UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ RENATO DA SILVA FILHO

AVALIANDO A RELAÇÃO EXISTENTE ENTRE OS ESCOPOS DE REQUISITOS E CÓDIGO: UM ESTUDO ENVOLVENDO SISTEMAS EM EVOLUÇÃO

MOSSORÓ - RN

2017

JOSÉ RENATO DA SILVA FILHO

AVALIANDO A RELAÇÃO EXISTENTE ENTRE OS ESCOPOS DE REQUISITOS E CÓDIGO: UM ESTUDO ENVOLVENDO SISTEMAS EM EVOLUÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação - associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Profº Dr. Francisco Dantas de Medeiros Neto

MOSSORÓ - RN

2017

S586a SILVA FILHO, JOSÉ RENATO DA .
AVALIANDO A RELAÇÃO EXISTENTE ENTRE OS ESCOPOS DE REQUISITOS E CÓDIGO: UM ESTUDO ENVOLVENDO SISTEMAS EM EVOLUÇÃO / JOSÉ RENATO DA SILVA FILHO. - 2017.
58 f. : il.

Orientador: FRANCISCO DANTAS DE MEDEIROS NETO.
Dissertação (Mestrado) - Universidade Federal Rural do Semi-árido, Programa de Pós-graduação em Ciência da Computação, 2017.

1. Software em Evolução. 2. Requisitos de Software. 3. Escopo. I. MEDEIROS NETO, FRANCISCO DANTAS DE, orient. II. Título.

JOSÉ RENATO DA SILVA FILHO

AVALIANDO A RELAÇÃO EXISTENTE ENTRE OS ESCOPOS DE REQUISITOS E CÓDIGO: UM ESTUDO ENVOLVENDO SISTEMAS EM EVOLUÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM:___ / ___ / _____.

BANCA EXAMINADORA

Dr. Francisco Dantas de Medeiros Neto
Orientador

Dra. Lyrene Fernandes Da Silva
Avaliadora Externa - UFRN

Dra. Karla Darlene Nepomuceno Ramos
Avaliador Interno - UERN

Aos meus pais, irmãs, minha namorada e a toda minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

Agradecimentos

Primeiramente a Deus, por ser o principal responsável por tudo isto que está se realizando.

Aos meus pais, por serem exemplos em minha vida e por terem se esforçado ao máximo me educando da melhor maneira possível, em especial minha mãe (in memoriam), pois mesmo não estando presente fisicamente é minha maior fonte de inspiração como pessoa.

À toda minha família, pelo carinho, apoio e momentos especiais que vivemos juntos, em especial minhas irmãs que contribuíram bastante para minha formação pessoal e profissional.

À minha namorada, pelo companheirismo, pelo apoio constante, pelas palavras de incentivo e pelas cansativas revisões de textos.

Aos meus amigos e colegas que contribuíram direta ou indiretamente para a conclusão dessa etapa da minha vida.

À Uern e Ufersa, aos professores que me instruíram durante todo o decorrer do curso, em particular ao professor Francisco Dantas, pela orientação, dedicação, prestatividade e por todo apoio para o desenvolvimento deste trabalho.

Nossas dúvidas são traidoras e nos fazem perder o que, com frequência, poderíamos ganhar, por simples medo de arriscar.

Resumo

Sistemas de software estão em constantes mudanças durante sua evolução, essas mudanças estão diretamente ligadas com a gestão de requisitos de seus produtos. Quando requisitos são bem gerenciados, eles evoluem de forma mais precisa e os custos de desenvolvimento são reduzidos, isso inclui evitar mudanças desnecessárias no código em evolução. Sabe-se que boa parte das mudanças demandadas durante a evolução do código está associada ao escopo dos seus elementos. Porém, não há estudos que investiguem se variações no escopo dos requisitos estão correlacionadas com variações no escopo das composições de código. O objetivo principal desta dissertação de mestrado é prover aos engenheiros de requisitos meios de monitorar a variação do escopo dos requisitos de software e correlacioná-los com o escopo dos elementos de código em evolução, o qual possui impacto negativo comprovado sobre o número de mudanças demandadas em artefatos de código em evolução. Neste contexto, foi formulado uma métrica para quantificar o escopo de requisitos e com base nela foi desenvolvido um *plug-in*, denominado MeRS, que fornece meios para automatizar esta quantificação. Por meio do MeRS foi conduzido um estudo exploratório com o objetivo de correlacionar os escopos nos níveis de requisitos e código. Para tanto, analisamos a evolução dos requisitos e código, com foco em escopo, de duas aplicações em evolução: NotePad e Mobile Media. Os resultados mostraram que os valores do escopo dos requisitos estão correlacionados com o escopo do código.

Palavras-chave: Software em evolução, Requisitos de Software, Escopo.

Abstract

Software systems are constantly changing during their evolution. These changes are directly linked with the management of their product requirements. When requirements are well managed, they evolve more accurately and development costs are reduced, which includes avoiding unnecessary changes to evolving code. It is known that much of the changes demanded during the evolution of the code is associated with the scope of its elements. However, there are no studies investigating whether variations in the scope of requirements are correlated with variations in the scope of code compositions. The main purpose of this master's thesis is to provide means to requirements engineers to monitor variation in the scope of software requirements and to correlate them with the scope of evolving code elements, which has a proven negative impact on the number of changes demanded in evolving code artifacts. In this context, a metric was formulated to quantify the scope of requirements and based on it a plug-in, called MeRS, was developed. MeRS provides means to automate this quantification. Using MeRS, an exploratory study was conducted to correlate the scopes at the requirements and code levels. To do so, we analyze the evolution of the requirements and code, focusing on the scope of two applications in evolution: NotePad and Mobile Media. The results showed that the requirements scope values are correlate with the scope of the code.

Keywords: Software Evolution, Software Requirement, Scope.

Lista de Figuras

Lista de tabelas

Lista de abreviaturas e siglas

IDE	Integrated Development Environment
LPS	Linha de Produto de Software
MeRS	Measuring Requirement Scope
TI	Tecnologia da Informação

Sumário

1 INTRODUÇÃO

Sistemas de software, mesmo após serem entregues ao usuário, tendem a evoluir por meio de mudanças nos requisitos já implementados ou por meio da implementação de novos requisitos. Segundo Paviotti (2011), a satisfação associada à qualidade de artefatos de software (*p.e.*, código fonte) em evolução está diretamente relacionada com uma boa definição dos requisitos desses produtos. Requisitos bem planejados resultam na geração e evolução de artefatos de software funcionando corretamente e entregues dentro do prazo. Quando bem conduzida, a gestão de requisitos contempla o processo de evolução com maior precisão e barateia custos de desenvolvimento, uma vez que identifica uma maior quantidade de problemas ainda na fase inicial do projeto. Todavia, os desenvolvedores de software se deparam com uma lacuna que precisa ser contornada: a falta de informações de como requisitos podem ser usados para rastrear e, consequentemente, guiar a evolução do código de sistemas em evolução, de modo a minimizar o número de mudanças nos artefatos codificados.

As atividades de controle da evolução dos artefatos de software (*p.e.* casos de uso e código) emergem como principais áreas do ciclo de vida de um sistema computacional (SOMMERVILLE, 2011). Entretanto, a natureza intangível e invisível do software, alinhado com seu crescente grau de complexidade, torna tais atividades difíceis de serem gerenciadas e executadas (DANTAS; GARCIA; WHITTLE, 2012). Em particular, entender como os requisitos podem sinalizar quais propriedades do código devem ser evitadas, de modo que mudanças indesejáveis no mesmo não ocorram, é algo desafiador.

A literatura reporta que boa parte das mudanças indesejadas durante evolução do código está associada ao escopo das propriedades de composição de seus elementos (DANTAS; GARCIA; WHITTLE, 2012). O estudo de Dantas, Garcia e Whittle (2012) também sugere que o escopo de artefatos de software é uma propriedade que deve ser observada em diferentes níveis de abstração. Porém, não há estudos que relacionem as variações do escopo de requisitos com o escopo de código e como a partir dos requisitos podemos gerenciar melhor as mudanças de código. O escopo de um determinado artefato de software, está relacionado a cobertura, em termos de dependência, que o artefato possui em relação a todos os artefatos do sistema. Assim sendo, o escopo de um requisito ou elemento de código nos revela que qualquer alteração em seus artefatos pode acarretar mudanças nos outros requisitos incluídos em seu escopo.

É importante destacar que o escopo dos requisitos e do código, ainda que sejam

intrínseco a evolução dos sistemas de software, são propriedades não explícitas aos engenheiros de software. Como já estudado (DANTAS; GARCIA; WHITTLE, 2012), o escopo de código, por exemplo, precisa ser automaticamente monitorado e ao passo que comece a apresentar variações anormais, é particularmente interessante que medidas preventivas sejam tomadas para que estabilidade dos elementos envolvidos não seja afetada negativamente. O problema é que o impacto das propriedades de requisitos na evolução do código ainda não possui um entendimento claro, porque não há estudos que relacionem as variações de escopo em diferentes níveis de abstração. Dessa forma, para lidar com os prováveis efeitos das propriedades de requisitos sobre o escopo do código, os engenheiros de requisitos precisam ter a sua disposição meios capazes de quantificá-las. Mesmo após a quantificação do escopo de requisitos em evolução, o seu impacto sobre as variações do escopo de código ainda continua indefinido, uma vez que não há estudos que estabeleça esta relação.

1.1 Problemática

Entendemos que o escopo de requisitos, assim como de elementos de código, precisam ser automaticamente monitorados e ao passo que comecem a apresentar variações anormais, é particularmente interessante que medidas preventivas sejam tomadas para que os artefatos evoluam da forma mais estável possível. No entanto, não podemos monitorar aquilo que não podemos mensurar. Apesar de termos métricas disponíveis para quantificar o escopo dos elementos de código (DANTAS; GARCIA; WHITTLE, 2012), não há mecanismos disponíveis para a medição do escopo dos requisitos.

Para que engenheiros de software possam traçar uma correlação entre os escopos de requisitos e código, faz-se necessário, inicialmente, a disponibilização de métricas capazes de quantificar o escopo de requisitos. De forma secundária, faz-se também necessário a existência de estudos que investiguem se o escopo dos requisitos e o escopo do código estão relacionados. Infelizmente, não há um entendimento na literatura sobre como o escopo de requisitos em evolução pode ser quantificado e como. Uma vez quantificados, eles podem ser usados para indicar potenciais mudanças indesejadas no código, causada pelo o escopo de seus elementos (DANTAS; GARCIA; WHITTLE, 2012).

Consequentemente, nosso primeiro problema, pode ser definido como:

> *O escopo de requisitos não é ainda investigado com profundidade, porque não há mecanismos que promovam a sua quantificação*

Diante da ausência de mecanismos que possam quantificar o escopo de requisitos

em evolução, nosso segundo problema pode ser definido como:

> *Mesmo após a quantificação do escopo de requisitos em evolução, a sua relação com o escopo de código ainda não foi investigada, uma vez que não há estudos que estabeleça esta relação*

O fato é que há uma falta de estudos empíricos voltados ao impacto do escopo de requisitos nos demais artefatos do software, em particular, o código. Esta lacuna existe principalmente porque não há meios capazes de quantificar o escopo dos requisitos. Neste caso, métricas para quantificação de tal escopo irão funcionar como uma base para investigarmos a relação do escopo de requisitos com o escopo de código em evolução e consequentemente, as mudanças demandadas.

1.2 Objetivos e Questão de Pesquisa

O objetivo principal deste trabalho é prover suporte aos engenheiros de requisitos para que eles possam mensurar de forma automática o escopo de requisitos de software em evolução. Em particular, temos os seguintes objetivos específicos:

1. Realizar uma revisão bibliográfica sobre tópicos relacionados à evolução de software, em particular, métricas para requisitos em evolução;
2. Entender o escopo dos elementos de requisitos de software;
3. Propor um mecanismo de medição com o objetivo de quantificar o escopo mapeado anteriormente;
4. Validar as métricas propostas, manualmente;
5. Formalizar métricas validadas;
6. Desenvolver uma ferramenta para automatizar a quantificação do escopo de requisitos;
7. Investigar o impacto do escopo de requisitos no escopo de código;
8. Documentar os resultados encontrados.

Diante do exposto e para atingir os objetivos de trabalho, temos como premissa maior responder a seguinte Questão de Pesquisa (QP):

QP: *O escopo dos elementos de requisitos estão relacionados com os escopo dos elementos de código?*

Ao responder esta questão de pesquisa estamos disponibilizando indicadores quantitativos sobre o impacto do escopo de requisitos no escopo de código e consequentemente, fornecendo indicadores aos engenheiros de requisitos para que eles possam prever possíveis mudanças desnecessárias e custosas dos artefatos de código.

1.3 Contribuições

Esta seção descreve brevemente as contribuições deste trabalho, são: (i) definição de mecanismos para quantificar o escopo de requisitos, (ii) desenvolvimento de um *plug-in* capaz de medir automaticamente o escopo dos requisitos e (iii) estudo empírico sobre a relação do escopo de requisitos com o escopo de código.

Definição de mecanismos para quantificar a estabilidade de requisitos. Foi proposta uma métrica para quantificação do escopo de requisitos de forma que pudéssemos avaliar quantitativamente a sua relação com o escopo de código. Tal medida soluciona a primeira questão exposta na problemática.

Desenvolvimento de um *plug-in* capaz de medir automaticamente o escopo dos requisitos. Para otimizar o processo de investigar a relação entre os escopos de requisitos e código em sistemas evolutivos, foi desenvolvido um *plug-in* para Eclipse, chamado *Measuring Requirement Scope* (MeRS), o qual lê os casos de uso em `xml` e tem a capacidade de fornecer o escopo de cada um dos requisito de forma automática.

Resultado empírico. Foi realizado um estudo exploratório com o objetivo de avaliar a correlação existente entre o escopo de requisitos e código. A investigação foi conduzida usando dois sistemas diferentes, o MobileMedia e o Notepad, e revelou que tais escopos estão diretamente ligados. Este resultado responde a segunda questão exposta na problemática e a QP que abrange a problemática do trabalho como um todo.

1.4 Estrutura do Documento

Este documento está organizado em mais 5 capítulos. O Capítulo 2 exibe o referencial teórico relacionado à pesquisa. No Capítulo 3 é discutida a rastreabilidade de requisitos. O processo de construção e automatização da métrica de requisitos é mostrado no Capítulo 4. A análise dos resultados é apresentada no Capítulo 5. Por fim, no Capítulo 6 são postas as considerações finais do trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo é apresentado o referencial teórico que norteia a condução de nosso estudo. Na Seção 2.1 abordamos requisitos de software, dando uma especial atenção às definições de requisitos funcionais e não-funcionais. Na sequência (Seção 2.2), falamos sobre escopo de requisitos. Na Seção 2.3, apresentamos técnicas para modelagens de requisitos. Em seguida (Seção 2.4), tratamos da evolução de requisitos de sistemas de software. Na Seção 2.5 é discutido sobre métrica de requisitos de software. A Seção 2.6 é dedicada a estabilidade de requisitos em sistemas de software. Por fim, a Seção 2.7 faz um resumo do capítulo e introduz o capítulo seguinte.

2.1 Requisitos de Software

De acordo com Sommerville (2011), a engenharia de requisitos é o processo de descobrir, analisar, documentar e verificar serviços que o sistema de software oferece e as restrições a seu funcionamento. Segundo Leite (1994 apud CYSNEIROS, 2001), por meio da engenharia de requisitos, é possível sistematizar o processo de definição dos requisitos. Tal sistematização é necessária devido a complexidade dos sistemas exigirem mais atenção no correto entendimento do problema antes do comprometimento de uma solução.

Posto isso, vimos que as definições da engenharia de requisitos giram em torno de uma correta compreensão e definição dos requisitos do sistema. Estes que são condições necessárias para obtenção de determinados objetivos. Requisitos de software são, em geral, classificados como funcionais e não-funcionais. Porém, a literatura nos apresenta outras classificações, onde requisitos funcionais são identificados como serviços e os não-funcionais como restrições (SOMMERVILLE, 2011).

Um requisitos é dito funcional quando expressam funções ou serviços que um software deve ou pode ser capaz de fornecer. As funções ou serviços são, em geral, processos que utilizam entradas para produzir saídas (CYSNEIROS, 2001). Requisitos não-funcionais referem-se a restrições de qualidade para um software ou para o processo de desenvolvimento deste sistema. Segurança, precisão, usabilidade, performance e manutenabilidade são exemplos de requisitos não-funcionais. A distinção entre esses tipos de requisitos nem sempre é clara, o que por uma parte se justifica pelo fato de requisitos não-funcionais estarem sempre associados a requisitos funcionais

(CYSNEIROS, 2001).

O trabalho de Eckhardt, Vogelsang e Fernández (2016) nos confirma essa percepção, pois discorre sobre o questionamento dos requisitos não-funcionais serem realmente não-funcionais. Os autores realizaram uma investigação classificando 530 requisitos não-funcionais de 11 especificações de requisitos industriais. Eles afirmaram que os requisitos são diferenciados por representar **como** o sistema deve fazer algo (não-funcional) em contraste com **o que** o sistema deve fazer (funcional). Diante disso, analisaram quais requisitos declarados como não-funcional dizia o que o sistema devia fazer e chegaram ao resultado de que algumas propriedades são mal interpretadas, tal como disponibilidade e desempenho. Devido a essas más interpretações, perceberam que a maioria dos requisitos nomeados de não-funcional descreve funcionalidades do sistema, ou seja, esses requisitos eram identificados por descreverem como o sistema deve fazer algo, porém suas informações indicavam o que o sistema deve fazer.

Portanto, percebemos que os requisitos não-funcionais tem uma maior complexidade para ser compreendido em relação aos funcionais. Por esse motivo e porque vamos investigar as propriedades de requisitos em evolução, as quais são mais facilmente identificadas nos requisitos funcionais, este trabalho irá tratar apenas dos requisitos funcionais.

2.2 Escopo de Requisitos

O documento de requisitos de um sistema de software é composto por requisitos funcionais e não-funcionais. Eles podem ser representados conjuntamente ou de forma separada, isso vai depender do nível de detalhes e da maneira que eles serão trabalhados. Cada um deles pode estar associado ou não a vários outros requisitos, podendo conter uma associação completa com todos os requisitos do sistema. Essa situação nos remete a um atributo da engenharia de software: escopo de requisitos.

O escopo de um requisito refere-se a uma porcentagem dos requisitos, que ele depende, pertencente ao sistema, ou seja, é uma parcela de requisitos associados ao requisito que o torna sujeito a sofrer as consequências por qualquer ação acometida a um dos seus integrantes. O escopo de requisito é calculado somando o número de suas ligações com outros requisitos e dividindo essa soma pelo número total de requisitos do sistema.

Assim sendo, o escopo de um requisito nos revela que qualquer alteração nesse

requisito pode acarretar mudanças nos outros requisitos incluídos em seu escopo. Essas mudanças em cascata podem comprometer o sistema, parcial ou completamente, devido a um possível custo e trabalho excessivo. Logo, faz-se interessante ter o total controle do escopo de cada requisito para que se possa moldá-lo da melhor maneira possível.

2.3 Modelagem de Requisitos

A engenharia de requisitos tem a modelagem como um de seus processos fundamentais. Moldar um requisito nada mais é do que analisar e formalizar uma necessidade levantada pelos envolvidos no desenvolvimento do sistema. Uma das vantagens proporcionada por essa prática esta relacionada à documentação do sistema, que tem sua importância já conhecida pela comunidade científica, pois a partir dela são alcançados vários benefícios, como o auxílio na manutenção, na evolução e no reuso de softwares.

Existem diversas abordagens para modelagem de requisitos que, basicamente, podem ser classificadas como: representações gráficas, representações textuais ou uma combinação de ambas (RIBEIRO, 2013). Este trabalho fez uso puramente da representação gráfica, especificamente do diagrama de casos de uso. Ele foi escolhido por, além de ser bastante conhecido e usado, ser capaz de importar e exportar seu código, em xml, de maneira simples. O código em xml se fez interessante por ter uma maior facilidade de manipulação por parte dos programas.

Casos de uso podem ser aplicados para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementado. Eles servem para ajudar a validar a arquitetura e verificar o sistema à medida que ele evolui durante seu desenvolvimento(BOOCH; RUMBAUGH; JACOBSON, 2006). Neste trabalho, os casos de uso são usados para expressar os requisitos funcionais do sistema.

Um conjunto de casos de uso de um mesmo sistema resulta num diagrama de casos de uso. Este possui, além dos requisitos/casos de uso, ligações unidirecionais (*include* e *extend*), as quais representam as possíveis dependência existentes entre eles. A relação *include* indica que o requisito base depende obrigatoriamente do requisito alvo, ao contrário da relação *extend* que indica que o requisito alvo pode depender do requisitos base.

Os casos de uso foram transformados em código xml com intuito de identificar

as dependências entre os requisitos e assim foi possível calcular o escopo dos requisitos. Para exemplificar essa modelagem, imaginemos um sistema de compras com os seguintes requisitos: **Efetuar compra**, **escolher forma de pagamento** e **inserir desconto**. Neste caso, teríamos um diagrama de caso de uso como o ilustrado na Figura 1.

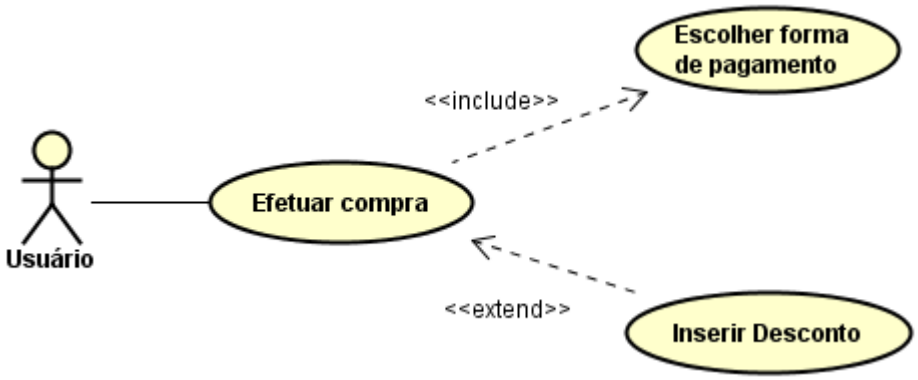

Figura 1 – Exemplo de Diagrama de casos de uso.

A partir do diagrama, utilizando a ferramenta ASTAH professional (VISION, 2017), é gerado o código xml com várias informações contidas em mais de 3000 linhas de código. Muitas dessas informações são referentes à representação gráfica, as que são importantes estão nas tags: <UML:UseCase>, <UML:Include> e <UML:Extend>. Elas trazem, respectivamente, as informações sobre os casos de uso, as ligações de *include* e de *extend*, tais como nomes e *id*. As duas últimas tags ainda apresentam os *ids* dos casos de uso relacionados na ligação.

As tags mencionadas referente ao sistema hipotético são apresentadas nas Figuras 2, 3 e 4. A primeira apresenta os nomes dos casos de uso, seus identificadores e, quando possuem, identificadores das ligações de *include* ou *extend*. As figuras seguintes apresentam, respectivamente, informações das ligações de *include* e de *extend*, tais como: identificadores da ligação, identificadores dos requisitos base e alvo que estabelecem a ligação e, caso possua, um nome para a ligação.

2.4 Evolução de Requisitos

Segundo Sommerville (2011), os requisitos do sistema são as descrições do que o sistema deve fazer, os serviços que oferecem e as restrições do seu funcionamento.

O resultado da definição dos requisitos de um software é o documento de requisitos que poderá ter diferentes níveis de abstração de modo que seja melhor

```
<UML:UseCase xmi.id="js8-9d8ccc15acb6e18ab492d128f66899c1" name="Escolher+forma+de+pagamento"
version="0" unSolvedFlag="false" isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
  <UML:ModelElement.namespace>
    <UML:Namespace xmi.idref="jak-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:ModelElement.namespace>
  <UML:ModelElement.visibility xmi.value="public"/>
  <UML:UseCase.include>
    <UML:Include xmi.idref="odn-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:UseCase.include>
</UML:UseCase>
<UML:UseCase xmi.id="k21-9d8ccc15acb6e18ab492d128f66899c1" name="Efetuar+compra" version="0"
unSolvedFlag="false" isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
  <UML:ModelElement.namespace>
    <UML:Namespace xmi.idref="jak-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:ModelElement.namespace>
  <UML:ModelElement.visibility xmi.value="public"/>
</UML:UseCase>
<UML:UseCase xmi.id="kce-9d8ccc15acb6e18ab492d128f66899c1" name="Inserir+Desconto" version="0"
unSolvedFlag="false" isRoot="false" isLeaf="false" isAbstract="false" isActive="false">
  <UML:ModelElement.namespace>
    <UML:Namespace xmi.idref="jak-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:ModelElement.namespace>
  <UML:ModelElement.visibility xmi.value="public"/>
  <UML:UseCase.extend>
    <UML:Extend xmi.idref="pkz-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:UseCase.extend>
</UML:UseCase>
```

Figura 2 – Casos de uso em xml.

```
<UML:Include xmi.id="odn-9d8ccc15acb6e18ab492d128f66899c1" name=""
version="0" unSolvedFlag="false">
  <UML:ModelElement.namespace>
    <UML:Namespace xmi.idref="jak-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:ModelElement.namespace>
  <UML:ModelElement.visibility xmi.value="public"/>
  <UML:ModelElement.stereotype>
    <UML:Stereotype xmi.idref="odp-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:ModelElement.stereotype>
  <UML:Include.base>
    <UML:UseCase xmi.idref="k21-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:Include.base>
  <UML:Include.addition>
    <UML:UseCase xmi.idref="js8-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:Include.addition>
</UML:Include>
```

Figura 3 – Ligações de include em xml.

compreendido pelos clientes, os quais possuem pouco ou nenhum conhecimento técnico, e desenvolvedores do projeto que, por sua vez, possuem um alto conhecimento técnico de sistemas de software e conhecem pouco do ambiente de negócio. A grande importância dos requisitos consiste em combinar a *expertise* dos envolvidos no projeto, o resultado são requisitos bem definidos em relação às regras de negócio e as restrições do sistema. Com isso, o sistema deverá responder de maneira satisfatória a expectativa dos clientes e, consequentemente, diminuir mudanças forçadas pelas necessidades não atendidas. Porém, ainda que se consigam requisitos iniciais bem definidos, eles normalmente tendem a evoluir, conforme afirma Sommerville (2011): os requisitos de sistemas de software estão sempre mudando.

```
<UML:Extend xmi.id="pkz-9d8ccc15acb6e18ab492d128f66899c1" name=""
version="0" unSolvedFlag="false">
  <UML:ModelElement.namespace>
    <UML:Namespace xmi.idref="jak-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:ModelElement.namespace>
  <UML:ModelElement.visibility xmi.value="public"/>
  <UML:ModelElement.stereotype>
    <UML:Stereotype xmi.idref="pl1-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:ModelElement.stereotype>
  <UML:Extend.extension>
    <UML:UseCase xmi.idref="kce-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:Extend.extension>
  <UML:Extend.base>
    <UML:UseCase xmi.idref="k21-9d8ccc15acb6e18ab492d128f66899c1"/>
  </UML:Extend.base>
</UML:Extend>
```

Figura 4 – Ligações de extend em xml.

A evolução de requisitos surge em consequência da evolução do sistema em geral. Se um sistema necessita de mudanças no seu código e no seu projeto arquitetural, então ele também necessita de mudanças no seu documento de requisitos. Muitas vezes, essas mudanças são guiadas pelas alterações na política de mercado ou até mesmo por alteração no ramo de negócios. Essa variedade e necessidade frequente de mudanças provoca a comunidade científica a investigar esse tema. Nesse contexto, encontramos trabalhos que tratam o problema de mudanças futuras, eles investigam maneiras de identificar e tratar essas mudanças, tal como os trabalhos a seguir.

O trabalho de Pimentel et al. (2011) trata da antecipação das mudanças de requisitos por meio do estudo do futuro. Para resolver o problema, já identificado anteriormente, os autores descreveram alguns métodos de previsão, descreveram também como métodos de previsão podem ser incluídos no levantamento de requisitos para reduzir a necessidade de mudanças posteriores em um sistema de software e, por fim, propuseram uma abordagem específica para realizar alterações em requisitos expressos por um *Goal Model*, com base em uma representação do futuro proporcionada pelo método *Futures Wheel*.

Ainda nesse trabalho são estabalecidos métodos de previsão, ou grupos de métodos, de acordo com que destina ser alcançado, são eles: (i) Coletar julgamento de especialistas; (ii) prever séries temporais e outras medidas quantitativas; (iii) compreender as ligações entre os eventos, tendências e ações; (iv) determinar um curso de ação na presença de incerteza; (v) retratar alternativas plausíveis para futuros; (vi) chegar a um entendimento se o futuro está melhorando; (vii) controlar alterações e suposições, e; (viii) determinar a estabilidade de um sistema (GORDON; GLENN, 2004 apud PIMENTEL et al., 2011).

No entanto, os autores afirmam que quatro dos oito objetivos estão fortemente relacionados com a elicitação de requisitos: (i), (ii), (iii) e (v); e se concentram no item (v) por julgar que tem um objetivo mais genérico e pode ser aplicado em uma amostra maior de sistemas. A estabilidade mencionada no item (viii) é apenas citada, não há análise alguma sobre ela, provavelmente porque o escopo do trabalho envolve apenas a elicitação de requisitos. Ainda que métodos de previsão sejam fortemente associados à evolução de requisitos, o escopo do trabalho é bem definido e não insere a evolução, na qual a estabilidade tem grande importância.

Em sequência, o trabalho de Pimentel et al. (2012) também aborda a antecipação de mudanças de requisitos usando a futurologia na elicitação de requisitos. Esse trabalho é uma continuação do anterior, ataca o mesmo problema, porém com a proposta de solução aprimorada. Os autores apresentam uma extensão da notação de modelagem *Futures Wheel*. Essa modelagem descreve eventos futuros e para cada evento suas consequências. Na extensão proposta foram incrementadas as consequências diretas, as quais diferenciam das consequências regulares por estarem relacionadas com o sistema. Com esse método de previsão é proposto um processo que compreende quatro etapas: planejar *Futures Wheel*, executar *Futures Wheel*, definir consequências diretas e analisar consequências diretas. O processo foi analisado em um estudo de caso realizado com um sistema de planejamento de rotas desenvolvido pela UFPE e como resultado mostrou que, para este caso em particular, a abordagem forneceu mais entradas para elicitação de requisitos que, por sua vez, forneceu um modelo de requisitos mais rico.

2.5 Métricas de Requisitos

No momento da evolução de um sistema, quando são exigidas mudanças, um ou mais novos processos são adicionados e o seus desempenhos deverão ser verificados, portanto, é necessária a realização de uma medição deste sistema. Software pode ser medido usando processo, produto, recursos e métricas de requisitos (ALI, 2006).

Segundo Bokhari e Siddiqui (2011), as métricas de software têm como finalidade fornecer uma avaliação quantitativa da extensão de que o produto, processo ou recurso possui determinados atributos. Eles complementam afirmando que as métricas de software dão informação geral sobre o desenvolvimento de produtos, como custo, tempo e informações de todas as fases; e seu objetivo é identificar e medir fatores essenciais que afeta o desenvolvimento de software.

Dada importância da medição de software e tendo ciência que erros descobertos

na fase inicial de projetos são mais facilmente corrigidos quando comparados a erros descobertos posteriormente, logo a comunidade cientifica percebeu a relevância do assunto e publicou alguns trabalhos evidenciando métricas de requisitos.

O trabalho de Bokhari e Siddiqui (2011) discorre sobre métricas para engenharia de requisitos e ferramentas de requisitos automatizadas. Esse trabalho pretende solucionar a questão da medição manual de indicadores de requisitos, na qual os autores afirmam que essa é uma tarefa demorada, tediosa e sujeita a erros. Para isso, eles apresentam métricas de requisitos para melhorar o processo de gerenciamento de requisitos e qualidade do produto, indicam quatros tipos de métricas que podem ser coletadas a partir dos requisitos: (i) métrica de tamanho; (ii) métricas de rastreabilidade de requisitos; (iii) métricas de completude de requisitos; e (iv) métrica de volatilidade de requisitos. Além disso, eles descrevem algumas ferramentas para medição de requisitos automatizada.

Enfatizando outro atributo de requisitos, o trabalho de Byun et al. (2014) explana sobre métricas para medir a consistência de requisitos por meio de seus objetivos e restrições. O problema evidenciado são as falhas de projetos resultantes da inconsistência de requisitos. Com isso, os autores propuseram métricas para indicar o quanto requisitos estão de acordo com objetivos e restrições do sistema, ou seja, medir a consistência dos requisitos. Na avaliação das métricas, em dois cenários diferentes, os autores apresentam como resultado alguns benefícios: a consistência de cada requisito pode ser medida com as métricas propostas e elas são flexíveis a mudanças.

No trabalho de Monperrus et al. (2013) é tratada a medição automatizada de modelos de requisitos. Os autores procuram preencher uma lacuna referente a uma abordagem que visa unificar as métricas de requisitos definidas na literatura. Para esse fim, eles analisaram 11 trabalhos sobre métricas de requisitos, os quais possuíam 138 especificações de métricas. A partir dessas especificações, eles desenvolveram um metamodelo de requisitos que dá suporte a medição de requisitos e uma lista consolidada de 78 métricas de requisitos. Compondo as contribuições do trabalho, além do metamodelo e da lista de requisitos, está a geração de uma ferramenta de medição automática de requisitos usando a abordagem de medição de requisitos dirigida por modelo.

2.6 Estabilidade de Requisitos

Foi notado que existem várias métricas, elas podem mensurar diferentes tipos de medidas, a estabilidade é um desses tipos de medidas de requisitos que pode ser mensurada com intuito de reduzir riscos na fase de requisitos e, consequentemente, no ciclo de desenvolvimento do projeto como um todo. De acordo com Hazan e Leite (2003), a estabilidade de requisitos é o grau de mudanças para a *Baseline* dos requisitos de software. Em outras palavras, podemos entender estabilidade como a capacidade de requisitos resistirem a mudanças, ou seja, quanto mais estável um requisito se apresenta, menos mudanças ele sofre ao longo da evolução do sistema.

Christopher e Chnadra (2012) afirmam que muitos projetos de software falham devido a requisitos instáveis e a falta de um eficiente gerenciamento de mudanças de requisitos. Song (2012) acrescenta dizendo que a qualidade e estabilidade dos requisitos desempenham um papel crítico em relação ao custo de trabalho e a qualidade do projeto. Ele ainda comenta que um requisito instável, inevitavelmente, irá causar atraso na entrega do sistema, sério declínio de qualidade, insatisfação do usuário, excesso de custos e outros problemas. Christopher e Chnadra (2012) complementam comentando que maximizando a estabilidade de requisitos, o impacto das mudanças é reduzido, ou seja, quanto maior a estabilidade, menos mudanças tendem a se propagar.

A partir deste cenário em que a estabilidade de requisitos se faz importante para o desenvolvimento de sistemas de software devido aos diversos motivos citados acima, foram encontrados alguns trabalhos que abordam este tema.

O trabalho de Christopher e Chnadra (2012) analisa a eficácia da estabilidade de requisitos na contagem de ponto de função. O problema estudado são os requisitos instáveis e a falta de um gerenciamento de mudanças de requisitos eficientes que são responsáveis pelas falhas de muitos projetos de softwares. Os autores descrevem o modelo de ponto de função e como calculá-lo, afirmam que o ponto de função calculado na fase de projeto é diferente do ponto de função calculado na codificação e demonstra como adaptar o modelo para a fase de projeto, de modo que seja possível prever o índice de estabilidade de requisitos usando ponto de função. Eles concluem certificando que o modelo de previsão para estabilidade de requisitos fornece a solução para medição de mudanças de requisitos com base na análise de ponto de função.

O trabalho de Song (2012) aborda o controle e medição da estabilidade de requisitos sobre projetos de software. Esse trabalho aborda o problema de requisitos instáveis, seus riscos e defeitos causados por ele. O autor faz uma análise detalhada dos fatores que afetam a estabilidade de requisitos, propõe um método de controle

de mudanças de requisitos e conduz uma pesquisa sobre a medida de estabilidade de requisitos com objetivo de compreender o grau da mudança de requisitos. Nesse trabalho o autor relata com maior ênfase sobre o método de controle proposto, o qual detalha como deve ser as ações dos envolvidos no projeto, e discorre superficialmente sobre a pesquisa realizada. Define duas métricas de estabilidade e para calculá-las extraiu dados estatísticos (requisitos iniciais, adicionados, removidos e modificados) de um projeto de software durante os seis primeiros meses de desenvolvimento, em intervalos regulares, tal como um mês. Contudo, ele não evidencia as métricas de estabilidade de requisitos, define as métricas usadas apenas em poucos parágrafos e deixa uma lacuna a ser preenchida sobre o tema. Ele apenas relata um estudo de caso, no qual são expostos os dados estatísticos que servem para calcular as métricas de estabilidade.

2.7 Resumo

Este capítulo apresenta uma revisão dos conceitos e trabalhos existentes, que amparam a condução desta pesquisa de mestrado. No Capítulo 3, abordaremos o tema rastreabilidade de requisitos, fundamental para conduzirmos a análise do impacto do escopo de requisitos no escopo de código.

3 RASTREABILIDADE DE REQUISITOS

Neste capítulo é contextualizado a rastreabilidade de requisitos, a qual já é reconhecida na literatura como um fator importante para a qualidade da gestão de requisitos e se fez necessária na análise de resultados deste trabalho. O capítulo é iniciado na Seção 3.1 com uma introdução e definições do assunto. Na Seção 3.2 é feita a classificação dos tipos de rastreabilidade. Logo em seguida são apresentados os tipos de geração de rastreabilidade, na Seção 3.3. Na Seção 3.4 são apresentados alguns trabalhos encontrados na literatura sobre o tema. Finalmente, é exibido um resumo do atual capítulo na Seção 3.5.

3.1 Visão Geral

A qualidade de um produto de software está associada diretamente a uma boa gestão de requisitos (Capítulo 1). Dentre as técnicas utilizadas para o gerenciamento dos requisitos, está a rastreabilidade de requisitos. Gotel e Finkelstein (1997 apud SILVA, 2016) definem rastreabilidade como um mecanismo capaz de descrever e seguir a vida de um requisito, em ambas as direções, por todo seu ciclo de vida, desde sua origem até a implantação e uso. Segundo Kotonya e Sommerville (1998 apud TRINDADE; LUCENA, 2016) , um requisito é rastreável quando é possível, a partir dele, descobrir quem o sugeriu, qual motivo da sua existência e quais são suas dependências.

Diante dessas definições podemos entender a rastreabilidade de requisitos como uma técnica que guarda informações dos requisitos para que seja possível, a partir de cada um deles, rastrear seus artefatos correspondentes nos diferentes níveis de desenvolvimento. Tanto o rastreamento para a modelagem de negócios (sentido para trás) quanto para o nível de código (sentido para frente).

De acordo com Ramesh e Jarke (2001 apud SAYÃO; LEITE, 2006), a rastreabilidade tem sido identificada na literatura como um fator de qualidade dos sistemas de software. ASSOCIAÇÃO (2009 apud MARQUES et al., 2010) complementa afirmando que é fundamental definir e manter esse mecanismo para a evolução de requisitos permanecer sob controle durante o processo de desenvolvimento.

Conforme Egyed e Grünbacher (2005 apud DELATER, 2013), a informação de rastreabilidade auxilia o processo de desenvolvimento de software de várias maneiras,

tais como: compreensão do programa, gerenciamento de mudanças, manutenção de software, reuso de software e prevenção de mal entendimentos.

3.2 Classificação de Rastreabilidade

A rastreabilidade pode ser identificada de diferentes formas. Os tipos de rastreabilidade existentes possuem, basicamente, duas classificações gerais: i) pré e pos-rastreabilidade, e ii) rastreabilidade horizontal e vertical (GENVIGIR, 2009).

A Figura 5 nos mostra a pré-rastreabilidade e a pós-rastreabilidade. A primeira está concentrada no ciclo de vida dos requisitos antes de serem incluídos na especificação de requisitos; e a segunda está concentrada no ciclo de vida dos requisitos após serem incluídos na especificação de requisitos (GOTEL; FINKELSTEIN, 1994 apud GENVIGIR, 2009).

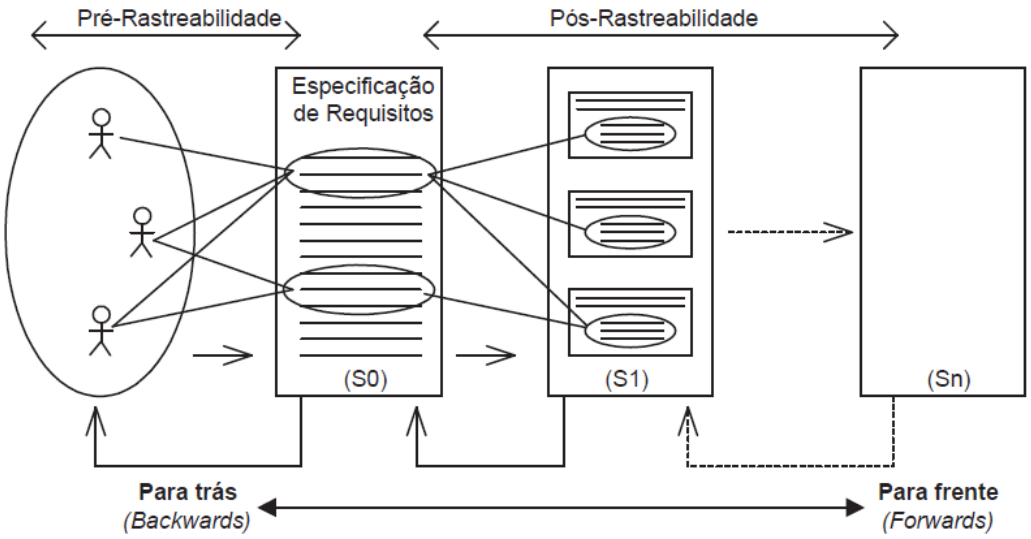

Figura 5 – Pré e Pos-Rastreabilidade.

Fonte: (GENVIGIR, 2009)

A principal diferença entre a pós e a pré-rastreabilidade envolve as informações com que elas podem lidar. A pós depende da habilidade de rastrear requisitos de um ponto de referência (*baseline*), neste caso, a especificação de requisitos. A pré-rastreabilidade depende da habilidade para rastrear requisitos de seus padrões originais ou fontes (clientes, usuários, normas, etc.) (GENVIGIR, 2009).

O segundo tipo de rastreabilidade, apresentado na Figura 6, nos mostra a rastreabilidade horizontal e vertical. A horizontal busca fazer ligações dos requisitos

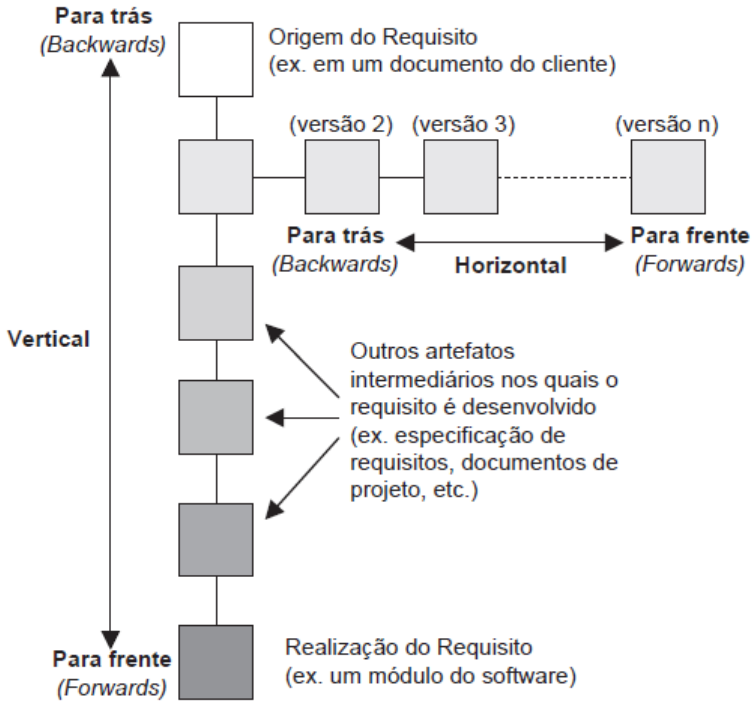

Figura 6 – Rastreabilidade Horizontal e Vertical.

Fonte: (GENVIGIR, 2009)

no decorrer das versões em uma mesma fase do ciclo de vida. Já a vertical relaciona os requisitos por diferentes fases do ciclo de vida, mas numa mesma versão.

Neste trabalho, para mapear as funções do sistema, foram usadas as duas classificação da técnica. Da primeira foi utilizada apenas a pós-rastreabilidade, pois lidamos com requisitos já especificados e documentados. Da segunda foi utilizada tanto a rastreabilidade vertical como a horizontal, uma vez que o intuito era relacionar os requisitos em diferentes níveis do ciclo de vida numa mesma versão e também acompanhar sua evolução no decorrer das versões em uma mesma fase do ciclo de vida. No nosso caso, os níveis de abstração trabalhados foram os de requisitos e de código.

3.3 Geração de Rastros

As relações de rastreabilidade esperadas nas diferentes fases do ciclo de vida do desenvolvimento de software, especificamente, na fase de requisito e de código são associações entre requisitos concisos e inseridos no documento de especificação de requisitos e os códigos correspondentes para implementação deles. A geração dessas relações pode ser muito complexa ou simples, isso depende de vários fatores, tais como:

a maneira que o sistema de software foi desenvolvido; a técnica utilizada e a ferramenta que dará suporte.

A maioria das ferramentas de engenharia e rastreabilidade de requisitos oferecem apenas um suporte limitado à rastreabilidade. É esperada a criação manual de rastreabilidade em inúmeras técnicas e abordagens, incluindo ferramentas, como: DOOR, RDT, RTM, RETH, CORE e PuLSE-BC. Contudo, essa prática realizada de forma manual é difícil, propensa a erros, demorada e complexa. Apesar das vantagens que podem ser obtidas, a rastreabilidade efetiva raramente é estabelecida manualmente (SPANOUDAKIS; ZISMAN, 2005). Para contornar esses obstáculos, algumas abordagens de geração de rastreabilidade com suporte automático ou semi-automático tem sido propostas.

A declaração manual de relações de rastreabilidade é normalmente suportada por componentes de visualização, nos quais os documentos a serem rastreados são exibidos e os usuários podem identificar os elementos a serem relacionados de uma maneira mais fácil. A maioria dessas abordagens afirma apoiar a geração de rastreabilidade semi-automática, uma vez que os relacionamentos são identificados manualmente, mas os links entre os elementos relacionados são gerados automaticamente pela ferramenta de suporte. Entretanto, numa visão geral, ainda são consideradas manuais por necessitarem que os usuários identifiquem e marque os elementos a serem rastreados (SPANOUDAKIS; ZISMAN, 2005).

As abordagens de geração de rastreabilidade semi-automatica buscam desvencilhar-se da identificação gerada única e diretamente pelos usuários e automatizar parte do processo. Elas podem ser divididas em dois grupos: (i) grupo de *links* pré-definidos, o qual concentra abordagens que gera os relacionamentos de rastreabilidade a partir de alguns links definidos por usuários anteriormente, algumas delas publicadas por Cleland-Huang e Schmelzer (2003), Cleland-Huang et al. (2002), Egyed (2003) e Egyed e Grunbacher (2002); e (ii) grupo orientado a processos, o qual concentra abordagens que possuem seus relacionamentos de rastreabilidade gerados como um resultado do processo de desenvolvimento de software, exemplos dessa abordagem estão nos trabalhos Pohl (1996a) e Pohl (1996b).

Com intuito de minimizar ainda mais a participação do usuário na identificação das relações de rastreabilidade, foram publicados trabalhos com abordagem completamente automática. Esses trabalhos não compartilham o emprego de uma mesma técnica, alguns deles utilizam técnicas de recuperação de informação (ANTONIOL et al., 2002) (HAYES; DEKHTYAR; OSBORNE, 2003) (MARCUS; MALETIC, 2003), outros regras de rastreabilidade (SPANOUDAKIS et al., 2004) (ZISMAN et al., 2003), integradores especiais (SHERBA; ANDERSON; FAISAL, 2003) e axiomas de inferência (PINHEIRO, 2000).

Deve ser notado que nenhuma das abordagens acima possa automatizar completamente a geração de relações de rastreabilidade, mas elas tomaram passos significativos para esta direção (SPANOUDAKIS; ZISMAN, 2005).

Diante de todo o exposto, percebemos que, em condições normais, a automatização do processo traz muitos ganhos. Quanto mais automatizado for o processo, mais eficiente ele será. No entanto, existem casos em que a geração manual é a melhor opção. Este caso se encaixa nas raras situações em que a rastreabilidade efetiva é alcançada de forma manual. A rastreabilidade nas Linhas de Produto de Software (LPS) representa esse acontecimento.

Como as LPS são construídas com intenção de adicionar ou remover componentes prontos de acordo com a necessidade dos envolvidos no projeto, esses componentes são bem especificados tanto nos requisitos como no código para que possa ocorrer a inserção ou remoção de forma mais simples. O código é construído com uso de diretivas de pré-processador, as quais permitem incrementar componentes independentes à *baseline* de forma *plug-and-play*. Essas diretivas, produzidas durante o desenvolvimento do código, fornecem informações dos requisitos e, automaticamente, faz o mapeamento das relações de rastreabilidade visto que cada componente é relacionado a uma função do sistema (requisito funcional).

O exemplo exibido na Figura 7 foi retirado da segunda versão do *MobileMedia*. Ele mostra um trecho de código que permite a inserção de um componente/requisito: ordena as fotos. A diretiva de pré-processador *#ifdef* verifica se a chave *includeSorting*, a qual representa a função de ordenar as fotos, foi definida e, em caso positivo, ordena as fotos chamando a *bubbleSort()* e passando por parâmetro uma lista de imagens. Caso a equipe de desenvolvimento decida por não inserir essa função no programa final, basta transformar o código em comentário.

```
// #ifdef includeSorting
// [EF] Check if sort is true (Add in the Scenario 02)
if (sort) {
    bubbleSort(images);
}
// #endif
```

Figura 7 – Trecho de código do *MobileMedia* exibindo diretivas de pré-processador que incluem a função ordenar foto.

Nesse exemplo, fica evidente a técnica de rastreabilidade mapeando os rastros do código para os requisitos, visto que a chave utilizada para incrementar esse requisito ao produto final serve também como seu identificador. Logo, partindo do código,

especificamente das chaves utilizadas nas diretivas de pré-processador, é possível realizar a geração de rastros manual entre os requisitos nos níveis de abstração de código e requisitos sem muita dificuldade.

A prática de estruturar as relações de rastreabilidade no decorrer do desenvolvimento de software também foi utilizada por Delater (2013), mas não no contexto de LPS. O autor utiliza tags nos requisitos e códigos com informações essenciais para estabelecer relações entre eles. Com isso ele garante que a rastreabilidade do sistema continuará atualizada no decorrer da evolução do sistema. A diferença entre essas duas práticas comentadas está no fato de que a última necessita de um cruzamento de informações entre as tags de requisitos e código para alcançar a rastreabilidade efetiva e a outra se beneficia por utilizar componentes prontos e independentes determinando quais funções do sistema eles exercem.

3.4 Trabalhos relacionados

Diversos trabalhos foram publicados apresentando a rastreabilidade como um dos mais importantes pré-requisitos para o desenvolvimento de software de qualidade (SAYÃO; LEITE, 2006). A seguir serão apresentados alguns trabalhos que abordaram esse tema.

No trabalho de Barboza, Filho e Souza (2013), foi desenvolvido uma proposta sobre rastreabilidade de requisitos legais no processo de contratação de soluções de Tecnologia da Informação (TI) na Administração Pública Federal. Devido à contratação de soluções de TI necessitar estar de acordo com os requisitos legais de uma Instrução Normativa, os autores apresentaram um modelo de referência que tem o objetivo de ajudar na compreensão de todo processo, além de colaborar na proposição de uma abordagem de rastreabilidade entre os requisitos legais e artefatos produzidos.

O artigo de Trindade e Lucena (2016) faz um estudo exploratório sobre a rastreabilidade de requisitos em metodologias ágeis. As autoras atacam os desafios de utilizar a rastreabilidade de requisitos em métodos ágeis, buscando levantar requisitos para uma ferramenta que permitisse a rastreabilidade e pudesse ser aplicada ao desenvolvimento ágil, considerando seus princípios.

Enfrentando esse mesmo problema, Silva (2016) apresentou um modelo de referência de rastreabilidade no âmbito de gestão de projetos em métodos ágeis e demonstrou como aplicá-lo em um contexto real. Para construir esse modelo, além

de realizar uma revisão da literatura sobre rastreabilidade e métodos ágeis, os conceitos foram empregados na análise dos dados de projeto de software de uma grande corporação.

3.5 Resumo

Neste capítulo tratamos da rastreabilidade de requisitos, buscando apresentar aspectos associados ao seu uso no contexto do desenvolvimento e da evolução de sistemas de software. Usando técnicas de rastreabilidade, podemos mapear os elementos de código que implementam cada um dos requisitos em evolução. Porém, o impacto associado às variações do escopo dos requisitos no código dos sistemas depende, fundamentalmente, da quantificação do escopo dos requisitos. No Capítulo 4 apresentamos mecanismos para quantificação do escopo de requisitos e um *plug-in* que faz uso destes mecanismos para quantificar automaticamente o escopo.

4 MEDINDO O ESCOPO DE REQUISITOS

Este capítulo propõe uma nova métrica para o escopo de requisitos. Por meio desta métrica, engenheiros de requisitos e desenvolvedores de software em geral terão como mensurar o escopo dos requisitos ao longo da evolução de aplicações de software. A métrica proposta foi implementada como um *plug-in* para o Eclipse. Na Seção 4.1 é abordado a medição de software e a automatização desse processo. A Seção 4.2 é dedicada a terminologia e formalização da métrica desenvolvida neste trabalho. O *plug-in* MeRS é apresentado na Seção 4.3. Por fim, a Seção 4.4 resume o atual do capítulo.

4.1 Medição de Software e Automatização

O avanço da tecnologia e o aumento na demanda por produção de software trazem consigo sistemas maiores e mais complexos e, por consequência, trazem também uma maior probabilidade de ocorrer problemas, principalmente, durante o desenvolvimento do software e durante sua evolução. Com intuito de minimizar esse problema e alcançar níveis cada vez mais altos de qualidade, deve-se obter e analisar dados quantitativos que descrevam a realidade do processo, ou seja, realizar a medição de software com a maior precisão possível.

Porém, tal procedimento envolve geralmente uma grande quantidade de informação, principalmente, no contexto de grandes projetos de software. Nesses casos, o registro e a análise manual dos dados tornam-se inviáveis, tanto pelo risco de ocorrência de erros decorrentes de falhas humanas quanto pelo custo envolvido.

Uma forma de contornar os problemas associados à medição manual é automatizar por meio de ferramentas o suporte às atividades de mensuração dos atributos de qualidade do software (BORGES, 2003). Em particular, a medição do escopo de requisitos não possui métricas definidas e nem tão pouco ferramentas de medição.

Uma das formas de automatizar o processo de medição de software é com o desenvolvimento de *plug-ins*. *Plug-ins* são bastante populares e utilizados por sua alta capacidade de adicionar novas funcionalidades e recursos em outros programas de maneira simples e pouco invasiva. Alguns ambientes de desenvolvimento integrado (do inglês, *Integrated Development Environment - IDE*), como o Eclipse, auxiliam a criação de *plug-ins*. Eles fornecem modelos pré-configurados para que o desenvolvedor possa

se preocupar apenas em implementar a funcionalidade pretendida.

A utilização de *plug-ins* em trabalhos publicados na literatura é bem comum, seja com o desenvolvimento de um novo (SILVA et al., 2006) ou com o aproveitamento de um de terceiros (PAREDES; ZORZO, 2012). Essa popularidade se justifica, visto os benefícios dos processos automatizados e as vantagens que geralmente os acompanha, como: não comprometer o funcionamento de software, ser leve e possuir fácil instalação e manuseio.

De forma mais pontual, a popularidade dos *plug-ins* se deve a:

- Extensibilidade: o *plug-in* pode ser dinamicamente estendido para incluir novos recursos.
- Desenvolvimento paralelo: uma vez que os recursos podem ser implementados como componentes separados, eles podem ser desenvolvidos em paralelo por equipes diferentes.
- Direção de desenvolvimento clara: uma vez que a estrutura de *plug-ins* idealmente fornece uma interface bem definida e uma boa documentação, os desenvolvedores possuem um roteiro claro para o desenvolvimento.
- Simplicidade: um *plug-in* geralmente possui uma função e, portanto, os desenvolvedores têm um foco único

4.2 Medição do Escopo de Requisitos

Esta seção apresenta a terminologia (Seção 4.2.1) utilizada para a definição da métrica do escopo de requisitos e como é feito o cálculo desta métrica (Seção 4.2.2).

4.2.1 Terminologia

Definição 1 (Versão e Requisitos). Os sistemas de softwares analisados são versionados, ou seja, possui diversas versões com diferentes quantidades de requisitos. Cada versão V_i possui um conjunto de requisitos (r_{in}) , onde i representa o número da versão e n representa o numero identificador do requisitos na versão i. Uma versão é expressa matematicamente pela equação 4.1:

$$V_i = \{ r_{i1} , r_{i2} , r_{i3} , \dots , r_{in} \mid i, n \in \mathbb{N}^* \} \tag{4.1}$$

Definição 2 (Dependência). Cada requisito (r_{in}) é modelado por um caso de uso e pode conter dependências (d_{ink}), onde i representa o número da versão e n representa o numero identificador do requisitos na versão i e k é o identificador da dependência do requisito i na versão n. Estas dependências são ligações entre os requisitos divididas em dois tipos: *include* e *extend*. Esses tipos podem ser compreendidos respectivamente por uma tupla (r_{io} , r_{if}) e uma tupla(r_{if} , r_{io}), onde r_{io} é o elemento origem e é r_{if} o elemento fim. A equação 4.2 mostra o conjunto de ligações de um requisito.

$$r_{in} = \{\, d_{in1} \,,\, d_{in2} \,,\, d_{in3} \,, \ldots \,,\, d_{ink} \mid i, n \in \mathbb{N}^* \,,\, k \in \mathbb{N} \,\} \tag{4.2}$$

4.2.2 Métrica de Escopo

Escopo de requisito pode ser representado como uma "mancha"no diagrama de requisitos. Ele contempla uma porcentagem da quantidade de requisitos, um requisito pode ser alterado se alguma mudança for realizada em algum dos requisitos incluídos em seu escopo. A partir de uma mudança em determinado requisito é acarretada mudanças em cascata nos outros requisitos que dependem dele direta ou indiretamente. Caso o requisito não tenha dependência alguma, seu escopo será 0. Do contrário, será somado, de maneira recursiva, o escopo de cada um dos requisitos que ele depende mais sua fração de participação no sistema (1/número total de requisitos) em relação aos outros requisitos. Simplificando, poderia dizer que será somado 1 por cada requisito que ele depende direta ou indiretamente e, em seguida, esse valor é divido pela quantidade total de requisitos. A equação 4.3 ilustra o cálculo do escopo (Esc) matematicamente.

$$Esc_{r_{in}} = \begin{cases} 0 \,,\, se\ r_{in} = \emptyset \\ \sum_{k=1}^{d} Esc_{r_{ik}} + \dfrac{1}{\sum_{j=1}^{t} r_{ij}} \mid r_{in} \notin Esc_{r_{ik}},\ se\ r_{in} \neq \emptyset \end{cases} \tag{4.3}$$

Onde:

- **d** é o número de dependências direta do requisito;
- **t** é o número total de requisitos.

Lê-se:

O escopo de um requisito n na versão i é igual a zero se o seu conjunto de dependências for vazio; e, caso não seja vazio, é igual ao somatório dos valores de escopo de cada um dos requisitos incluídos no conjunto de dependências mais a fração de um pelo número total de requisitos.

Por meio da métrica apresentada na equação 4.3, calcula-se o escopo do requisito e verifica se ele possui dependências. O número de elementos dos quais um requisito depende é dividido pelo número total de elementos de requisitos da versão analisada do sistema de software. Logo, com base na Figura 8, podemos dizer que na Versão 2 o requisito `Procurar` possui uma dependência e o cálculo do seu escopo será 1 dividido por 4 (número total de elementos da versão). Em outras palavras, dizemos que o cálculo do seu escopo é $Esc_{Procurar\ por\ proximo}$ + 1/4, como o escopo de `Procurar por próximo` é zero, então o resultado é 1/4 . Em termos percentuais dizemos que o escopo é igual a 25%. Na Versão 3, como aumentou o número total de requisitos do sistema em 3 e o número de dependência permaneceu o mesmo, o cálculo seria: $Esc_{Procurar\ por\ proximo}$ + 1/7 = 0 + 1/7 = 1/7.

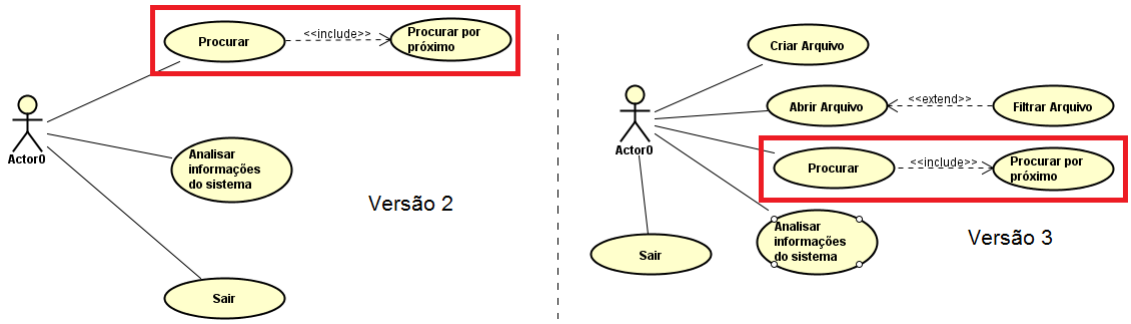

Figura 8 – Casos de uso da Versão 2 e Versão 3 do NotePad.

4.3 O Plug-in MeRS

O *plug-in* MeRS (*Measuring Requirement Scope*) foi desenvolvido com a IDE Eclipse Neon. Ele permite calcular o escopo de cada requisito de uma determinada versão de software em evolução. O MeRS realiza a leitura de um ou mais diagramas de casos de uso para que possa comparar os valores do escopo em cada versão. Para acessar o *plug-in*, deve-se buscar na barra de menu a opção `MeRS` e selecionar `Calcular Escopo` ou pressionar o seu atalho, `crtl+6`, como é representado na Figura 9.

A Figura 10 ilustra o funcionamento do *plug-in* desenvolvido. O MeRS tem como entrada diagramas de casos de uso no formato `xml`. Estes podem ser inseridos apenas colocando os seus nomes, caso estejam na pasta XML pré-definida no projeto. Caso contrário, deverá ser inserido o caminho completo de onde está cada arquivo, acompanhado do seu nome.

Uma vez lidos, tais diagramas são processados. Na etapa de processamento, identifica-se cada caso de uso, suas ligações e sua versão correspondente; bem como

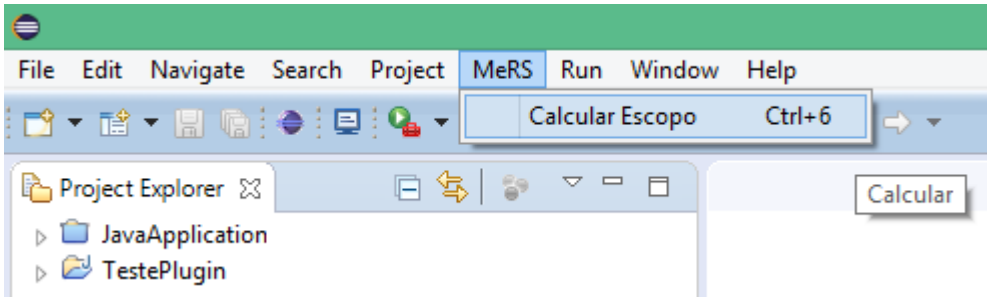

Figura 9 – Interface da ferramenta MeRS.

calcula-se os valores do escopo de requisitos de cada caso de uso referente à versão analisada. Esse cálculo é realizado por meio de uma função recursiva, a qual percorre todas as possíveis ligações a partir de determinado caso de uso e, a cada novo requisito alcançado, incrementa ao escopo calculado a porcentagem correspondente àquele requisito.

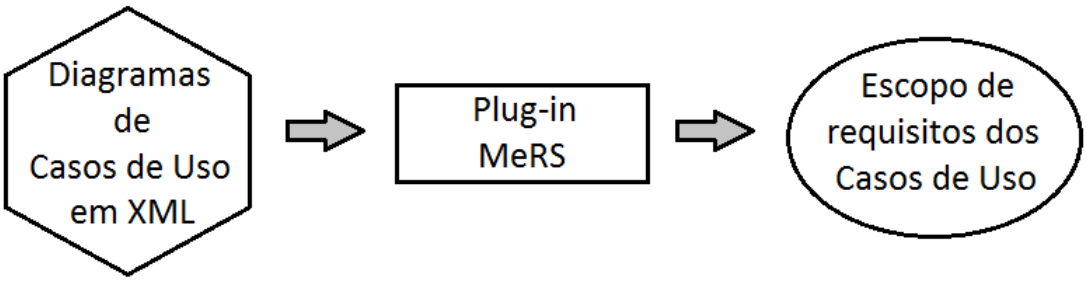

Figura 10 – Processo de obtenção do escopo de requisitos.

O MeRS fornece como saída um documento de texto contendo os nomes que identificam os requisitos e valores do escopo de cada um dos requisitos. Caso a consulta tenha sido a mais de uma versão da aplicação, o escopo dos requisitos são apresentados separados por versões ordenadas na mesma sequência que foram inseridas na entrada. Um exemplo da saída é mostrado na Figura 11.

4.4 Resumo

Este capítulo apresentou uma terminologia para o cálculo do escopo de requisitos, dando suporte para definição de uma métrica para escopo, implementada via *plug-in* para Eclipse Neon. De posse de tal ferramental, foi possível investigar a relação existente entre o escopo de requisitos e código (ver Capítulo 4).

Metrica de Escopo.txt

```
1ª Versão Inserida


|-------------------------------------------------------|
| Identificador do Requisito             | Escopo(%%)   |
|-------------------------------------------------------|
| Informações sobre o sistema            |         0,00 |
| Sair                                   |         0,00 |
| Procurar                               |        25,00 |
| Procurar por próximo                   |         0,00 |
|-------------------------------------------------------|


2ª Versão Inserida


|-------------------------------------------------------|
| Identificador do Requisito             | Escopo(%%)   |
|-------------------------------------------------------|
| Informações sobre o sistema            |         0,00 |
| Sair                                   |         0,00 |
| Procurar                               |        14,29 |
| Procurar por próximo                   |         0,00 |
| Novo Arquivo                           |         0,00 |
| Abrir Arquivo                          |        14,29 |
| Filtrar Arquivo                        |         0,00 |
|-------------------------------------------------------|

```

Figura 11 – Exemplo de saída do *plug-in* MeRS.

5 ANÁLISE DOS RESULTADOS

A garantia da conformidade do software com os seus requisitos é uma exigência básica da indústria de desenvolvimento de sistemas de software. Porém, tal conformidade nem sempre é alcançada, devido principalmente à dificuldade de garantir que o código, que dá vida aos requisitos, evolua de forma estável (*i.e.*, livre de mudanças indesejadas). Neste contexto, acreditamos que o escopo dos requisitos está diretamente ligado ao escopo dos elementos de código que, por sua vez, exerce um papel determinante no número de mudanças demandadas durante a evolução de sistemas de software (DANTAS; GARCIA; WHITTLE, 2012). Com o objetivo de identificar a relação existente entre o escopo dos requisitos e do código e, consequentemente, contribuir para minimizar mudanças indesejadas, discutimos esta potencial relação analisando os requisitos e o código de duas aplicações de software em evolução, descritas na Seção 5.2. Na Seção 5.1 apresentamos nossa motivação para conduzirmos tal estudo. O objetivo e a questão de pesquisa são apresentadas na Seção 5.3. Os procedimentos metodológicos e a discussão dos resultados do nosso estudo exploratório propriamente ditos seguem nas Seções 5.4 e 5.5, respectivamente. Nossa investigação apresenta algumas ameaças discutidas na Seção 5.6. Finalmente, o resumo do capítulo é apresentado na Seção 5.7.

5.1 Motivação

O escopo de código tem sido um dos principais responsáveis pela maioria das mudanças indesejáveis que acontecem nos artefatos de código em evolução (DANTAS; GARCIA; WHITTLE, 2012). A conclusão da pesquisa dos referidos autores está alinhada a um dos principais desafios na área de desenvolvimento de software das últimas décadas que é o estudo e a melhoria da qualidade das mudanças demandadas durante o processo de evolução do software, bem como a quantidade de mudanças desnecessárias associadas a esta evolução.

Considerando que o escopo de código possui um papel importante em determinar o número de mudanças prejudiciais a evolução do código, faz-se necessário investigar se o escopo de requisitos está de algum modo correlacionado com variações no escopo de código. Esta potencial relação abriria espaço para controlar as variações do escopo de código e consequentemente as mudanças demandadas durante a evolução a partir do documento de requisitos.

À medida que o escopo de um requisito aumenta há uma forte tendência que

estabilidade do código seja afetada negativamente, uma vez que mudança realizada nos elementos incluídos no escopo deste requisito exigirá também mudança neste requisito. Porém, a identificação de um potencial relacionamento entre o escopo dos requisitos e do código, depende fundamentalmente de rastrear os requisitos no código. Ou seja, faz-se necessário identificar onde, no código, os requisitos são implementados. Este conhecimento, além de ser muito importante para os desenvolvedores, é fundamental para correlacionar variações do escopo nos dois níveis de abstração durantes a evolução de sistemas de software.

5.2 Sistemas Analisados

O nosso estudo teve como base a análise do escopo de requisitos e de código dos sistemas MobileMedia e Notepad, com intuito de detectar a relação entre os valores dos escopos entre os níveis de requisitos e código. Os sistemas foram desenvolvidos em Java e disponibilizados em versões, de acordo com sua evolução. MobileMedia é uma Linha de Produto de Software (LPS) que possui 7 versões e foi desenvolvida para manipular mídias em dispositivos móveis. Essas mídias são fotos, músicas e filmes. A Tabela 1 apresenta as descrição das versões do MobileMedia. No MobileMedia, cada produto contém um conjunto aleatório de funcionalidades, tais funcionalidades relacionadas às mídias que ele gerencia, como: adicionar, remover, editar, ordenar, contar.

Tabela 1 – Resumo dos cenários do MobileMedia (Adaptado de Figueiredo et al. (2008)).

Versão	Descrição
V1	Núcleo do MobileMedia.
V2	Novo recurso adicionado para contar o número de vezes que uma foto foi visualizada e ordenar foto por maior frequência de visualização. Novo recurso adicionado para editar a legenda da foto.
V3	Novo recurso adicionado para permitir aos usuários especificar e visualizar suas fotos favoritas.
V4	Novo recurso adicionado para permitir aos usuários manter múltiplas cópias de fotos.
V5	Novo recurso adicionado para enviar foto para outros usuários por SMS.
V6	Novo recurso adicionado para carregar, tocar e organizar musicas. O gerenciamento da foto foi transformado em um recurso alternativo. Todas as funcionalidades estendidas também foram fornecidas.
V7	Novo recurso adicionado para gerenciar vídeos

O NotePad também é uma Linha de Produto de Software (LPS). Ele é uma aplicação de editores de texto, constituída em 6 versões, desenvolvida em um curso

sobre projeto orientado a *features*, da Universidade do Texas (ANDRADE, 2013). A Tabela 2 apresenta as funcionalidades do Notepad, versão a versão.

Tabela 2 – Resumo dos cenários do NotePad.

Versão	Descrição
V1	Núcleo do NotePad.
V2	Novo recurso adicionado para procurar por segmentos de textos e procurar por próximas ocorrências dele.
V3	Novo recurso adicionado para criar, abrir e filtrar um arquivo.
V4	Novo recurso adicionado para permitir salvar um arquivo no mesmo local em que se encontra e salvar em um local diferente. Novo recurso adicionado para imprimir um arquivo.
V5	Novo recurso adicionado para formatar um arquivo. Novo recurso adicionado para configurar fonte e permitir quebra de linha no arquivo
V6	Novo recurso adicionado para copiar, cortar e colar elementos de texto.

5.3 Objetivo e Questão de Pesquisa

Há uma lacuna referente à correlação existente entre o escopo dos requisitos e o escopo dos elementos de código de sistemas em evolução. As mudanças no escopo dos artefatos de software, em nível de requisitos ou código, ocorrem recorrentemente durante a evolução dos sistemas. Neste contexto, precisamos de um mecanismo para entender completamente o impacto que o escopo dos requisitos exerce sobre o escopo de código. A literatura já atesta que o escopo dos elementos de código é responsável por boa parte das mudanças indesejadas nos artefatos de software. Porém, nada se sabe sobre possíveis relações entre os escopos nos níveis de código e requisitos.

Neste contexto, o objetivo principal deste estudo é investigar a relação existente entre o escopo de requisitos e código. Buscamos controlar variações no escopo dos elementos de código a partir do escopo de requisitos e estaremos contribuindo para minimizar a ocorrência de mudanças indesejadas no código em evolução. Em particular, temos como premissa maior responder a seguinte Questão de Pesquisa (QP):

QP: *O escopo dos elementos de requisitos estão relacionados com o escopo dos elementos de código?*

Ao responder esta questão de pesquisa estamos disponibilizando indicadores quantitativos do impacto do escopo de requisitos sobre o escopo de código de sistemas

em evolução. Estando o escopo dos requisitos relacionado com o escopo de código, estaremos, a partir dos requisitos, contribuindo para o controle de mudanças no código.

5.4 Procedimentos Metodológicos

O presente estudo usa uma abordagem quantitativa para seu desenvolvimento, uma vez que o objetivo estabelecido é medir a correlação entre os escopos de requisitos e código. Logo, torna-se necessário investigar, por meio de medições, a existência ou não deste relacionamento. Uma pesquisa quantitativa quantifica os dados para responder um questionamento, um problema de pesquisa. A quantificação, nesse caso, se dá via análise dos resultados. Pesquisas quantitativas são usadas em situações nas quais você pretende validar estatisticamente uma hipótese sem, necessariamente, entender as motivações por trás das respostas.

Nosso estudo quantitativo foi construído apoiado em cinco etapas: (1) Mapeamento dos Requisitos; (2) Mapeamento do Escopo; (3) Medição da Métrica; (4) Desenvolvimento do *Plug-in* e (5) Avaliação dos resultados. Inicialmente, na Etapa 1, os modelos de casos de usos (ver Apêndices A e B) dos dois sistemas (Seção 5.2) foram extraídos usando a ferramenta *astah professional* (astah.net/editions/professional). A realização desse processo foi possível devido a capacidade da ferramenta de ler diagramas de casos de uso e exportar o código dele em formato `xml`.

Na Etapa 2, compreendemos o escopo dos requisitos ao longo da evolução do MobileMedia. Este mapeamento foi fundamental para entender o processo de espalhamento do escopo e prover base para a definição de uma métrica capaz de quantificá-lo. Este mapeamento foi realizado de forma manual. Na Etapa 3, uma métrica para escopo de requisitos foi definida e formalizada, utilizando o conhecimento adquirido no mapeamento do escopo ao longo da evolução do MobileMedia.

Na Etapa 4, foi desenvolvido o *plug-in* que automatiza a aplicação da métrica proposta e viabiliza o cálculo automático do escopo de requisitos a partir das versões de casos de uso de um sistema de software em evolução. O *plug-in* recebe como entrada casos de uso em `xml` e retorna o escopo de cada requisito identificado.

Na etapa de avaliação (Etapa 5), além do MobileMedia, foi introduzida a aplicação Notepad. Como estamos trabalhando com LPS, a qual possui como característica a utilização de *features* e é composta por produtos com diferentes composições, podemos escolher quais *features* participarão de uma versão e qual será a sequência dessas versões. Desse modo, escolhemos uma sequência de versões onde a quantidade de requisitos fosse crescente no decorrer da evolução.

A avaliação da Etapa 5 só foi possível devido a uma operação de rastreamento entre os níveis de requisitos e código. As funcionalidades que foram implementadas ao longo da evolução dos sistemas utilizaram diretivas de pré-processamento `ifdef` e ifndef. Dessa forma, o código relativo a cada requisito implementado vinha precedido e sucedido por tais diretivas o que facilitava a identificação do código que implementava cada um dos requisitos. Após o processo de rastreamento, o escopo de código de todos os elementos envolvidos na implementação dos requisitos foram mapeados (DANTAS; GARCIA; WHITTLE, 2012) e correlacionados com o escopo dos requisitos.

Para os cálculos do escopo foram utilizadas métricas de escopo de requisitos (ver Capítulo 4) e escopo de código (DANTAS; GARCIA; WHITTLE, 2012). A métrica definida por Dantas, Garcia e Whittle (2012) mede o escopo dos elementos de código considerando a mesma linha de raciocínio aplicada na métrica de escopo de requisitos. Para cada um dos elementos de código (*p.e.*, métodos) foi contabilizada a percentagem de cobertura, por meio da divisão dos elementos de códigos envolvidos em uma dada composição pelo número de elementos de código totais da versão do sistema.

5.5 Discussão dos Resultados

As Figuras 12 e 13 ilustram, respectivamente, a variação dos valores do escopo de requisitos ao longo da evolução nas 7 versões do MobileMedia e da mesma forma a variação ao longo da evolução nas 6 versões do NotePad. Podemos verificar que existem valores de escopo constantes nulo; outros que apresentam uma queda regular, diminuindo a cada versão com certa proporcionalidade; e um que possui aumento significativo nas últimas versões.

Para as duas aplicações, os valores de escopo constantes são reflexos de requisitos com escopo igual a zero, ou seja, referem-se aos requisitos que não possuem dependências com nenhum outro requisito e, portanto, possuem sempre o escopo nulo. Como podemos observar, os requisitos `Informações sobre o sistema` e `Sair`, Figura 14, permanecem sem nenhuma ligação com outros requisitos enquanto o sistema evolui, nesse caso, da Versão 1 para a Versão 2.

A diminuição dos valores de escopo identificada nas Figuras 12 e 13 ocorre devido ao número de requisitos adicionados ser maior do que o aumento do número de ligações do requisito em questão. Como o denominador da métrica é o número total de requisitos, quando esse valor sobe e o numerador não o acompanha, o resultado é um valor menor em relação ao anterior. A diminuição se faz regular pelo fato do número de ligações desses requisitos serem iguais, exatamente 1. Ou seja, quando a métrica divide essa única ligação pela quantidade de requisitos, a qual vai aumentando a cada versão, o valor do escopo cai de forma proporcional para esses requisitos. Um exemplo desse

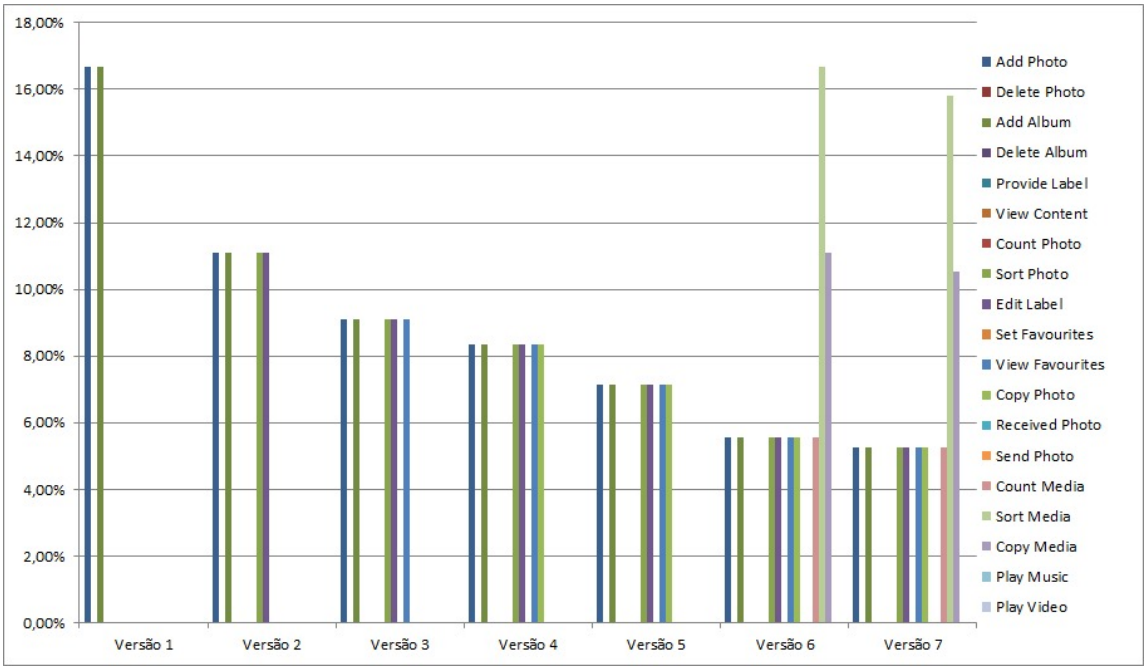

Figura 12 – Escopo de requisitos do MobileMedia.

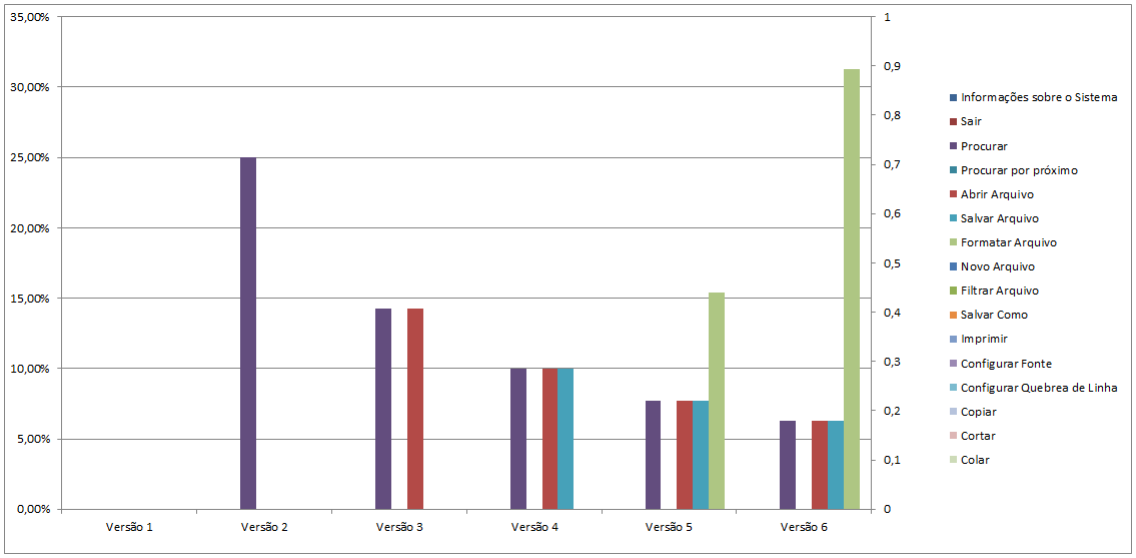

Figura 13 – Escopo de requisitos do NotePad.

fato é mostrado na Figura 15, onde o requisito `Procurar` se mantém com apenas uma ligação durante a evolução do sistema.

Para o NotePad, o requisito `Formatar Arquivo` foge às regras anteriores, pois é o único que tem um aumento na porcentagem do seu escopo. Na Versão 5 o seu número de dependências é igual a 2. Já na Versão 6, 3 novos requisitos passam a existir e são relacionados ao `Formatar Arquivo`, ficando o requisito dependente de cinco requisitos. Neste caso, diferentemente do anterior, o número de ligações (numerador) acompanha o aumento do número de requisitos (denominador), foram adicionados igualmente

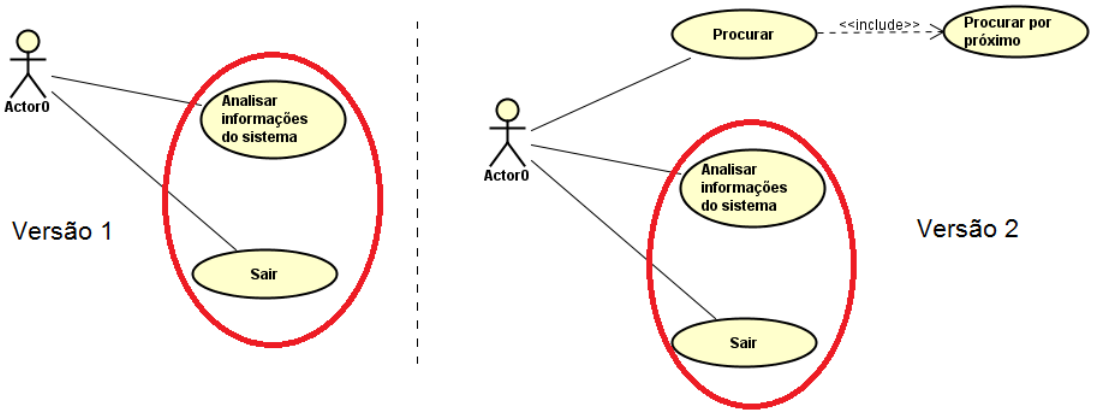

Figura 14 – Casos de uso da Versão 1 e Versão 2 do NotePad.

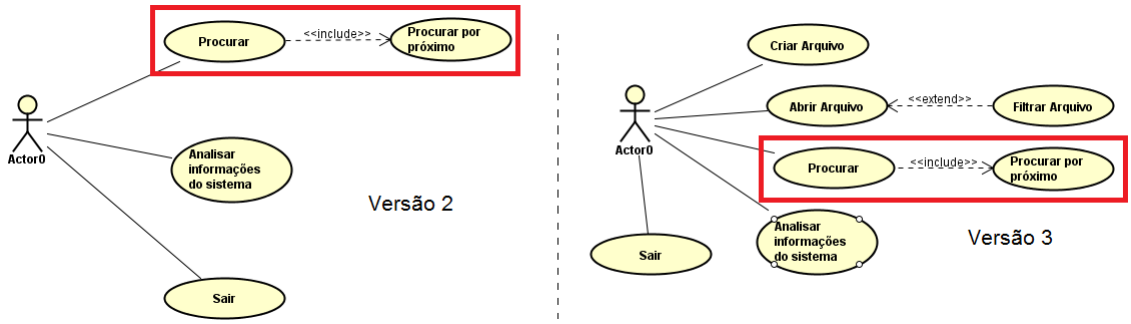

Figura 15 – Casos de uso da Versão 2 e Versão 3 do NotePad.

três novos requisitos ao sistema e três ligações ao requisito citado, resultando em um aumento na porcentagem do escopo desse requisito em relação a versão anterior, como é ilustrado na Figura 16.

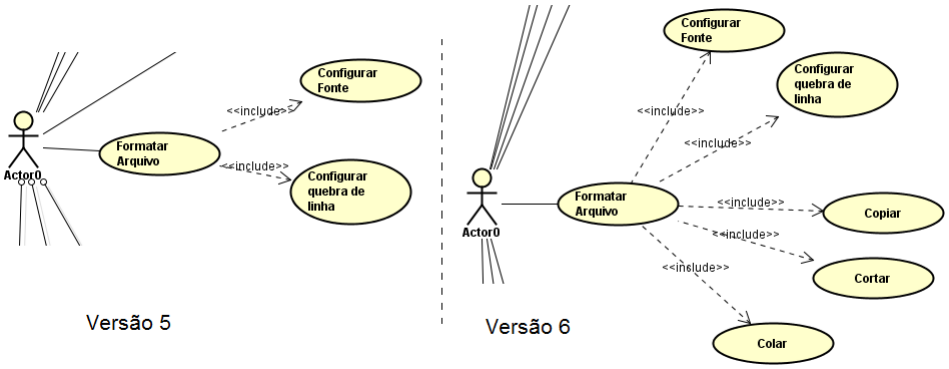

Figura 16 – Casos de uso da Versão 5 e Versão 6 do NotePad.

Para o MobileMedia, o escopo dos requisitos apresenta uma queda em sua maioria. No entanto, nas verões 6 e 7, o escopo das funcionalidades `CopyMedia` e `PlayVideo` sofreram uma significativa elevação, devido principalmente, ao forte acoplamento com os requisitos existentes.

O Escopo dos Requisitos e do Código estão relacionados. Conforme podemos observar nas Figuras 17 e 18 as variações do escopo dos requisitos sinalizam como o escopo dos elementos de código irá se comportar. O escopo do requisito `Formatar Arquivo` da Versão 5 do Notepad aumenta consideravelmente na Versão 6 da aplicação, ao mesmo tempo, o escopo de código apresenta o mesmo comportamento. Os demais permanecem com escopo em queda constante tanto no nível de requisitos como de código. A maior queda acontece com o requisito `Procurar`, que cai consideravelmente da Versão 2 para a Versão 3, isso devido ao aumento do número total de requisitos (denominador da fórmula), esse número quase dobra o seu valor, vai de 4 para 7 requisitos. Um comportamento similar pode ser observado no MobileMedia. Conforme ilustra a Figura 18, o escopo dos requisitos atinge seu pico nas versões 6 e 7 do MobileMedia, que são as versões com um maior escopo de código. Para ambos os cenários, a explicação para tal comportamento reside no fato de que as dependências já identificadas em nível de requisitos, ao ser implementadas, demandam um número extremamente elevados em termos de composições de elementos de código. Como base neste estudo, podemos afirmar em resposta a nossa questão de pesquisa (Seção 5.3) que o escopo em nível de requisitos tende a afetar o escopo de código.

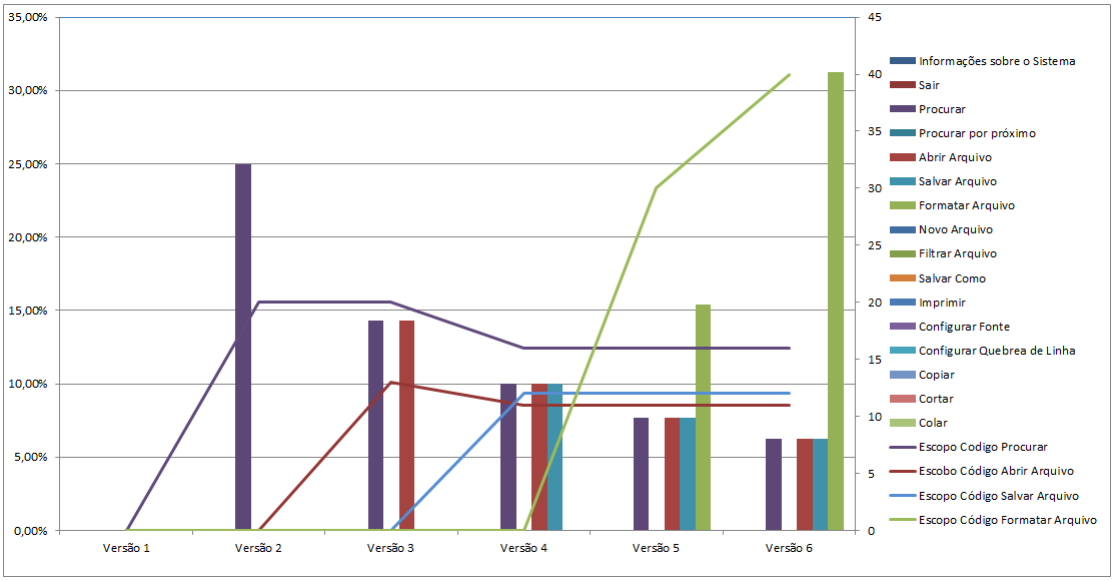

Figura 17 – Gráfico da porcentagem de escopo de requisitos e código do NotePad.

5.6 Ameaças a Validação

Fatores difíceis de serem controlados e que podem afetar o resultado da investigação são conhecidos como ameaças à validade interna de um estudo. A validade interna, portanto, está preocupada em atribuir a causalidade à intervenção e descartar possíveis explicações alternativas.

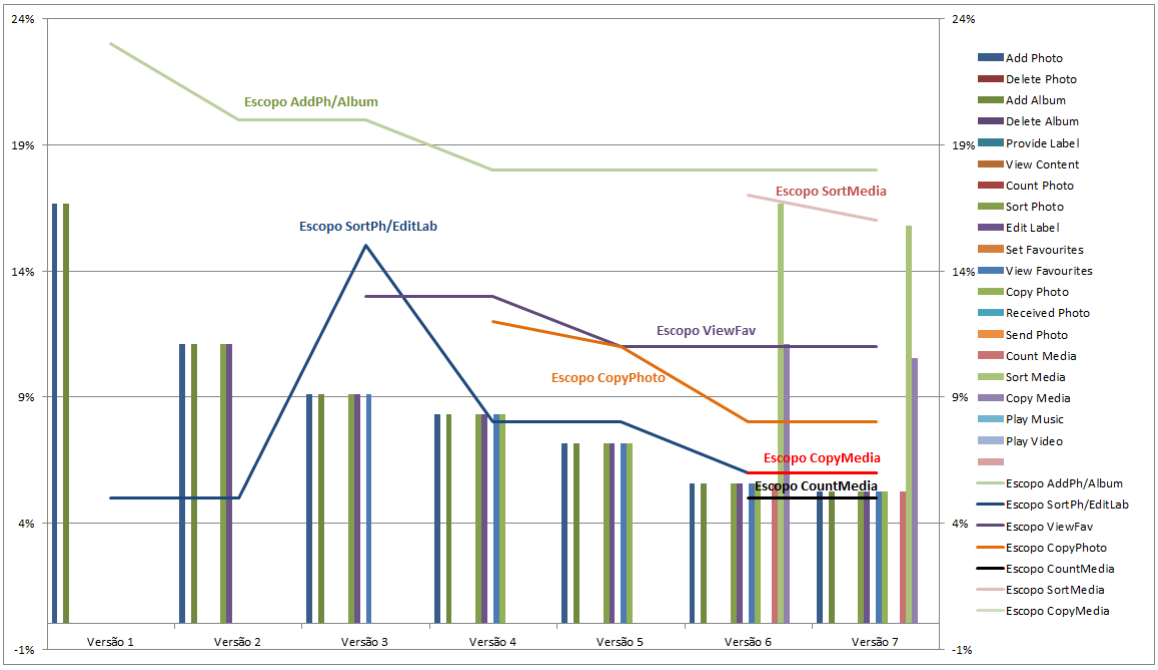

Figura 18 – Gráfico da porcentagem de escopo de requisitos e código do MobileMedia.

Um segundo aspecto da validade do estudo diz respeito à medida em que o estudo nos permite generalizar os resultados para aplicações de software diferentes daquelas a partir das quais a amostra foi desenhada ou para populações semelhantes em diferentes configurações ou em momentos diferentes. Isto é conhecido como validade externa. A validade externa é sobre como os resultados são significativos quando aplicados ao mundo real.

O estudo aqui apresentado possui algumas ameaças que precisam ser discutidas. No que diz respeito às ameaças internas, com o objetivo de minimizar as ameaças associadas à definição da métrica, foi realizada uma validação matemática da métrica proposta. Os casos de usos do NotePad foram gerados por um integrante externo, mestre em Computação, com intuito de não obter casos de uso tendenciosos ao favorecimento do resultado.

Para minimizar a ameaça externa a generalização dos resultados, foram utilizados sistemas reais, trabalhados em estudos anteriores e com casos de uso bem definidos, ou seja, casos de usos bem documentados durante todas as fases evolutivas da aplicação. Além disso, as aplicações também possuem cenários ricos para analisar a variação do escopo em diferentes aspectos, tais como os níveis de abstração de requisito e de código. Desta forma, por meio dessa variação foi possível observar as diferenças entre os resultados. No entanto, ainda é necessário realizar outras avaliações com outras aplicações similares de modo a fornecer maiores evidências sobre as nossas conclusões.

5.7 Resumo do Capítulo

Este capítulo apresentou um estudo sobre o impacto do escopo de requisitos no escopo do código. Uma solução computacional, baseada em *plug-in*, foi construída para a mensuração do escopo dos requisitos. Com base na análise dos resultados foi verificado que o escopo de requisitos está diretamente relacionado com o escopo dos elementos de código. No Capítulo 6, apresentamos as nossas considerações finais.

6 CONSIDERAÇÕES FINAIS

Este trabalho relatou uma investigação exploratória sobre a relação existente entre o escopo de requisitos e o escopo de código em artefatos de software em evolução. Através deste estudo preliminar, foi possível analisar a correlação entre requisitos e código de sistemas tendo por base o escopo de cada um. Torna-se evidente que os programadores devem ter ciência do impacto que o escopo de requisitos gera sobre o código e de como gerenciá-lo de modo a minimizar a ocorrência de mudanças indesejáveis durante a evolução.

A investigação só foi possível graças à formalização de uma métrica para quantificação do escopo de código. Uma vez formalizada, a métrica serviu de base para a implementação do *plug-in* MeRS. Por meio do MeRS, os desenvolvedores podem fornecer a versão `xml` dos casos de uso e terão o valor do escopo dos requisitos do sistemas em foco. Eles poderão gerenciar os valores obtidos de forma eficiente para evitar mudanças indesejadas no código durante a evolução do sistema, uma vez que existe uma correlação entre os escopos de requisitos e de código, ou seja, uma boa gestão do escopo de requisitos propicia um bom controle do escopo de código e, transitivamente, uma maior chance de não ocorrer mudanças indesejadas nos elementos de composição do código.

6.1 Dificuldades encontradas

Durante o desenvolvimento do trabalho encontramos algumas limitações. Em particular, entendemos que o documento de requisitos é essencial para que seja possível avaliar impacto de qualquer alteração solicitada e assim facilitar a manutenção do sistema em questão. No entanto, não encontramos aplicações em evolução com a documentação de requisitos disponível e adequada.

Supostamente, a documentação formal deve descrever o sistema e, assim, tornar sua compreensão mais fácil para as pessoas que fazem as mudanças. Porém, na prática, identificamos que a documentação formal, principalmente a de requisitos, nem sempre existe e quanto existe nem sempre é atualizada, e, portanto, não reflete exatamente o código do programa.

Outra limitação do nosso trabalho foi encontrar aplicações com várias versões, com artefatos nos níveis de requisitos e código Java disponíveis para análise. A ausência

de aplicações de software com a documentação de requisitos necessária dificultou o aprofundamento do nosso estudo e de nossas conclusões.

6.2 Trabalhos Futuros

Como trabalhos futuros, vislumbramos a inclusão de novas aplicações para validar os resultados aqui apresentados, bem como a sugestão de novas heurísticas que expressem o relacionamento existente entre os escopos nos dois níveis de abstração.

Ainda para trabalhos futuros, poderiam ser investigadas aplicações que não são Linhas de Produto de Software, pois essas novas aplicações possuem uma evolução mais fixa em relação às LPS, isto é, não tem a flexibilidade de inserção e remoção de componentes durante a evolução do sistema como as LPS. Com uma análise detalhada sobre essas novas aplicações, poderia ser afirmado que o escopo de requisitos e o escopo de componentes de código estão relacionados tanto em sistemas de LPS como em outros sistemas em evolução comuns.

Também cabe uma análise estatística dos dados coletados. Chegamos a aplicar um teste de correlação. No entanto, o tamanho da amostra limitou a generalização dos resultados. O estudo estatístico mais aprofundado poderia indicar a necessidade de ajustes nas métricas. Finalmente, a evolução da ferramenta tem como meta a inclusão de outras formatos como leitura além do `xml`.

REFERÊNCIAS

ALI, M. J. Metrics for requirements engineering. *UMEA University*, 2006. Citado na página 20.

ANDRADE, H. Software product line architectures: Reviewing the literature and identifying bad smells. 2013. Citado na página 39.

ANTONIOL, G. et al. Recovering traceability links between code and documentation. *IEEE transactions on software engineering*, IEEE, v. 28, n. 10, p. 970–983, 2002. Citado na página 27.

BARBOZA, L. da S.; FILHO, A. d. A. G.; SOUZA, R. A. C. D. Uma proposta sobre rastreabilidade de requisitos legais no processo de contratação de soluções de ti na administração pública federal. In: *ER@ BR*. [S.l.: s.n.], 2013. Citado na página 29.

BOKHARI, M. U.; SIDDIQUI, S. T. Metrics for requirements engineering and automated requirements tools. In: CITESEER. *Proceedings of the 5th National Conference*. [S.l.], 2011. Citado 2 vezes nas páginas 20 e 21.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. [S.l.]: Elsevier Brasil, 2006. Citado na página 16.

BORGES, E. P. Um modelo de medição para processos de desenvolvimento de software. *Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais. Belo Horizonte*, 2003. Citado na página 31.

BYUN, J. et al. Metrics for measuring the consistencies of requirements with objectives and constraints. *Requirements Engineering*, Springer, v. 19, n. 1, p. 89–104, 2014. Citado na página 21.

CHRISTOPHER, D. F. X.; CHNADRA, E. Analyzing the efficacy of requirements stability based on function point modeling. In: ESRSA PUBLICATIONS. *International Journal of Engineering Research and Technology*. [S.l.], 2012. v. 1, n. 9 (November-2012). Citado na página 22.

CLELAND-HUANG, J. et al. Automating speculative queries through event-based requirements traceability. In: IEEE COMPUTER SOCIETY. *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*. [S.l.], 2002. p. 289–298. Citado na página 27.

CLELAND-HUANG, J.; SCHMELZER, D. Dynamically tracing non-functional requirements through design pattern invariants. In: *Workshop on Traceability in Emerging Forms of Software Engineering, in conjunction with IEEE International Conference on Automated Software Engineering*. [S.l.: s.n.], 2003. v. 10. Citado na página 27.

CYSNEIROS, L. M. *Requisitos não funcionais: da elicitação ao modelo conceitual*. Tese (Doutorado) — PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, 2001. Citado 2 vezes nas páginas 14 e 15.

DANTAS, F.; GARCIA, A.; WHITTLE, J. On the role of composition code properties on evolving programs. In: ACM. *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. [S.l.], 2012. p. 291–300. Citado 4 vezes nas páginas 10, 11, 37 e 41.

DELATER, A. *Tracing requirements and source code during software development*. Tese (Doutorado), 2013. Citado 2 vezes nas páginas 24 e 29.

ECKHARDT, J.; VOGELSANG, A.; FERNÁNDEZ, D. M. Are non-functional requirements really non-functional?: an investigation of non-functional requirements in practice. In: ACM. *Proceedings of the 38th International Conference on Software Engineering*. [S.l.], 2016. p. 832–842. Citado na página 15.

EGYED, A. A scenario-driven approach to trace dependency analysis. *IEEE Transactions on Software Engineering*, IEEE, v. 29, n. 2, p. 116–132, 2003. Citado na página 27.

EGYED, A.; GRUNBACHER, P. Automating requirements traceability: Beyond the record & replay paradigm. In: IEEE. *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*. [S.l.], 2002. p. 163–171. Citado na página 27.

FIGUEIREDO, E. et al. Evolving software product lines with aspects: an empirical study on design stability. In: ACM. *Proceedings of the 30th international conference on Software engineering*. [S.l.], 2008. p. 261–270. Citado 2 vezes nas páginas 6 e 38.

GENVIGIR, E. C. Um modelo para rastreabilidade de requisitos de software baseado em generalização de elos e atributos. *Instituto Nacional de Pesquisas Espaciais*, 2009. Citado 2 vezes nas páginas 25 e 26.

HAYES, J. H.; DEKHTYAR, A.; OSBORNE, J. Improving requirements tracing via information retrieval. In: IEEE. *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*. [S.l.], 2003. p. 138–147. Citado na página 27.

HAZAN, C.; LEITE, J. C. S. P. Indicadores para a gerência de requisitos. In: *WER*. [S.l.: s.n.], 2003. p. 285–301. Citado na página 22.

MARCUS, A.; MALETIC, J. I. Recovering documentation-to-source-code traceability links using latent semantic indexing. In: IEEE. *Software Engineering, 2003. Proceedings. 25th International Conference on*. [S.l.], 2003. p. 125–135. Citado na página 27.

MARQUES, A. B. et al. Um estudo experimental sobre abordagens de apoio à rastreabilidade de requisitos. *VI WAMPS, Campinas–SP*, p. 158–165, 2010. Citado na página 24.

MONPERRUS, M. et al. Automated measurement of models of requirements. *Software Quality Journal*, Springer, v. 21, n. 1, p. 3–22, 2013. Citado na página 21.

PAREDES, L. N.; ZORZO, S. D. Privacy mechanism for applications in cloud computing. *IEEE Latin America Transactions*, IEEE, v. 10, n. 1, p. 1402–1407, 2012. Citado na página 32.

PAVIOTTI, C. R. *MCReF:MÉTRICA DE COMPLEXIDADE DE REQUISITOS FUNCIONAIS*. Dissertação (Mestrado) — UNIMEP, 2011. Citado na página 10.

PIMENTEL, J. et al. Towards anticipating requirements changes through studies of the future. In: IEEE. *Research Challenges in Information Science (RCIS), 2011 Fifth International Conference on*. [S.l.], 2011. p. 1–11. Citado na página 19.

PIMENTEL, J. et al. Anticipating requirements changes-using futurology in requirements elicitation. *International Journal of Information System Modeling and Design (IJISMD)*, IGI Global, v. 3, n. 2, p. 89–111, 2012. Citado na página 20.

PINHEIRO, F. A. Formal and informal aspects of requirements tracing. In: *WER*. [S.l.: s.n.], 2000. p. 1–21. Citado na página 27.

POHL, K. Pro-art: Enabling requirements pre-traceability. In: IEEE. *Requirements Engineering, 1996., Proceedings of the Second International Conference on*. [S.l.], 1996. p. 76–84. Citado na página 27.

POHL, K. *Process-centered requirements engineering*. [S.l.]: John Wiley & Sons, Inc., 1996. Citado na página 27.

RIBEIRO, F. G. C. *Modelagem de requisitos de software de tempo-real usando SysML e MARTE*. 2013. Citado na página 16.

SAYÃO, M.; LEITE, J. C. S. do P. Rastreabilidade de requisitos. *RITA*, v. 13, n. 1, p. 57–86, 2006. Citado 2 vezes nas páginas 24 e 29.

SHERBA, S. A.; ANDERSON, K. M.; FAISAL, M. A framework for mapping traceability relationships. In: *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*. [S.l.: s.n.], 2003. p. 32–39. Citado na página 27.

SILVA, I. A. D. et al. Lighthouse: coordination through emerging design. In: ACM. *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*. [S.l.], 2006. p. 11–15. Citado na página 32.

SILVA, R. P. d. *Modelo de rastreabilidade de requisitos aplicada a gestão de projetos em métodos ágeis*. Dissertação (Mestrado) — Brasil, 2016. Citado 2 vezes nas páginas 24 e 29.

SOMMERVILLE, I. *Engenharia de Software, 9ª edição, Tradução: Kalinka Oliveira, Ivan Bosnic*. [S.l.: s.n.], 2011. Citado 4 vezes nas páginas 10, 14, 17 e 18.

SONG, L. G. The requirements stability of control and measure about software project. In: TRANS TECH PUBL. *Applied Mechanics and Materials*. [S.l.], 2012. v. 121, p. 4218–4223. Citado na página 22.

SPANOUDAKIS, G.; ZISMAN, A. Software traceability: a roadmap. *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Co, v. 3, p. 395–428, 2005. Citado 2 vezes nas páginas 27 e 28.

SPANOUDAKIS, G. et al. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, Elsevier, v. 72, n. 2, p. 105–127, 2004. Citado na página 27.

TRINDADE, G. O. da; LUCENA, M. Rastreabilidade de requisitos em metodologias ágeis: um estudo exploratório. 2016. Citado 2 vezes nas páginas 24 e 29.

VISION, C. *Astah Professional*. 2017. Disponível em: <http://astah.net/editions/professional>. Citado na página 17.

ZISMAN, A. et al. Tracing software requirements artifacts. In: *Software Engineering Research and Practice*. [S.l.: s.n.], 2003. p. 448–455. Citado na página 27.

APÊNDICE A – Diagrama de Casos de Uso do NotePad

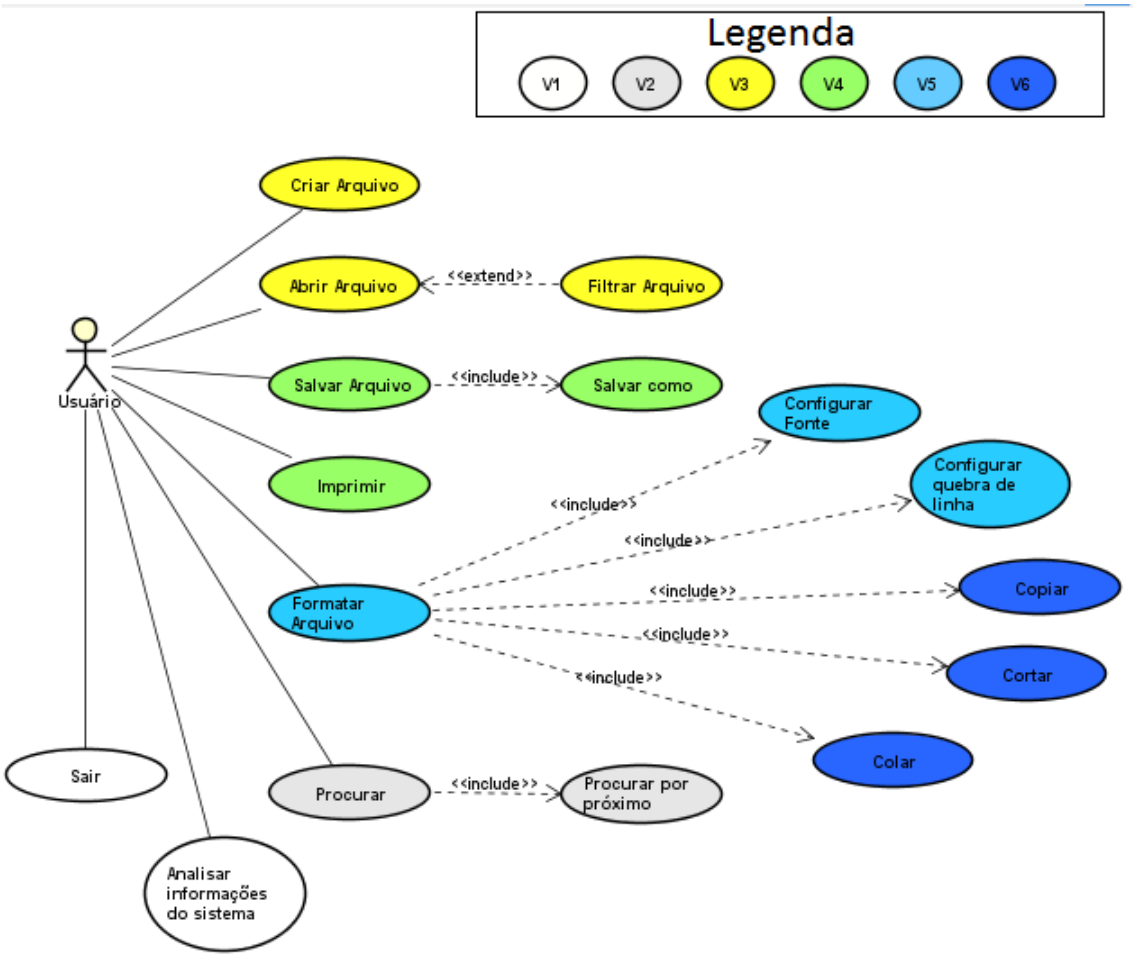

Figura 19 – Diagrama de Casos de Uso do NotePad com versões identificadas.

APÊNDICE B – Diagrama de Casos de Uso do MobileMedia

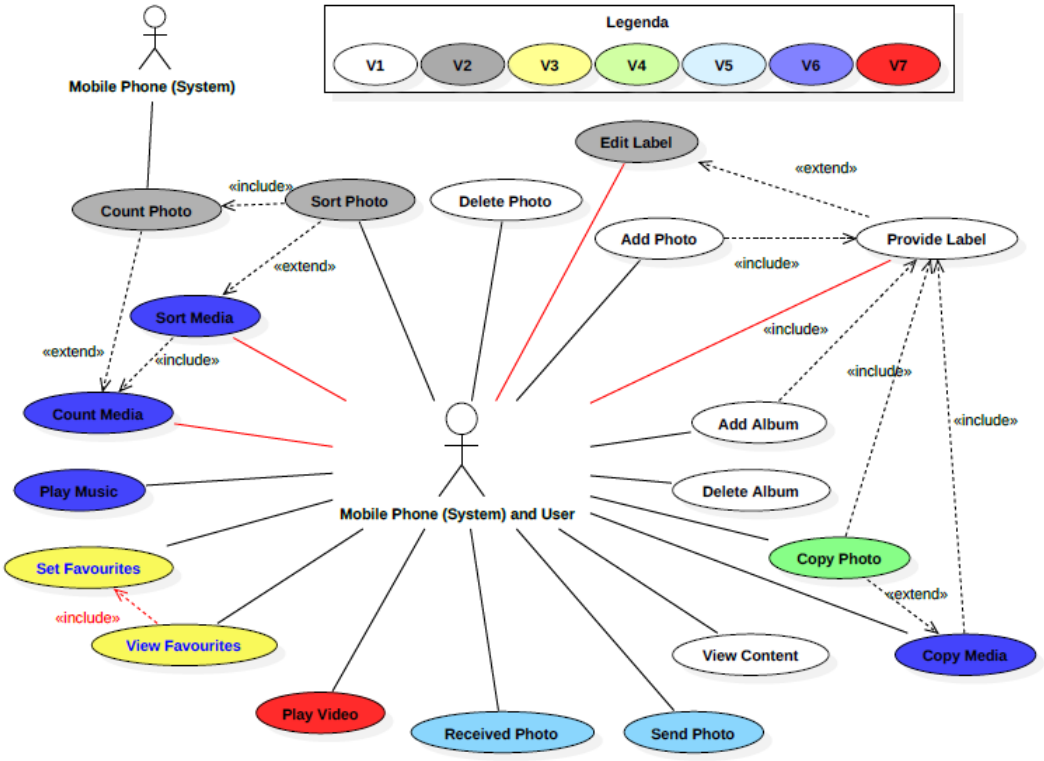

Figura 20 – Diagrama de Casos de Uso do MobileMedia com versões identificadas.