Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Visualizando Comunalidades e Variabilidades em uma Família de Produtos de Software

Denise Alves da Costa

Número de Ordem PPGCC: M001

Teresina-PI, Agosto de 2017

Denise Alves da Costa

Visualizando Comunalidades e Variabilidades em uma Família de Produtos de Software

Dissertaçao de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Orientador: Pedro de Alcântara dos Santos Neto

Teresina-PI

Agosto de 2017

Denise Alves da Costa
Visualizando Comunalidades e Variabilidades em uma Família de Produtos de Software/ Denise Alves da Costa. – Teresina-PI, Agosto de 2017-
59 p. : il. (algumas color.) ; 30 cm.

Orientador: Pedro de Alcântara dos Santos Neto

Dissertaçao (Mestrado) – Universidade Federal do Piauí – UFPI
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação, Agosto de 2017.

1. Linha de Produto de Software. 2. Abordagem Extrativa. 3. Detecção de Variabilidades 4. Visualização de Software I. Pedro de Alcântara dos Santos Neto. II. Universidade Federal do Piauí. III. Visualizando Comunalidades e Variabilidades em uma Família de Produtos de Software

CDU 02:141:005.7

Denise Alves da Costa

Visualizando Comunalidades e Variabilidades em uma Família de Produtos de Software

Dissertaçao de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

. Teresina-PI, de de 20 :

Pedro de Alcântara dos Santos Neto
Orientador

Professor
Raimundo Santos Moura - UFPI

Professor
Thiago Carvalho de Sousa - UESPI

Professor
Lincoln Souza Rocha - UFC

Teresina-PI
Agosto de 2017

Agradecimentos

Agradeço a Deus acima de todas as coisas por ter me dado a capacidade de realizar este trabalho e ter me proporcionado todas as condições necessárias para esse fim.

Agradeço ao meu pai, Ruimar Portela Costa, pelo carinho, à minha mãe, Elzimar Alves da Silva, por sua grande dedicação.

Agradeço ao meu esposo, Wislanildo Gonçalves Dias Júnior, pelo grande incentivo e apoio que me proporcionou desde o início desse projeto.

Agradeço ao meu orientador, Pedro de Alcântara dos Santos Neto, pelas oportunidades concedidas e ensinamentos repassados.

À Universidade Federal do Piauí (UFPI) por ter aberto as portas para a realização deste projeto.

Aos amigos e colegas da UFPI pela parceria e ajuda nos mais diversos momentos.

Aos professores do programa que tiveram o cuidado e a presteza em ministrar de forma a enriquecer nossos conhecimentos.

A CAPES e FAPEPI pelo apoio financeiro para realização deste trabalho de pesquisa.

A todos que de alguma forma tornou esse sonho possível, muito obrigada!

"A pessoa sábia está sempre ansiosa e pronta para aprender."
(Provérbios 18:15)

Resumo

Linha de Produto de Software (LPS) é uma forma de desenvolvimento de software, na qual existe uma coleção de sistemas que compartilham um conjunto de artefatos, mas possuem componentes customizados para clientes específicos. Muitas são as vantagens de se utilizar uma LPS, como por exemplo, a melhor gestão das aplicações. Entretanto, muitos também são os desafios para a sua adoção por parte de uma empresa. Embora a engenharia de LPS ofereça menores custos à fábrica de software, a sua implantação ainda é um processo oneroso. Existem hoje diversas abordagens distintas para a implantação da linha de produto em um processo de desenvolvimento, definidas para as diferentes necessidades do mercado. Uma delas denomina-se abordagem extrativa, na qual a LPS será construída a partir de produtos de software individuais pré-existentes. Empresas que já possuem diversas variantes de um produto, por exemplo, têm dificuldade em adotar LPS devido, dentre outros motivos, aos próprios custos decorrentes da migração de produtos individuais para uma plataforma única de desenvolvimento. Para realizar essa transição, são necessárias algumas atividades como identificar o que é comum e o que é variante dentre os produtos de software, reunir todo o código em um único local e prover um ambiente único de desenvolvimento. A fim de auxiliar o processo de adoção de LPS, este trabalho propõe um método para apoiar as empresas na primeira das etapas acima citadas, que é identificar as comunalidades e variabilidades existentes nos produtos que fazem parte do portfólio da empresa. Para alcançar esse objetivo, o método propõe uma ferramenta de visualização dessas comunalidades e variabilidades. De posse dessa informação, o gerente de projetos possui maiores subsídios para guiar o planejamento da reengenharia de seus produtos. Também foi realizada uma avaliação experimental na qual pode-se observar a eficiência da ferramenta desenvolvida sendo utilizada por acadêmicos e profissionais da área.

Palavras-chaves: Linha de Produto de Software. Abordagem Extrativa. Detecção de Variabilidades. Visualização de Software.

Abstract

Software Product Line (SPL) is an approach to develop collections of systems that share a set of artefacts, but also have customizations to individual customers. Despite the advantages of the SPL approach, such as improvement of the applications management, the adoption is still a challenge. Although SPL engineering can be applied to save development costs, its implantation is still a costly process. Nowday, there are three approachs to product line implatation in a development process that are defined according to the varying need of the companies. In the extractive approach, the SPL is built from single pre-existing software product. Companies with already existing variants of a product may have difficulties to adopting SPL due, among others, to own migration cost from single products to a single development platform. Some activities must be executed to perform the transition, as to identify the common and the variable elements among the products, to gather all the code in a unique local and provide an environment to integrated development. In order to reduce the adoption costs, this work proposes a method to support the companies in the first activity presented above, that is the identification of commonalities and variabilities of the products that are part of the company product portfolio. To achieve this objective, the method proposes visualization techniques to support the process. This information can help stakeholders to guide the reengineering planning of your products.

Keywords: Software Product Line. Extractive Approach. Variabilities Detection. Software Visualization.

Lista de ilustrações

Lista de abreviaturas e siglas

AST	Árvore Sintática Abstrata
LPS	Linha de Produto de Software
PDV	Ponto de Venda
UFPI	Universidade Federal do Piauí

Sumário

1 Introdução

De acordo com (IEEE, 1990) produto de software é um conjunto completo de programas, procedimentos, documentos e dados prontos para entrega a um usuário ou cliente. Devido à sua grande capacidade de contribuição, os produtos de software estão ganhando cada vez mais espaço nas empresas e instituições em todo o mundo. Fábricas de software possuem cada vez mais clientes, sendo necessárias, portanto, metodologias e ferramentas apropriadas para gerenciar seus produtos de maneira adequada.

A Engenharia de Software é a disciplina que busca oferecer um caráter profissional ao desenvolvimento do software. Ela estabelece alguns modelos e diretrizes para a construção de um sistema conforme a qualidade exigida pelo cliente (SOMMERVILLE, 2011). Ainda de acordo com o autor Sommerville (2011), o desenvolvimento de software com foco no reuso de código tornou-se amplamente usado. Nas fábricas de software, essa necessidade decorre do fato de um grupo de clientes compartilharem as mesmas demandas, ou demandas semelhantes. Por exemplo, se uma fábrica tem um cliente com um sistema de Ponto de Venda (PDV) e fecha negócio com outro cliente que também precisa de um PDV, ela poderá reusar o projeto já existente e fazer as adaptações necessárias para o novo cliente.

Dentro dessa questão do reuso, surgiu o conceito de Linha de Produto de Software (LPS). Trata-se de um paradigma de desenvolvimento de sistemas, no qual para um grupo de produtos semelhantes (chamado de família), existe uma plataforma comum (artefatos em comum) para reuso e pontos variáveis de acordo com a necessidade de cada cliente (POHL; BöCKLE; LINDEN, 2005). A adoção da LPS em contraste com o paradigma tradicional (um projeto diferente a cada novo cliente), traz diversas vantagens para a fábrica de software, dentre elas o menor custo com manutenção.

Apesar da área de LPS se apresentar como uma solução eficiente para a indústria, muitas organizações ainda não a adotaram. Diversas equipes iniciam desenvolvendo um sistema único para um cliente e à medida que novos contratos são fechados, novos sistemas vão sendo criados. Nessa fase, para fazer a migração para uma LPS, é necessário um grande investimento extra por parte da organização (LINDEN; ROMMES; SCHMID, 2007).

O apoio no processo de implantação de LPS tem sido tema de diversas pesquisas acadêmicas. Assunção e Vergilio (2014) construíram um mapeamento sistemático de trabalhos nessa área. Os autores conseguiram relacionar um total de 141 trabalhos entre os anos de 2002 e 2013, com um crescimento considerável a partir de 2011, mostrando ser uma área em ascensão. Dentre várias classificações que os autores utilizaram para os trabalhos, uma é realizada de acordo com a fase do processo de migração que os mesmos abordam, que são: Fase de Detecção, referente à localização de características (*features*); Fase de

Análise, referente à identificação de comunalidades e variabilidades; e por fim, Fase de Transformação, referente à própria implantação da LPS. Considerando essa divisão, este trabalho de mestrado é focado na Fase de Análise, a qual ficou em segundo lugar no número de trabalhos desenvolvido nesse período (47%) (ASSUNÇÃO; VERGILIO, 2014).

1.1 Motivação e Justificativa

É comum encontrar empresas de software que possuam diversos clientes com produtos semelhantes, mas com distinção em partes dos seus componentes. Isso decorre principalmente de solicitações de customizações específicas para clientes, criando assim versões alternativas de funcionalidades de um sistema para atender a uma demanda bem específica. Nesse contexto, o uso de LPS poderá ser iniciado a partir de uma abordagem de implantação chamada extrativa (KRUEGER, 2001). Essa abordagem provê o reuso dos artefatos já existentes e portanto, tende a ser menos onerosa para a empresa, se comparada com as demais (apresentadas na Seção 2.2) (CLEMENTS, 2002). Muitos estudos tem sido apresentados dentro deste campo, em diversos locais, como conferências, workshops, revistas científicas, dissertações de mestrado e teses de doutorado, dentre outros, de acordo com Assunção et al. (2017), o que mostra grande interesse nessa área. Ainda de acordo com os autores, uma das áreas mais promissoras para pesquisas futuras é a criação de ferramentas de apoio e automação. Esses motivos levaram à concepção deste trabalho com foco em tal abordagem.

Ainda que a abordagem extrativa se mostre como a solução mais promissora, a mesma também tem os seus desafios. Para fazer a extração da linha de produto a partir das aplicações já existentes é necessário executar as tarefas relacionadas a seguir (não necessariamente nessa ordem), além daquelas próprias da construção de qualquer LPS:

- Identificar as comunalidades e as variabilidades dos sistemas. Em outras palavras, é necessário identificar os artefatos que pertencem a todas as aplicações e que podem ser generalizados para quaisquer produtos e os artefatos que estão presentes em apenas uma parte das aplicações, ou que não podem ser generalizados para todos os produtos a serem gerados. Na teoria de Linha de Produtos de Software essas tarefas fazem parte do processo chamado Engenharia de Domínio (POHL; BöCKLE; LINDEN, 2005).

- Levantamento de *features*. Todas as funcionalidades pertencentes a todos os sistemas devem ser identificadas.

- Mapeamento de *features*. Construir a associação entre os artefatos e as *features* identificadas.

A partir das informações anteriormente citadas é possível então construir a Linha de Produto de Software, com todos os artefatos necessários e com a definição dos produtos a serem gerados.

Como já dito anteriormente, apesar de seus diversos benefícios, existem grandes desafios para a adoção de LPS, como o grande consumo de tempo e o alto esforço nas atividades descritas acima (KRUEGER; JUNGMAN, 2009). Nos últimos anos, muitos pesquisadores da engenharia de software têm se voltado a desenvolver pesquisas em LPS para impulsionar essa adoção (DURA; YILMAZ, 2009), inclusive com o desenvolvimento de ferramentas para oferecer o devido suporte (JÚNIOR, 2008) (MARTINEZ et al., 2017). Tendo em vista essas questões, o trabalho apresentado nesta dissertação de mestrado propõe mais um ferramenta para auxiliar as empresas no processo de planejamento da adoção de LPS utilizando a abordagem extrativa, por meio da visualização das comunalidades e variabilidades dos produtos atuais.

Este trabalho foi aprovado e apresentado no VI Workshop de Teses e Dissertações do Congresso Brasileiro de Software (WTDSoft 2016). Essa participação no WTD foi importante para discutir a temática junto a especialistas da área e contribuiu para a reformulação do projeto.

1.2 Objetivos

O trabalho de pesquisa aqui apresentado visa propor um método automatizado para auxiliar a detecção das diferenças e semelhanças entre produtos distintos de uma mesma família de software, o que será feito por meio de técnicas de visualização. Uma ferramenta deve exibir uma estrutura unificada de todos os produtos destacando os elementos em comuns e os elementos variantes.

O objetivo geral deste trabalho é criar um método e uma ferramenta que auxilie as empresas de software a fazer a transição de produtos individuais para uma Linha de Produto de Software, a fim de reduzir os custos de implantação de LPS em uma empresa que já possua produtos desenvolvidos com variabilidades para diferentes clientes.

A fim de atingir o objetivo proposto, foram também definidos alguns objetivos específicos:

1. Identificar os artefatos comuns e variantes de uma família de produtos;
2. Unificar as estruturas dos projetos de cada produto;
3. Prover a visualização da estrutura única dos projetos, diferenciando as comunalidades das variabilidades;
4. Permitir a visualização do quanto determinado elemento do projeto é comum a todos os projetos, ou a apenas uma parte deles;

5. Realizar avaliações do método proposto.

1.3 Metodologia de pesquisa

Nesta seção serão explicadas as etapas para o desenvolvimento da pesquisa e como cada etapa foi realizada. A pesquisa está dividida em 6 fases, conforme relacionado abaixo:

- Fase 1: Identificar os artefatos dos sistemas de software existentes a serem analisados
- Fase 2: Desenvolver o método a ser utilizado para a detecção das comunalidades e variabilidades nos artefatos selecionados
- Fase 3: Implementar uma ferramenta para a identificação e classificação em comunalidade ou variabilidade
- Fase 4: Avaliar a eficiência do método e ferramenta em um contexto acadêmico
- Fase 5: Avaliar a eficiência do método e ferramenta em um contexto industrial

1.3.1 Fase 1: Identificar os artefatos dos sistemas de software existentes a serem analisados

Nesta fase foi escolhido o tipo de artefato a ser utilizado para o desenvolvimento do método. A proposta deste trabalho consiste em criar um método para visualizar as comunalidades e variabilidades de uma família de produto de software. Portanto, o artefato a ser utilizado deveria teve a necessidade de ser um elemento comumente encontrado nos ambientes de desenvolvimento e que fosse constantemente utilizado. Dessa forma, concluiu-se que o próprio código fonte das aplicações deveria ser utilizado, pois consiste no principal artefato de um sistema.

1.3.2 Fase 2: Desenvolver o método a ser utilizado para a detecção das comunalidades e variabilidades nos artefatos selecionados

O foco desta fase foi o estudo para a identificação das técnicas mais adequadas para o desenvolvimento do método a fim de possibilitar a identificação, classificação e visualização automáticas das comunalidades e variabilidades. O objetivo era realizar pesquisas de trabalhos de outros autores a fim de encontrar uma técnica nova e que se mostrasse promissora.

1.3.3 Fase 3: Implementar uma ferramenta para a identificação e classificação em comunalidade ou variabilidade

Nessa fase foi realizada a implementação da ferramenta utilizando o método desenvolvido na etapa anterior, a partir dos artefato selecionado na fase 1. A ferramenta, chamada reCOVER, foi desenvolvida para a plataforma web. Os detalhes de sua implementação encontram-se descritos na Seção 4.2 deste trabalho.

1.3.4 Fase 4: Avaliar a eficiência do método e ferramenta em um contexto acadêmico

A avaliação foi realizada com alunos de graduação e pós-graduação da UFPI. Os detalhes dessa atividade são descritos no Capítulo 5.

1.3.5 Fase 5: Avaliar a eficiência do método e ferramenta em um contexto industrial

A avaliação foi realizada com um grupo de analistas de sistemas de uma empresa que desenvolve software na área de gestão de saúde. Os detalhes dessa atividade são descritos no Capítulo 5.

1.4 Contribuições Científicas

Este trabalho visa contribuir para o ambiente acadêmico, científico e industrial, de diferentes maneiras. As principais contribuições são:

- Proposta de abordagem para detecção de comunalidades e variabilidades em uma família de produtos software;
- Implementação de uma visualização das comunalidades e variabilidades em uma família de produtos de software, seguindo a definição da abordagem;
- Avaliação experimental do uso da abordagem em um contexto acadêmico e no contexto industrial.

1.5 Organização do Texto

Este trabalho está dividido em quatro capítulos além da Introdução e Conclusão. O Capítulo 2 trata do Referencial Teórico com um embasamento necessário para o entendimento do trabalho. O Capítulo 3 apresenta o Estado da Arte, apresentando uma análise dos trabalhos recentes que abordam pesquisas relacionadas ao trabalho aqui proposto. O Capítulo 4

apresenta a solução proposta. O Capítulo 5 detalha uma avaliação experimental inicial das ideias propostas neste trabalho. Por fim, as conclusões e direções para trabalhos futuros são descritos no capítulo final.

2 Referencial Teórico

Alguns conceitos são cruciais para o entendimento deste trabalho e para possibilitar a sua formulação e desenvolvimento. Este capítulo apresenta esses conceitos. Na Seção 2.1 será apresentado como o reuso vem sendo adotado, algumas de suas vantagens e desvantagens. Na Seção 2.2 será introduzido o conceito de LPS com destaque para as principais atividades e abordagens de implantação. A Seção 2.3 traz o conceito de clone, define os tipos de clones existentes e discorre sobre ferramentas de detecção. Por fim, a Seção 2.4 define visualização de software e descreve sobre sua importância para as equipes de desenvolvimento.

2.1 Reuso de Software

Por vezes, a Engenharia de Software busca nos diversos ramos da indústria e engenharia inspiração para desenvolver seus métodos e técnicas para lidar com software. Assim como um engenheiro civil não projeta um novo tijolo a cada construção que irá erguer, engenheiros de software defendem que nem sempre os componentes de software precisam ser construídos do zero a cada novo sistema.

De acordo com Sommerville (2007), o desenvolvimento de software com foco no reuso de artefatos tornou-se amplamente usado desde o ano 2.000. Nas fábricas de software, essa necessidade decorre do fato de existir um grupo de clientes que compartilha as mesmas demandas, ou demandas semelhantes. Por exemplo, se uma fábrica tem um sistema para ser usado em um PDV e fecha negócio com outro cliente que também necessita informatizar seu ponto de venda, ela poderá reusar o projeto já existente, ou partes dele e fazer as adaptações necessárias para o novo cliente.

Reuso de software traz menores custos de produção e manutenção, visto que ao invés de criar algo diferente a cada novo cliente, será feito um reaproveitamento de sistemas anteriores com a realização de cópias (RUBIN; CHECHIK, 2013). Além disso, será possível melhorar a qualidade, pois sistemas que já estão no mercado há algum tempo tendem a ser mais estáveis do que outro recém-criado. De acordo com Sommerville (2007), a engenharia de software baseada em reuso é uma abordagem que tenta aproveitar ao máximo as vantagens que o reuso pode oferecer.

Uma das práticas de reuso mais utilizadas consiste em copiar unidades de software a serem reutilizadas, as quais podem ser:

- A aplicação inteira. Nesse caso, um produto inteiro é copiado de um cliente para o outro, seguido de adaptações para as diferentes necessidades, caso existam.

- Componentes de software. Algumas vezes, apesar do tipo de aplicação a ser desenvolvida ser diferente de outras já existentes, ela inclui funcionalidades, fluxos ou subprocessos semelhantes. Nesse caso, serão reusadas apenas partes do sistema.

- Objetos e funções. Aqui serão copiadas porções menores, que executam apenas uma única função, como uma classe ou um método. Nesse contexto está uma das formas mais comuns de reuso, onde são vistas bibliotecas ou classes que desempenham determinada tarefa, como operações matemáticas ou em estruturas de dados, por exemplo.

Apesar de seus benefícios, o reuso de software também possui alguns problemas. Dentre eles podemos citar a ausência de ferramentas alinhadas com a práxis industrial, em algumas áreas, para o apoio efetivo ao desenvolvimento de software (SOMMERVILLE, 2007), (ROSSI, 2004).

Por muitos anos o reuso tornou-se então uma boa solução para fábricas de software com diversos clientes com demandas semelhantes. Entretanto, com o passar do tempo, o simples copiar-e-colar passou a apresentar alguns inconvenientes. Manter o mesmo código em vários produtos diferentes, embora represente uma economia, ainda se mostrava como uma abordagem onerosa quando as empresas possuíam muitos clientes com necessidades parecidas (APEL et al., 2016), (RUBIN et al., 2012). De acordo com Pohl, Böckle e Linden (2005), sem um planejamento adequado para o reuso, seus custos podem ser maiores que desenvolver artefatos a partir do zero. Nesse contexto, surgiu o conceito de Linhas de Produto de Software. De acordo com Pohl, Böckle e Linden (2005), a partir de quatro produtos torna-se menos custoso para uma empresa manter uma LPS do que tratar os produtos de forma independente.

2.2 Linhas de Produto de Software

LPS é um paradigma de desenvolvimento de sistemas no qual existe um grupo de softwares semelhantes (chamado de família), uma plataforma comum de artefatos para reuso e componentes variáveis de acordo com a necessidade de cada cliente (POHL; BöCKLE; LINDEN, 2005). Os artefatos são agrupados e mapeados em *features*, de acordo com suas funcionalidades. A partir dessa plataforma é possível gerar produtos de acordo com configurações previamente definidas (*features* que deverão conter).

LPS é uma das técnicas de reuso de software mais promissoras atualmente. Por meio de uma linha, componentes dos sistema não precisam estar duplicados para serem reaproveitados. Em outras palavras, deverão existir componentes únicos que poderão ser usados ao mesmo tempo por várias aplicações que necessitem utilizá-los.

A adoção da LPS em contraste com o paradigma tradicional (um projeto diferente a cada novo cliente), traz diversas vantagens para a fábrica de software, dentre elas, o menor custo com manutenção (KNAUBER et al., 2001). Por exemplo, em uma empresa que adote a LPS, ao receber um novo cliente, a equipe não precisa criar um novo projeto para ele, pois basta selecionar dentro da linha as funcionalidades, ou *features*, correspondentes aos requisitos em questão, mas se alguma *feature* nova for necessária, ela deve então ser incluída na LPS e posteriormente selecionada para o cliente. Outra vantagem pode ser observada quando a equipe necessita atualizar alguma funcionalidade do sistema para correção de erros ou melhora da performance. Nesse caso, basta fazer a alteração em um único local na LPS e então a mudança será refletida para todos os produtos que fazem parte dessa linha.

2.2.1 Principais atividades de LPS

LPS faz uso de três principais atividades, como mostra a Figura 1 (CLEMENTS; NORTHROP, 2002). Na primeira, Desenvolvimento de Ativos, são desenvolvidos os artefatos reusáveis que farão parte das aplicações. Aqui são incluídos todos os artefatos que possam pertencer ao desenvolvimento de um sistema, como documentação de requisitos, elementos de código e até planejamento de testes. Essa atividade também é chamada de Engenharia de Domínio. A segunda, Desenvolvimento de Produtos, é responsável pela construção dos produtos a partir dos artefatos da atividade anterior; também é conhecida como Engenharia de Aplicação. A terceira atividade, Gerenciamento, como o próprio nome sugere, inclui as práticas de gerenciamento organizacional e técnico relacionadas à abordagem.

A Engenharia de Domínio relaciona-se à definição do escopo com identificação das aplicações que farão parte da LPS, ao processo de identificação do que é comum e do que é variável em uma família de sistemas, além da construção da plataforma única de desenvolvimento, que deverá seguir o *framework* proposto por Pohl, Böckle e Linden (2005), incluindo os subprocessos: engenharia de requisitos do domínio, projeto do domínio, desenvolvimento do domínio e teste do domínio. A engenharia de aplicação, por sua vez, concentra-se nas atividades relacionadas a geração dos produtos com o máximo reuso dos artefatos, incluindo sua documentação e os subprocessos: engenharia de requisitos de aplicação, projeto da aplicação, construção da aplicação e teste da aplicação.

2.2.2 Variabilidades

Em uma família de produtos de software, ou em uma LPS, existem as comunalidades, que são os artefatos reutilizáveis, e as variabilidades, que são as características que podem ser alteradas de produto para produto. Por meio das variabilidades é que os produtos são personalizados para cada cliente diferente, conforme sua demanda.

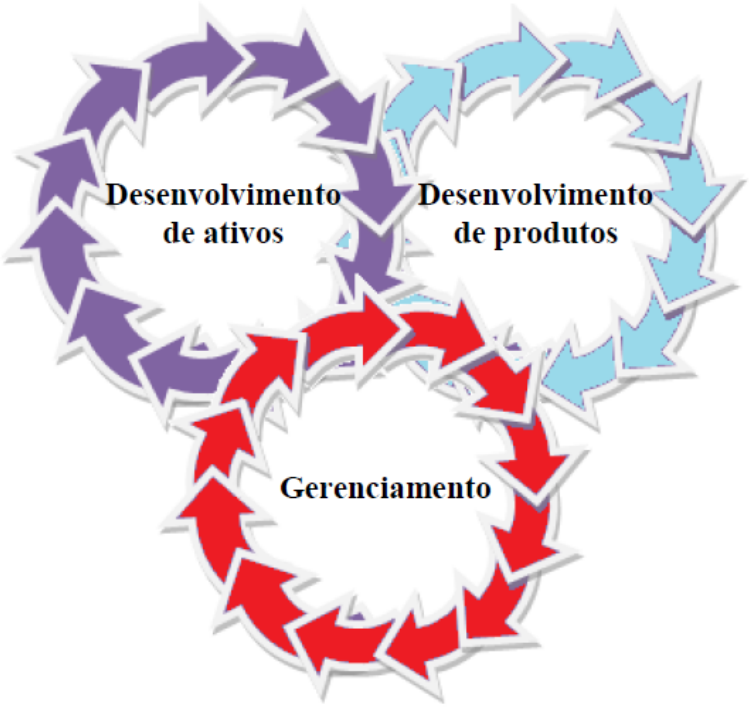

Figura 1 – Principais atividades de uma LPS. (IANZI, 2013)

Uma variabilidade é composta por um ponto de variação e suas variantes (POHL; BöCKLE; LINDEN, 2005). Um ponto de variação é uma marcação indicando onde os produtos podem variar de um para outro. Variante é, por sua vez, a representação de uma das variações possíveis. Por exemplo, em um sistema de PDV, um ponto de variação poderia ser "Tipo de pagamento", enquanto que variantes poderiam ser "Boleto"e "Cartão de Débito".

2.2.3 Abordagens de Implantação de LPS

Existem diferentes abordagens de implantação de uma LPS em uma fábrica de software: a pró-ativa, a reativa e a extrativa (KRUEGER, 2001). A abordagem a ser escolhida dependerá da situação em que os sistemas se encontram e da estratégia de negócios da empresa. Abaixo são explicadas cada uma.

- Abordagem pró-ativa: a LPS é planejada e construída para cobrir todo o escopo dos produtos que farão parte dela. Inicialmente deverão ser levantados e analisados todos os requisitos de todos os produtos, identificar o que é comum e o que é variável e então iniciar o processo de construção da plataforma única de desenvolvimento.

- Abordagem reativa: a LPS se inicia com um, ou alguns produtos e cresce gradativamente à medida que novas aplicações são requisitadas.

- Abordagem extrativa: a LPS é construída a partir de diversos produtos pré-existentes, ou em outras palavras, ela é extraída a partir de um grupo de produtos. O desenvolvi-

mento aqui reaproveita os artefatos já construídos pela equipe enquanto mantinham os sistemas de maneira independente.

A abordagem pró-ativa pode ser usada em casos onde a fábrica ainda não iniciou o desenvolvimento de nenhum sistema de uma família, mas que já sabe que serão necessários não apenas um, mas vários produtos. Pode ser usada ainda em casos onde a equipe já possui produtos desenvolvidos, mas prefere iniciar o planejamento da LPS do zero ao invés de reaproveitá-los, como sugere a extrativa. A abordagem reativa é indicada para equipes que tenham até três produtos, mas que pretendam futuramente expandir esse número. Nesse caso, ela constrói uma LPS com poucos produtos e mantém a arquitetura preparada para abranger novos produtos. Por fim a abordagem extrativa talvez é a de maior interesse hoje para as empresas. Isso se dá pelo fato de que as empresas muitas vezes iniciam um produto especificamente para um cliente e posteriormente novos clientes vão surgindo e os projetos vão sendo copiados e colados, sem planejamento e visão de LPS (FISCHER et al., 2014). Após algum tempo, a equipe percebe o problema do gerenciamento necessário para tantos produtos e então decide adotar a linha de produto, sendo o caminho da abordagem extrativa o mais curto e viável (POHL; BöCKLE; LINDEN, 2005), (KRüGER et al., 2016).

Muitas empresas estão escolhendo adotar a LPS por meio da abordagem extrativa. Por exemplo, Rincón et al. (2016) trazem um relato de experiência de adoção de LPS utilizando tal abordagem em uma pequena companhia de desenvolvimento de software. No trabalho é relatado que o principal objetivo da adoção é um melhor aproveitamento do conceito de reuso, integrando todas as aplicações e dessa forma reduzir custos de desenvolvimento. Também relatam que a abordagem extrativa foi escolhida por permitir o reaproveitamento dos artefatos já desenvolvidos.

Alves et al. (2010) buscaram analisar em seu trabalho quais as estratégias de adoção de LPS são mais abordadas em artigos científicos, na área de engenharia de requisitos. De acordo com seu trabalho, 39% das pesquisas encontradas tratam da abordagem extrativa, perdendo apenas para a pró-ativa, com 57%. A pesquisa mostra que existe um número considerável de pessoas em busca de auxiliar esse tipo de adoção, mas que o campo oferece espaço para mais trabalhos, uma vez que a maioria está focada na abordagem pró-ativa.

Apesar de o conceito de LPS já existir há mais de duas décadas (MACALA; STUCKEY; GROSS, 1996), ela ainda não é largamente adotada devido às poucas técnicas e ferramentas de apoio. A transição é considerada arriscada se a equipe não estiver preparada e não dispor de técnicas e investimentos adequados para tal (KRüGER et al., 2016). A criação de mecanismos de apoio para auxiliar a abordagem extrativa poderá ajudar diversas empresas que possuem vários produtos similares e gastam muito tempo e recursos para manterem um nível de atendimento às necessidades de seus clientes em relação a funcionalidades específicas.

2.3 Detecção de clones

De maneira geral, quando as empresas fecham novos contratos e assim precisam customizar seus produtos de softwares para os novos clientes, é comum que se adote o simples copiar e colar de pedaços de código que possam ser similares entre as demandas, o que chamamos de clones. Entretanto, à medida que o número de produtos cresce torna-se difícil o gerenciamento desses códigos, pois cada atualização a ser feita precisa ser realizada em cada um dos produtos, o que pode ocasionar problemas como o esquecimento de algum dos produtos que contenham aquele código.

Tendo em vista esse cenário, diversas ferramentas tem sido propostas para detecção dos clones em produtos, utilizando técnicas como Similaridade baseada em Tokens (KAMIYA; KUSUMOTO; INOUE, 2002), Árvore Sintática Abstrata (AST) (BAXTER et al., 1998) e Grafo de Dependências (LIN et al., 2014). E para empresas que desejam adotar a LPS utilizando a abordagem extrativa é importante o uso de ferramentas como essas para que tomem ciência do conteúdo que os produtos possuem em comum e o que os mesmos tem de diferente.

Existem quatro tipos de clones que tais ferramentas podem detectar (KHODAI; PERUMAL; KANMANI, 2010), conforme especificado abaixo. Normalmente, cada uma dedica-se apenas a um dos tipos, (JIANG et al., 2007),(KOMONDOOR; HORWITZ, 2001).

- Tipo I: Textos exatamente iguais. Código são diferenciados por espaços em branco ou comentários;
- Tipo II: Estrutura igual. Nesse caso, os espaços em brancos e comentários são desconsiderados, mas mudanças nos nomes de identificadores já são suficientes para tornar dois trechos de código diferentes;
- Tipo III: Espaços em branco, comentários e nomes de identificadores não são importantes. Se os códigos possuem a mesma estrutra sintática, então são considerados clones;
- Tipo IV: Este tipo de clone não compara as linhas de código em si, mas o que fazem. Nesse caso não importam os comandos utilizados, se os blocos de código fazem a mesma coisa, então já podem ser considerados clones.

2.4 Visualização de Software

De acordo com Sellier e Mannion (2007), visualização é definida como a formação mental de uma imagem. Visualização de Informação, por sua vez é o uso de representações visuais e interativas de dados utilizando meios computacionais, a fim de favorecer sua compreensão

(CARD; MACKINLAY; SHNEIDERMAN, 1999). Seu objetivo deve ser fornecer uma representação natural e compreensiva dos dados sob a perspectiva dos stakeholders. A visualização de Software, por conseguinte, é uma subárea da visualização da Informação que foca no entendimento humano de um software computacional (PRICE; BAECKER; SMALL, 1993).

A visualização de software tem sido útil para a percepção de informações de diversas naturezas dentro deste contexto. Dentre os benefícios encontrados, podemos citar como por exemplo a facilitação da busca de informação. Espera-se que com uma imagem mental como representação do software, ao se buscar determinada informação ela será mais facilmente encontrada, do que se a busca fosse realizada analisando todo o sistema da maneira natural. Portanto, ultimamente tem-se encontrado diversos trabalhos desenvolvendo técnicas de visualização para auxiliar especialmente engenheiros de software em suas tarefas essenciais pertencente às fases de desenvolvimento, testes e manutenção (WILDE; GERMAN, 2016), (KULA et al., 2014), (ABDEEN et al., 2014), sendo considerada uma área bem estabelecida e frequentemente abordada.

Há alguns anos, modelos de representação de variabilidades de software também têm sido propostos nas diversas fases do desenvolvimento de uma LPS, desde seu planejamento até após sua construção (LOESCH; PLOEDEREDER, 2007), (THIEL; HEIN, 2002), (SELLIER; MANNION, 2007). A variedade dos produtos e a complexidade dos artefatos variantes dificultam a compreensão da organização da família de software. A técnica pode ser utilizada com o objetivo de fornecer uma visão clara e compreensível das partes comuns e distintas entre os diferentes produtos.

Mattila et al. (2016) fez em seu trabalho uma revisão sistemática na literatura sobre visualização de software e cita variantes em linhas de produto como um dos temas explorados por artigos de visualização. Podemos citar alguns pontos interessantes levantados em seu trabalho: os métodos de visualização mais populares são em gráfico e em formato de árvore; o aspecto mais estudado é o entendimento da estrutura do software (37 de um total de 83 artigos); a fonte de dados mais comumente utilizada é o próprio código-fonte da aplicação.

Como dito anteriormente, para a implantação da abordagem extrativa de LPS, é necessário fazer o levantamento dos artefatos que farão parte da plataforma comum e os que serão pontos de variação. Este trabalho propõe uma visualização comparativa dos produtos de uma mesma família utilizando a própria estrutura hierárquica dos pacotes e classes dos projetos. Espera-se com isso favorecer o entendimento das diferenças e similaridades, uma vez que o usuário (membro de uma equipe de desenvolvimento) já está familiarizado com tal disposição.

2.5 Considerações Finais

Neste capítulo foi tratada a importância que o reuso tem no desenvolvimento de software e como o mesmo precisa ser planejado, sob pena de ter efeito contrário ao esperado, ou seja, aumentar os custos. Foi apresentada uma das formas recentes de se aplicar o reuso de forma sistemática e que se propõe a ser eficaz, denominada LPS. Com o uso dessa abordagem, espera-se que as equipes tenham o máximo proveito do conceito de reuso. Também foi apresentado o conceito de LPS, comparando-a com o desenvolvimento de sistemas únicos. Foram tratados os principais processos e atividades envolvidos e as diferentes maneira de como as empresas podem iniciar uma linha de produto em suas organizações.

3 Estado da Arte

Este capítulo busca descrever outros trabalhos que, assim como este, procuram apoiar o processo de adoção de LPS por meio da detecção de elementos comuns e variantes. Com a pesquisa foi possível observar que muitos trabalhos relacionados vem sendo desenvolvidos no tema proposto nos últimos anos. Os principais trabalhos encontrados serão detalhados a seguir.

3.1 Trabalhos Relacionados

Os trabalhos relacionados nesta seção encontram-se agrupados em subseção conforme o tipo de proposta que apresentam em sua pesquisa.

3.1.1 Busca de Similaridades

Richenhagen et al. (2016) efetua em seu trabalho a busca de similaridades entre versões diferentes de softwares utilizando análise semântica baseada em testes de software. Para a sua pesquisa o autor utiliza como entrada especificações de teste, que consiste em uma ou mais sequências de teste. O algoritmo foi desenvolvido utilizando um autômato finito determinístico cujos estados estão relacionados aos dados de entrada e saída dos casos de teste selecionados. Assim como este trabalho, seu objetivo é diminuir o custo do processo de identificação das similaridades entre diferentes produtos de software. Entretanto, para seu trabalho foi realizada uma análise semântica dos elementos de software, enquanto que este trabalho foca apenas em análise sintática.

3.1.2 Apoio à LPS

Nesta subseção reunimos os trabalhos que assim como este, se propõem a auxiliar o uso de LPS. Alguns têm o enfoque em ambientes que já utilizam uma linha de produto, outros se preocupam com as equipes que ainda pretendem implantá-las. Os trabalhos estão a seguir identificados:

3.1.2.1 Visualização em LPS

Nesta subseção estão os trabalhos encontrados que assim como o nosso, são focados em visualização aplicada em um ambiente com produtos de software variantes. Entretanto diferentemente do nosso, tratam de uma LPS já construída, ou seja, de produtos que já estão integrados em uma linha de produto, enquanto que nós nos preocupamos com as fábricas que ainda estão planejando essa adoção.

Nestor et al. (2007) apresentam em seu trabalho uma pesquisa acerca da visualização de software aplicada a LPS, encorajando a sua aplicação na prática. Os autores apresentam as técnicas de visualização que são mais comuns nesse ambiente, apontando as suas principais limitações e também citam as áreas as quais afirmam ser mais importantes para maiores pesquisas. Entretanto, o seu trabalho não desenvolveu nenhum método específico para resolver a problemática, diferentemente desta pesquisa de mestrado. Já o trabalho de Loesch e Ploedereder (2007) propõe um novo método com o objetivo de otimizar o gerenciamento de variabilidades em uma LPS por meio de sua visualização. Entretanto, seu trabalho se distingue do nosso no fato de não gerar visualização de código, mas sim de funcionalidades dos sistemas, as *features*.

3.1.2.2 Adoção de LPS

Nesta subseção apresentamos os trabalhos que buscam auxiliar as empresas que pretendem adotar futuramente uma LPS, assim como este trabalho de mestrado.

Botterweck et al. (2008) em seu trabalho apresentam um meta-modelo e ferramenta, os quais propõem técnicas de visualização para apoiar o desenvolvimento de LPS. Dentre as tarefas que os autores se propõem a auxiliar estão: configuração de um produto variante, compreender as consequências de possíveis alterações na arquitetura dos sistemas e representar as *features* de alto custo. Por outro lado, este trabalho auxilia o engenheiro no sentido de dar subsídios para que o mesmo possa compreender o que pode ser alterado e o que pode ser mantido na estrutura dos sistemas para a futura adoção da LPS. Diferentemente do trabalho desta dissertação, que trata da visualização de elementos de código, eles abordam a visualização em termos de *features* e pontos de variação entre elas. Além disso, seu método foca em fábricas que já utilizam a LPS e não apenas planejando a adoção, como foi feito neste trabalho.

Bakar et al. (2017) apresenta em seu trabalho um experimento conduzido para extração de *features* a partir das especificações de requisitos, utilizando processamento de linguagem natural. Seu objetivo é auxiliar o processo de reuso, permitindo à equipe de desenvolvimento o entendimento das *features* comuns e diferentes entre os produtos analisados. Entretanto, diferentemente do nosso trabalho, que analisa código-fonte, Bakar et al. (2017) trabalham com *features* especificadas nos requisitos em linguagem natural.

Klatt, Küster e Krogmann (2013) apresentam um método para identificar pontos de variação em produtos semelhantes. O método inicia pela construção de diagramas de representação dos sistemas similares, em seguida os modelos são comparados afim de identificar as diferenças de implementação. A partir desta comparação, é criado então o modelo de pontos de variação em granularidade fina, que deverá ser passado aos engenheiros de LPS. Por último, a abordagem proposta sugere ainda recomendações acerca da nova implementação dos sistemas, a fim de construir a LPS. Neste trabalho não é criado

diagramas, pois as comparações se dão no próprio nível de código dos sistemas. Além disso, neste trabalho é gerado uma estrutura de visualização, o que não é feito por Klatt, Küster e Krogmann (2013).

Os autores em Martinez et al. (2015) apresentam um conjunto de princípios para a construção de um *framework* com o objetivo de apoiar de ponta-a-ponta a adoção de LPS utilizando a abordagem extrativa. Os princípios foram estabelecidos para abranger o processo de transição a partir de qualquer tipo de artefato, seja ele código-fonte, arquivo texto, ou até imagem. Estes princípios foram: (1) os artefatos poderão ser decompostos em unidades distintas, denominadas Elementos; (2) é possível calcular uma métrica de similaridade entre cada par de elementos; (3) um novo artefato pode ser construído a partir de um conjunto de Elementos. Os autores criaram um *framework* com o intuito de avaliar a proposta. Os resultados do experimento apresentaram uma boa eficácia em todo o processo. O trabalho de mestrado aqui apresentado se distingue no sentido de que este provê uma infraestrutura de visualização concentrada na etapa de planejamento de adoção da LPS, recurso que é ausente no trabalho de (MARTINEZ et al., 2015).

O trabalho apresentado por Eyal-Salman e Seriai (2016) propõe uma abordagem para a definição de uma arquitetura de LPS a partir de uma família de sistemas. O seu método é constituído de duas fases: a primeira recebe como entrada a descrição de todas as *features* dos sistemas, assim como a configuração de cada produto (as *features* pertencentes a cada); com esses dados, infere-se quais são as *features* obrigatórias e quais os pontos de variação nas *features*. Esses dados são usados como entradas na segunda fase, juntamente com o mapeamento das *features* nos códigos das aplicações, na qual são inferidos as porções de código que são obrigatórias e as que são pontos de variação. O trabalho desenvolvido neste mestrado se diferencia desse no sentido de que o nosso método recebe como entrada o código fonte de cada uma das aplicações, e não a descrições das suas *features* como em (EYAL-SALMAN; SERIAI, 2016). Em um ambiente real, os dois métodos poderiam ser utilizados em conjunto, um para detecção do código comum e variante e outro para identificação das *features* comuns e variantes.

Em Ajila (2005) é apresentado um *framework* que abrange desde a identificação das *features* em cada produto, a construção da arquitetura da linha de produto e até a derivação de produtos individuais. O *framework* utiliza o Processo de Desenvolvimento Unificado (Unified Software Development Process - USP) (JACOBSON et al., 1999) e a construção do modelo de *features* de acordo com a FODA (Feature-Oriented Domain Analysis) (KANG et al., 1990). Como trabalho futuro o autor apresenta a construção da ferramenta que implemente o *framework* proposto.

Dado um único produto de software, Bastarrica, Rivas e Rossel (2007) propõem detectar os possíveis pontos de variação em relação a outros produtos. A arquitetura da aplicação deverá então ser posteriormente atualizada para uma Arquitetura de Linha de Produto

(Product Line Architeture - PLA). Na avaliação do método foi admitido pelos autores que a abordagem não é eficiente em todos os cenários, mas que é possível lidar com esses impasses. Seu trabalho propõe o projeto da LPS a partir de apenas um único produto, e não de uma família, diferentemente do abordado nesta dissertação. Dessa forma, o seu método é mais adequado para uma adoção segundo a abordagem reativa; a pesquisa descrita nesta dissertação, por sua vez se torna mais adequada para a abordagem extrativa.

3.1.3 Análise de Código

Nesta subseção agrupamos os trabalhos que não tratam especificamente do contexto de LPS, mas que buscam no código os pontos de variação entre versões diferentes de um mesmo produto, assim como o que é proposto neste trabalho.

Ryssel, Ploennigs e Kabitzsch (2010) apresentam uma abordagem automática para identificação de pontos de variação em sistemas embarcados pré-existentes e para reorganizá-los em um conjunto de modelos. Os sistemas analisados utilizam algoritmos modelados por meio de blocos de função (VYATKIN; AMERICA, 2007). Para identificar os elementos variantes, o trabalho primeiramente busca pelas similaridades, as organiza em clusters e então usa esses grupos para pesquisar por variantes em potencial. Entretanto seu foco está nos sistemas embarcados, enquanto que a ferramenta desenvolvida neste trabalho trata de sistemas em geral escritos na linguagem Java.

Duszynski, Knodel e Becker (2011) apresentam uma método que recebe como entrada o código-fonte de todos os sistemas e então constrói modelos abstratos para cada um dos produtos. Em seguida, o método compara os modelos dos produtos para encontrar as similaridades e diferenças entre eles. Os resultados são exibidos em matrizes de ocorrências, e diagramas de barras, enquanto que a reCOVER gera uma árvore comparativa. Com isso o seu trabalho tende a facilitar a identificação de um elemento (como comum ou variante) por parte dos usuários; por outro lado, sua visualização não permite saber a relação entre esses elementos, que é justamente a proposição deste trabalho, que é uma visualização em estrutura hierárquica.

Assim como este trabalho, o trabalho de Nunes et al. (2012) também busca por similaridades e variações em produtos membros da mesma família de software escritos na linguagem Java. Os autores fazem comparações das aplicações variantes umas com as outras e também com as diferentes versões de cada uma, utilizando heurísticas sensíveis a história. O método recebe como entrada os mapeamentos das *features* (no código) de cada uma das aplicações e, por meio de comparação entre grupo de *features*, consegue inferir os grupos obrigatórios e opcionais, gerando um novo projeto Java com um pacote para cada. Este trabalho se distingue do apresentado por desconsiderar versões antigas do software, uma vez que acredita-se que essa inclusão possa atrapalhar no processo de identificação do que é comum e variante, uma vez que se tratam de versões legadas.

3.1.3.1 Detecção de Comunalidades

Existem ainda trabalhos com o foco exclusivamente em detectar os códigos que estão duplicados em sistemas diferentes, o que equivalem às comunalidades procuradas nesta pesquisa. Tais trabalhos buscam não apenas duplicações do tipo copiar-e-colar, mas também trechos com pequenas variações, ou mesmo aqueles que executam a mesma operação, mas de maneiras diferentes.

Oliveira, Fernandes e Figueiredo (2015) trazem em seu trabalho uma avaliação a respeito das ferramentas que fazem esse tipo de investigação. Foram listadas e analisadas vinte ferramentas ao todo, o que indica uma preocupação no mercado com a problemática. O objetivo do trabalho foi avaliar se as ferramentas eram capazes de detectar quatro tipos diferentes de código duplicado: tipo I, representando um simples copiar-e-colar; tipo II, copiar-e-colar com pequenas alterações nos identificadores, tipos, valores, dentre outros; tipo III, copiar-e-colar com variações nas operações ou nos blocos de código; e por fim, tipo IV, trechos de código que realizam a mesma tarefa, mas com implementações diferentes. Apesar de muitos pesquisadores estarem buscando uma solução efetiva para o problema, ela ainda não foi encontrada para todos os quatro de tipos de duplicação, como concluem os autores.

Para identificar oportunidades de reuso de ativos entre softwares distintos, Oliveira et al. (2016) propuseram o método, JReuse, para calcular o grau de similidades de componentes de software. O cálculo é feito a partir da análise léxica dos componentes de um conjuntos de sistemas. De acordo com os autores, um grau de pelo menos 80% de semelhança entre duas unidades (do mesmo tipo) é um forte indício de que se trata de código duplicado. Uma ferramenta foi desenvolvida para permitir a aplicação do método, a qual foi avaliada por meio de experimentos. Porém, diferentemente do trabalho deste mestrado, Oliveira et al. (2016) não fornece uma estrutura de visualização.

Baxter et al. (1998) em seu artigo apresentam um método para detectar clones em sistemas de software utilizando a AST. Os autores realizam uma análise bottom-up na árvore (assim como o trabalho aqui apresentado) e à medida em que sobem na estrutura, se for encontrado um clone que abrange outro, este será desconsiderado. Além de clones idênticos, o trabalho também investiga "clones próximos", que são trechos de código quase idênticos. A abordagem foi avaliada por meio de um experimento com 19 sistemas escritos em C. Dentre outras conclusões, o experimento mostrou que os sistemas mais recentes eram os que possuíam maior números de clones e não foi encontrada correlação entre o tamanho dos softwares com a quantidade de clones encontrados. O trabalho dos autores não utiliza a técnica de visualização como o proposto neste mestrado; além disso seu trabalho foca em sistemas escritos em C, diferentemente da ferramenta proposta aqui, que trabalha com aplicações Java.

3.2 Considerações Finais

A pesquisa bibliográfica permitiu identificar que o problema abordado neste trabalho de mestrado é de relevância no meio acadêmico. Diversos trabalhos tem sido desenvolvidos na tentativa de oferecer maior apoio às equipes que pretendem adotar a LPS em seu desenvolvimento, mas que consideram os custos de implantação como uma barreira para a sua implantação.

Por outro lado, apesar de ter sido encontrados diversos trabalhos que executam alguns pontos semelhantemente ao aqui proposto, não foram encontrados trabalhos que realizassem tudo conforme é proposto aqui, ou seja, que propusesse um método e ferramenta para auxiliar a adoção de LPS por meio da abordagem extrativa, analisando os códigos fonte de aplicações em Java, relacionadas aos produtos existentes, e gerando uma visualização hierárquica em estrutura de árvore. Essas questões mostram a relevância deste trabalho que se encontra em uma área de interesse de muitas organizações, mas que propõe uma metodologia nova de tratar o problema.

4 Solução Proposta

Neste capítulo está descrito o método proposto neste trabalho, visando a geração da visualização para comunalidades e variabilidades em projetos de software. Também neste capítulo é apresentado um exemplo de situação em que é possível aplicar este método, a fim de que o leitor obtenha melhor entendimento do mesmo.

4.1 Método

Técnicas que permitem o reconhecimento e análise de códigos comuns são bastante úteis no processo de migração para uma linha de produto de software (TONSCHEIDT, 2015). Antes que se comece o planejamento da LPS é necessário que a equipe de desenvolvimento conheça claramente o que de fato é comum e o que é diferente entre seus diversos produtos de software. Seguindo essa ideia, este trabalho pretende oferecer tal informação de código comum por meio da visualização de software com relação à estrutura dos projetos.

Neste trabalho foi proposto um esquema de visualização do processo da reengenharia de produtos individuais para uma linha de produto, que é apresentado na Figura 2. Do lado esquerdo da figura estão representados os sistemas inicialmente construídos separadamente (retângulos com uma ponta dobrada). A partir da análise desses produtos deverão ser identificadas as variabilidades e as comunalidades entre os mesmos, representadas na figura por peças de quebra-cabeça. Em termos de código, de posse do que é comum e do que é variável, a equipe poderá então projetar a nova arquitetura da LPS, representada pelos blocos azuis. Finalmente, após o projeto estar pronto, a linha de produto pode ser enfim construída, representada na figura por um retângulo único contendo todos os produtos existentes previamente.

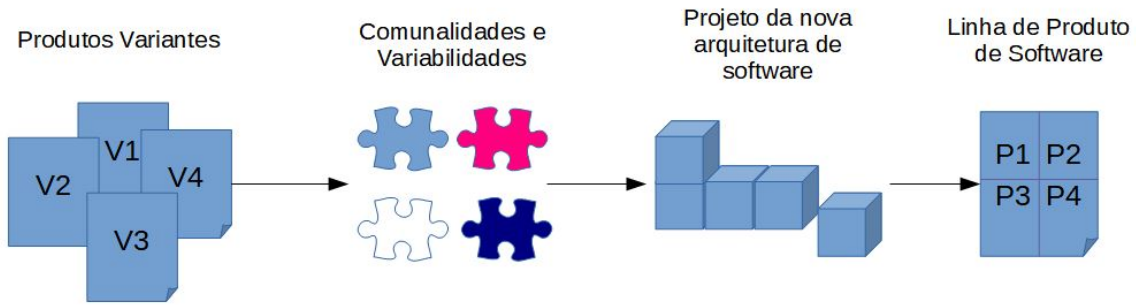

Figura 2 – Processo de adoção de LPS por meio da abordagem extrativa.

Para empresas com produtos de software com dezenas ou centenas de milhares de linhas de código, entende-se que a segunda etapa do processo apresentado se torne inviável de

ser executado de forma manual, sem o auxílio de uma ferramenta que automatize parte da atividade ou por completo, pois trata-se de uma tarefa dispendiosa (RICHENHAGEN et al., 2016). Por conta disso, espera-se que o método e a ferramenta propostos neste trabalho, gere uma redução no esforço envolvido nesse processo de transição para uma futura LPS.

Dessa forma, foi desenvolvido neste trabalho um método focado nessa problemática, voltado para sistemas orientados a objetos. O método recebe como entrada o código-fonte de um conjunto de produtos pertencentes à mesma família e que mantêm a estrutura de seus projetos de maneira similar. O método gera como saída uma visualização das comunalidades e variabilidades dessa família.

A seguir relacionamos os passos a serem seguidos para a execução do método.

1. Selecionar as aplicações cujas comunalidades e variabilidades se deseja descobrir (realizado por experts do negócio);
2. Identificar todos elementos de pacotes e classes dos projetos (de maneira automatizada);
3. Gerar a AST de cada classe (de maneira automatizada);
4. Comparar as classes por meio de seus membros, atributos e métodos (de maneira automatizada);
5. Definir limiares para determinação de elementos comuns ou variantes (escolhidos pela equipe de desenvolvimento);
6. Agrupar classes que possuem semelhança igual ou superior ao limiar estabelecido. Esse passo deve envolver todas as classes da aplicação. Aquelas que não puderem ser agrupadas com nenhuma outra, deve formar um grupo de apenas uma única classe cada (de maneira automatizada);
7. Para cada nível da árvore, começando no maior e indo em direção ao menor (de maneira automatizada):

   Agrupar os pacotes que possuírem quantidade de elementos (classes ou pacotes) pertencentes a um mesmo grupo igual ou superior ao limiar estabelecido
8. Gerar a visualização da árvore final encontrada (de maneira automatizada).

Este trabalho realiza uma análise sintática nos códigos dos produtos de software. Alguns algoritmos já propostos na literatura utilizam análise semântica das similaridades (RICHENHAGEN et al., 2016), (RUBIN; CHECHIK, 2012) (RYSSEL; PLOENNIGS; KABITZSCH, 2010). Entretanto, este trabalho, assim como diversos outros ((BERGER; RENDEL; RUMPE, 2014), (RUBIN; CZARNECKI; CHECHIK, 2013), (FISCHER et

al., 2014), (DUSZYNSKI; KNODEL; BECKER, 2011)), é focado na análise sintática do código, pelo fato de possuir uma implementação mais simples e adequada ao contexto do projeto.

4.1.1 Selecionar aplicações

Uma família de produtos de software normalmente inicia-se com o desenvolvimento de um único sistema e, à medida que o número de clientes cresce, são geradas customizações específicas para suas necessidades, criando assim versões alternativas do produto inicial. Algumas fábricas de software reutilizam o sistema que já possuem e realizam as adaptações aos requisitos para os diferentes clientes. Dessa forma, com o passar do tempo, a mesma se depara com uma gama de produtos diferentes, mas com boa parte dos artefatos em comum.

Essas aplicações são candidatas a serem submetidas ao método aqui proposto. Sistemas que possuem funcionalidades semelhantes, mas que foram desenvolvidos de maneira totalmente independente, não se aplicam a este método. Por outro lado, sistemas que foram criados com cópia e colagem de elementos de software, mas que possuem hierarquia de pacotes diferentes até podem ser submetidos ao método, mas não é o recomendado, pois a visualização se tornaria complexa, qualidade oposta à proposta pela visualização de software, que é justamente simplificar o entendimento. Tal fator configura-se como uma limitação do nosso trabalho.

Portanto, para a aplicação deste método, as aplicações a serem selecionadas devem ter arquiteturas semelhantes, com a mesma hierarquia de componentes, embora o nome deste não importe. Este é o caso de equipes que para atender determinado cliente, copiam o projeto já existente e apenas altera, remove ou adiciona componentes de acordo com os requisitos desse novo cliente. A seleção é uma decisão em nível de negócio, mas que deve respeitar tais detalhes técnicos.

4.1.2 Identificação das Variabilidades e Comunalidades

A fim de detectar as variabilidades e comunalidades da família de software, é realizada uma comparação entre seus códigos fonte. O algoritmo proposto realiza uma análise bottom-up, ou seja, a verificação inicia-se nos membros das classes, passa pelos pacotes, indo em direção ao pacote raiz da aplicação. Ao final, deve ser gerada uma visualização das estruturas dos projetos de maneira unificada, informando os nós que são comuns a todos, os que são comuns a apenas uma parte e aqueles que pertencem a um produto apenas.

Conceitualmente, a análise é feita em uma estrutura de árvore, na qual no nível mais baixo está o pacote raiz do projeto e nos níveis mais altos os membros de classes, que são os atributos e métodos, de maneira hierárquica. No Nível 0 é criado um nó-raiz genérico

para a família em questão. No Nível 1 estão os nós referentes aos pacotes localizados na raiz do projeto. Os filhos de cada um desses nós são os seus pacotes ou suas classes internas, e assim por diante. No entanto, quando um nó refere-se a uma classe, seus filhos são atributos e métodos. Essa é a estrutura utilizada para a análise da comparação dos produtos, com a visualização limitando-se a apenas exibir os pacotes e as classes.

Para que dois elementos sejam considerados comuns, deve ser comparado cada um dos seus componentes, situados no nível imediatamente superior. Ou seja, se é desejado saber se um pacote em um nível X na Aplicação A é comum com relação ao pacote Y na Aplicação B, então devem ser comparados seus elementos filhos (pacotes e classes), considerando apenas aqueles que estão respectivamente nos níveis X+1 e Y+1, em cada uma das aplicações. O cálculo de similaridade entre um pacote X e um pacote Y é realizado segundo a fórmula a seguir:

$$S = \frac{2n_{EC}}{n_{EX} + n_{EY}}$$

onde n_{EC} é o número de elementos comuns a X e Y; n_{EX} é o número total de elementos de X e n_{EY} é o número total de elementos de Y. Assim, se todos os elementos de X e de Y forem comuns o resultado da fórmula será 1%.

A comparação entre as aplicações se inicia do nível mais alto (folhas) e vai em direção ao nível mais baixo (raiz) da árvore. Primeiramente é realizada uma comparação geral de todas as classes, verificando o quanto elas possuem em comum em termos de atributos e métodos. Se a porcentagem dos elementos em comum for igual ou superior a um dado limiar, elas serão consideradas comum. Por exemplo, se as classes A e B possuem uma porcentagem de similaridade maior que o limiar estipulado, então elas são agrupadas em um mesmo cluster. Esse processo é feito para todas as classes entre todas as aplicações.

Tendo-se realizado o agrupamento de todas as classes, deve-se a analisar os pacotes dos projetos. Para a adequada construção da árvore, cada agrupamento de classes deverá pertencer, hierarquicamente, a um agrupamento de pacotes. Para isso verifica-se os pacotes-pai de cada uma das classes do cluster, em cada um dos produtos individualmente. Dessa forma, para cada grupo de classes reunidas deve ser verificado se é possível também agrupar os pacotes-pai de cada uma. Os pacotes só poderão ser agrupados caso possuam uma porcentagem de similaridade também maior ou igual a um certo limiar. Caso contrário estarão em grupos diferentes.

Ao se fazer a análise dessa forma é possível perceber que em um mesmo grupo podem haver classes com pais diferentes, caso em que os pacotes aos quais pertencem as classes não puderem ser agrupados. Entretanto, não é possível fazer um mesmo grupo de classes ser filho de dois agrupamentos de pacotes, uma vez que isso viola o conceito de árvore, pois em uma árvore cada nó pode ter apenas um único pai. Para resolver esse problema, na árvore de visualização duplicamos o nó referente ao grupo de classes com mais de um pai, de forma que cada um é mostrado como filho de pais diferentes.

Assim, após o agrupamento de todos os pacotes, é o momento de descer mais um nível da árvore, seguindo o mesmo procedimento, verificando se os pacotes-pais de cada um dos elementos de um grupo podem ser agrupados com base no limiar. O processo é então repetido quantas vezes forem necessárias até atingirmos o nó raiz. Com base nesses agrupamento de acordo com as variabilidades e comunalidades do sistema, é gerada então uma árvore para visualização das diferenças entre os produtos.

4.2 Ferramenta

A fim de explorar o método proposto e viabilizar sua análise, foi desenvolvida uma ferramenta chamada reCOVER (nome criado para conter o acrônimo de COmmunalities and Variabilities ExtRaction) com base no mesmo. A ferramenta recebe os códigos-fonte de aplicações escritas na linguagem Java e de posse desses artefatos, é feita então a análise de similaridades. Tal linguagem foi escolhida por ser amplamente utilizada por equipes de desenvolvimento em geral (PRECHELT, 2000). Dessa forma, para realizar as análises utilizou-se a própria AST do Java (KUHN; THOMANN, 2006) com o auxílio da ferramenta JDT (Java Development Tools) da IDE Eclipse (FOUNDATION, 2017).

Como pode ser visto na Figura 3, cada nó da árvore representa um pacote ou uma classe do projeto. Os pacotes são representados pelo ícone de pasta e as classes pelo símbolo de visto. Foi escolhida uma visualização 2D adotando um layout de árvore de diretório. A expansão da árvore no sentido horizontal permite melhor aproveitamento do espaço da tela e consequentemente uma visualização mais adequada. Exibir dezenas de nós sob uma orientação vertical da árvore causaria, por exemplo, sobreposição dos elementos ou teríamos a largura de nosso layout extremamente grande.

Figura 3 – Visão dos produtos gerada pela reCOVER.

A execução da ferramenta é realizada em três passos, mostrados na Figura 4. Em a) é possível vermos a página inicial da reCOVER. O primeiro passo, representado por b),

consiste em selecionar as aplicações que farão parte da visualização. Para isso, deve ser feito o *upload* das pastas de cada projeto compactadas individualmente com a extensão .zip. Após o *upload*, o usuário tem a oportunidade de renomear os projetos dentro da ferramenta para nomes que considere mais adequados (em *c)*), uma vez que inicialmente são considerados os nomes originais dos projetos, mas estes podem não ser adequados à correta identificação no momento da comparação, ou ainda podem ser muito grandes, o que atrapalha a etapa de visualização.

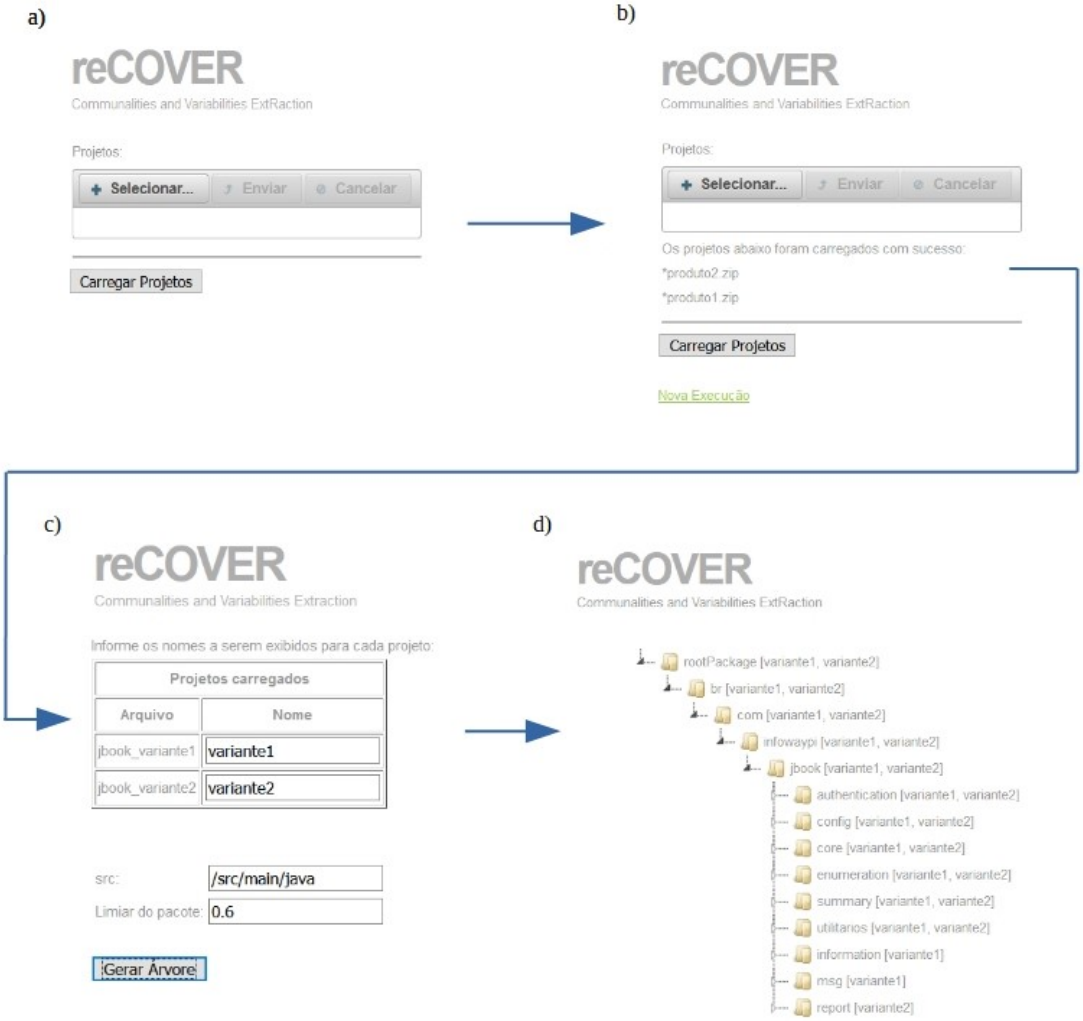

Figura 4 – Fluxo de execução da reCOVER.

Nesse mesmo passo, é possível ainda informar o diretório do código a ser comparado dentro dos projetos. Por padrão, o campo é preenchido de acordo com a estrutura utilizada em projetos Maven[1], considerando apenas o código principal, ignorando assim o diretório de testes. Entretanto, o usuário poderá alterar o valor do campo de acordo com a estrutura utilizada em sua equipe, ou ainda alterar o diretório de forma a incluir os testes. Na mesma tela, existe o campo "Limiar do pacote", que é o valor da porcentagem utilizado para determinarmos se um pacote é comum ou se é uma variabilidade. Ao clicar no botão

[1] https://maven.apache.org

"Gerar Árvore"a reCOVER fará toda a abstração do código necessária e fará o cálculo das similaridades de cada classe e pacote. Nesse passo também será feita a associação das variabilidades encontradas com cada um dos produtos analisados.

Por fim, temos o terceiro e último passo que é geração da árvore em si, formada pelos pacotes e classes de todos os projetos dos quais foi realizado o *upload*, representado pela imagem d. Selecionando um dos nós da árvore, o usuário poderá visualizar um gráfico de pizza com a sua respectiva composição. Em outras palavras, o gráfico mostra, em porcentagem, o quanto do nó pertence apenas a determinado projeto, o quanto pertence a apenas parte dos projetos e o quanto do nó é comum a todos os projetos. A Figura 5 mostra um exemplo desse gráfico para uma classe chamada *FluxoFinanceiro*. Cada setor do gráfico corresponde a uma determinada porção da classe (ou pacote) que pertence a um certo conjunto de produtos.

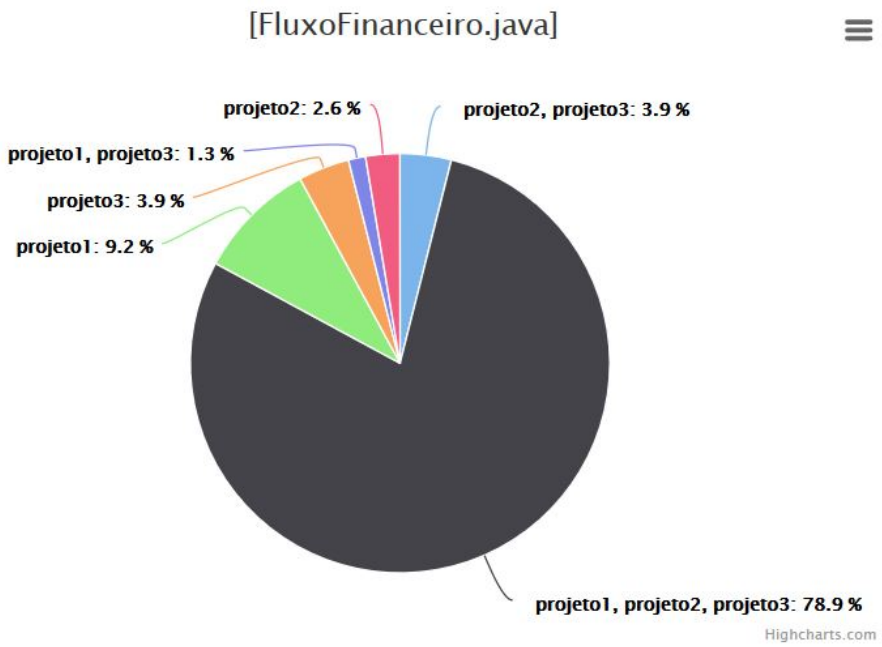

Figura 5 – Fluxo de execução da reCOVER.

4.2.1 Tecnologias utilizadas

A ferramenta com a implementação do método proposto neste trabalho foi desenvolvida sob a plataforma Web, utilizando a linguagem *Java* e o *framework JSF*[2] com *Prime Faces*[3]. A árvore foi projetada utilizando o plugin *jsTree*[4] e os gráficos a partir da biblioteca *HighCharts*[5].

[2] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[3] https://www.primefaces.org/
[4] https://www.jstree.com/
[5] https://www.highcharts.com

4.3 Exemplo para Ilustração

Para melhor entendimento do trabalho, nesta seção é apresentado um exemplo de aplicação do método e de sua ferramenta em um cenário fictício. Nesse exemplo será considerado o limiar de similaridade de 60%.

Suponha que uma empresa desenvolvedora de soluções para universidades pretenda adotar os conceitos de LPS para facilitar seu trabalho. O primeiro passo é então identificar as aplicações já existentes que deverão fazer parte da linha. A Figura 6 mostra a estrutura das respectivas aplicações usadas para a estruturação da linha. Considerando que a arquitetura de seus sistemas são semelhantes, o método aqui proposto pode então ser aplicado.

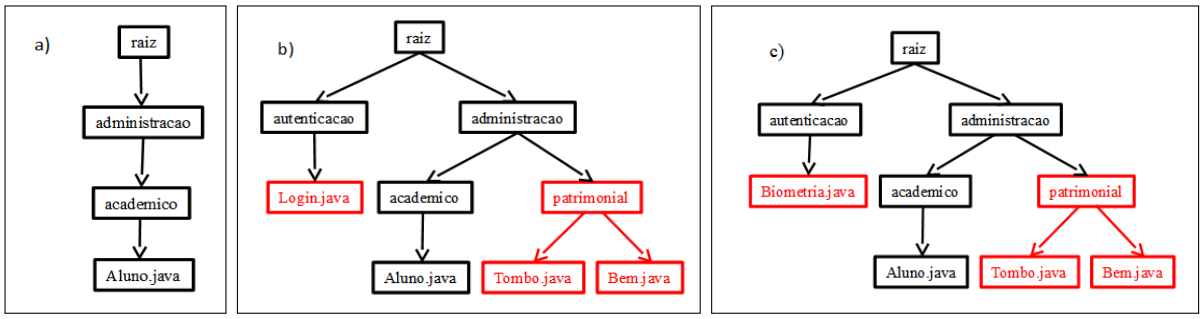

Figura 6 – Exemplo de aplicações que devem formar uma LPS: a) *Universidade1*, b) *Universidade2*, c) *Universidade3*.

O próximo passo é descobrir as comunalidades e variabilidades das aplicações. Para isso, a reCOVER executa um algoritmo que inicia a comparação a partir das classes. Para este exemplo, vamos considerar que as classes com o mesmo nome também possuem o mesmo conteúdo, ou seja, são comuns. Podemos notar que serão criados ao todo cinco grupos, um para cada classe comum: *Aluno*, *Bem*, *Tombo*, *Login* e *Biometria*, como mostra a Figura 7. Tendo-se realizado todos os devidos agrupamentos, deve-se analisar agora os pais dos itens agrupados e checa-se se esses elementos são comuns ou não, começando pelos grupos com as classes no maior nível, no caso o Nível 3. A classe *Aluno* pertence ao pacote chamado *academico* nas três aplicações. O grau de similaridade deste pacote é de 100 % entre todos, pois seu conteúdo é igual. Portanto, devemos agrupar esses pacotes também, colocando esse grupo como pai daquele, como mostra a Figura 8. A classe *Tombo*, por sua vez pertence ao pacote *patrimonial* nas duas versões em que aparece, o mesmo acontece com a classe *Bem* e esse pacote também possui 100% de seus elementos iguais. Por isso também é feito o agrupamento desse pacote.

A comparação prossegue então para os grupos cujos elementos estão no Nível 2, em uma busca em largura. Nesse exemplo verifica-se que a classe *Login* está sozinha em um grupo, portanto o pacote ao qual pertence, *autenticacao* já pode ser adicionado como nó pai sem a necessidade de cálculos adicionais de similaridade. Observe que, apesar da Aplicação 3 conter um pacote com o mesmo nome, eles não são comuns, pois seu conteúdo é

totalmente diferente. Com isso, na árvore resultante, esses dois pacotes serão considerados uma variabilidade. Essas atualizações podem ser visualizadas na Figura 9

O pacote *academico* pertence ao pacote *administracao* nas três aplicações, sendo que o conteúdo deste último difere entre as três versões, enquanto que nas aplicações 2 e 3 os pacotes possuem semelhança de 100%. A comparação do pacote da Aplicação 1 com o pacote da Aplicação 2, por exemplo, resulta numa similaridade de 66,67%, mas que ainda sim supera o limiar considerado, sendo *administracao* classificado portanto, como uma comunalidade.

Com todos os elementos do Nível 2 já associados com seus respectivos pais, a verificação deverá partir então para o próximo nível, o 1. Aqui todos elementos já se relacionam diretamente com o pacote raiz, não sendo necessárias, portanto, mais comparações; simplesmente associamos todos os pacotes nesse nível diretamente ao raiz, como mostra a Figura 10.

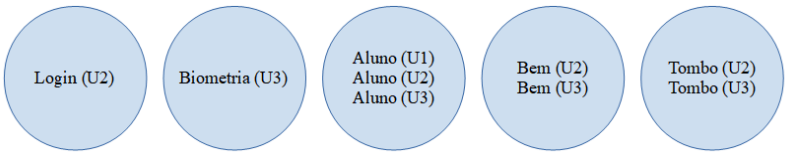

Figura 7 – Exemplo de agrupamento de classes comuns.

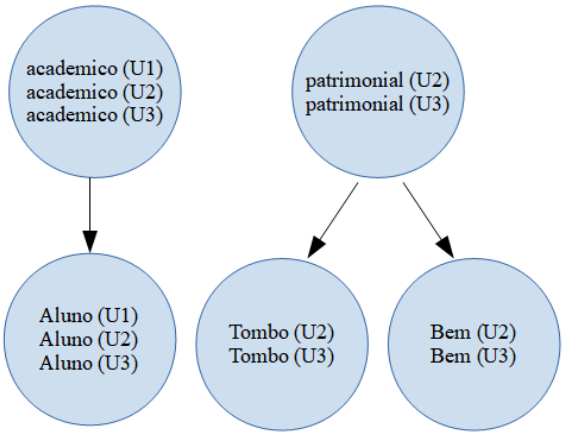

Figura 8 – Exemplo de agrupamento de pacotes comuns.

4.4 Considerações Finais

Este capítulo apresentou o método proposto neste trabalho para apoiar a adoção de LPS em uma família de software. Foi também apresentada a ferramenta que implementa o método no contexto de produtos de software desenvolvidos em Java. Foi explicado em que consiste cada uma das etapas de análise e um cenário foi apresentado a fim de exemplificar sua execução. No próximo capítulo será apresentado um cenário de avaliação do método e

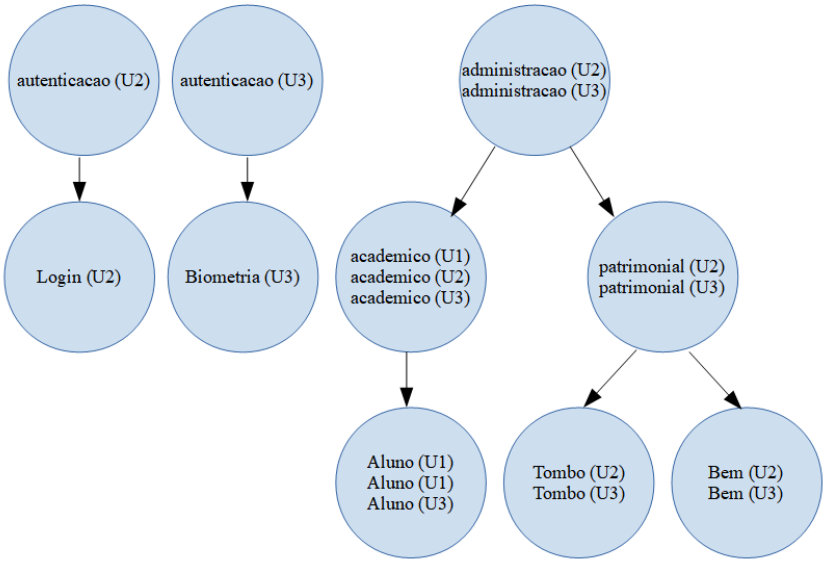

Figura 9 – Exemplo de agrupamento de pacotes.

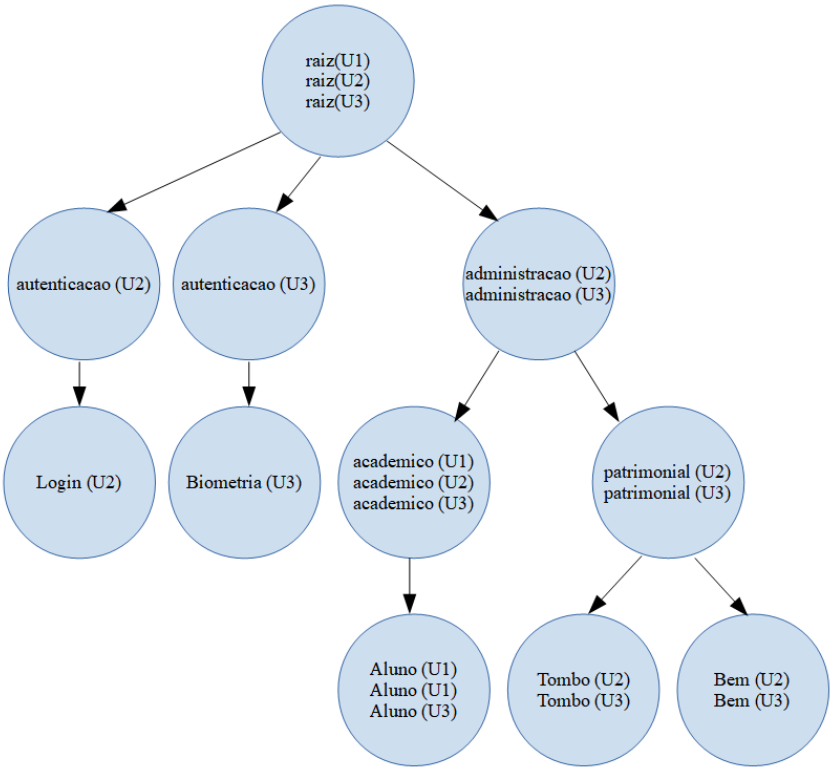

Figura 10 – Exemplo de agrupamento de pacotes.

da sua ferramenta, envolvendo tanto o ambiente acadêmico quanto um ambiente industrial de desenvolvimento de software.

5 Avaliação Experimental

Neste capítulo é apresentado o procedimento de avaliação do método e da ferramenta desenvolvidos durante este trabalho de mestrado com o objetivo de realizar uma avaliação inicial do método aqui proposto. São apresentados a estruturação da avaliação, passos para a sua realização e uma avaliação das ameaças associadas.

5.1 Planejamento

Nesta seção são definidos os objetivos do experimento executado, detalhando materiais e métodos utilizados, caracterização dos participantes e hipóteses.

5.1.1 Definição dos objetivos

O propósito da avaliação experimental realizada no contexto desta pesquisa é verificar se o método e a ferramenta aqui propostos podem de fato auxiliar as equipes de desenvolvimento de software no processo de identificação das partes comuns e variantes de uma família de produtos de software. O estudo experimental aqui realizado pode ser classificado como experimento, sendo realizado em dois ambientes controlados, sendo um na academia e outro na indústria.

O experimento objetivou avaliar os prós e contras com relação ao trabalho desenvolvido. Os participantes do experimento deveriam utilizar a ferramenta para gerar a visualização das comunalidades e variabilidades de uma família de produto de software e avaliar o quanto a mesma foi útil para esse processo.

Para a avaliação foi realizado um experimento constituído de duas fases. A primeira fase foi realizada com membros da academia, mais especificamente com estudantes da Universidade Federal do Piauí (UFPI) e foi denominado de experimento piloto, pois serviu como base e treino para a execução em um ambiente industrial. Além disso, permitiu averiguar a utilização da reCOVER por estudantes. A segunda fase contou com participantes da indústria, mais especificamente com desenvolvedores de uma empresa que atua na criação de sistemas para a área da saúde e que possui o contexto adequado para tal avaliação.

5.1.2 Questões de Pesquisa

O estudo experimental foi projetado para responder as questões abaixo. Para medirmos tais capacidades, foram elaboradas questões direcionadas, sobre as quais é detalhado mais à frente.

- A visualização gerada pela reCOVER melhora a identificação de um elemento como comunalidade ou variabilidade? A métrica associada a tal questão é o número de itens corretamente identificados, sejam itens comuns ou variáveis, nos produtos de software de uma família.

- A visualização da reCOVER auxilia na identificação do nível de similaridade de uma classe (ou pacote) em cada um dos produtos em questão? A métrica associada a tal questão é o número de acertos associados à percepção de uma classe em uma família.

5.1.3 Hipótese

A visualização de software busca favorecer a compreensão de produtos de software de uma maneira geral, em diferentes perspectivas e usando diferentes artefatos. Dessa forma a hipótese deste trabalho está relacionado a um aumento de percepção sobre um software, especialmente no que se refere às comunalidades e variabilidades. Formalizando, temos:

- Hipótese nula, H_0: o uso da ferramenta reCOVER não afeta a percepção das comunalidades e variabilidades de uma família de produto de software. Hipótese alternativa, H_1: o uso da ferramenta afeta a percepção das variabilidades e comunalidades entre produtos de software.

Acredita-se que em produtos de software de um modo geral, o fato de se criar diferentes versões, para se atender diferentes clientes, com diferentes necessidades, naturalmente gera mudanças contínuas na estrutura do software que tornam a percepção de tais diferenças entre os produtos algo muito difícil para as equipes de desenvolvimento.

5.1.4 Seleção de Variáveis

As variáveis independentes são: a ferramenta reCOVER e o sistema utilizado em cada contexto. A variável dependente é o nível de percepção das diferenças entre os produtos de software.

5.1.5 Seleção dos Participantes

O experimento aqui descrito contou com a participação de alunos da UFPI e de analistas de sistemas de uma empresa desenvolvedora de sistemas na área da saúde. Os alunos convidados são todos integrados ao ambiente de pesquisa, com bons conhecimentos da linguagem Java, que é a utilizada pela ferramenta desenvolvida. Antes do início do experimentos, os participantes preencheram um questionário de caracterização (disponível no Apêndice A), a partir do qual foram obtidos os dados apresentados a seguir. Como dito anteriormente, o experimento contou tanto com participantes da academia quanto da

indústria. Primeiramente serão apresentados os dados dos participantes do piloto e em seguida da indústria.

Participaram da fase piloto ao todo sete pessoas. Todos os voluntários eram estudantes da área de Ciência da Computação e dois afirmaram também já serem analistas de sistemas, conforme o gráfico mostrado na Figura 11. Já com relação à escolaridade, quatro possuíam curso superior incompleto (dos quais três já estavam no 8º período do curso), enquanto que três eram alunos de mestrado (um no 1º período e 2 no 3º), o gráfico da Figura 12 mostra os números em porcentagens.

Os conhecimentos em Java dos participantes também são relevantes para a pesquisa, pois para a execução do experimento esse critério é indispensável. A Figura 13 apresenta esses dados, que mostra que a maioria declara possuir conhecimentos avançados nessa linguagem.

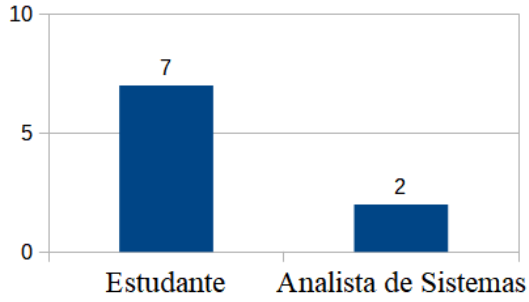

Figura 11 – Ocupação dos participantes do experimento piloto.

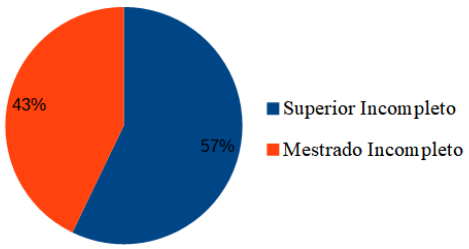

Figura 12 – Escolaridade dos participantes do experimento piloto.

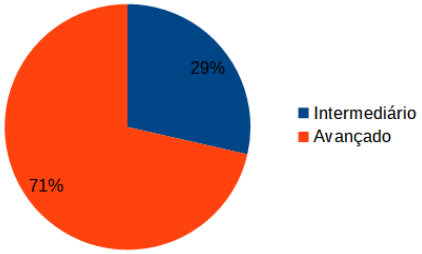

Figura 13 – Nível de conhecimento em Java dos participantes do experimento piloto.

Na execução do experimento na empresa de tecnologia, contou-se com a participação de três analistas de sistemas. A rotina em um ambiente de desenvolvimento impediu que

mais pessoas pudessem participar da pesquisa. Quanto à escolaridade, um possuía curso superior incompleto, outro superior completo e outro tinha mestrado completo, conforme gráfico da Figura 14.

Todos os participantes já possuíam um período de tempo considerável na empresa. O funcionário mais novo possuía três anos na casa, enquanto que o mais antigo já estava no cargo há mais de seis anos. Dessa forma, como já era de se esperar, os indivíduos já tinham um bom conhecimento da linguagem Java, como mostra o gráfico da Figura 15.

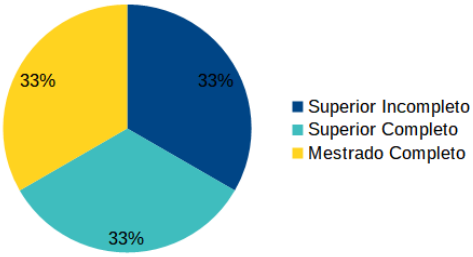

Figura 14 – Escolaridade dos participantes do experimento na indústria.

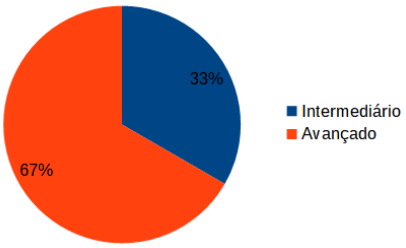

Figura 15 – Nível de conhecimento em Java dos participantes do experimento na indústria.

5.1.6 Sistemas alvo

Para a execução do experimento dentro da indústria foi escolhido o próprio sistema em que a equipe trabalhava, o que tornaria o experimento mais proveitoso, tanto para a empresa, que poderia conhecer melhor seus sistemas, quanto para o trabalho do mestrado, que teria uma execução em um cenário real. Entretanto, para a execução com os participantes da academia isso não foi possível, pois não havia um sistema comum em que todos os participantes trabalhassem. Portanto, para esses deveria ser utilizado um sistema pequeno, simples o suficiente para não causar problemas de entendimento e assim não atrapalhar a execução do experimento. Por outro lado, o sistema utilizado não poderia ser demasiadamente simples, senão poderia invalidar a execução; em outras palavras, deveria ter uma quantidade de classes que permitisse uma análise como a proposta.

Para esse ambiente, foi escolhido o sistema JBook, uma aplicação de gerenciamento de biblioteca desenvolvida por uma empresa de software para controlar seus livros e revistas. Na tentativa de tornar mais eficiente o entendimento do sistema, foram removidas alguns

Tabela 1 – Dados relativos aos sistemas utilizados no experimento realizado

Ambiente	Min. LOC	Max. LOC	Min. Pac.	Max. Pac.	Min. Clas.	Max. Clas.
Academia	1.598	1.773	12	13	20	22
Indústria	93.569	123.193	99	350	1.089	1.489

pacotes e classes, preservando aqueles que eram mais relevantes para o contexto. Para viabilizar a execução do experimento foram criadas dois produtos diferentes a partir do original a fim de possibilitar a comparação entre os mesmos. As variações foram criadas de maneira aleatória, alternando em alteração, exclusão ou inserção de classe, pacotes, atributos e métodos.

Para fins de comparação, mostramos na Tabela 1 algumas métricas dos sistemas utilizados: número de linhas de código, quantidade de pacote e quantidade de classes.

5.1.7 Treinamento

Antes de execução de cada uma das etapas foi realizada uma apresentação da reCOVER, seu objetivo, modo de usar e uma breve explicação de como a mesma realiza as comparações, quando os participantes também poderiam tirar dúvidas que pudessem vir a surgir no decorrer da explanação. Em seguida, foram passadas as tarefas a serem executadas por eles e o manual da ferramenta, a fim de evitar interferências durante a execução. Assim, em caso de dúvidas os participantes poderiam consultá-lo.

Uma vez que os participantes da academia não conheciam nada a respeito do sistema selecionado para sua avaliação, foi explicado de maneira breve seu objetivo e apresentados alguns pacotes e classes mais relevantes para o negócio. Esse esclarecimento foi necessário a fim de tornar a execução mais parecida com o contexto real no qual a reCOVER seria executada, ou seja, um ambiente de desenvolvimento onde todos os envolvidos conhecessem o sistema alvo. Com relação à execução na indústria essa etapa foi desnecessária, tendo em vista que todos já estavam familiarizados com o sistema em uso, tendo ocorrido apenas uma breve apresentação da reCOVER.

5.1.8 Projeto do Estudo

Foi elaborado um conjunto de tarefas a serem distribuídas a cada um dos voluntários da pesquisa. Essas tarefas consistiam em perguntas a respeito da comparação entre os diferentes produtos de um sistema alvo. Após a execução do experimento, as respostas de cada participante foram verificadas a fim de avaliarmos sua corretude. As questões deveriam ser respondidas inicialmente sem o uso da ferramenta, a partir da própria percepção dos participantes com o auxílio de uma IDE de desenvolvimento, em seguida os mesmos deveriam utilizar a reCOVER para responder as mesmas perguntas a fim de

confirmarem seus palpites iniciais ou refutar o que se acreditava.

Acredita-se que tal sequência de execução pode adicionar um viés na análise dos dados, uma vez que na segunda fase os participantes estariam respondendo as perguntas pela segunda vez e por consequência, já possuíam experiência naqueles casos. Para sanar o problema, uma alternativa seria dividi-los em dois grupos e cada um executaria as tarefas de maneira diferente, com e sem o uso da reCOVER. Entretanto tal abordagem seria desapropriada devido ao pequeno número de indivíduos, o que levaria a uma maior limitação da análise dos dados.

Para compor o conjunto de tarefas, foram escolhidas questões que pudessem destacar alguns pontos considerados fortes na ferramenta e outras que pudessem por à prova outros pontos que acreditava-se não serem tão fortes assim. Por isso as tarefas deveriam ser feitas com e sem o uso da ferramenta, para fins de comparação.

Antes da realização de cada fase, os participantes deveriam preencher um formulário de caracterização de perfil. Esse formulário é útil para avaliarmos os resultados obtidos de acordo com as características de quem executou. Após a execução os mesmos deveriam também preencher um questionário para colher *feedbacks* a respeito da ferramenta, disponível no Apêndice B. Essas questões são úteis a fim descobrir a percepção de outros desenvolvedores sobre a ferramenta e método aqui propostos.

5.1.9 Execução do Experimento

A Figura 16 resume as atividades efetuadas durante esse experimento. Primeiramente, seleção dos participantes, em seguida aplicação de questionário de caracterização, posteriormente o treinamento, execução das tarefas sem apoio da reCOVER seguida da execução com a ferramenta e por fim, o preenchimento de questionário pós-experimento, para colher os *feedbacks* dos participantes com relação à ferramenta.

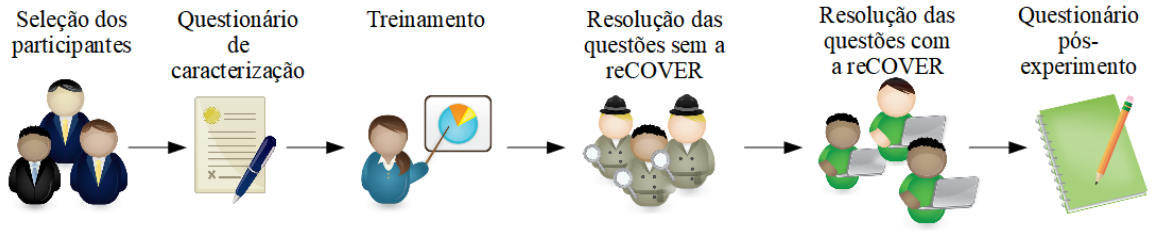

Figura 16 – Atividades desenvolvidas na avaliação experimental

Após as devidas apresentações, cada participante realizava as tarefas em um computador diferente. Cada um dos mesmos era responsável por contabilizar o tempo necessário para executar todas as tarefas, com e sem o uso da reCOVER. Ao final, os mesmos devolveram todas as perguntas com suas repostas de acordo com a sua perspectiva.

Para avaliação da eficácia da ferramenta foram elaboradas as questões abaixo, tanto para o experimento na academia, quanto na indústria. Em seguida, realizou-se uma verificação a fim de saber se as respostas estavam corretas. Para cada contexto, foram feitas as devidas adaptações para as classes solicitadas.

1. Localizar 3 pacotes comuns aos projetos;
2. Localizar 3 classes comuns aos projetos;
3. Procurar 3 classes presentes nos projetos mas que sofreram mudanças entre as versões
4. Encontrar 3 classes exatamente idênticas em todos os projetos;
5. Localizar o pacote que menos sofreu mudanças (incluindo seu conteúdo) entre todas as versões;
6. Procurar a classe X no projeto 2 e informe se ela se repete no outro sistema e caso afirmativo, diga o quanto são semelhantes (em %);
7. Informar, com relação às classes Y de cada um dos projetos, se são semelhantes ou diferentes.

5.1.9.1 Execução na Academia

Os gráficos a seguir mostram os resultados obtidos na execução no ambiente acadêmico. O gráfico da Figura 17 mostra o resumo das repostas relativas à primeira questão, apresentando a relação entre o número acertos e o número de participantes. Utilizando a reCOVER, todos os participantes conseguiram identificar corretamente todos os três pacotes solicitados na questão. Por outro lado, sem utilizar a ferramenta, apenas quatro conseguiram acertar na totalidade, contra três que acertaram somente dois.

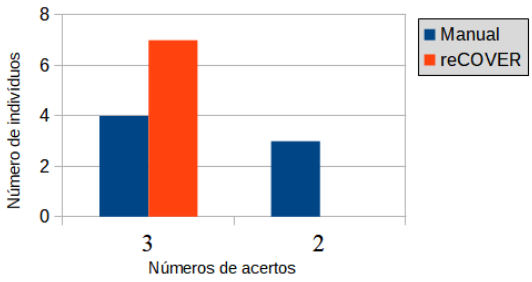

Figura 17 – Número de acertos com relação à questão 1 para o experimento na academia

A Figura 18 apresenta os dados relativos à questão 2. Tanto com a ferramenta quanto sem a ferramenta, todos os participantes conseguiram solucionar a questão em sua totalidade.

A Figura 19 apresenta os resultados para a questão de Número 3. Para esse problema observa-se que o uso da ferramenta gerou um ganho significativo com relação à execução

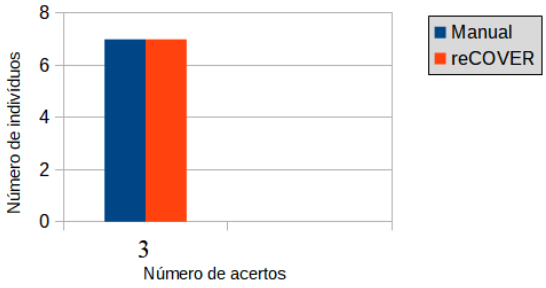

Figura 18 – Número de acertos com relação à questão 2 para o experimento na academia

sem o seu uso. Com a reCOVER, dobrou o número de participantes que acertaram todas as classes solicitadas, enquanto que a porcentagem de indivíduos que acertaram apenas duas questões caiu pela metade.

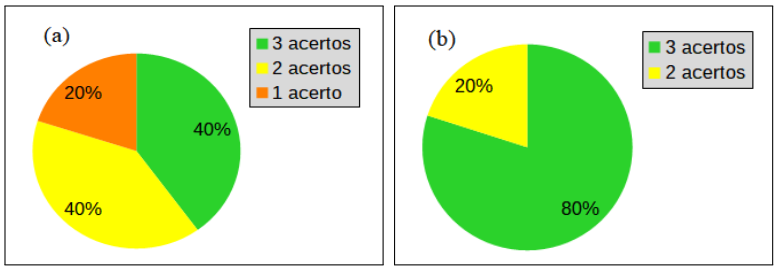

Figura 19 – Porcentagens de acertos com relação à questão 3 para o experimento na academia. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

O gráfico da Figura 20 apresenta os dados obtidos para a questão de número 4, a qual solicita a localização de três classes exatamente idênticas. O gráfico mostra uma diferença significativa dos resultados obtidos. De maneira semelhante à questão anterior, o número de participantes que acertou todas as classes solicitadas foi o dobro em relação àqueles que não usaram a ferramenta.

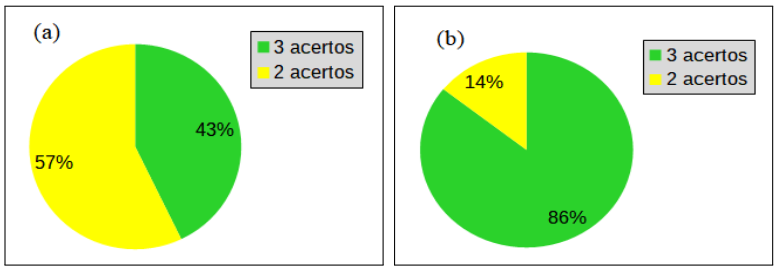

Figura 20 – Porcentagens de acertos com relação à questão 4 para o experimento na academia. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

A Figura 21 apresenta os dados para a questão 5, que solicitava ao participante informar o pacote que menos sofreu mudanças. Com relação a essa questão, o uso da ferramenta não apresentou grandes diferenças, embora tenha apresentado uma taxa pequena de melhoria no resultado final.

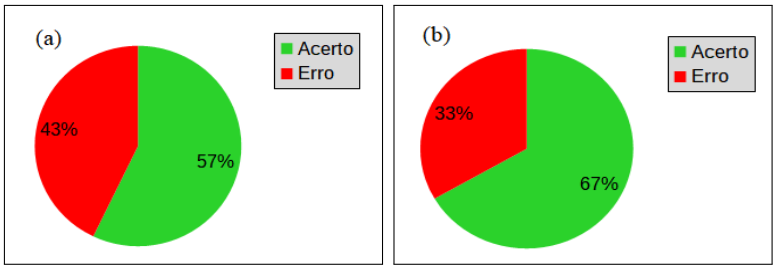

Figura 21 – Porcentagens de acertos com relação à questão 5 para o experimento na academia. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

A Figura 22 apresenta os dados relativos à questão 7, que indagava se determinada classe era semelhante ou não. Apesar do uso da ferramenta ter melhorado a taxa de acerto, que foi inferior a 50%, a mesma ainda foi superior em comparação com a execução sem a ferramenta.

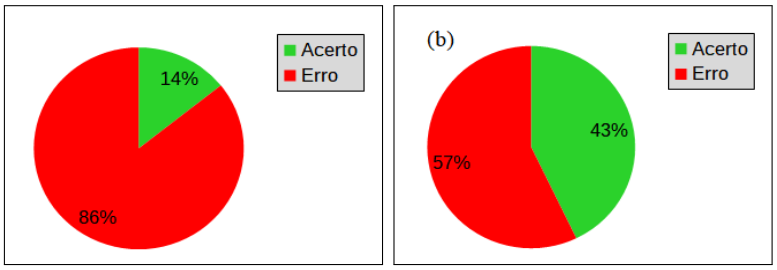

Figura 22 – Porcentagens de acertos com relação à questão 7 para o experimento na academia. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

Com relação à questão 6, foi proposta uma análise comparativa de uma mesma classe com duas versões diferentes nos dois sistemas utilizados, as quais possuíam cerca de 66,7% de semelhança. Dos sete participantes seis afirmaram que as classes possuíam pelo menos cerca de 90% de similaridade, e um afirmou 80%, o que destoou bastante dos dados reais. Já com o uso da ferramenta, todos conseguiram identificar acertadamente a referida porcentagem, o que mostra uma eficiência considerável no uso da ferramenta para esse tipo de questão.

5.1.9.2 Execução na Indústria

Infelizmente, pelo baixo número de participantes da indústria, não é possível obter conclusões precisas a respeito da aplicabilidade da ferramenta nesse contexto. Entretanto, é possível obter alguns indícios de suas vantagens nesse ambiente.

Os gráficos a seguir discutidos mostram os resultados obtidos na avaliação realizada em um ambiente industrial. O gráfico da Figura 23 mostra o resumo das repostas relativas à primeira questão, apresentando a relação entre o número de acertos e o número de participantes. Não houve grandes melhorias comparando-se as duas execuções para esse caso.

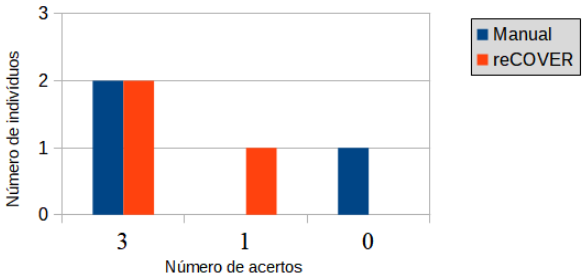

Figura 23 – Número de acertos com relação à questão 1 para o experimento na indústria

A Figura 24 apresenta os dados relativos à questão 2. Com relação a essa questão, houve uma maior melhoria se comparada à anterior. Sem o auxílio da ferramenta, um dos participantes acertou apenas um dos itens, outro acertou dois e outro acertou três. Já com o uso da reCOVER, dois participantes conseguiram acertar os três itens e apenas um acertou duas.

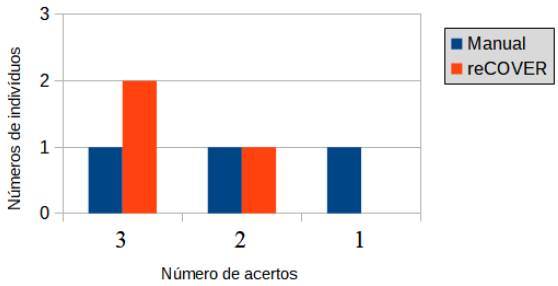

Figura 24 – Número de acertos com relação à questão 2 para o experimento na indústria

A Figura 25 apresenta os resultados para a questão de número 3. Para essa questão, observa-se que o uso da ferramenta teve um ganho significativo com relação à execução sem o seu uso. Sem a ferramenta, dois dos três participantes não acertaram nenhuma das classes solicitadas e apenas um acertou duas classes. Com a reCOVER, dobrou o número de participantes que acertaram duas classes e não houve mais participante que tivesse zerado essa questão.

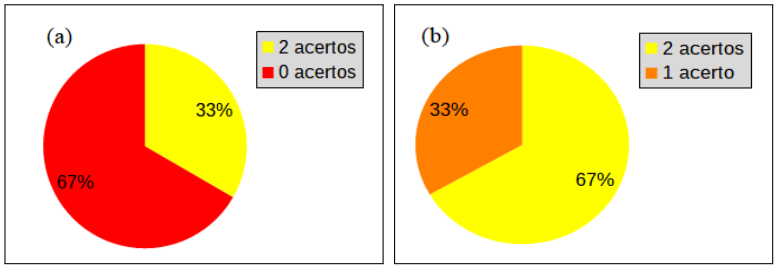

Figura 25 – Porcentagens de acertos com relação à questão 3 para o experimento na indústria. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

O gráfico da Figura 26 apresenta os dados obtidos para a questão de número 4, a qual solicita a localização de três classes exatamente idênticas. O gráfico também nos mostra

uma diferença significativa dos resultados obtidos. Sem o auxílio da ferramenta, cada um dos participantes conseguiu acertar apenas uma classe. Por outro lado, com a reCOVER, dois dos participantes conseguiram acertar todos os itens e um acertou duas questão.

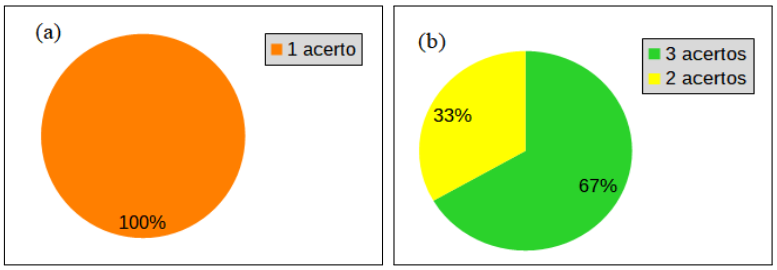

Figura 26 – Porcentagens de acertos com relação à questão 4 para o experimento na indústria. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

A Figura 27 apresenta os dados para a questão 5, que solicitava ao participante informar o pacote que menos sofreu mudanças. Com relação a essa problemática o uso da ferramenta não apresentou ganhos de acertos. Os participantes não conseguiram acertar a questão nem com o auxílio da ferramenta, nem sem a ajuda.

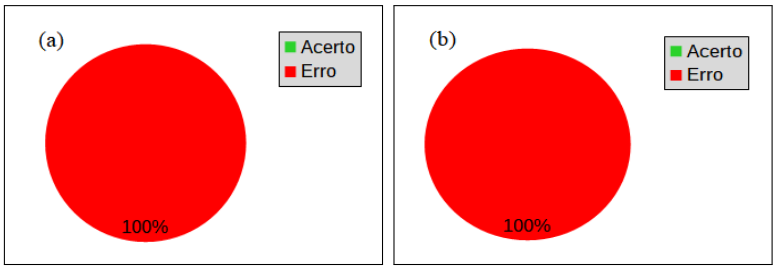

Figura 27 – Porcentagens de acertos com relação à questão 5 para o experimento na indústria. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

A Figura 28 apresenta os dados relativos à questão 7, que indagava se determinada classe era semelhante ou não. Sem o uso da ferramenta, apenas um dos participantes não conseguiu classificá-la corretamente. Já com o uso da reCOVER, a taxa de acerto foi de 100 %.

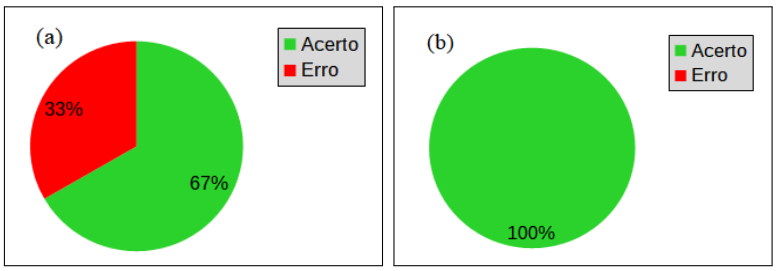

Figura 28 – Porcentagens de acertos com relação à questão 7 para o experimento na indústria. a) *Sem o uso da reCOVER*, b) *Com o uso da reCOVER*

Com relação à questão 6, foi proposta uma análise comparativa de uma mesma classe com suas três versões diferentes nos três sistemas utilizados. Na época de preparação do

Tabela 2 – Tempo gasto em cada uma das etapas do experimento

Ambiente	Método	Menor	Maior	Média
Academia	Manual	00:16:05	00:30:33	00:21:12
Academia	reCOVER	00:05:08	00:10:37	00:08:34
Indústria	Manual	00:13:00	00:19:00	00:16:55
Indústria	reCOVER	00:19:15	00:24:00	00:22:05

experimento, essas classes eram comuns de acordo com o limiar adotado nesse trabalho. Entretanto, no dia a execução do experimento, a equipe já havia feito bastantes alterações nesses arquivo o que acabou por torná-las variabilidades. Esse fato comprometeu a análise que se pretendia fazer com a questão proposta, que era verificar a similaridade em todos os sistemas.

Por isso, foi realizada apenas a análise dessa comparação entre os dois sistemas que conservavam essa classe com o maior índice de similaridade, as quais possuíam cerca de 81,3% de semelhança. Dos três participantes, apenas um indicou uma porcentagem próxima a esse valor, que foi 80%; outro participante informou se tratar de 60%, e o terceiro disse acreditar que as classes possuíam apenas 20% de semelhança, o que destoou bastante da porcentagem real. Já com o uso da ferramenta, um dos participantes não informou a porcentagem (talvez por esquecimento), outro informou 62,5% e outro ainda 100%.

5.2 Duração das execuções

A fim de se mensurar o esforço necessário para realizar cada uma das etapas realizadas, foi solicitado que cada participante contabilizasse o tempo despendido para realizar as tarefas em cada uma das fases. Os resultados podem ser conferidos na Tabela 2. Pode-se verificar que no ambiente acadêmico a resolução das tarefas com o apoio da reCOVER teve grande ganho de tempo em relação à execução manual. Lembramos que nessa etapa os participantes não conheciam os sistemas analisados, então dessa forma, o cálculo automatizado da ferramenta foi de grande ajuda.

Esse ganho de tempo, por outro lado não foi observado no ambiente industrial. De acordo com a tabela, pode-se observar que em geral o tempo da execução sem a ferramenta foi maior se comparado com a mesma execução no ambiente acadêmico. Isso se deve ao fato de que os participantes já conheciam o sistema, e dessa forma, apesar das aplicações serem bem maiores, eles poderiam responder às questões sem grandes esforços.

Por outro lado, pode-se afirmar que o aumento no tempo da execução utilizando a reCOVER não foi tão alto, sendo esse de cerca de 30%. Pois, embora esse tempo tenha sido maior, nessa etapa os acertos das questões também foram maiores, o que pode justificar

o uso da ferramenta. É importante ressaltar que a diferença de tamanho dos sistemas avaliados é muito grande. Enquanto que um encontra-se na casa de milhares de linha de código, o outro encontra-se na casa de centena de milhares de linhas de código.

5.3 *Feedbacks*

Após a execução dos experimentos, foi aplicado um questionário a fim de descobrir a opinião dos participantes sobre a reCOVER. Uma das perguntas deste questionário indagava a respeito do quanto cada um achava que a ferramenta foi útil para a visualização das semelhanças e diferenças entre os sistemas. Os dados obtidos são mostrados na Figura 29. Todos os participantes da academia escolheram o valor máximo. Enquanto isso, 66,6% dos voluntários da indústria escolheram o valor 3 e 33,3% a nota 4.

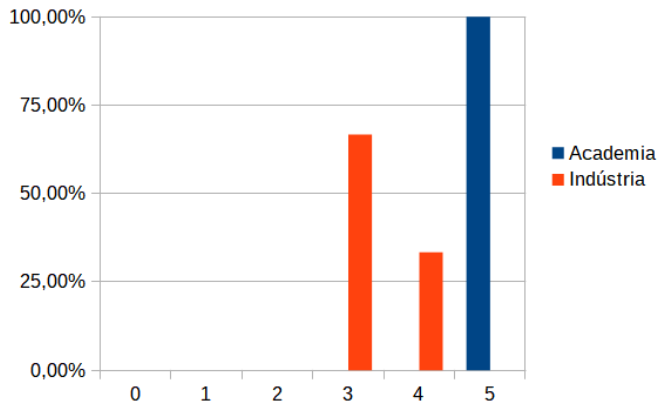

Figura 29 – Resposta dos participantes à pergunta: "Em uma escala de 0 a 5, como você considera que a reCOVER foi útil para a visualização das semelhanças e diferenças entre os sistemas abordados?"

Outra pergunta do questionário era com relação aos pontos positivos da reCOVER sob a perspectiva dos participantes. A maioria deles afirmou que a ferramenta facilita e agiliza a identificação das semelhanças entre os projetos.

Já com relação aos pontos negativos, a maioria dos participantes afirma ser a usabilidade o maior problema. São alguns dos relatos de melhoria possíveis a serem adotados na ferramenta a estrutura adotada na visualização, e os passos necessários para analisar determinado elemento na árvore de visualização.

Com relação às sugestões de melhorias na ferramenta, no geral, cada participante apontou questões distintas. Entretanto, chama a atenção o fato de dois dos mesmos afirmarem que seria mais interessante se o cálculo da similaridade incluísse todos os níveis acima do elemento em questão, e não apenas do nível imediatamente superior.

5.4 Discussão dos resultados

Os dados levantados com os experimentos permitiram identificar os pontos fortes e fracos da ferramenta desenvolvida. A seguir detalha-se as conclusões de acordo com as questões propostas nos experimentos.

A primeira questão pedia que os participantes citassem três pacotes comuns aos sistemas. Apesar de que no ambiente industrial a melhoria obtida para a resolução do problema tenha sido pequena, no meio acadêmico (o qual possuía número maior de participantes) foi possível notar resultados consideravelmente melhores com o uso da reCOVER.

A segunda questão era referente a localizar três classes comuns entre os projetos. O experimento na academia permitiu concluir que mesmo sem a ferramenta, os participantes não possuíam dificuldades para a identificação. Por outro lado, com uma equipe profissional de desenvolvimento foi possível identificar uma vantagem no uso da reCOVER para auxílio da resolução do problema.

As questões 3 e 4 foram as que possibilitaram maior percepção de benefício de uso da ferramenta. Ambas tratavam de verificar o quanto determinada classe possui de semelhança e diferença entre os sistemas alvo. A execução utilizando a reCOVER tanto no experimento na academia quanto na indústria apresentou dados significativamente melhores com relação à manual, principalmente na indústria. O resultado positivo decorre do fato de essas informações serem facilmente encontradas na estrutura de visualização escolhida e difíceis de serem calculadas sem um apoio automatizado.

O pior desempenho para o uso da reCOVER ocorreu para o problema da questão de número 5, que pedia o pacote com menos mudanças de um sistema para outro. Acredita-se que esse resultado tenha ocorrido devido à complexidade associada à busca desse tipo de informação na visualização gerada pela ferramenta.

A questão 7, por sua vez, apresentou resultados diferentes na indústria e na academia. Na primeira, não houve ganho significativo no uso da ferramenta. Na academia, por outro lado, houve uma grande benefício na resolução desse tipo de problema.

Com isso, uma conclusão associada à avaliação realizada, é que os pontos mais fortes da ferramenta consistem em detectar elementos comuns e o quanto de código que tal elemento possui de comum e diferente entre produtos de uma mesma família de software. Por outro lado a ferramenta não demonstrou um bom desempenho com relação a questões como identificação dos elementos que menos mudaram de uma versão para outro do sistema.

5.5 Ameaças à validade

Desde o planejamento do experimento aqui descrito buscou-se adotar técnicas de modo a tornar o mesmo o mais imparcial possível. Entretanto, sabe-se que em situações cotidianas podem ocorrer circunstâncias que possam comprometer os estudos pretendidos. Por isso é importante realizar uma estudo acerca das ameaças à validade associada a um estudo experimental (MALTERUD, 2001). Portanto, nesta seção, apresentamos alguns pontos que podem ter interferido nos resultados obtidos.

5.5.1 Validade Interna

A validade interna refere-se ao grau da acurácia dos resultados encontrados com relação à realidade observada. Ou seja, a validade interna avaliar se algum fator pode ter afetado a experimentação e assim distorceu o fato real. Nesta subseção e discutido o que se considera ameaças à validade interna.

No experimento em ambiente industrial, cada participante analisou os códigos que estavam em seu próprio computador de trabalho. Dessa forma, há a possibilidade de que estes não estivessem sincronizados, fazendo com que cada um estivesse analisando códigos diferentes e consequentemente gerando resultados diferentes, mas igualmente corretos. Porém, na aferição das respostas dos mesmos, estava sendo observado um único código fonte para cada uma das aplicações, tendo em vista que não tínhamos acesso ao código na máquina de cada um dos participantes. Esse fato pode ter ocasionado correção indevida. Entretanto, espera-se que isso possa ter tido um impacto mínimo, uma vez que entende-se que embora em alguns momentos seus códigos se diferencie durante a fase de implementação, eles sejam constantemente sincronizados a fim de minimizar os erros de integração. Além disso, as questões envolviam muitos pacotes e classes diferentes, o que pode minimizar a taxa de possíveis inconsistências.

5.5.2 Validade Externa

A validade externa refere-se à capacidade de os resultados encontrados serem generalizados para outras populações de indivíduos. Assim, nesta subseção discutimos fatores que podem dificultar a generalização dos dados encontrados em nosso experimento.

Nem sempre é possível contar com um forte apoio de uma empresa de software para execução de atividades científicas como a aqui proposta. Por exemplo, a empresa parceira escolhida para este projeto não pôde disponibilizar um grande número de profissionais para participar do experimento proposto, pois a participação dos mesmos implica em uma pausa no trabalho por eles desenvolvido. Assim, o fato de apenas três profissionais terem participado da prática pode ter acarretado em dados não reais, pois a amostra é pequena.

A falta de espaço na indústria implica ainda em outro problema. Para sanar tal ausência, recrutou-se alguns alunos de pesquisa da UFPI para participarem da avaliação. Entretanto este não é uma amostra real do público alvo da ferramenta, que é dirigida para equipes de desenvolvimento em fábricas de software.

Conclusões e Trabalhos Futuros

Este capítulo apresenta as principais conclusões do trabalho, relembrando as contribuições esperadas e delineando os próximos passos a serem executados.

Contribuições

Este trabalho teve como objetivo propor um método e uma ferramenta para auxiliar as empresas na adoção de uma LPS através da identificação das variabilidades e comunalidades em uma família de produtos. Nas seções anteriores foi discutido a respeito da importância do reuso planejado e como a LPS pode ser uma boa alternativa para as empresas. Também foi abordada a pertinência de trabalhos acerca de visualização de software e como essa pode auxiliar no entendimento de produtos. Espera-se que este trabalho possa auxiliar equipes a identificar com mais facilidade o código que produtos de uma mesma família compartilham entre si e o código que é individualizado, existente em apenas algumas versões. Espera-se ainda que essa facilitação possa contribuir de maneira positiva para o processo de planejamento de adoção da LPS por meio da abordagem extrativa.

Trabalhos estudados durante a pesquisa bibliográfica permitiram observar que de fato o custo no processo de adoção de uma LPS na indústria é uma preocupação compartilhada por diversos pesquisadores, o que pode apontar semelhante preocupação por parte das grandes equipes de desenvolvimento que consideram a adoção. Tal percepção obtida com o levantamento encorajou os autores deste trabalho a prosseguirem nessa linha de pesquisa, tendo em vista a sua relevância, pertinência e interesse por parte da academia e indústria.

Durante o decorrer deste mestrado foi desenvolvida a ferramenta reCOVER, a qual consegue gerar uma visualização das porções comuns e variantes do código de uma família de produtos. Um experimento foi realizado em duas etapas: na academia e na indústria, com sistemas reais (adaptado para a execução com os alunos). O experimento ajudou a atestar o potencial de eficiência da ferramenta, seus pontos fortes e fracos. A reCOVER demonstrou maior potencial na identificação dos elementos de código que são comuns aos projetos, bem como auxilia a identificação de elementos variantes entre os produtos de uma família.

Durante a realização do experimento, com o uso da ferramenta no ambiente industrial, foi possível exibir informações não conhecidas pela própria equipe de desenvolvimento, que é composta por pessoas que participaram do projeto desde o seu início. De um modo geral, a percepção das diferenças entre as três versões existentes na empresa participante do experimento já não se mostrava de forma tão clara nem mesmo para os desenvolvedores

do projeto. A visualização apresentada na reCOVER permitiu à equipe entender melhor as semelhanças entre produtos e suas diferenças, conhecimento que por vezes é deixado de lado frente às rotinas de trabalho, uma vez que as três variantes já existem há bastante tempo. O produto mais recente possui dois anos de criação e os outros dois possuem mais de quatro anos de desenvolvimento. Além disso, a ferramenta desenvolvida neste trabalho foi apreciada pela equipe de desenvolvimento da empresa, afirmando ser de relevância para o seu negócio.

Trabalhos Futuros

Considerando o estado atual do trabalho aqui apresentado, abaixo são propostos alguns trabalhos futuros que podem seguir a partir deste mestrado.

1. Aplicação do experimento em outros ambientes industriais que possibilitem a abrangência de um maior número de participantes;
2. Melhoria da interface da ferramenta, de modo a propiciar o melhor entendimento das questões propostas neste trabalho;
3. Extensão do método para abranger as outras atividades que envolvem a adoção de LPS.

Referências

ABDEEN, H. et al. The package blueprint: Visually analyzing and quantifying packages dependencies. *Science of Computer Programming*, Elsevier, v. 89, p. 298–319, 2014. Citado na página 13.

AJILA, S. A. Reusing base product features to develop product line architecture. In: *IRI -2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.* [S.l.: s.n.], 2005. p. 288–293. Citado na página 17.

ALVES, V. et al. Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, v. 52, n. 8, p. 806 – 820, 2010. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584910000625>. Citado na página 11.

APEL, S. et al. *Feature-Oriented Software Product Lines*. [S.l.]: Springer, 2016. Citado na página 8.

ASSUNÇÃO, W. K. G. et al. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering*, Feb 2017. Disponível em: <https://doi.org/10.1007/s10664-017-9499-z>. Citado na página 2.

ASSUNÇÃO, W. K. G.; VERGILIO, S. R. Feature location for software product line migration: A mapping study. In: ACM. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*. [S.l.], 2014. p. 52–59. Citado 2 vezes nas páginas 1 e 2.

BAKAR, N. H. et al. Extracting software features from online reviews to demonstrate requirements reuse in software engineering. In: *6th International Conference of Computing and Informatics*. [S.l.: s.n.], 2017. p. 184–190. Citado na página 16.

BASTARRICA, M. C.; RIVAS, S.; ROSSEL, P. O. From a single product architecture to a product line architecture. In: *Chilean Society of Computer Science, 2007. SCCC '07. XXVI International Conference of the*. [S.l.: s.n.], 2007. p. 115–122. ISSN 1522-4902. Citado na página 17.

BAXTER, I. D. et al. Clone detection using abstract syntax trees. In: IEEE. *ICSM - 1998. Proceedings of the International Conference on Software Maintenance*. [S.l.], 1998. p. 368–377. Citado 2 vezes nas páginas 12 e 19.

BERGER, C.; RENDEL, H.; RUMPE, B. Measuring the ability to form a product line from existing products. *arXiv preprint arXiv:1409.6583*, 2014. Citado na página 22.

BOTTERWECK, G. et al. Visual tool support for configuring and understanding software product lines. In: *2008 12th International Software Product Line Conference*. [S.l.: s.n.], 2008. p. 77–86. Citado na página 16.

CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. *Readings in information visualization: using vision to think*. [S.l.]: Morgan Kaufmann, 1999. Citado na página 13.

CLEMENTS, P. Being proactive pays off. *IEEE Software*, IEEE, v. 19, n. 4, p. 28–30, 2002. Citado na página 2.

CLEMENTS, P.; NORTHROP, L. *Software product lines: Practices and Patterns*. [S.l.]: Addison-Wesley, 2002. Citado na página 9.

DURA, Ö.; YILMAZ, A. E. Software product line development: A review on practical issues and challenges. In: IEEE. *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*. [S.l.], 2009. p. 736–742. Citado na página 3.

DUSZYNSKI, S.; KNODEL, J.; BECKER, M. Analyzing the source code of multiple software variants for reuse potential. In: *2011 18th Working Conference on Reverse Engineering*. [S.l.: s.n.], 2011. p. 303–307. ISSN 1095-1350. Citado 2 vezes nas páginas 18 e 23.

EYAL-SALMAN, H.; SERIAI, A.-D. Toward recovering component-based software product line architecture from object-oriented product variants. In: *Proceedings of the Software Engineering and Knowledge Engineering (SEKE)*. [S.l.: s.n.], 2016. Citado na página 17.

FISCHER, S. et al. Enhancing clone-and-own with systematic reuse for developing software variants. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. [S.l.: s.n.], 2014. p. 391–400. ISSN 1063-6773. Citado 2 vezes nas páginas 11 e 23.

FOUNDATION, T. E. *Eclipse Java development tools (JDT)*. 2017. [Online; acessado em 15-08-2017]. Disponível em: <http://www.eclipse.org/jdt/>. Citado na página 25.

IANZI, A. *Definição de Escopo em Linhas de Produto de Software: uma Abordagem Semiautimática utilizando Anotaçao Linguística*. 2013. Citado 2 vezes nas páginas 13 e 10.

IEEE. *IEEE Standard Glossary of Softaware Engeneering Terminology*. [S.l.], 1990. Citado na página 1.

JACOBSON, I. et al. *The unified software development process*. [S.l.]: Addison-wesley Reading, 1999. v. 1. Citado na página 17.

JIANG, L. et al. Deckard: Scalable and accurate tree-based detection of code clones. In: *Proceedings of the 29th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (ICSE '07), p. 96–105. ISBN 0-7695-2828-7. Disponível em: <http://dx.doi.org/10.1109/ICSE.2007.30>. Citado na página 12.

JÚNIOR, R. A. d. L. *Comparação entre Ferramentas para Linha de Produtos de Software*. 2008. Monografia apresentada ao curso de Engenharia da Computação da Universidade Federal de Pernambuco. Citado na página 3.

KAMIYA, T.; KUSUMOTO, S.; INOUE, K. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. In: IEEE. *IEEE Transactions on Software Engineering*. [S.l.], 2002. p. 654–670. Citado na página 12.

KANG, K. C. et al. *Feature-oriented domain analysis (FODA) feasibility study*. [S.l.], 1990. Citado na página 17.

KHODAI, E.; PERUMAL, A.; KANMANI, S. Clone detection using textual and metric analysis to figure out all types of clones. *International Journal of Computer Communication and Information System*, v. 2, n. 1, p. 99–103, 2010. Citado na página 12.

KLATT, B.; KÜSTER, M.; KROGMANN, K. A graph-based analysis concept to derive a variation point design from product copies. In: *International Workshop on Reverse Variability Engineering*. [S.l.: s.n.], 2013. p. 1–8. Citado 2 vezes nas páginas 16 e 17.

KNAUBER, P. et al. Quantifying product line benefits. In: SPRINGER. *International Workshop on Software Product-Family Engineering*. [S.l.], 2001. p. 155–163. Citado na página 9.

KOMONDOOR, R.; HORWITZ, S. Using slicing to identify duplication in source code. In: *Proceedings of the 8th International Symposium on Static Analysis*. London, UK, UK: Springer-Verlag, 2001. (SAS '01), p. 40–56. ISBN 3-540-42314-1. Disponível em: <http://dl.acm.org/citation.cfm?id=647170.718283>. Citado na página 12.

KRUEGER, C. Easing the transition to software mass customization. In: *Software Product-Family Engineering*. [S.l.]: Springer, 2001. p. 282–293. Citado 2 vezes nas páginas 2 e 10.

KRUEGER, C. W.; JUNGMAN, M. N. *Software customization system and method*. [S.l.]: Google Patents, 2009. US Patent 7,543,269. Citado na página 3.

KRüGER, J. et al. Extracting software product lines: A cost estimation perspective. In: *Proceedings of the 20th International Systems and Software Product Line Conference*. New York, NY, USA: ACM, 2016. (SPLC '16), p. 354–361. ISBN 978-1-4503-4050-2. Disponível em: <http://doi.acm.org/10.1145/2934466.2962731>. Citado na página 11.

KUHN, T.; THOMANN, O. *Abstract Syntax Tree*. 2006. [Online; acessado em 27-Junho-2016]. Disponível em: <http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/>. Citado na página 25.

KULA, R. G. et al. Visualizing the evolution of systems and their library dependencies. In: IEEE. *Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on*. [S.l.], 2014. p. 127–136. Citado na página 13.

LIN, Y. et al. Detecting differences across multiple instances of code clones. In: *Proceedings of the 36th Internacional Conference on Software Engineering*. [S.l.]: ACM, 2014. (ICSE 2014). Citado na página 12.

LINDEN, F. J. van der; ROMMES, E.; SCHMID, K. *Software Product Lines in Action*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2007. Citado na página 1.

LOESCH, F.; PLOEDEREDER, E. Optimization of variability in software product lines. In: *11th International Software Product Line Conference (SPLC 2007)*. [S.l.: s.n.], 2007. p. 151–162. Citado 2 vezes nas páginas 13 e 16.

MACALA, R. R.; STUCKEY, L. D.; GROSS, D. C. Managing domain-specific, product-line development. *IEEE Software*, v. 13, n. 3, p. 57–67, May 1996. ISSN 0740-7459. Citado na página 11.

MALTERUD, K. Qualitative research: standards, challenges, and guidelines. *The lancet*, Elsevier, v. 358, n. 9280, p. 483–488, 2001. Citado na página 45.

MARTINEZ, J. et al. Bottom-up technologies for reuse: Automated extractive adoption of software product lines. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. [S.l.: s.n.], 2017. p. 67–70. Citado na página 3.

MARTINEZ, J. et al. Bottom-up adoption of software product lines: a generic and extensible approach. In: ACM. *Proceedings of the 19th International Conference on Software Product Line*. [S.l.], 2015. p. 101–110. Citado na página 17.

MATTILA, A.-L. et al. Software visualization today: Systematic literature review. In: *Proceedings of the 20th International Academic Mindtrek Conference*. [S.l.: s.n.], 2016. p. 262–271. Citado na página 13.

NESTOR, D. et al. Visualisation of variability in software product line engineering. In: *International Workshop on Variability Modelling of Software-intensive Systems*. [S.l.: s.n.], 2007. Citado na página 16.

NUNES, C. et al. History-sensitive heuristics for recovery of features in code of evolving program families. In: *Proceedings of the 16th International Software Product Line Conference - Volume 1*. New York, NY, USA: ACM, 2012. (SPLC '12), p. 136–145. ISBN 978-1-4503-1094-9. Disponível em: <http://doi.acm.org/10.1145/2362536.2362556>. Citado na página 18.

OLIVEIRA, J. et al. A method based on naming similarity to identify reuse opportunities. In: *Proceedings of the Brazilian Symposium on Information Systems (SBSI)*. [S.l.: s.n.], 2016. Citado na página 19.

OLIVEIRA, J. A. d.; FERNANDES, E. M.; FIGUEIREDO, E. Evaluation of duplicated code detection tools in cross-project context. In: *Proceedings of the 3rd Workshop on Software Visualization, Evolution, and Maintenance (VEM)*. [S.l.: s.n.], 2015. p. 49–56. Citado na página 19.

POHL, K.; BöCKLE, G.; LINDEN, F. J. v. d. *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 3540243720. Citado 6 vezes nas páginas 1, 2, 8, 9, 10 e 11.

PRECHELT, L. An empirical comparison of c, c++, java, perl, python, rexx and tcl. *IEEE Computer*, v. 33, n. 10, p. 23–29, 2000. Citado na página 25.

PRICE, B. A.; BAECKER, R. M.; SMALL, I. S. A principled taxonomy of software visualization. *Journal of Visual Languages & Computing*, Elsevier, v. 4, n. 3, p. 211–266, 1993. Citado na página 13.

RICHENHAGEN, J. et al. Test-driven semantical similarity analysis for software product line extraction. In: *Proceedings of the 20th International Systems and Software Product Line Conference*. New York, NY, USA: ACM, 2016. (SPLC '16), p. 174–183. ISBN 978-1-4503-4050-2. Disponível em: <http://doi.acm.org/10.1145/2934466.2934483>. Citado 2 vezes nas páginas 15 e 22.

RINCóN, L. et al. Extractive spl adoption applied into a small software company. In: *2016 XLII Latin American Computing Conference (CLEI)*. [S.l.: s.n.], 2016. p. 1–8. Citado na página 11.

ROSSI, A. C. *Representação do componente de software na FARCSoft: ferramenta de apoio à reutilização de componentes de software*. Dissertação (Mestrado) — Escola Politécnica, Universidade de São Paulo, São Paulo, 2004. Citado na página 8.

RUBIN, J.; CHECHIK, M. Combining related products into product lines. In: *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2012. (FASE'12), p. 285–300. ISBN 978-3-642-28871-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-28872-2_20>. Citado na página 22.

RUBIN, J.; CHECHIK, M. A framework for managing cloned product variants. In: *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013. (ICSE '13), p. 1233–1236. ISBN 978-1-4673-3076-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2486788.2486971>. Citado na página 7.

RUBIN, J.; CZARNECKI, K.; CHECHIK, M. Managing cloned variants: A framework and experience. In: *Proceedings of the 17th International Software Product Line Conference*. New York, NY, USA: ACM, 2013. (SPLC '13), p. 101–110. ISBN 978-1-4503-1968-3. Disponível em: <http://doi.acm.org/10.1145/2491627.2491644>. Citado na página 22.

RUBIN, J. et al. Managing forked product variants. In: *Proceedings of the 16th International Software Product Line Conference - Volume 1*. New York, NY, USA: ACM, 2012. (SPLC '12), p. 156–160. ISBN 978-1-4503-1094-9. Disponível em: <http://doi.acm.org/10.1145/2362536.2362558>. Citado na página 8.

RYSSEL, U.; PLOENNIGS, J.; KABITZSCH, K. Automatic variation-point identification in function-block-based models. In: *Proceedings of the Ninth International Conference on Generative Programming and Component Engineering*. New York, NY, USA: ACM, 2010. (GPCE '10), p. 23–32. ISBN 978-1-4503-0154-1. Disponível em: <http://doi.acm.org/10.1145/1868294.1868299>. Citado 2 vezes nas páginas 18 e 22.

SELLIER, D.; MANNION, M. Visualising product line requirement selection decision inter-dependencies. In: *Second International Workshop on Requirements Engineering Visualization (REV 2007)*. [S.l.: s.n.], 2007. p. 7–7. Citado 2 vezes nas páginas 12 e 13.

SOMMERVILLE, I. *Engenharia de Software*. 8. ed. [S.l.]: Pearson Education do Brasil, 2007. Citado 2 vezes nas páginas 7 e 8.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: Pearson Education do Brasil, 2011. Citado na página 1.

THIEL, S.; HEIN, A. Modeling and using product line variability in automotive systems. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 19, n. 4, p. 66–72, jul. 2002. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2002.1020289>. Citado na página 13.

TONSCHEIDT, K. *Leveraging Code Clone Detection for the Incremental Migration of Cloned Product Variants to a Software Product Line: An Explorative Study*. 2015. Bachelor Thesis. Citado na página 21.

VYATKIN, V.; AMERICA, I. S. of. *IEC 61499 function blocks for embedded and distributed control systems design*. [S.l.]: ISA, 2007. Citado na página 18.

WILDE, E.; GERMAN, D. Merge-tree: Visualizing the integration of commits into linux. In: IEEE. *Software Visualization (VISSOFT), 2016 IEEE Working Conference on*. [S.l.], 2016. p. 1–10. Citado na página 13.

Apêndices

APÊNDICE A – Formulário de perfil de participantes

Endereço de e-mail: _____

Nome Completo: _____

Como você define seus conhecimentos com relação à linguagem Java?

( ) Básico ( ) Intermediário ( ) Avançado ( ) Não conhece

Quanto tempo de experiência você possui com desenvolvimento de sistemas?

( ) Não possuo ( ) Menos de 1 ano ( ) Entre 1 e 2 anos ( ) Mais de 2 anos

Ocupação:

( ) Estudante

( ) Analista de Sistemas/Desenvolvedor/Programador

( ) Outros

Escolaridade:

( ) Doutorado Completo ( ) Doutorado Incompleto

( ) Mestrado Completo ( ) Mestrado Incompleto

( ) Superior Completo ( ) Superior Incompleto

( ) Ensino Médio

Formação:

( ) Ciência da Computação e afins ( ) Outros

Caso seja estudante, informe o nome da sua Instituição de Ensino e qual ano/período está cursando: _____

Caso trabalhe, informe o nome da empresa em que trabalha e há quanto tempo está vinculado a ela: _____

APÊNDICE B – Questionário Pós-experimento

Nome Completo: _____

Em uma escala de 0 a 5, como você considera que a reCOVER foi útil para a visualização das semelhanças e diferenças entre os sistemas abordados? (0 - não muito útil; 5 - muito útil) _____

Quais os pontos positivos observados com o uso da ferramenta?

_____

_____

_____

_____

Quais os pontos negativos observados com o uso da ferramenta?

_____

_____

_____

_____

Você gostaria de sugerir melhorias para a ferramenta utilizada? Quais?

_____

_____

_____

_____