea_link (%s, %s, %s, %s) """ % (cls.Id, cls.Type,
cls.Source, cls.Target)
            command = command + """VALUES (%s, %s, %s, %s) """
            parameters = (ref.id, ref.type, ref.source.ticketid, ref.target.ticketid)
        result = cls.Execute(db, env, command, parameters)
        return result is not None
    Insert = classmethod(Insert)
```