LINHA DE PRODUTOS PARA VISUALIZAÇÃO DE SOFTWARE: UMA INFRAESTRUTURA PARA APOIAR ATIVIDADES DE COMPREENSÃO POR MEIO DA CONSTRUÇÃO DE MECANISMOS DE VISUALIZAÇÃO

Marlon Alves da Silva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadora: Cláudia Maria Lima Werner

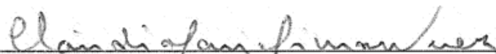
Rio de Janeiro
Dezembro de 2012

LINHA DE PRODUTOS PARA VISUALIZAÇÃO DE SOFTWARE: UMA INFRAESTRUTURA PARA APOIAR ATIVIDADES DE COMPREENSÃO POR MEIO DA CONSTRUÇÃO DE MECANISMOS DE VISUALIZAÇÃO

Marlon Alves da Silva

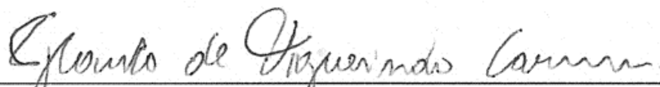
DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:



Prof.ª Cláudia Maria Lima Werner, D. Sc.

Prof. Toacy Cavalcante de Oliveira, D. Sc.



Prof. Glauco de Figueiredo Carneiro, D. Sc.

Prof. Leonardo Gresta Paulino Murta, D. Sc.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2012

Silva, Marlon Alves da

Linha de Produtos para Visualização de Software: uma infraestrutura para apoiar atividades de compreensão por meio da construção de mecanismos de visualização/ Marlon Alves da Silva. – Rio de Janeiro: UFRJ/COPPE, 2012.

XII, 111 p.: il.; 29,7 cm.

Orientadora: Cláudia Maria Lima Werner

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2012.

Referências Bibliográficas: p. 100-103.

1. Linha de Produtos de Software. 2. Visualização de Software. 3. Engenharia de Software. I. Werner, Cláudia Maria Lima *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A Deus e a minha esposa Dayana

AGRADECIMENTOS

A Deus, por estar sempre presente em minha vida operando novos milagres e ensinando a cada o momento o passo a ser dado. À Dayana, minha amada esposa, que dedicou seu carinho, seu tempo, suas preciosas horas de sono, para estar me acompanhando e auxiliando nesta jornada. Em meu coração fica a certeza de que o levantar nos momentos mais difíceis só foi possível por ter a sua presença ao meu lado nesta caminhada.

Aos meus pais, que me criaram ensinando valores que podem fazer a diferença na sociedade em que vivemos. A todos os meu parentes e familiares, que direta ou indiretamente participaram da minha vida, tanto nos momentos de celebração como nos momentos de tempestade. Aos amigos que, mesmo sem saber, cultivaram mais alegria no meu dia-a-dia.

À minha orientadora Cláudia Werner, por todo o ciclo de ensino que vivenciei no grupo de reutilização e por todo o esforço e sacrifícios feitos em prol desta defesa de dissertação. Ao Marcelo Schots, pela dedicação em também participar, aconselhar e contribuir com o desenvolvimento deste trabalho. A todo o grupo de Reutilização da COPPE/UFRJ onde amigos e colegas de pesquisas me fizeram crescer e aprender cada vez mais.

À FAPERJ, pelo apoio financeiro durante o mestrado por meio do programa mestrado Nota 10. Os artigos aprovados tanto no WMSWM 2012 como no WTDSoft 2011 são frutos deste apoio. À CAPES e ao CNPq, pelo auxílio em eventos e na apresentação de trabalhos. Aos professores Toacy Cavalcante, Glauco Carneiro e Leonardo Murta por terem aceitado participar desta banca de defesa de mestrado. Aos funcionários do PESC, pela atenção e colaboração durante todos os processos necessários ao longo deste ciclo.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

LINHA DE PRODUTOS PARA VISUALIZAÇÃO DE SOFTWARE: UMA INFRAESTRUTURA PARA APOIAR ATIVIDADES DE COMPREENSÃO POR MEIO DA CONSTRUÇÃO DE MECANISMOS DE VISUALIZAÇÃO

Marlon Alves da Silva

Dezembro/2012

Orientadora: Cláudia Maria Lima Werner

Programa: Engenharia de Sistemas e Computação

Diferentes metodologias e ferramentas foram propostas no campo de visualização de software para melhorar o entendimento sobre as propriedades do sistema, uma das atividades mais difíceis e custosas durante a manutenção de software. No entanto, a construção de tais mecanismos de visualização é considerada de alta complexidade, dado o conhecimento necessário, esforço e tempo para o seu desenvolvimento. Neste sentido, este trabalho apresenta uma infraestrutura para a construção de mecanismos de visualização de software, que é denominada Linha de Produtos para Visualização de Software (LPVS), com o objetivo de proporcionar maior flexibilidade na criação e combinação de visualizações para apoiar as atividades de manutenção de software, com menos tempo e esforço na sua construção. Neste sentido, um estudo de observação do uso da infraestrutura desenvolvida neste trabalho também é descrito com o objetivo de caracterizar as principais dificuldades, limitações e benefícios da mesma.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SOFTWARE VISUALIZATION PRODUCT LINE: AN INFRAESTRUCTURE FOR SUPPORTING COMPREHENSION ACTIVITIES BY VISUALIZATION MECHANISMS GENERATION

Marlon Alves da Silva

December/2012

Advisor: Cláudia Maria Lima Werner

Department: Systems and Computing Engineering

Different methodologies and tools have been proposed in the software visualization field to improve the understanding about system properties, one of the most difficult and expensive activities during software maintenance. Nevertheless, the construction of these visualization mechanisms is considered of high complexity, given the necessary knowledge, effort and time for their development. In this sense, this work presents an infrastructure for building software visualizations, which is called Software Visualization Product Line (SVPL), aiming at providing greater flexibility in creating and combining visualizations to support software maintenance activities, with less time and effort in its construction. In this sense, an observational study of the infrastructure's usage, developed in this work, is also described in order to characterize the main difficulties, limitations and benefits.

ÍNDICE

CAPÍTULO 1 - Introdução ... 1
1.1 Motivação ... 1
1.2 Problema ... 2
1.3 Objetivo ... 3
1.4 Organização ... 5
CAPÍTULO 2 - Fundamentação Teórica ... 6
2.1 Introdução ... 6
2.2 Visualização de Software ... 6
2.2.1 Visualização na Engenharia de Software ... 8
2.2.2 Exemplos de Visualização de Software ... 9
2.2.3 Estrutura de Mecanismos de Visualização de Software ... 16
2.3 Trabalhos Relacionados ... 17
2.3.1 Luthier ... 17
2.3.2 Mondrian ... 18
2.3.3 CogZ ... 19
2.3.4 Model Driven Visualization ... 20
2.3.5 Many Eyes ... 23
2.3.6 Análise dos Trabalhos ... 24
2.4 Linha de Produtos de Software ... 26
2.4.1 Principais conceitos ... 29
2.4.2 Engenharia de Domínio ... 32
2.4.3 Engenharia de Aplicação ... 34
2.5 Considerações Finais ... 35
CAPÍTULO 3 - Abordagem Proposta ... 36
3.1 Introdução ... 36
3.2 Visão Geral da Abordagem ... 36
3.2.1 Análise de Domínio ... 38
3.2.2 Gerador de Componentes ... 39
3.2.3 *Wizard* de Visualização ... 40
3.3 Desenvolvimento da Infraestrutura da LPVS ... 40
3.3.1 Análise de Domínio para a LPVS ... 41
3.3.2 Gerador de Componentes ... 49
3.3.3 *Wizard* de Visualização ... 57
3.4 Cenário de Utilização ... 63
3.4.1 Análise da necessidade ... 63
3.4.2 Criação do novo componente ... 64
3.4.3 Evolução do modelo de características ... 69
3.4.4 Utilização da nova funcionalidade ... 73
3.5 Considerações Finais ... 77
CAPÍTULO 4 - Estudo de Observação da LPVS ... 79
4.1 Introdução ... 79
4.2 Objetivos ... 80
4.3 Definição do Estudo ... 80
4.4 Procedimento de Execução ... 81

4.5 Resultados e Observações 82
4.5.1 Tarefa 1 84
4.5.2 Tarefa 2 85
4.5.3 Tarefa 3 87
4.5.4 Tarefa 4 88
4.5.5 Observações Gerais 89
4.6 Avaliação realizada pelos participantes 91
4.7 Validade 93
4.8 Considerações Finais 94
CAPÍTULO 5 - Conclusão 96
5.1 Epílogo 96
5.2 Contribuições 97
5.3 Limitações 98
5.4 Trabalhos Futuros 99
Referências 100
APÊNDICE A – FORMULÁRIO DE CONSENTIMENTO 104
APÊNDICE B - QUESTIONÁRIO DE CARACTERIZAÇÃO 105
APÊNDICE C – TAREFAS DO ESTUDO 106
APÊNDICE D – QUESTIONÁRIO DE OBSERVAÇÃO 110
APÊNDICE E – GABARITO DAS TAREFAS 111

ÍNDICE DE FIGURAS

Figura 1.1 Fluxo do processo de visualização (adaptado de Diehl, 2007) 2
Figura 2.1. Visualização em radiografia (D'Ambros et al., 2007) 10
Figura 2.2. Visualização em relógio (D'Ambros et al., 2007) 11
Figura 2.3. Visualização da chamada de métodos em uma thread (Trumper et al., 2010) 11
Figura 2.4. Janela Principal do Vizz3D (Carlsson, 2006) 13
Figura 2.5. Colorações utilizadas: status da linha (a), tipo de construção (b) e autor (c) (Voinea et al., 2005) 14
Figura 2.6. Visão geral da ferramenta CVSscan (Voinea et al., 2005) 15
Figura 2.7. Exemplo de uso do Luthier na análise do comportamento de um software de desenho gráfico (Campo & Price, 1998) 18
Figura 2.8. Uma simples visualização da ideia da abordagem (Meyer, 2006) 19
Figura 2.9. Mapeamentos dos conceitos da ontologia para a representação visual (Falconer et al., 2009) 20
Figura 2.10. Arquitetura de referência para MDV (Bull, 2008) 22
Figura 2.11. Nove visualizações do Many Eyes (Viégas et al., 2007) 23
Figura 2.12. Esquema de Linha de Produtos de Software (adaptado de Clements & Northrop, 2002) 28
Figura 2.13. Visão geral dos processos inerentes a LPS (adaptado de Pohl et al., 2005) 29
Figura 2.14. Quantidade de produtos possíveis (Pereira et al., 2011) 30
Figura 2.15. Desenvolvimento da Base de Ativos (adaptado de Clements & Northrop, 2002) 33
Figura 2.16. Desenvolvimento do Produto (Clements & Northrop, 2002) 34
Figura 3.1. Visão Geral da LPVS 38
Figura 3.2. Visão da área de trabalho de modelo de características no Odyssey 43
Figura 3.3. Modelo de características da LPVS - diagrama sumarizado 45
Figura 3.4. Modelo de características da LPVS - diagrama específico 47
Figura 3.5. Áreas de criação de regras de composição (A) e associação de características com componentes (B) 48
Figura 3.6. Exportação do modelo de características no formato XMI para uso pelos demais módulos da LPVS 49
Figura 3.7. Arquitetura do EvolTrack (Werner *et al.*, 2011) 50
Figura 3.8. Modelo usado no gerador para componente de fonte de dados 52
Figura 3.9. Resultado da geração do componente de fonte de dados 53
Figura 3.10. Código da classe Extractor gerado com ponto de implementação em destaque 54
Figura 3.11. Modelo usado no gerador para componente de representação visual 55
Figura 3.12. Resultado da geração do componente de representação visual 56
Figura 3.13. Interface web da base de componentes da LPVS 57
Figura 3.14. Modos de funcionamento do *wizard* de visualização 58
Figura 3.15. Verificação de consistência da seleção de características 59
Figura 3.16. Parametrização de características de ambiente operacional no modo de configuração detalhado 60
Figura 3.17. Tela de preferências do *wizard* de visualização 61
Figura 3.18. Seleção e configuração do produto de visualização 62

Figura 3.19. Produto gerado pela LPVS (visão dos componentes e documentação) 62
Figura 3.20. Resultado da execução do produto da LPVS (análise estrutural de um sistema) 63
Figura 3.21. Modelo UML utilizado no gerador de componentes da LPVS para a criação do componente EvolTrack-JUnit 64
Figura 3.22. Projeto do componente gerado 65
Figura 3.23. Classe JUNITConstants (constantes adicionadas em destaque) 65
Figura 3.24. Classe PreferenceConstants (constantes adicionadas em destaque) 65
Figura 3.25. Classe JUNITPrefPage e código da tela de preferência adicionados (em destaque) 66
Figura 3.26. Algoritmo de extração de dados 67
Figura 3.27. Método para transformação das informações extraídas num modelo UML 68
Figura 3.28. Carga do novo componente no núcleo de artefatos 68
Figura 3.29. Inclusão de novas características para a representação da nova funcionalidade 69
Figura 3.30. Inclusão de regra de composição entre características 70
Figura 3.31. Inclusão de nova característica no diagrama sumarizado 71
Figura 3.32. Inclusão do novo componente no modelo 72
Figura 3.33. Associação das novas características com o novo componente 72
Figura 3.34. Exportação do modelo atualizado no formato XML 73
Figura 3.35. Iniciação do Wizard em modo detalhado 74
Figura 3.36. Novas características apresentadas no Wizard 74
Figura 3.37. Seleção das características para geração de novo produto 75
Figura 3.38. Instalação do mecanismo de visualização 75
Figura 3.39. Novas opções para análise de resultados de testes unitários 76
Figura 3.40. Resultados dos testes na forma de metáfora 3D 77
Figura 4.1. Experiência dos participantes em desenvolvimento de software (gráfico da esquerda) e em visualização de software (gráfico da direita) 83
Figura 4.2. Ambiente de experiência em desenvolvimento de software dos participantes. 84

ÍNDICE DE TABELAS

Tabela 2.1. Quadro resumo das abordagens analisadas 25
Tabela 4.1. Questionário de Caracterização, por participante. 83
Tabela 4.2. Resultado da execução da tarefa 1. 85
Tabela 4.3. Divergências encontradas na tarefa 1. 85
Tabela 4.4. Resultado da execução da tarefa 2. 86
Tabela 4.5. Divergências encontradas na tarefa 2. 86
Tabela 4.6. Resultado da execução da tarefa 3. 87
Tabela 4.7. Divergências encontradas na tarefa 3. 87
Tabela 4.8. Resultado da execução da tarefa 4. 88
Tabela 4.9. Divergências encontradas na tarefa 4. 89
Tabela 4.10. Resultado geral da execução das tarefas. 89
Tabela 4.11. Percentual de tarefas segundo divergências. 91

CAPÍTULO 1 - INTRODUÇÃO

1.1 Motivação

A era da informação tem mudado fortemente a forma de organização e o cotidiano de pessoas e organizações. Com a grande quantidade de dados gerados e consumidos, a demanda por sistemas de informação que auxiliem no processamento e monitoramento dos mesmos cresce a cada dia. Assim, a atividade de desenvolvimento de software tem envolvido novas estruturas, processos e tecnologias almejando manipular essa massa de informação, entretanto, aumentando também a complexidade desta atividade.

Nesse sentido, empresas e companhias têm absorvido em suas estruturas sistemas complexos suportando as atividades chave de seus negócios, tornando-os críticos para a saúde da organização (Martin et al., 1997). Além disso, o volume crescente de dados produzidos pelas organizações criam a necessidade de formas que facilitem o processamento, interpretação e análise dos mesmos. Dessa forma, técnicas, metodologias e ferramentas para auxiliar no monitoramento e controle destes sistemas e informações têm se tornado alvo de pesquisas e uma necessidade urgente.

Com esta finalidade, surgiu o campo de visualização de software, que utiliza mecanismos computacionais como instrumentos para o desenvolvimento de visualizações, auxiliando os usuários a obterem um melhor entendimento de eventos complexos. Consequentemente, a visualização se tornou uma disciplina da ciência da computação (Diehl, 2007), que pode ser definida não somente como um método computacional, mas também como um processo de transformação da informação numa forma que permita a percepção de características que estão "escondidas" nos dados, além de ampliar a exploração e análise dos mesmos (Gershon, 1994).

A visualização exerce uma função crucial de suporte do raciocínio humano na compreensão da informação por meio da utilização de metodologias e técnicas computacionais. Assim, pode-se observar a existência de campos do desenvolvimento de software que podem ser beneficiados pelas pesquisas e ferramentas da área de visualização, devido ao caráter abstrato de informação que o software representa. O processo de visualização compreende três etapas principais, ilustradas na Figura 1.1 (Diehl, 2007):

- **Aquisição de dados**: Existem várias fontes de informação, logo, existem formas diferentes de extraí-las dependendo do tipo de fonte.
- **Processamento**: Tipicamente, a informação extraída é muito grande e *in natura* para ser imediatamente apresentada a um individuo, assim, necessita-se desta etapa de processamento para reduzir a quantidade de informação com base no foco e contexto do indivíduo.
- **Representação Visual**: Os dados resultantes das etapas anteriores são mapeados em um modelo visual, isto é, transformados em informações gráficas e/ou geométricas para serem exibidos em algum dispositivo de saída.

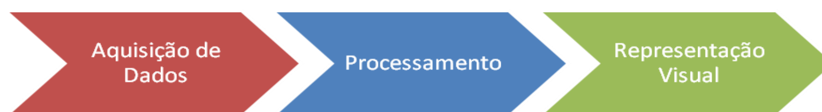


Figura 1.1 Fluxo do processo de visualização (adaptado de Diehl, 2007)

Dessa forma, a visualização de software tem se tornado uma ferramenta para análise e compreensão de sistemas, processos e dados advindos de todas as fases do ciclo de desenvolvimento. Tanto pesquisadores (Koschke, 2002) como membros da indústria (Bassil & Keller, 2001) citam benefícios e vantagens advindos do uso de mecanismos de visualização em atividades de engenharia de software, obtendo economia de dinheiro e tempo, melhor compreensão do software, melhor gerenciamento de complexidade e aumento de produtividade.

1.2 Problema

Atividades de Engenharia de Software são influenciadas fortemente pela compreensão dos diversos artefatos e processos envolvidos, por exemplo, no campo manutenção de software (Mayrhauser & Vans, 1995). Diversas abordagens de visualização de software foram criadas de forma a atenderem diferentes objetivos neste contexto. No entanto, a construção destes mecanismos tipicamente envolve alta complexidade, grande demanda de tempo e altos custos (Anslow et al., 2008), o que dificulta a difusão do uso dos mesmos apesar dos conhecidos benefícios.

Atualmente, as abordagens existentes raramente se preocupam com questões de integração e reúso dos seus mecanismos para serem reaproveitados em diferentes

cenários além dos inicialmente projetados, restringindo o uso do mecanismo de visualização (Schafer & Mezini, 2005). Dessa forma, todo esforço empregado no desenvolvimento da ferramenta de visualização tem o seu retorno e alcance limitados, dado o seu uso isolado, isto é, com pouca ou nenhuma possibilidade de reutilização parcial/integral dos mecanismos em outros cenários.

Além disso, estudos como o de Petrillo et al. (2012) mostram que a implementação destes mecanismos de visualização, frequentemente, deixa a desejar em questões como manutenção e suporte. Em parte, isso é agravado pela complexidade envolvida e também pela falta de preocupação com reutilização citados anteriormente.

Desta forma, é importante observar a necessidade de se incorporar reutilização e flexibilidade nos mecanismos de visualização, de forma que possam ser reaproveitados em cenários variados, diminuindo o tempo e custo de desenvolvimento de novas visualizações. Neste sentido, também vale ressaltar a relevância de se possuir ferramentas que (i) agreguem qualidade e estabeleçam uma arquitetura (padrão de construção) aos mecanismos de visualização, facilitando a manutenção dos mesmos; e que (ii) simplifiquem o processo de desenvolvimento dos mesmos, além de reduzir o tempo total de construção. Estes atributos são considerados chave e essenciais para facilitar, disseminar e ampliar os benefícios do uso de visualização em Engenharia de Software.

1.3 Objetivo

Diante dos problemas expostos anteriormente, esta dissertação tem como objetivo fornecer uma infraestrutura para a geração rápida de mecanismos de visualização, focados predominantemente em reutilização e em padrões que melhorem as condições de manutenção dos mesmos. Assim, almeja-se oferecer para usuários de visualizações de software (testadores, programadores, analistas, etc.) uma forma de construí-las rapidamente, não exigindo profundo conhecimento acerca de como são implementadas, e entregá-las prontas para uso.

Além disto, para buscar melhor qualidade nos mecanismos gerados, a infraestrutura também dever atuar na padronização dos mecanismos de visualização gerados, facilitando as futuras atividades de manutenção nos mesmos. Em se tratando dos altos custos do desenvolvimento de ferramentas de visualização, a infraestrutura deve focar fortemente na reutilização de forma que as soluções de visualização possam ser compostas por partes reutilizáveis, diminuindo custos de desenvolvimento e tempo.

Nesse sentido, o foco em reúso também deve possibilitar a expansão da infraestrutura, dado que novas formas de visualização surgem naturalmente com o avanço tecnológico, a criatividade humana e as necessidades de compreensão do software.

Para atender a estes objetivos, podem se aplicar conceitos e técnicas de linha de produtos de software, abordagem baseada em componentes inspirada na tradicional teoria de linha de montagem, na construção de mecanismos de visualização de software, proporcionando uma forma simples e rápida de geração dos mesmos, além de oferecer a flexibilidade para configuração e extensibilidade para que se amplie a variedade de visualizações a serem geradas. Segundo Clements & Northrop (2002), o uso desta abordagem garante a redução de custos no desenvolvimento de software ao longo do tempo, ao passo que se baseia na utilização de um núcleo (expansível) de componentes, cuja composição resulta num software bem definido.

Com base em trabalhos da literatura sobre geração de visualização de software, foram identificadas algumas técnicas importantes aplicadas neste domínio para auxiliar no desenvolvimento das mesmas. Normalmente, estas técnicas ou privilegiavam a capacidade de customização em detrimento da reutilização e facilidade de construção, ou atentavam mais para a simplificação do processo de geração em detrimento da reutilização e poder de configuração (Petrillo et al., 2012). Assim, o uso de linhas de produtos de software nesse domínio parece promissor, pois busca agregar os fatores citados acima.

Dessa forma, esta dissertação busca o desenvolvimento desta infraestrutura para geração de mecanismos de visualização de software utilizando a técnica de linha de produtos de software, denominando-a **L**inha de **P**rodutos para **V**isualização de **S**oftware (LPVS). Este trabalho inclui a realização de uma prova de conceito da aplicação da infraestrutura no domínio de visualizações que apóiem atividades de manutenção de software, dado que a amplitude de visualizações possíveis dentro de Engenharia de Software é muito grande, e não é objetivo desta abordagem esgotar todas as possibilidades de geração de visualizações nesta área.

Nesse âmbito, estabelece-se como meta desta dissertação o provimento de uma infraestrutura que (i) contribua para a geração rápida e simples de mecanismos de visualização de software voltados para manutenção; (ii) seja pautada em reutilização para redução de tempo e custo no desenvolvimento de visualizações; (iii) tenha capacidade de expandir os mecanismos de visualização a serem gerados e (iv) possua ferramentas para padronizar os mecanismos gerados. Em conformidade com estas

metas, também precisa-se de (v) um conjunto inicial de visualizações em manutenção de software suportadas pela infraestrutura e (vi) um exemplo de uso da mesma no domínio de manutenção de software para avaliação dos objetivos alcançados.

1.4 Organização

Esta dissertação está organizada em cinco capítulos. O presente capítulo apresentou a motivação para o desenvolvimento deste trabalho, bem como os objetivos da pesquisa.

O Capítulo 2 apresenta uma visão geral da área de visualização de software, sua história e seus benefícios, discutindo sua aplicação no contexto da engenharia de software, em especial, na utilização dentro do processo de desenvolvimento de software. Além disto, analisa alguns trabalhos relacionados à construção de mecanismos de visualização, elicitando atributos e pontos a serem tratados por este trabalho de pesquisa. Por fim, trata também da conceituação da técnica de linha de produtos de software, discutindo sua utilização no contexto da geração de visualizações de software.

No Capítulo 3, é proposta uma abordagem que visa atender aos objetivos citados anteriormente e aos requisitos identificados no Capítulo 2, voltado para a geração de visualização de software. Assim, é apresentada a solução proposta, denominada LPVS (Linha de Produtos para Visualização de Software), junto com a infraestrutura (e detalhes de implementação) desenvolvida para suportá-la.

O Capítulo 4 descreve as etapas de definição, planejamento, execução e análise de um estudo de observação realizado para avaliar a abordagem. Este estudo buscou analisar atributos qualitativos do uso da LPVS, como simplicidade de uso e o apoio à atividade de geração, e atributos quantitativos, como o tempo total para a geração.

Por fim, o Capítulo 5 contém as considerações finais deste trabalho, bem como as contribuições da dissertação, algumas limitações identificadas e as perspectivas de trabalhos futuros.

CAPÍTULO 2 - FUNDAMENTAÇÃO TEÓRICA

2.1 Introdução

Este capítulo apresenta os principais conceitos e uma visão geral do campo de visualização de software e sua aplicação na área de Engenharia de Software. Além disso, a complexidade do desenvolvimento destes mecanismos é explicitada com a apresentação e análise de outros trabalhos neste cenário. Ao final deste capítulo, formas de se melhorar este quadro são discutidas com o estudo de uma técnica de reutilização denominada Linha de Produtos de Software (LPS).

A Seção 2.2 discute brevemente sobre os principais conceitos e desafios de visualização de software, apresentando algumas abordagens e aplicações da área. A Seção 2.3 discute sobre trabalhos relacionados, sob a ótica da geração de mecanismos de visualização, ressaltando pontos positivos e negativos dos mesmos. A Seção 2.4 introduz a técnica de Linha de Produtos de Software, descrevendo suas características, processos e benefícios. A Seção 2.5 encerra este capítulo, sumarizando o estudo da literatura realizado e apontando as principais metas a serem alcançadas por este trabalho de pesquisa.

2.2 Visualização de Software

A área de visualização busca representar dados e informações graficamente, por meio de técnicas e abstrações, de forma que a capacidade cognitiva do ser humano (derivada da sua memória, percepção e raciocínio) seja estimulada para facilitar a compreensão de um determinado assunto (Diehl, 2007).

Segundo Diehl (2007), existem duas grandes disciplinas dentro da visualização: a visualização científica, que processa dados físicos, e a visualização da informação, que processa dados abstratos. Como o software está ligado a informação, considera-se que ele faz parte da visualização da informação, onde a maior parte das técnicas se baseia em um dos dois seguintes princípios, ou ambos:

- **Exploração Interativa**: Para explorar dados, visualizações interativas devem permitir que o usuário, primeiramente, tenha uma visão geral e, depois, aplique filtros e zoom para obter detalhes sobre demanda. Este princípio foi chamado por Shneiderman (1996) de “mantra de busca da informação” (*Information*

Seeking Mantra) e envolve técnicas como *zoom*, filtragem, *minimap*, entre outras.

- **Foco + Contexto**: Uma visualização detalhada de uma parte da informação – o foco – é incorporada dentro de uma visualização do contexto, isto é, uma informação mais refinada sobre as partes relacionadas ao foco. Assim, técnicas de foco + contexto proveem detalhamento e uma visão geral ao mesmo tempo. Exemplos deste princípio aparecem nas técnicas de agrupamento e detalhamento.

O objetivo da visualização é otimizar o uso da percepção e raciocínio visual do ser humano, ao tratar de fenômenos que não são prontamente compreendidos, utilizando técnicas e ferramentas computacionais (Chen, 2006). Pesquisadores em visualização de software desenvolvem e investigam métodos e usos de representações gráficas computacionais de vários aspectos do software.

Neste trabalho, é utilizada a definição de visualização de software como a visualização de artefatos relacionados ao software e ao seu processo de desenvolvimento, o que envolve, por exemplo, documentação do projeto e mudanças no código-fonte. Assim, a visualização de software está compreendida em três nichos principais (Diehl, 2007):

- **Estrutura**: Refere-se à parte estática e aos relacionamentos do sistema, isto é, aqueles que são computados ou inferidos sem a execução do sistema. Isto inclui o código-fonte, a organização do projeto de software e as estruturas de dados.
- **Comportamento**: Refere-se à execução do sistema com dados reais ou simulados. Esta pode ser vista como uma sequência de estados, onde em cada um existe o código em execução e os dados utilizados.
- **Evolução**: Refere-se ao processo de desenvolvimento de software como um todo e, em particular, enfatiza o fato de que o código-fonte (que compõe a estrutura do software) sofre mudanças ao longo do tempo, seja para acrescentar funcionalidades, seja para efetuar correções.

O campo de visualização de software tem muita influência em diversas atividades que são realizadas em áreas relacionadas ao ciclo de vida do software. A

próxima seção mostra como essa técnica pode influenciar na área de Engenharia de Software.

2.2.1 Visualização na Engenharia de Software

A visualização de software permite que um *stakeholder* do desenvolvimento (desenvolvedor, analista, testador, etc.) crie um modelo mental de um sistema, auxiliando-o a entender melhor o seu projeto, suas funcionalidades e outras características relacionadas. Uma das áreas mais apoiadas por visualização de software é o campo de manutenção de software, que representa mais da metade do custo de um software e, dentro do seu conjunto de atividades, cerca de metade do tempo é gasto buscando-se compreender artefatos do software (Sharafi, 2011). Dessa forma, mecanismos de visualização são tidos como importantes meios para aumentar a compreensão durante as atividades executadas em manutenção.

A análise da arquitetura de software, estrutura de um sistema englobando seus elementos, propriedades e relacionamentos (IEEE, 2000), pode ser apoiada por essas técnicas. A arquitetura de um sistema é base de um desenvolvimento; compreender as mudanças ocorridas ao longo do tempo, descobrindo desvios do planejado inicialmente, são benefícios alcançados por meio da visualização de software (Schots *et al.*, 2010).

Além disso, o código-fonte também é um artefato essencial do desenvolvimento e manutenção de software, guardando a maior parte do valor intelectual produzido. Em repositórios de código, existe todo um histórico de modificações, interações entre pessoas e a estrutura do próprio código em si, que guardam propriedades intrínsecas do desenvolvimento e seu processo.

Sistemas de rastreamento de *bug*s e de gerenciamento de projetos são outras fontes que registram dados do ciclo de vida de um software, como as atividades realizadas, o tempo decorrido, sua situação atual, falhas e erros ocorridos, orçamento, documentação, entre outros. Nestes dados, informações sobre módulos mais instáveis de um software, desvios de processos, qualidade dos artefatos produzidos e medição de andamento de atividades são exemplos de benefícios que podem ser obtidos por meio de visualização, reiterando todo o suporte desta disciplina nas atividades de Engenharia de Software (Mayrhauser & Vans, 1995).

Neste sentido, a seção subsequente apresenta exemplos de abordagens e mecanismos de visualização de software, ilustrando os benefícios alcançados e os problemas de compreensão atendidos.

2.2.2 Exemplos de Visualização de Software

Nesta subseção, são apresentados alguns trabalhos de visualização de software aplicados a tarefas de engenharia de software. Isto remete às atividades de projeto, arquitetura, manutenção e depuração de software, onde a compreensão dos artefatos auxilia no melhor desenvolvimento das mesmas.

2.2.2.1 *Bug's Life* (D'Ambros et al., 2007)

Neste trabalho, os autores enaltecem o fato de que a visualização pode ser crucial para lidar com dados brutos que precisam ser analisados. Em face disto, são consideradas informações providas por sistemas de *bug tracking* (rastreamento de *bugs*) que armazenam dados sobre problemas enfrentados por diferentes *stakeholders* envolvidos em um projeto.

A escolha pelas informações capturadas por estes sistemas se deve à notável importância desempenhada no desenvolvimento de software. O seu uso é feito por diferentes tipos de *stakeholders*, como desenvolvedores, testadores, usuários finais, avaliadores de qualidade, entre outros, tornando muito diversificados os dados obtidos.

Além disso, também podem ser utilizados para análise retrospectiva de sistemas (no contexto de evolução de software), cujo objetivo é compreender os módulos mais problemáticos de um sistema ao longo de sua vida. Nesse contexto, *bugs* são considerados a entidade primária desta abordagem, podendo mudar e evoluir com o passar do tempo. Em especial, busca-se a análise do ciclo de vida dos *bugs*, isto é, de sua história e diversos estados. Para tal, foram apresentadas duas técnicas de visualização: Radiografia e Relógio.

A primeira trata a informação acerca de *bugs* no nível do sistema, isto é, num nível mais abstrato. Ela provê indicações de quais partes do software estão sendo afetadas por diferentes tipos de *bugs* em determinado ponto do tempo, oferecendo suporte para a análise da "saúde" do sistema.

Os princípios desta visualização se baseiam numa representação matricial, onde cada linha ilustra um componente do sistema e, cada grupo de linhas, um produto do sistema. As colunas representam o tempo (da esquerda para a direita), e cada uma corresponde a um intervalo parametrizável. A cor das células exibe o número de *bugs* afetando um componente num determinado instante onde, quanto mais escuro, maior a

intensidade. Assim, o objetivo desta visualização (apresentada na Figura 2.1) é descobrir onde e quando os *bugs* estão concentrados.

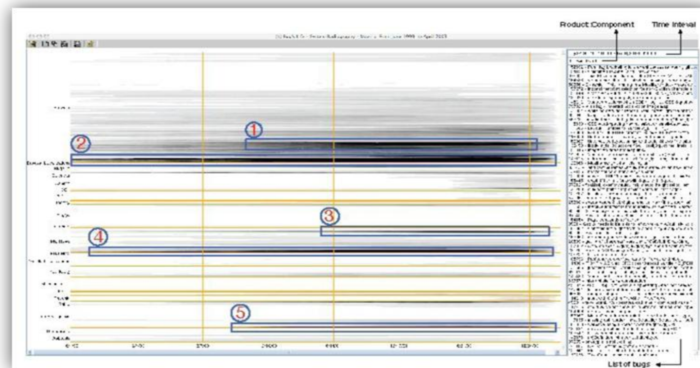


Figura 2.1. Visualização em radiografia (D'Ambros et al., 2007)

A segunda, em concordância com a visualização em radiografia que fornecia uma visão geral do sistema, propõe facilitar a análise de determinados *bugs*. Os objetivos desta visualização são a caracterização dos *bugs,* afetando componentes e a detecção dos mais críticos. Para tal tarefa, é assumido que a criticidade de um *bug* não depende somente da sua severidade e prioridade, apesar de sua importância no processo. Como exemplo, os autores citam a recorrência de determinado *bug* como um fator que indica um problema mais profundo que o esperado.

Dessa forma, esta visualização se baseia na metáfora de um relógio para representar a temporalidade de um *bug*. Ela é formada por três camadas: a camada de **status**, onde todos os status que o *bug* passou ao longo de sua vida é visualizado como setores. O tamanho e a posição do setor indicam duração e posição cronológica, respectivamente; a camada de **atividade**, que mostra as modificações nas propriedades do *bug* por meio de uma barra preta posicionada de acordo com a data de ocorrência; a camada de **severidade**, que indica quão severo ou prioritário é um determinado *bug*, onde cores mais escuras significam uma maior criticidade. A Figura 2.2 ilustra esta visualização.

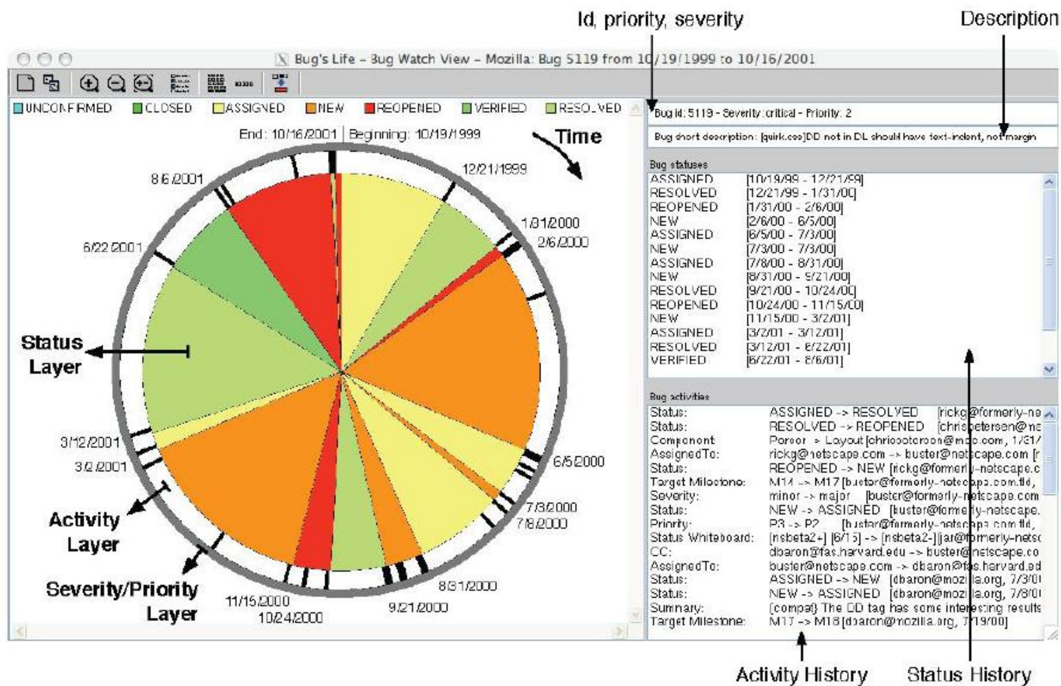


Figura 2.2. Visualização em relógio (D'Ambros et al., 2007)

2.2.2.2 *Software Diagnostics* (Trumper et al., 2010)

Este trabalho apresenta uma técnica de compreensão de software para apoiar a análise e entendimento do comportamento de execução de sistemas *multithreads*. O fluxo apresentado na abordagem deste artigo remete a duas etapas: a primeira consiste no registro da chamada de métodos de um software em execução e, a segunda, a transformação destes dados em formas visuais de forma a permitir que os desenvolvedores explorem o comportamento de um sistema após sua execução.

Como sistemas *multithreads* podem gerar diversas *threads* em tempo de execução, a análise comportamental se torna bastante complexa; logo, esta ferramenta permite a seleção de um subconjunto representativo de *threads* para ser analisado. A Figura 2.3 mostra a técnica de visualização aplicada a uma única *thread*.

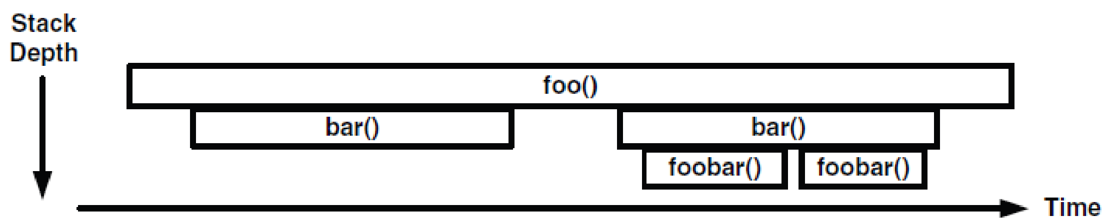


Figura 2.3. Visualização da chamada de métodos em uma thread (Trumper et al., 2010)

Sendo assim, este tipo de abordagem auxilia desenvolvedores a acompanhar, monitorar e até mesmo testar se o comportamento de um sistema é o esperado segundo as circunstâncias a que o mesmo é colocado.

2.2.2.3 Vizz3D (Carlsson, 2006)

Vizz3D é uma ferramenta de visualização de grafos desenvolvida na Universidade de Växjö. Ela é utilizada para visualizar diferentes aspectos de sistemas de software em 3D, baseado na análise estática de código-fonte. Os diferentes elementos de software (por exemplo, classes, interfaces, pacotes, métodos, atributos) e seus relacionamentos (como chamadas, herança, composição e agregação) são visualizados pelo Vizz3D como grafos interativos, consistindo de nós e arestas. Um nó pode ser simplesmente uma esfera ou um cubo, ou uma forma mais complexa (como, por exemplo, uma casa).

Como projetos complexos de software costumam ser grandes, os grafos criados podem conter milhares de nós e arestas, resultando num esforço grande para a construção da visualização. Isso acaba limitando algumas ferramentas, pois o desempenho no processamento é um fator importante para a visualização, já que, ao prejudicar a interação do usuário com os mecanismos visuais, perde-se a conexão cognitiva e a orientação. Esta é uma limitação do Vizz3D.

Vizz3D permite a utilização de diferentes metáforas e leiautes, de forma que o usuário configure a visualização do grafo. Este pode ser rotacionado, movido e até sofrer *zoom* para aproximação e distanciamento do foco. A visualização é exibida em uma GUI (*Graphical User Interface*), construída na janela principal que contém menus, ícones, painéis e o *canvas*, conforme ilustra a Figura 2.4.

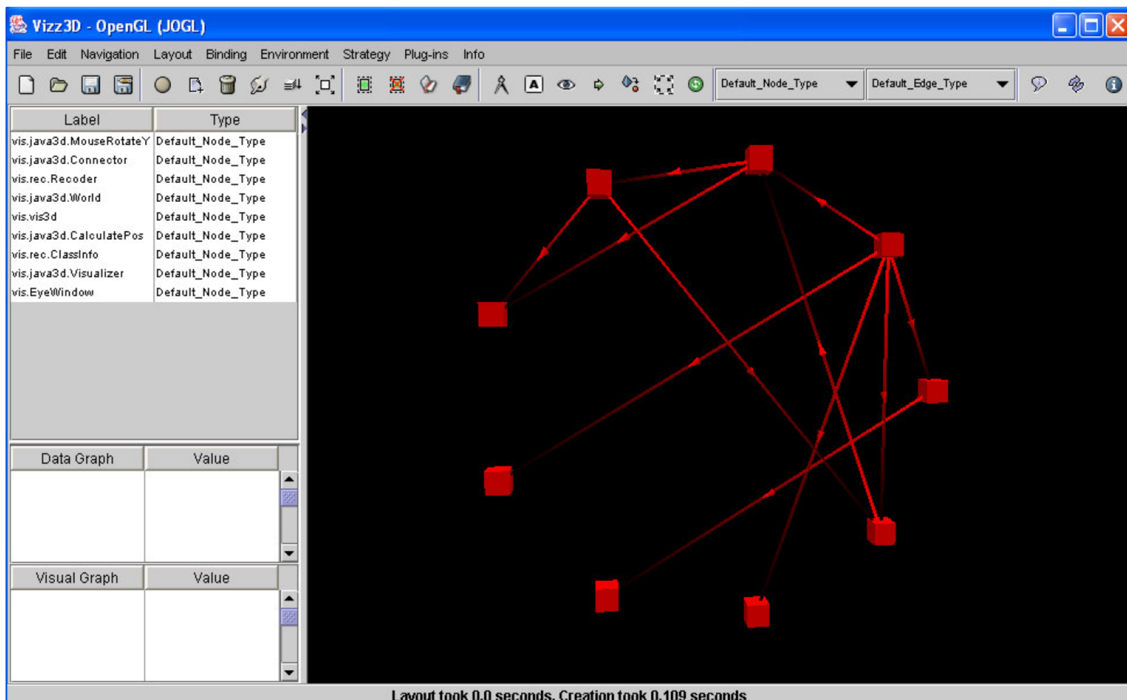


Figura 2.4. Janela Principal do Vizz3D (Carlsson, 2006)

2.2.2.4 CVSscan (Voinea et al., 2005)

O CVSscan é uma ferramenta de apoio ao processo de manutenção e entendimento de software que, através da técnica de visualização, expõe ao engenheiro diversos aspectos de uma determinada aplicação ao longo do tempo. Entretanto, ao contrário das demais ferramentas citadas anteriormente, o CVSscan utiliza uma abordagem baseada em métricas de linhas de código para a visualização do software.

Nesta abordagem, pontos na tela são utilizados para representar, sob algum tipo de perspectiva, linhas de código fonte (tais pontos são denominados *pixel lines*). Cores distinguem as possíveis variações de uma perspectiva. Por exemplo, na ferramenta CVSscan existem basicamente três perspectivas, ou dimensões, sob as quais as linhas de código são classificadas: *status* da linha, tipo de construção e autor.

A perspectiva de *status* da linha classifica-a em uma das seguintes categorias: constante (i.e. linha inalterada em relação à versão anterior), a ser inserida, modificada e removida. A perspectiva de tipo de construção classifica funcionalmente a linha de acordo com a linguagem de programação utilizada. Por exemplo, se a linha for um comentário no programa, será classificada com uma categoria de mesmo nome. Já a perspectiva de autor classifica a linha de acordo com o autor da linha. Cada autor que realiza alterações no software terá uma cor diferente. A Figura 2.5 ilustra algumas colorações utilizadas para cada perspectiva.

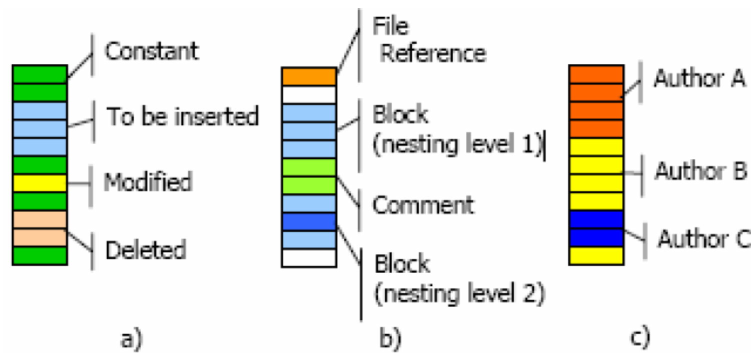


Figura 2.5. Colorações utilizadas: status da linha (a), tipo de construção (b) e autor (c) (Voinea et al., 2005)

Todas estas linhas de código utilizadas são originadas a partir de sistemas de controle de versão. Nesta implementação, apenas o sistema CVS é suportado pela ferramenta. Outra ferramenta, chamada CVSgrab, é responsável por extrair as informações do CVS e repassar para o CVSscan para o devido processamento. Desta forma, pode-se reparar que houve uma tentativa de se criar uma arquitetura modular para que, no futuro, outros sistemas de controle de versão pudessem ser utilizados com o CVSscan. A ferramenta suporta a análise de arquivos fonte escritos nas linguagens C e Perl (Wall et al., 2000).

Neste contexto, cada ponto discreto (i.e. versão) no ciclo de vida de um arquivo é representado pela ferramenta a partir de uma tupla (identificação da versão, autor, data, código). Então, para comparar versões consecutivas de um determinado arquivo, a ferramenta utiliza uma aplicação externa. Assim, a partir da saída desta aplicação, o CVSscan rotula cada linha de acordo com as categorias de *status* de linha citadas anteriormente.

A Figura 2.6 ilustra como ocorre o processo de visualização na ferramenta. É interessante notar que a ferramenta não utiliza indentação e tamanho da linha para representar uma possível estrutura para o arquivo. Ao contrário, utiliza-se de um tamanho fixo de linhas, maior ou igual ao maior número de linhas atingido pelo arquivo ao longo do tempo, e cores para codificar a estrutura.

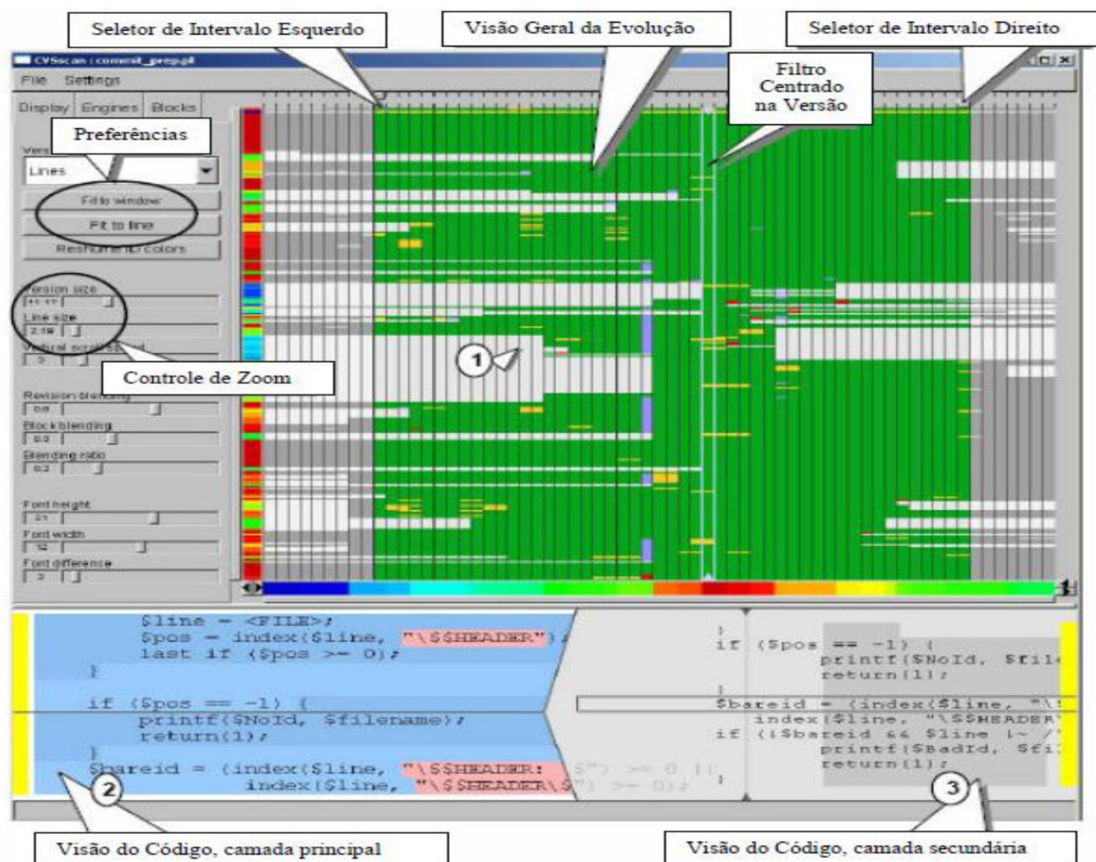


Figura 2.6. Visão geral da ferramenta CVSscan (Voinea et al., 2005)

Cada linha vertical representa uma versão do arquivo, e cada linha horizontal representa uma linha do arquivo, conforme pode ser observado na parte central da ferramenta (apontada como "*Evolution Overview*"). Adicionalmente, podem ser observadas métricas que complementam a visualização. Na borda esquerda da parte central, uma barra vertical representa o tamanho da versão em linhas, informação codificada por meio de cores. Na borda inferior da parte central, uma barra horizontal é utilizada para representar o autor responsável por cada versão.

Outra funcionalidade interessante é exemplificada na Figura 2.6. Ao passar o *mouse* sobre uma parte da visualização (marcado como (1) na figura), ocorre uma apresentação do código referente a esta passagem (marcado como (2) e (3) na figura). Além disto, a ferramenta oferece alguns recursos adicionais, como *zoom*, alteração do tamanho das fontes exibidas, alteração do tamanho das linhas exibidas, seleção dos intervalos de tempo para exibição (através dos seletores *de Intervalo*), dentre outros.

É importante ressaltar que toda análise realizada pela ferramenta (e, consequentemente, todo resultado gerado por esta) tem como único foco arquivos e suas

respectivas linhas. Isto é, a ferramenta é capaz apenas de representar, ao longo do tempo, a evolução de um único arquivo por vez, ou seja, representar linhas que foram acrescidas, removidas ou alteradas ao longo da execução do projeto, o que limita o potencial de visualização de um conjunto de informações.

Neste sentido, a seção subsequente apresenta uma visão geral da estrutura destes mecanismos de visualização de software, salientando os processos envolvidos e as dificuldades existentes.

2.2.3 Estrutura de Mecanismos de Visualização de Software

Conforme foi apresentado no Capítulo 1, o fluxo normal para mecanismos de visualização se baseia em três processos principais (Diehl, 2007): aquisição de dados, processamento e representação visual. Dessa forma, os mecanismos de visualização se estruturam de diferentes formas para executar estes três processos. Anslow et al. (2008) citam que essas estruturas envolvem alta complexidade**,** dado que os processos envolvidos necessitam de conhecimentos específicos de:

- **Especialistas Visuais**, que se ocupam de como construir representações gráficas com recursos tecnológicos;
- **Especialistas de Fontes de Dados**, que se preocupam em como extrair dados das diversas fontes utilizadas no ciclo de vida do software, tais como repositório de código, bases de erros, sistemas de gerenciamento de projeto, entre outros;
- **Especialistas em Informações,** que conhecem a fundo os dados e o seu contexto sabendo como agrupar, classificar, limpar e medir os mesmos, de forma que sabem como agregar valor aos dados extraídos enriquecendo a informação que poderá ser interpretada após a representação visual.

Assim, a restrição no uso de mecanismos de visualização, de forma massiva, nas fases de vida do software é fortemente influenciada pela especificidade de conhecimentos necessários, o custo e o tempo demandados para a criação destes mecanismos. Neste sentido, Schafer & Mezini (2005) relatam fatores complicadores nestas ferramentas de visualização como a falta de flexibilidade e integração ao se tratar de possibilidade de reaproveitamento em diferentes cenários, mostrando a baixa preocupação com questões de reúso nestes mecanismos, além de corroborar com os problemas na geração (custo e tempo) citados anteriormente.

Outro estudo (Petrillo et al., 2012) atentou para os atributos de manutenção e suporte destes mecanismos. Analisando um conjunto de 52 ferramentas de visualização, ficou constatada a alta complexidade de se executar manutenção e utilizar estes mecanismos devido à falta de suporte, documentação e arquiteturas mal planejadas e/ou de baixa qualidade.

Dessa forma, observa-se a necessidade da incorporação de reúso na construção de mecanismos de visualização favorecendo, assim, a redução de custos e desenvolvimento mais rápido das mesmas. Além disso, a estruturação dos mesmos por meio de uma arquitetura que favoreça tanto a reutilização como a integração é considerada chave para uma melhor manutenção e suporte destes mecanismos. Nesse contexto, a próxima seção apresenta um conjunto de trabalhos relacionados à construção de mecanismos de visualização, mostrando aspectos ligados à flexibilidade, reutilização e forma de interação com o interessado no mecanismo.

2.3 Trabalhos Relacionados

Esta seção apresenta algumas ferramentas e abordagens para apoiar a geração de visualizações. Entretanto, estes trabalhos não atendem a todas as necessidades mencionadas no Capítulo 1, o que será abordado no próximo capítulo com a apresentação de uma abordagem proposta nesta dissertação. Desta forma, foram analisadas as seguintes ferramentas/abordagens dado o foco na geração de mecanismos de visualização: *Luthier* (Campo & Price, 1998), *Mondrian* (Meyer, 2006) e *CogZ* (Falconer *et al.*, 2009), *Model Driven Visualization* (Bull, 2008) e *Many Eyes* (Viégas *et al.*, 2007).

2.3.1 Luthier

Luthier é um framework destinado a apoiar a construção de ferramentas visuais para a análise dinâmica de programas. Como um framework, este trabalho permite a construção de mecanismos especializados baseados em extensões e implementações de interfaces de sua estrutura. Através desta organização, é possível construir visualizações altamente complexas, entretanto, ainda necessita de pessoas especializadas para operar no nível de código fonte, envolvendo maior quantidade de tempo e alto custo.

O funcionamento básico está em torno de uma infraestrutura ou esqueleto de uma família de aplicações, projetado para ser reutilizado. Basicamente, aplicações específicas são construídas especializando as classes do framework para fornecer a

implementação de alguns métodos, enquanto a maior parte da funcionalidade da aplicação é herdada.

Este tipo de técnica possibilita alta flexibilidade em customização para as visualizações geradas, ao passo que cria a restrição de usuários que compreendam a estrutura organizacional do framework e sejam capazes de codificar os pontos necessários, baseado em heranças e interfaces, para a correta customização das funcionalidades. Além disso, este framework captura a informação baseado em meta-objetos, monitorando objetos reais em tempo de execução de uma aplicação. Isto demonstra o foco na análise comportamental de um software, abdicando de outros aspectos ligados ao processo de desenvolvimento.

A Figura 2.7 apresenta uma visualização gerada a partir da ferramenta onde é analisado um software para desenho gráfico, mostrando na área mais a esquerda o fluxo de informação enquanto, na mais a direita, o resultado da execução.

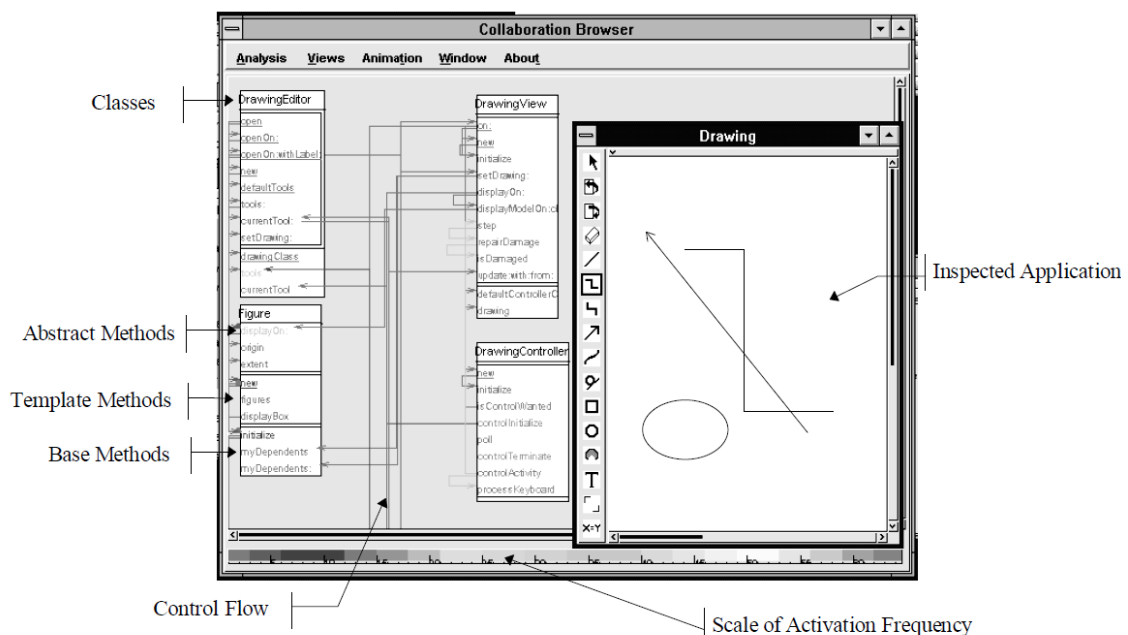


Figura 2.7. Exemplo de uso do Luthier na análise do comportamento de um software de desenho gráfico (Campo & Price, 1998)

2.3.2 Mondrian

Frequentemente, não é muito simples saber como um conjunto de dados deve ser visualizado. Por isso, esta abordagem argumenta que, para os usuários que não sabem como aparecerá a solução final, um meio que os permita tentar diversas ideias com o mínimo de esforço é necessário. Nesta tese, é apresentado um novo modelo de visualização que foca na rápida prototipação de visualizações.

A Figura 2.8 mostra a essência desta abordagem. Neste modelo, uma visualização é definida de forma declarativa utilizando *scripts*. No *script*, é feita uma referência ao modelo, que provê dois métodos: o método #allClasses, que retorna uma coleção de definições de classes, e o método #allInheritances, que retorna uma coleção de definições de heranças.

```
view := ViewRenderer new.
view nodes: model allClasses using: RectangleShape withBorder.
view edges: model allInheritances
     from: #superclass
     to: #subclass
     using: LineShape new.
view layout: TreeLayout new.
view open.
```

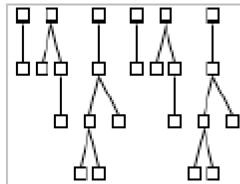


Figura 2.8. Uma simples visualização da ideia da abordagem (Meyer, 2006)

Este tipo de abordagem gera uma grande flexibilidade na criação de visualizações, porém ela não se vale de uma linguagem padrão (como o QVT, da OMG), mas utiliza uma descrição própria. Com isso, existe a dificuldade de se aprender a linguagem e dominá-la a ponto de tornar a criação de visualizações eficiente. Isso não é algo trivial, em especial, quando se assume que os utilizadores da visualização serão pessoas com os diversos tipos de interesse e conhecimento. Nesse sentido, consegue-se alto nível de customização da visão, utilizando-se de *scripts* para tal atividade, além de reutilização de técnicas de visualização para diferentes dados. Porém, este trabalho também não possibilita a customização do formato do dado coletado para a visualização, assim como necessita de certa especialização do usuário final para a criação dos *scripts*, aumentando a complexidade da interação com a ferramenta.

2.3.3 CogZ

Ontologias fornecem um entendimento comum e partilhado sobre um domínio específico. Os membros deste domínio representam instâncias dos conceitos dentro da ontologia. Por exemplo, se "país" é definido como um conceito numa ontologia OWL (*Web Ontology Language*), então "Brasil" e "Chile" são potenciais instâncias desta classe. A visualização da informação auxilia no entendimento e compreensão de espaços de informação complexa como ontologias.

Apesar de existirem trabalhos que tratem da visualização de ontologias, este foca na investigação de técnicas para a visualização das instâncias associadas a elas. Este trabalho estende um conjunto de ferramentas (denominado CogZ) para o rápido desenvolvimento de visualizações específicas para ontologias.

A Figura 2.9 apresenta a ferramenta, mostrando linhas que fazem o mapeamento dos conceitos (elementos à esquerda) para as representações visuais (elementos à direita). Assim, a interação com o usuário é facilitada pelo mecanismo *drag-and-drop* (arrastar e soltar), todavia, o conhecimento para realizar o mapeamento pode não ser trivial dependendo do tipo de visão. Dessa forma, consegue-se a reutilização da visualização e customização no nível da visão, ao passo que, no nível da fonte de dados, o formato permanece fixo em modelos OWL.

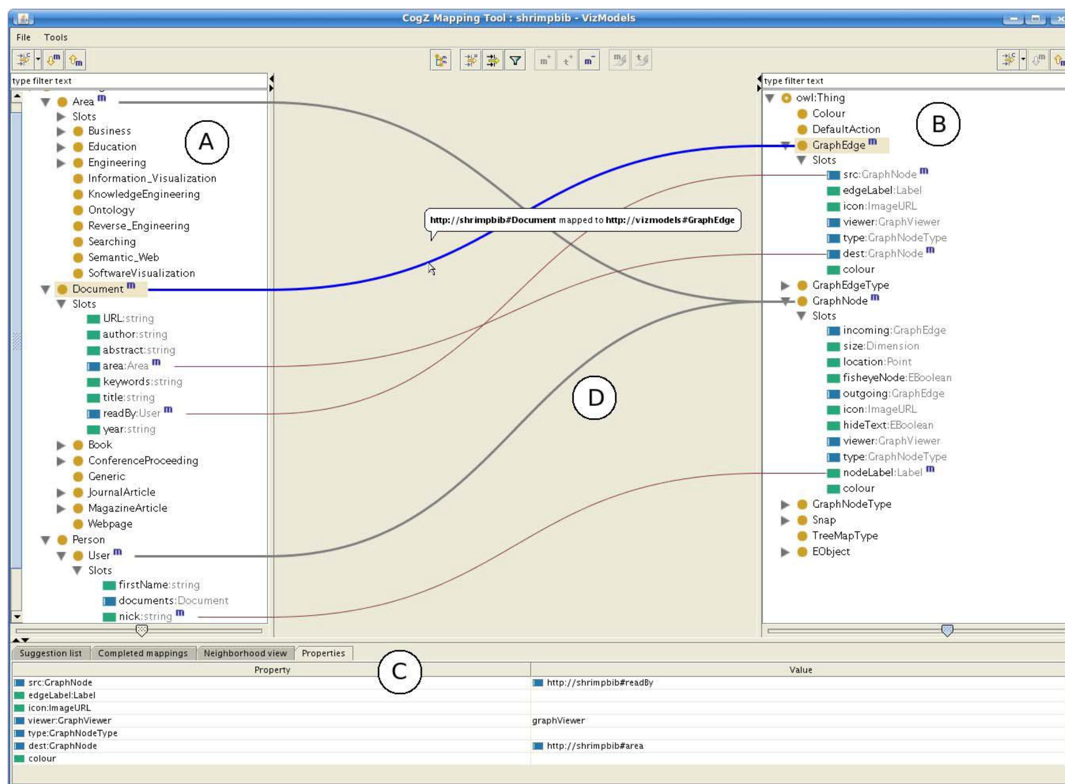


Figura 2.9. Mapeamentos dos conceitos da ontologia para a representação visual (Falconer et al., 2009)

2.3.4 Model Driven Visualization

Ferramentas para a compreensão de programas são recursos valiosos para a navegação e o entendimento de grandes sistemas de software. *Package explorers*, visões de *fan-in / fan-out*, grafos de dependência e análise de cobertura são exemplos de

contribuições da comunidade de compreensão de software (Bull, 2008). Enquanto alguns destes projetos trouxeram grandes avanços, outros deixaram de ser adotados porque não possuíam integração com outras ferramentas existentes ou possuíam um *design* de interface pouco atraente.

Assim, a criação de interfaces altamente customizáveis para a visualização de software pode ser considerada uma importante forma de aumentar a usabilidade destes mecanismos visuais. Com isso, tomando emprestadas as ideias de MDE (*Model Driven Engineering*), movendo o nível de abstração da implementação para o projeto, almeja-se aumentar a eficiência da construção de visualizações de software, provendo recursos para que pesquisadores customizem suas próprias visualizações sem atentarem para detalhes de implementação.

As iniciativas de MDA e MDE da OMG (*Object Modeling Group*) descrevem como modelos podem ser usados tanto para o projeto como para a entrega de um software. Um modelo é considerado uma descrição sistemática de um sistema e, nas abordagens de MDE, eles são construídos para atender a um problema sem a necessidade da preocupação com a implementação.

Assim, desenvolvendo uma analogia com o tema de visualização abordado neste trabalho, a junção destas ideias pode proporcionar a criação de modelos customizáveis de visualizações, deixando a implementação dos componentes visuais que os implementam e algoritmos complexos de geração gráfica para os especialistas em programação gráfica.

Para isso, Bull (2008) desenvolveu uma arquitetura para este tipo de abordagem, denominada MDV (*Model Driven Visualization*), derivada do MDE, onde foram apresentados os principais componentes para a execução. A Figura 2.10 apresenta esta arquitetura.

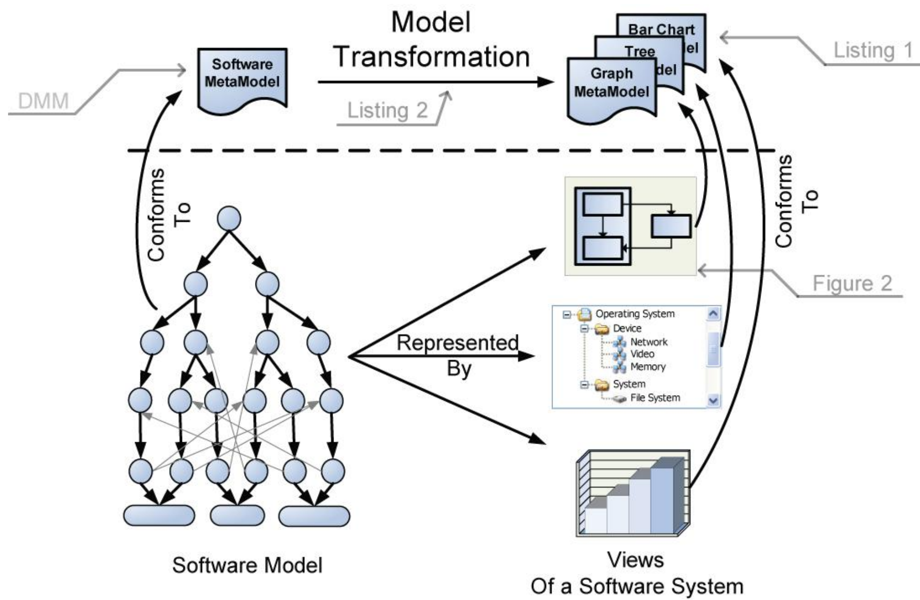


Figura 2.10. Arquitetura de referência para MDV (Bull, 2008)

A arquitetura retratada é baseada na existência de meta-modelos de domínio (*Software MetaModel*, na Figura 2.10) que representam o tipo de dado a ser visualizado (neste caso, o software) e meta-modelos de visualização (*Graph/Tree/Bar MetaModel*, na Figura 2.10), que representam como as visualizações são formadas (na Figura 2.10, existem os meta-modelos para a construção de grafos, árvores e gráficos de barra).

Entretanto, o elemento principal dessa arquitetura são as transformações (*Model Transformation*, na Figura 2.10), afinal, elas farão a passagem dos elementos de domínio para os elementos visuais, envolvendo assim a maior complexidade do processo. Assim, a flexibilidade é garantida por este componente que, normalmente, será descrito por uma DSL (*Domain Specific Language*) e executado por um motor de transformação. Dentro dessas linguagens, algumas se destacam por sua utilização para transformação de modelos como a ATL (*Atlas Transformation Language*) e a QVT (*Query-View-Transformation*).

Dessa forma, a abordagem MDV auxilia a geração de visualizações de software, focando principalmente na customização por meio de transformações. Apesar de existir certo grau de reusabilidade neste tipo de abordagem devido ao reaproveitamento de modelos, o aprofundamento nesta propriedade no nível de implementação não é especificado, dependendo de outras técnicas mais ligadas ao desenvolvimento para reúso e ao desenvolvimento com reúso.

2.3.5 Many Eyes

Many Eyes é um website que permite que qualquer pessoa submeta dados, crie visualizações interativas e compartilhe seus resultados. Os arquitetos do *Many Eyes* o descrevem da seguinte forma: "o objetivo do *Many Eyes* é dar suporte a colaboração ao redor de visualizações em larga escala através da promoção de um estilo social de análise de dados no qual a visualização não serve somente como uma ferramenta de descoberta para indivíduos, mas como um meio para estimular discussões entre usuários" (Viégas et al., 2007). O website oferece suporte a um número de técnicas de visualização incluindo: gráficos (barra, linha, pilha e bolha), mapas, *tag clouds*, *treemaps* e grafos (vistos na Figura 2.11).

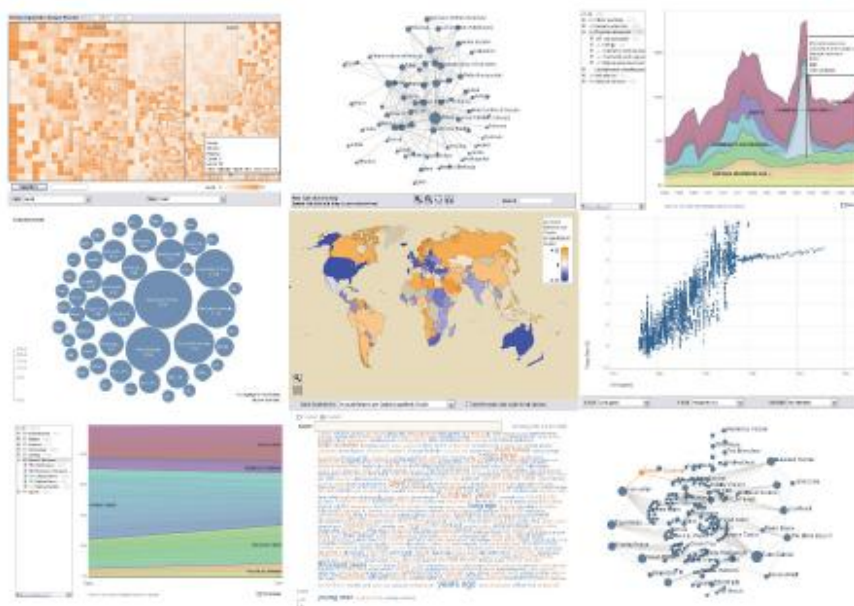


Figura 2.11. Nove visualizações do Many Eyes (Viégas et al., 2007)

Para utilizar as visualizações providas pelo *Many Eyes*, um arquivo de texto delimitado por vírgulas deve primeiramente ser submetido ao website e uma técnica de visualização deve ser selecionada. Diferentes técnicas de visualização possuem diferentes restrições. Por exemplo, para se criar uma visão de nós e associações, os dados precisam conter pelo menos duas colunas com conjuntos de interseção. Após a submissão, o usuário seleciona a coluna para os nós fonte e para os nós destinos e, com isso, o *Many Eyes* irá gerar uma visualização, aplicar um *layout*, e renderizá-lo utilizando um *applet* Java com suporte a operações de *zoom*.

Os dados submetidos são mapeados em elementos visuais dependendo da escolha de visualização realizada. Vale atentar que, apesar de toda a ideia de geração

automática de visualização, o Many Eyes ainda oferece pouca flexibilidade ao manter um conjunto de visualizações e formatos pré-determinados para os dados, além de não permitir a configuração do mapeamento dos elementos de domínio nos elementos visuais. Como ponto forte, existe a simplicidade na interação com o usuário por meio da seleção dos dados e visões desejados, além da reutilização das visualizações existentes em diferentes conjuntos de dados.

2.3.6 Análise dos Trabalhos

Os critérios descritos a seguir foram definidos a partir da análise dos trabalhos relacionados e das necessidades de reutilização, customização e geração de visualizações, que atendam diferentes *stakeholders*, expostas neste trabalho. Para tal, utiliza-se a definição de visualização como sendo uma etapa de **captura de dados**, que é a base fornecedora de dados para a visualização; uma etapa de **visão**, que é a representação visual que exprime um aspecto da informação; uma etapa de **transformação**, que faz a relação entre as duas etapas anteriores, permitindo a flexibilidade e reutilização. Desta forma, as ferramentas descritas nesta seção foram comparadas de acordo com os seguintes critérios:

- **C1. Flexibilidade na captura de dados:** indica a possibilidade de se adequar, customizar e reutilizar um mecanismo de visualização gerado com uma diferente fonte/formato de dado conforme o interesse do usuário. Serão atribuídos os valores “Sim” ou “Não”.

- **C2. Flexibilidade na visão:** indica a possibilidade de se adequar, customizar e reutilizar um mecanismo de visualização gerado com uma diferente forma de visão conforme o interesse do usuário. Serão atribuídos os valores “Sim” ou “Não”.

- **C3. Forma de configuração do processo de transformação dentro da visualização:** indica a forma como é realizada a transformação entre os dados providos por uma fonte e a respectiva representação visual. Serão atribuídos valores descritivos conforme a forma utilizada.

- **C4. Interação do usuário final com a ferramenta:** indica a forma como o usuário interage com a referida ferramenta/abordagem. Serão atribuídos valores descritivos conforme a forma utilizada.

A Tabela 2.1 classifica e caracteriza as abordagens de acordo com os critérios enunciados.

Tabela 2.1. Quadro resumo das abordagens analisadas

	Abordagens				
Critérios	Luthier	Mondrian	CogZ	MDV	Many Eyes
C1	Sim	Não	Não	Sim	Não
C2	Sim	Sim	Sim	Sim	Não
C3	Código-fonte	Scripts	Mapeamento variável	Linguagem de Transformação	Mapeamento fixo
C4	Programação	Construção de scripts	Mapeamento *drag-and-drop*	Modelagem	Seleção da visão

A partir desta comparação, conclui-se que as abordagens pesquisadas, em sua maioria, buscam a flexibilidade dos mecanismos de visualizações construídos em pelo menos uma das etapas (entre captura de dados e visão) de visualização. Poucos dos trabalhos estudados (somente Luthier e MDV) atentaram para a flexibilidade, adaptação e reutilização na etapa de customização dos formatos/fontes de dados a serem visualizados. Este requisito é importante, pois sua ausência gera uma dependência e falta de flexibilidade que podem tornar mais difícil a análise de dados de fontes com formatos específicos.

Com relação à customização da visão, as abordagens que trataram desta questão, o fizeram por quatro meios: *scripts*, mapeamento variável, linguagens de transformação e código-fonte. Todos acabam implicando na necessidade de conhecimento específico das representações visuais e/ou esforço de desenvolvimento para a geração. Assim, o critério de interação com o usuário é prejudicado pela necessidade destas especificidades. Neste quesito, em contrapartida, a abordagem *Many Eyes* abdica de

certa flexibilidade e oferece uma simples utilização por meio da simples seleção dos dados e visões.

Dessa forma, a abordagem desenvolvida neste trabalho, explicitada no próximo capítulo, busca manter uma interação simplificada com o usuário, por meio de escolhas e seleção de características, ao mesmo tempo em que possibilita mecanismos de customização tanto para a fonte de dados como para visão focando na reutilização, na flexibilidade e no desenvolvimento rápido de visualizações. Para isto, fundamenta-se na técnica de desenvolvimento de Linhas de Produtos de Software para o projeto e construção de um ambiente de geração de visualizações focado em reutilização, customização, qualidade do produto e na divisão do processo de desenvolvimento em duas etapas: para o reúso e com reúso. Nesse contexto, a próxima seção discute Linha de Produtos de Software, uma técnica baseada em reutilização para introduzir e ampliar o reúso em mecanismos de visualização de software, além de oferecer flexibilidade para adaptação dos mesmos.

2.4 Linha de Produtos de Software

A forma como produtos são construídos tem mudado significamente com o passar do tempo, principalmente em função do advento de novas tecnologias e processos. Na literatura, existe o exemplo da invenção da linha de montagem por Ford, que possibilitou a produção para um mercado em massa muito mais barato que a tradicional criação individual (Pohl *et al.*, 2005).

Entretanto, esta forma de produção passou a não atender as necessidades do mercado quando os clientes começaram a requerer maior customização dos seus produtos. Assim, ao se tratar de produtos, inclusive software, há a necessidade de uma forma de produção que consiga construir massivamente ao mesmo tempo em que customizações possam ser aplicadas a produtos específicos.

Dessa forma, a técnica de Linha de Produtos de Software (LPS) busca a geração de produtos de software em escala possibilitando a existência de variações em cada um dos objetos criados, para atender demandas específicas. Com isso, a LPS busca a construção de uma família de sistemas focando em dois princípios fundamentais (Pohl et al., 2005):

- **Variabilidade:** a produção em larga escala de produtos adaptados às necessidades de clientes individuais;

- **Núcleo Comum:** base comum de tecnologias em que todos os produtos de uma determinada família são construídos.

Alguns estudos (Pohl et al., 2005; Clements & Northrop, 2002) apresentam benefícios providos pelo uso do paradigma de desenvolvimento de LPS. Entre eles, podem ser destacados:

- **a redução do custo de desenvolvimento**, ao se reutilizar artefatos do núcleo comum a diferentes tipos de sistemas, implicando na redução do custo individual de cada sistema;
- **aumento de qualidade**, dado que os artefatos do núcleo comum utilizados em todos os sistemas são produzidos para diferentes sistemas, passam por um processo de avaliação e teste maior, que aumenta a chance de detecção de erros e sua correção;
- **redução de *time-to-market***, considerando a reutilização dos artefatos do núcleo comum a médio e longo prazo;
- **redução de esforço de manutenção**, dado que a correção feita num componente pode ser propagada a todos os produtos que o utilizam com a substituição do mesmo, reduzindo a probabilidade de um alto impacto devido ao baixo acoplamento entre os artefatos que compõem o produto;
- **suporte à evolução**, onde novas funcionalidades podem ser acrescentadas a toda a família de sistemas por meio da inclusão de novos artefatos ao núcleo comum;
- **suporte à complexidade**, favorecendo um melhor gerenciamento de sistemas complexos através de uma estrutura de desenvolvimento com reutilização, onde o reúso e o baixo acoplamento mitigam o crescimento excessivo de complexidade;
- **melhora na estimativa de custos**, devido ao uso de uma mesma infraestrutura para a produção de vários sistemas de uma mesma família. Assim, o custo inicial do desenvolvimento da infraestrutura é dividido por um número estimado de produtos a serem gerados, acrescendo-se as necessidades de customizações e criação de variabilidades.

Logo, uma LPS pode ser definida como um conjunto de sistemas compartilhando um conjunto de características comuns e gerenciáveis que satisfazem

necessidades específicas de um segmento do mercado em particular (domínio) e que são desenvolvidos a partir de um conjunto comum de artefatos principais (Clements & Northrop, 2002). Isto envolve estratégia e planejamento de reutilização que levam aos benefícios apresentados anteriormente (esquematizado na Figura 2.12).

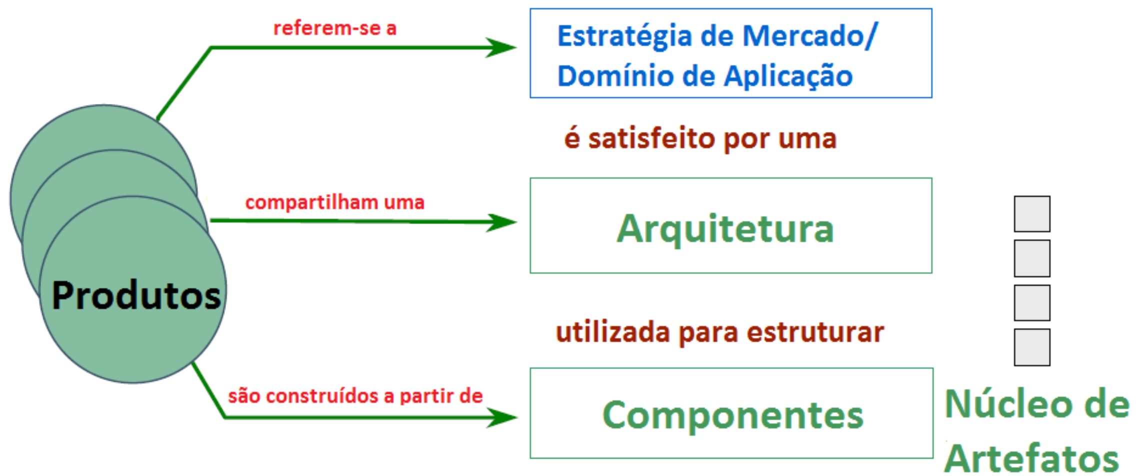


Figura 2.12. Esquema de Linha de Produtos de Software (adaptado de Clements & Northrop, 2002)

A organização que implementa uma LPS planeja o escopo da família de produtos e projeta os produtos para que levem vantagem das partes comuns entre os vários produtos. Durante o desenvolvimento de uma LPS, as diferenças específicas entre produtos também é planejada e pontos de variação são construídos nos artefatos da LPS. Estes são os locais onde a variação entre os membros da LPS irá ocorrer. Esta fase é denominada Engenharia de Domínio (*Domain Engineering*).

Em contrapartida, existe uma fase posterior, denominada Engenharia de Aplicação (*Application Engineering*), que se refere aos processos envolvidos na criação de produtos a partir de uma LPS existente. Outra terminologia utilizada relaciona engenharia de domínio ao desenvolvimento para reúso, e a engenharia de aplicação ao desenvolvimento com reúso. A Figura 2.13 ilustra os processos e pessoas envolvidas numa LPS.

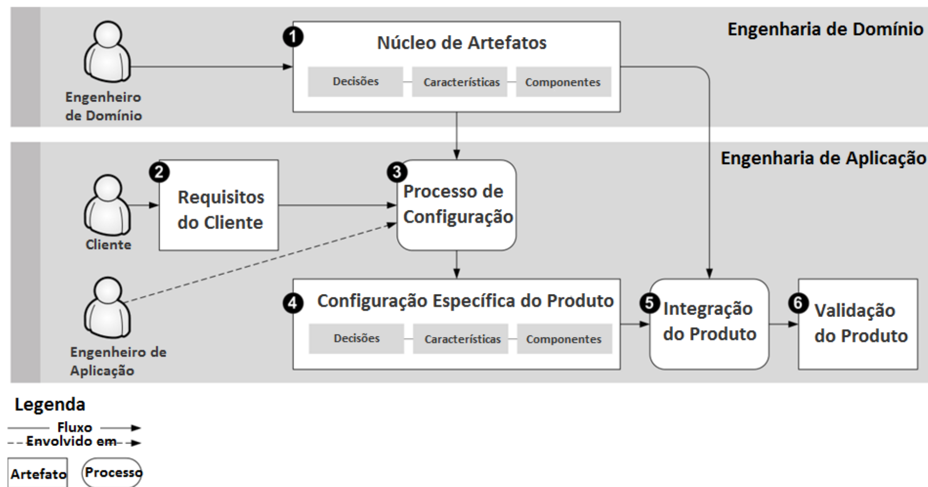


Figura 2.13. Visão geral dos processos inerentes a LPS (adaptado de Pohl et al., 2005)

As próximas subseções tratam mais detalhadamente da abordagem de LPS, iniciando-se por um resumo dos principais conceitos, seguindo a apresentação destas duas fases destacadas, ilustrando a construção e o funcionamento de uma LPS.

2.4.1 Principais conceitos

Nesta subseção, são tratados alguns dos principais conceitos (Pereira *et al.*, 2011) envolvidos na criação de LPSs: variabilidade, gerenciamento de variabilidade, modelo de características e conhecimento de configuração.

2.4.1.1 Variabilidade

Variabilidade está relacionada às possibilidades de alteração ou customização de um sistema. Durante a construção de um software, a sua variabilidade é restringida, como indicado na Figura 2.14. Na fase inicial, a quantidade de sistemas possíveis é normalmente grande, dado que o escopo ainda não está bem definido e as restrições são mínimas.

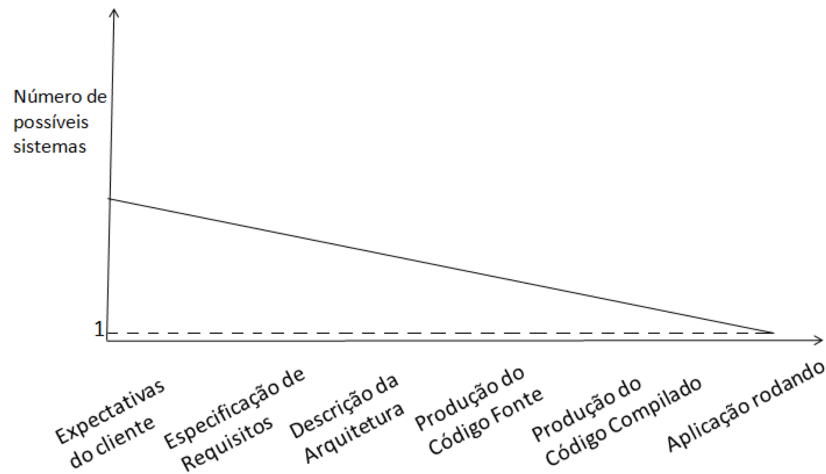


Figura 2.14. Quantidade de produtos possíveis (Pereira et al., 2011)

Ao passo que o projeto vai sendo executado, as possibilidades vão se reduzindo até que, em tempo de execução, exista apenas um sistema. Em cada etapa do desenvolvimento, decisões de projeto são feitas onde cada uma restringe o número de possíveis sistemas. Com a abordagem de LPS, as variabilidades são projetadas e modeladas segundo uma arquitetura de referência, conjunto de padrões arquiteturais predefinidos que norteiam a geração de todos os produtos da LPS, e resolvidas antes da derivação e instalação dos produtos. Esta arquitetura de referência é formada por pontos fixos denominados invariantes e, principalmente, pontos de variação e variantes.

Um Ponto de Variação representa uma alteração funcional num módulo de software, enquanto uma Variante corresponde a uma opção do conjunto de possíveis instâncias que um ponto de variação poderá originar.

2.4.1.2 Gerenciamento de Variabilidades

O objetivo principal da construção de uma LPS é alcançar flexibilidade suficiente para atender às novas funcionalidades e necessidades. Como adaptar uma arquitetura existente para suportar certo ponto de variação envolve alta complexidade, existe a carência por um processo de gestão que antecipe e planeje uma arquitetura de referência para suportar adequadamente o domínio.

Segundo Ali Babar *et al.* (2010), a gestão da variabilidade consiste nas seguintes tarefas:

- **Identificando Variabilidades**: o objetivo deste processo é identificar a diferença entre os produtos, isto é, os pontos de variação e as variantes, além das características que são compartilhadas por todos os produtos.
- **Introduzindo a variabilidade no sistema**: após a identificação da variabilidade, o sistema deve ser projetado de tal forma que ela possa ser introduzida.
- **Agrupando as variantes**: fase que resulta em um conjunto de variantes associadas a um ponto de variação. A coleção de variantes pode ser implícita, baseando-se no conhecimento dos desenvolvedores ou usuários para escolherem variantes adequadas quando necessário, ou explícita, implicando que o sistema decida qual variante usar. A coleção pode ser fechada, significando que não pode ser adicionada nova variante, ou aberta, quando permite novas adições.
- **Vinculando o sistema a uma variante**: associa um ponto de variação particular de um sistema com uma de suas variantes. Este vínculo pode ser feito internamente ou externamente, na perspectiva de sistemas. Uma ligação interna implica que o sistema é capaz de vincular uma variante particular, ao passo que se a ligação é externa, o sistema necessita de outras ferramentas para realizar a vinculação.

2.4.1.3 Modelo de Características (do inglês *Feature Model*)

A variabilidade e as partes comuns entre os produtos de uma LPS podem ser expressos em termos de características. Apresentado originalmente pelo método *Feature Oriented Domain Analysis* (FODA), uma característica pode ser definida como uma propriedade do sistema visível ao usuário final.

Este conceito pode ser usado para agrupar um conjunto de requisitos relacionados, sendo importante notar que existe uma relação n-para-n entre características e requisitos. Por esta relação com os requisitos, o conceito de característica pode também aproximar, em termos de comunicação, o usuário final e os colaboradores da construção da LPS. Deste modo, o modelo de características procura apresentar uma visão geral de alto nível das principais características comuns e variáveis de uma LPS.

2.4.1.4 Conhecimento de Configuração

O modelo de características, isoladamente, representa apenas a modelagem de um domínio específico, mas não detalha a forma como os produtos deste domínio serão gerados a partir do núcleo de artefatos da LPS. Esse mapeamento entre o modelo de características e os artefatos de implementação é chamado de Conhecimento de Configuração, do inglês *Configuration Knowledge* (Czarnecki & Eisenecker, 2000).

Existem diferentes nomes para a representação do conhecimento de configuração, tais como modelo de componentes, modelo de família, modelo arquitetural, entretanto, todos eles buscam associar os artefatos deste modelo com as características do modelo de características.

No conhecimento de configuração, o mapeamento é essencialmente um conjunto de regras que definem que artefatos de implementação (componentes, classes, arquivos de recursos, etc) entram em cada produto da linha. Em um ambiente de desenvolvimento orientado a objetos, uma classe chamada *Menu* (que trata dos pratos disponíveis num restaurante) estaria representada como uma regra que diz se a característica *Menu* estiver selecionada em um produto, a classe entraria nesse produto também, estabelecendo uma relação de requerimento (uma característica implica em artefato de implementação).

No processo de geração de produtos de uma LPS, existe um motor de resolução de expressões lógica, que verifica a consistência das restrições presentes no modelo de características, além dos artefatos de implementação necessários na instanciação. Após a realização de todas as verificações, a saída do processo, para cada produto, pode ser uma lista dos artefatos de implementação habilitados no produto, ou um projeto individual com uma cópia de cada artefato de implementação, ou um pacote dos produtos gerando os executáveis finais, facilitando a instalação e uso dos produtos.

2.4.2 Engenharia de Domínio

Engenharia de Domínio (ED) é o processo de engenharia de linhas de produtos de software no qual as características e variabilidades da LPS são definidas e construídas. O processo de ED é composto por cinco subprocessos chave (Pohl *et al.*, 2005): **gerenciamento do produto**, onde define-se o escopo da família de sistemas; **engenharia de requisitos do domínio**, que abrange a elicitação e documentação dos requisitos dos produtos da LPS, compreendendo o modelo de características, pontos de variação, variantes e invariantes; **projeto do domínio**, onde é definida a arquitetura de

referência da LPS; **concretização do domínio**, que envolve o desenvolvimento e formação do núcleo de artefatos de implementação para reúso; e **teste do domínio**, responsável pela verificação e validação dos artefatos implementados quanto as funcionalidades e características especificadas anteriormente.

Todos estes processos contribuem para o principal alvo da engenharia de domínio de uma LPS, o desenvolvimento do núcleo de artefatos para reutilização, que acontece na forma de um ciclo de vida, resultando em uma base de ativos, cujo conjunto compõe a plataforma da LPS (Schmid *et al.*, 2007). Nesta fase são definidos os aspectos comuns e a variabilidade da LPS e, com isso, planejados e desenvolvidos artefatos reutilizáveis que os contemplem, segundo uma arquitetura de referência que permita a expansão da LPS (Pohl *et al.*, 2005). A Figura 2.15 mostra o núcleo desta fase juntamente com suas entradas e saídas.

Figura 2.15. Desenvolvimento da Base de Ativos (adaptado de Clements & Northrop, 2002)

As setas na Figura 2.15 indicam que não existe momento certo de se adicionar uma restrição (*constraint*) ou novos padrões (*patterns*) no desenvolvimento, dado que necessidades e novos requisitos surgem durante o ciclo de vida, e que estas entradas afetam diretamente os artefatos do núcleo. Em alguns contextos, os artefatos desta base de ativos surgem de produtos existentes e, em outros, estes podem ser projetados e desenvolvidos do zero.

Esta base de ativos inclui, mas não está limitada à arquitetura e sua documentação, especificações, componentes de software, ferramentas como geradores

de componentes ou aplicação, modelos de desempenho, cronogramas, orçamentos, planos de teste, casos de teste, planos de trabalho e descrições de processo (Clements & Northrop, 2002). Apesar da possibilidade de criação de artefatos que possam ser utilizados em todos os produtos sem quaisquer adaptações, muitas vezes, algumas adaptações são necessárias para aumentar sua eficácia no contexto mais amplo da LPS.

Logo, técnicas de suporte a variação dos principais artefatos utilizados ajudam a controlar as adaptações necessárias e a manter as diferenças entre os produtos de software gerados (Bachmann & Clements, 2005).

2.4.3 Engenharia de Aplicação

Engenharia de Aplicação (EA) é o processo de engenharia de linhas de produtos de software responsável por derivar produtos a partir do núcleo de artefatos estabelecido na engenharia de domínio. Explora a variabilidade da LPS e garante a correta montagem e seleção dos artefatos de acordo com as necessidades do produto.

Na atividade de construção de um produto específico, estes são gerados a partir dos artefatos armazenados na base de ativos. De posse do conhecimento de configuração (plano de produção ou *production plan*, em inglês), que detalha como os artefatos de implementação serão utilizados para a montagem do produto, o engenheiro de software pode concatenar as peças da LPS. A Figura 2.16 ilustra esta etapa de geração do produto, desde a chegada das necessidades dos usuários até a seleção e montagem do respectivo produto.

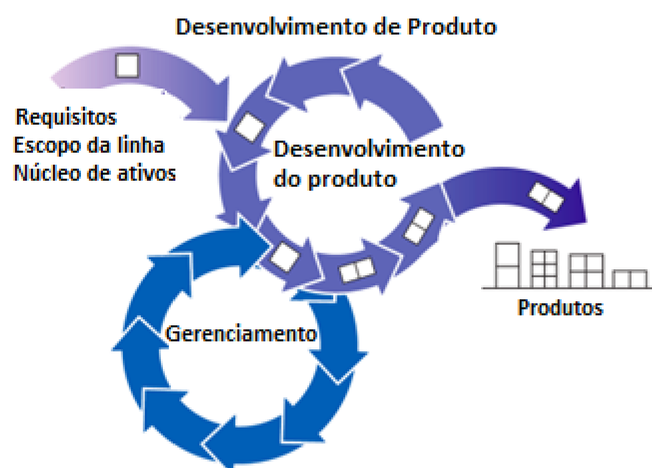


Figura 2.16. Desenvolvimento do Produto (Clements & Northrop, 2002)

Como na Figura 2.15, as setas na Figura 2.16 indicam iterações entre as partes envolvidas. Por exemplo, um produto que tem partes comuns, previamente não reconhecidas em relação a outro produto da LPS, vai criar a necessidade de atualização da base de ativos para explorar essa semelhança através de reutilização em futuros produtos.

2.5 Considerações Finais

Neste capítulo, foi apresentado brevemente o conceito e a disciplina de visualização de software, incluindo as principais características, motivações, dificuldades e alguns exemplos de abordagens. Neste contexto, foi discutido o importante papel da reutilização, flexibilidade e customização sobre a geração de um determinado mecanismo de visualização causando, ao longo do seu desenvolvimento, maior qualidade, menor *time-to-market*, maior facilidade e incentivo ao uso de visualizações, melhor compreensão do software e da informação.

É possível perceber que poucas abordagens buscaram atender os atributos citados anteriormente para mecanismos de visualização e, as que almejaram tal meta, focaram em técnicas que não privilegiam tão profundamente a qualidade do produto e/ou sobrecarregam o usuário dos mecanismos com necessidades de conhecimentos e habilidades não inerentes à visualização de software. Assim, considerando as abordagens estudadas nesse capítulo, incluindo seus problemas e seus pontos positivos, foi proposta a abordagem de Linha de Produtos para Visualização de Software (LPVS), que é apresentada no próximo capítulo.

CAPÍTULO 3 - ABORDAGEM PROPOSTA

3.1 Introdução

No capitulo anterior foram apresentadas diferentes abordagens que tinham por objetivo auxiliar na geração de mecanismos de visualização de software. Através de uma análise realizada a partir destes trabalhos (Seção 2.3.6), aspectos positivos e negativos foram detectados, bem como pontos comuns e pontos divergentes. Isto auxiliou na formalização de uma nova abordagem que atendesse a necessidade de se construir visualizações para software, de forma a contribuir para que o mesmo seja analisado, compreendido e difundido com uma maior facilidade entre os diversos *stakeholders*. Alguns atributos como flexibilidade, fácil interação, capacidade de customização e qualidade, apontados na análise, são essenciais para ampliar a usabilidade destes mecanismos.

Com isso, este trabalho concretiza o conceito de LPVS (Linha de Produtos para Visualização de Software) (Silva *et al.*, 2012) por meio de uma infraestrutura que aplica as técnicas de Linha de Produtos de Software sobre o domínio de visualização de software, almejando-se a rápida criação, flexibilidade e reusabilidade de mecanismos de visualização de software para diversos cenários.

Este capítulo está organizado da seguinte forma: a Seção 3.2 trata, primeiramente, de descrever a visão geral da abordagem, apresentando o objetivo e descrição dos principais módulos envolvidos; a Seção 3.3 mostra o projeto de implementação de cada um dos módulos envolvidos na infraestrutura da LPVS; a Seção 3.4 apresenta um exemplo de utilização da abordagem num cenário típico de Engenharia de Software; e a Seção 3.5 conclui o capítulo discorrendo sobre o conteúdo apresentado.

3.2 Visão Geral da Abordagem

Motivado por uma ferramenta de visualização pautada em reutilização, construída pelo grupo de reutilização da COPPE/UFRJ, denominada EvolTrack (Werner *et al.*, 2011), este trabalho buscou ampliar o alcance do reúso nesta área por meio de uma abordagem que almejasse a geração de famílias de mecanismos de visualizações.

A Figura 3.1 apresenta uma visão geral da abordagem, compreendida em três etapas principais da LPVS: análise de domínio, instanciação do domínio (representado pelo desenvolvimento dos componentes, qualquer artefato reutilizável, e formação do núcleo inicial de artefatos), e derivação de aplicação (denotado por um *wizard* de visualização de software). Traçando um paralelo com as atividades clássicas de LPS (Clements & Northrop, 2002), as duas primeiras etapas desta abordagem estão inseridas no contexto da Engenharia de Domínio (ED), enquanto a última etapa está inserida no cenário da Engenharia de Aplicação (EA). Esta estrutura modularizada favorece uma separação de papéis ao longo das atividades da LPVS:

- **Analista de domínio**: responsável pela análise e definição do domínio de visualização da LPVS, bem como pela modelagem e administração de suas características;
- **Produtor de componentes**: incumbido do projeto e desenvolvimento de componentes que implementam as características definidas pelo analista de domínio e formam o núcleo inicial de componentes da LPVS;
- ***Stakeholder* de visualização de software**: interessado na utilização de mecanismos de visualização para melhorar atributos ligados à compreensão de software em suas atividades. Seleciona as características modeladas pelo analista de domínio e utiliza, em suas atividades, o pacote de componentes que as implementa.

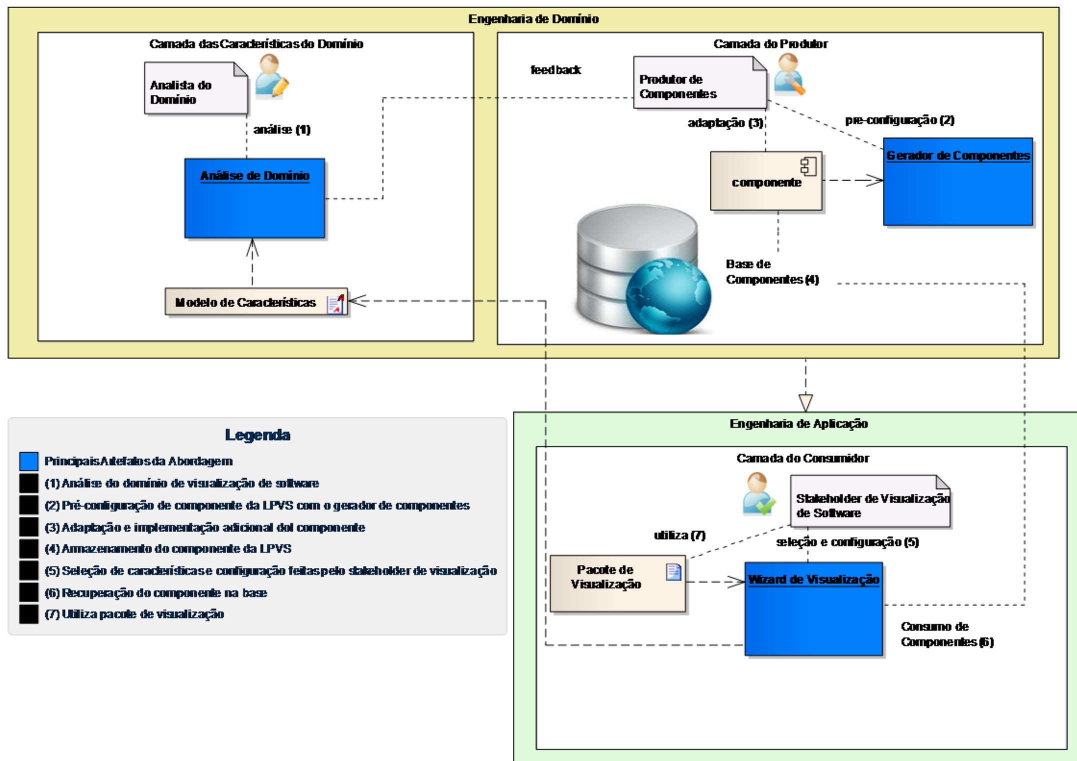


Figura 3.1. Visão Geral da LPVS

A divisão de atividades por papéis, retratada na Figura 3.1, contribui com a redução da especialização necessária dos *stakeholders* para a criação de mecanismos de visualização de software. Como descrito no capítulo anterior, em algumas abordagens, era necessário a manipulação de linguagens específicas de domínio, codificação, entre outras atividades que aumentam a complexidade da geração. Com a divisão de papéis da LPVS, a única preocupação do *stakeholder* consiste na seleção de características primordiais para apoiarem suas tarefas, deixando para a LPVS a atribuição da geração do mesmo.

As subseções, a seguir, descrevem, em linhas gerais, a função de cada módulo principal (marcado em tom mais escuro na Figura 3.1) que compõe a LPVS.

3.2.1 Análise de Domínio

Análise de Domínio (AD) é um subprocesso de ED no qual as características da LPS (seus pontos de variação, variantes e invariantes) são definidas e projetadas (Clements & Northrop, 2002). Nesta fase, o analista de domínio, que possui conhecimentos acerca das atividades dos *stakeholders*, define o escopo dos produtos de

visualização da LPVS, isto é, as necessidades de compreensão que auxiliariam as atividades dos *stakeholders*.

Estas necessidades se transformam em características que são projetadas na forma de **invariantes**, funcionalidades obrigatórias para todos os mecanismos de visualização; **pontos de variação**, onde uma funcionalidade pode ser especializada para atender melhor as necessidades do *stakeholder*; e **variantes**, que implementam as funcionalidades que especializam os pontos de variação.

Todas as características são organizadas em um artefato denominado **modelo** de características, onde são descritas as relações entre as mesmas, isto é, os relacionamentos de hierarquia, associação, inclusão e exclusão. Isto é essencial, dado que certas representações visuais impõem restrições nos tipos e formatos de dados possíveis para este mapeamento. Assim, é imprescindível que as restrições estejam descritas.

Vale atentar para que esta etapa de análise de domínio seja realizada de forma contínua durante todo o ciclo de vida da LPVS. Após a realização inicial, onde são definidas as necessidades iniciais, é normal que surjam outros requisitos por parte dos *stakeholders* havendo, dessa forma, uma retroalimentação no modelo de características com novas funcionalidades, restrições, relacionamentos que precisam ser gerenciados.

3.2.2 Gerador de Componentes

Sendo uma LPS, a LPVS precisa de um núcleo de componentes que realize o esperado pelas funcionalidades presentes no modelo de características. Para este fim, é necessário que haja uma arquitetura que possibilite a comunicação entre os componentes construídos para variadas finalidades de forma que trabalhem integrados. Neste sentido, atenta-se para que a arquitetura contemple o fluxo de visualização explicado no Capítulo 2, isto é, que aborde a integração entre módulos responsáveis por fontes de dados, processamento e representação.

Além disso, a qualidade dos componentes é um dos atributos requeridos para que todo o mecanismo de visualização de software opere adequadamente. É importante atentar para este fato, devido à observação feita por Petrillo *et al.* (2012), onde o suporte e a manutenção destes mecanismos eram agravados pela alta complexidade e baixa preocupação com qualidade.

Dessa forma, este módulo busca ser um ferramental para apoio ao desenvolvimento de componentes da LPVS, gerando código no padrão arquitetural definido e pontos de implementação específicos para a adaptação dos mesmos. Com isso, procura-se evitar desvios arquiteturais e tempo com absorção da arquitetura e desenvolvimento. Com a finalização do desenvolvimento dos componentes, estes são armazenados numa base específica para que possam ser gerenciados, consultados e recuperados posteriormente para a composição dos mecanismos de visualização de software.

3.2.3 *Wizard* de Visualização

Este módulo lida com os aspectos técnicos da EA, procurando prover um mecanismo que permita uma forma de selecionar e configurar as características da LPVS advindas do modelo de características da AD. A partir deste módulo, o *stakeholder* de visualização pode escolher e configurar as características que o mecanismo deve possuir para atender suas necessidades e apoiar suas atividades.

Assim, o *wizard* de visualização verifica a seleção do *stakeholder* para que sempre sejam gerados produtos consistentes. Essa interação de alto nível de abstração faz com que o *wizard* retire do *stakeholder* a necessidade de conhecer os componentes responsáveis pela implementação e toda a arquitetura existente. Logo, é criada uma camada que integra desde a seleção das características até a busca pelos componentes requeridos na base, de forma transparente ao *stakeholder*, realizando também a composição e empacotamento para a entrega de um único pacote de visualização que atenda ao solicitado pelo *stakeholder*.

O fluxo de atividades do *wizard* é estabelecido da seguinte maneira: ele recebe o modelo de características como entrada para controlar as restrições, composições e configurações das mesmas, permitindo aos *stakeholders* escolherem aquelas que os atendam; baseado nesta seleção, ele verifica se as regras de composição estão em conformidade com o projeto original do modelo; em caso positivo, o *wizard* recupera os componentes, configura-os e empacota-os num pacote de visualização pronto para ser utilizado.

3.3 Desenvolvimento da Infraestrutura da LPVS

Nesta seção são detalhadas as implementações de cada um dos módulos descritos anteriormente, tendo como foco o domínio de manutenção de software. Assim,

são tratadas a forma, metodologia e tecnologias utilizadas para a concretização de cada módulo. De forma análoga à seção anterior, as subseções a seguir discorrem sobre as principais etapas da abordagem da LPVS.

3.3.1 Análise de Domínio para a LPVS

No âmbito desta pesquisa, apesar de a visualização de software poder ser aplicada em diversas áreas e atividades da Engenharia de Software, atentou-se particularmente para o campo de manutenção de software, devido ao alto custo das atividades envolvidas nesta área e o valor da compreensão de programas e suas propriedades na execução das mesmas (Xiong *et al.*, 2009).

Sendo assim, o foco inicial para a LPS, porém não restrito, são mecanismos de visualizações focados para auxiliar, parcialmente ou totalmente, atividades relacionadas com a manutenção de software. Estas, de acordo com Weiss & Lai (1999), podem ser classificadas como adaptativas, corretivas, perfectivas e preventivas.

- **Atividades adaptativas**: consistem na adaptação do software para mudanças no ambiente como hardware e sistema operacional. O termo ambiente tem o amplo significado de todo fator externo que influencia um sistema de fora para dentro.
- **Atividades corretivas:** lidam com o reparo de falhas e defeitos encontrados, normalmente relacionados com erros de projetos, erros lógicos e erros de código.
- **Atividades perfectivas:** referenciam novos requisitos dos *stakeholders*. Envolvem tanto melhorias funcionais como questões associadas à desempenho.
- **Atividades preventivas:** envolvem recursos para a melhoria da manutenibilidade do sistema, como atualização de documentação, refatoração e modularidade.

Alguns trabalhos da literatura de manutenção de software (Voinea & Telea, 2007; D'Ambros *et al.*, 2007; Cottam *et al.*, 2008; Fisher *et al.*, 2003; Trumper *et al.*, 2010; Storey *et al.*, 2008) mostram a possibilidade de ganho potencial nas atividades pertinentes a esta área por meio de mecanismos que as apóiem através de técnicas e métodos de visualização. Alguns exemplos deste apoio são ilustrados a seguir:

- na **depuração do software**, mostrando a distribuição de diferentes tipos de problemas sobre a estrutura de um sistema, assim como auxiliando no

entendimento de testes realizados, possibilitando melhorias arquiteturais, processuais (especialmente na priorização de problemas) e na qualidade como um todo;

- na **estrutura do software**, apresentando a relação entre as diferentes partes e funcionalidades de um sistema, juntamente com indicadores de erosão e desvio arquitetural e, até mesmo, um conjunto extenso de métricas que contribuam para a análise da qualidade de um software;
- no **comportamento do software**, apoiando o entendimento e análise da execução de um sistema, averiguando a conformidade entre o que está sendo obtido e o que foi acordado inicialmente, descobrindo possíveis anomalias e falhas (entre elas, de desempenho); e
- no **desenvolvimento do software**, suportando a colaboração entre as equipes envolvidas durante o ciclo de vida de um sistema, favorecendo a análise da interação e comunicação entre as pessoas e times, além do levantamento de áreas de conhecimento, no contexto deste software, por parte dos colaboradores.

Esta pesquisa orientou a construção do modelo de características que é utilizado pelos módulos seguintes da LPVS. Este modelo foi desenvolvido utilizando a ferramenta Odyssey (Reuse, 2012) que atua, entre outras áreas, na engenharia de domínio. Ela foi desenvolvida pelo grupo de Reutilização de Software da COPPE/UFRJ e possui todas as funcionalidades para administrar o modelo de características, inclusive, em suas futuras manutenções e evoluções.

A Figura 3.2 apresenta uma visão do ambiente Odyssey voltada para a construção do modelo de características, no painel de ED. A marcação A mostra as possibilidades de relacionamentos que pode ser construídos entre as características, por exemplo, associação, herança, dependência, alternativa, entre outras. A marcação B apresenta diferentes tipos de características que podem ser incluídas no modelo, entre elas, conceituais, representando entidades abstratas; funcionais, representando funcionalidades efetivas; operacionais, parametrizando funcionalidades para determinados ambientes. A marcação C exibe a representação de uma característica dentro do diagrama. A marcação D lista as características do modelo. A marcação E mostra a área de regras de composição, onde podem ser estabelecidas regras de exclusão ou inclusão de características com relação à seleção das mesmas. Para todas as

representações ilustradas, é utilizada a notação Odyssey-FEX descrita em Blois *et al.* (2006).

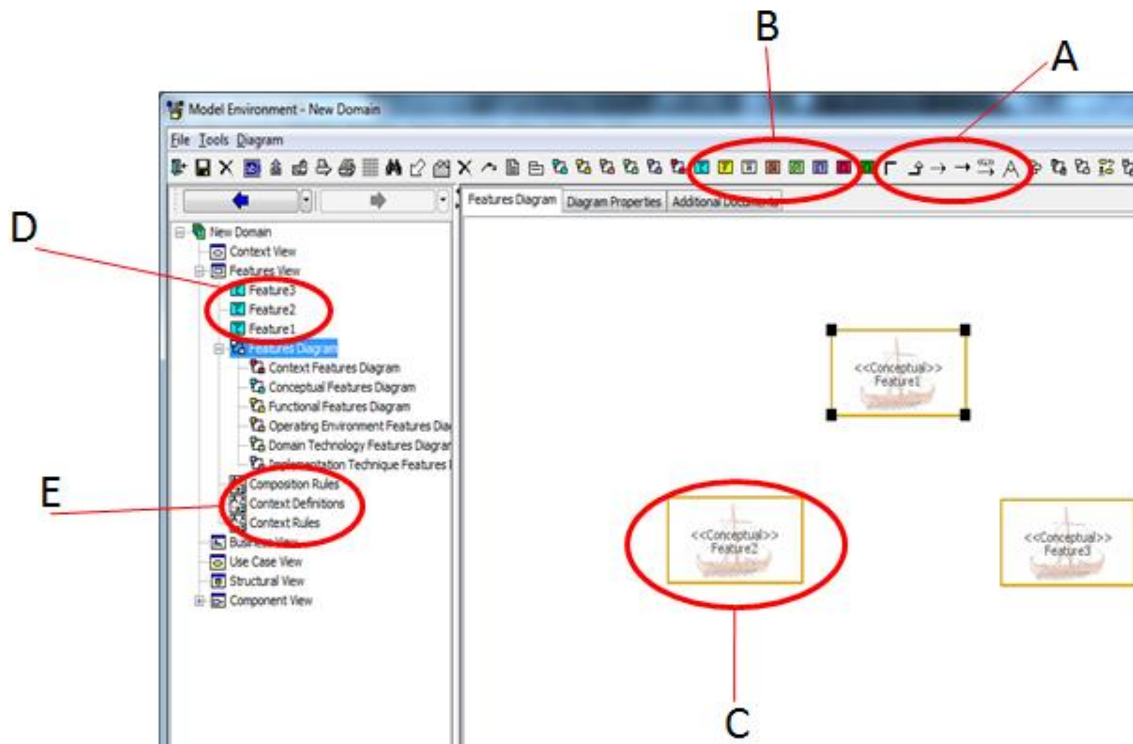


Figura 3.2. Visão da área de trabalho de modelo de características no Odyssey

Devido à dimensão das características para representação do escopo de visualização aplicada a manutenção de software, decidiu-se pela representação do modelo de características em dois diagramas que são interrelacionados por regras de composição definidas no próprio Odyssey. Com esta divisão, diminui-se a carga de informação que seria apresentada por um único diagrama, reduzindo também a complexidade futura no momento da utilização.

O primeiro diagrama (Figura 3.3) busca sumarizar as funcionalidades da LPVS em características mais abstratas, de forma que *stakeholders* menos experientes possam compreender com maior facilidade o objetivo e contexto das mesmas. Neste diagrama, em especial, pode ser visto o escopo da LPVS em três variantes da característica de Análise de Manutenção de Software (*Software Maintenance Analysis*): análise estrutural (*structure analysis*), análise de evolução de métricas (*evolutionary maintenance analysis*) e análise social (*social analysis*). Estes três características acabam por possuir também suas variantes, especificando suas funcionalidades cada vez mais. Na Seção 3.4, é visto como o modelo e a LPVS podem ser expandidos para atenderem a outras áreas (por exemplo, depuração de software).

A análise estrutural busca, principalmente, visualizar a distribuição arquitetural de um sistema, verificando desvios e a organização do software. A análise de evolução de métricas trata do monitoramento de indicadores de qualidade de determinadas propriedades do software, alertando sobre situações a serem observadas. A análise social observa as pessoas e suas interações durante o desenvolvimento de software, sendo importante para análise dos processos envolvidos, melhor distribuição e condução das atividades e, até mesmo, levantamento das áreas de conhecimento da equipe de desenvolvimento.

Vale ressaltar que neste diagrama sumarizado não se associa nenhuma das características com seus elementos de implementação, isto é, componentes. Estas características são associadas a outras de um diagrama específico, por meio de regras de composição (descritas na Figura 3.3 por meio de sequências de letras e números no canto inferior das características), onde estas são ligadas aos componentes que as implementam.

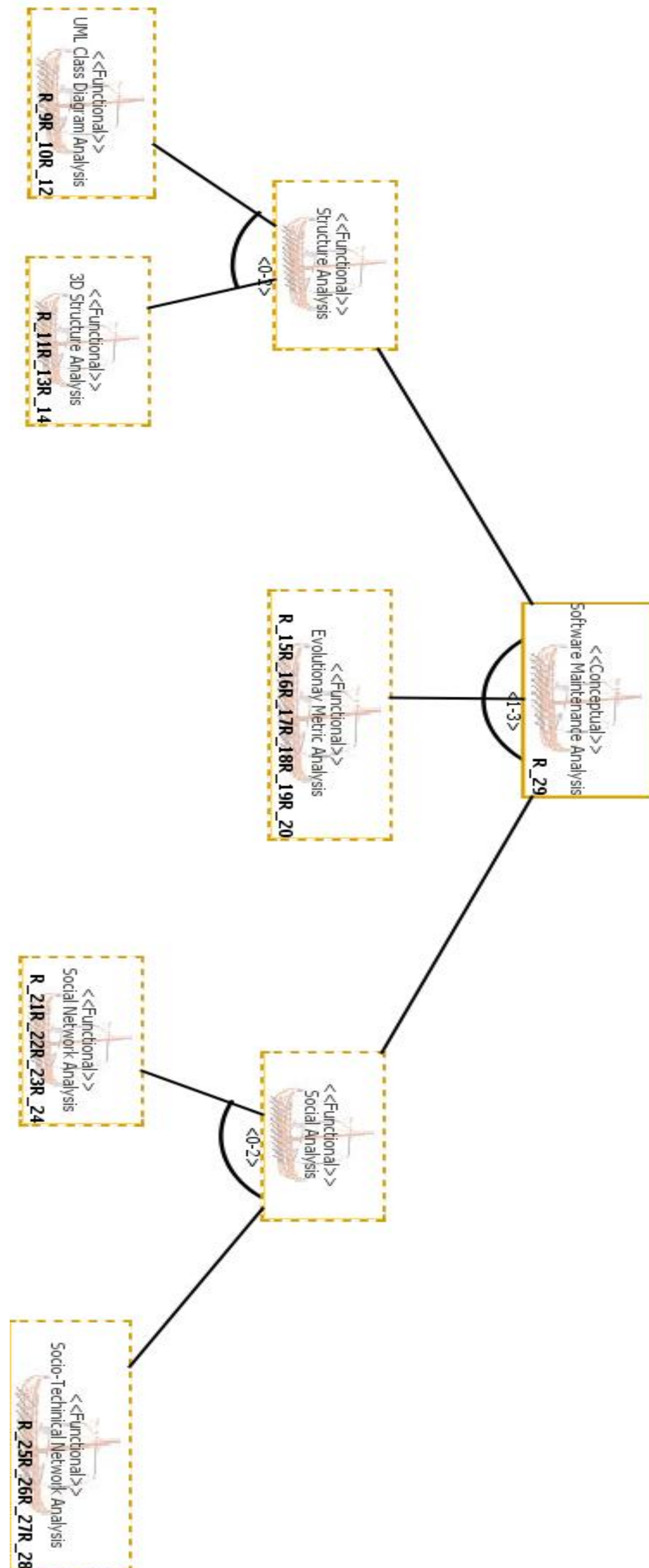


Figura 3.3. Modelo de características da LPVS - diagrama sumarizado

O segundo diagrama (Figura 3.4) detalha as possibilidades de criação de mecanismos de visualização, concentrando as características com funcionalidades específicas no contexto do fluxo de visualização. Neste diagrama, nota-se que a formação do produto de visualização de software se dá pela agregação de três características mandatórias (bordas contínuas): o **Provedor de Dados** (*Data Provider*), uma representação conceitual da fonte de dados; o **Extrator de Dados** (*Data Extractor*), um mecanismo técnico que extrai dados; e a **Representação Visual** (*Visual Representation*), que é uma metáfora utilizada para representar dados de forma a elicitar informação e entendimento sobre um tópico. Existe uma característica opcional (bordas pontilhadas), denominada Processamento de Dados (*Data Processing*), que pode ser utilizada para transformar os dados extraídos antes de serem mapeados em objetos visuais. Caso não adotada, o fluxo dos dados obtidos é transformado diretamente para a respectiva representação visual.

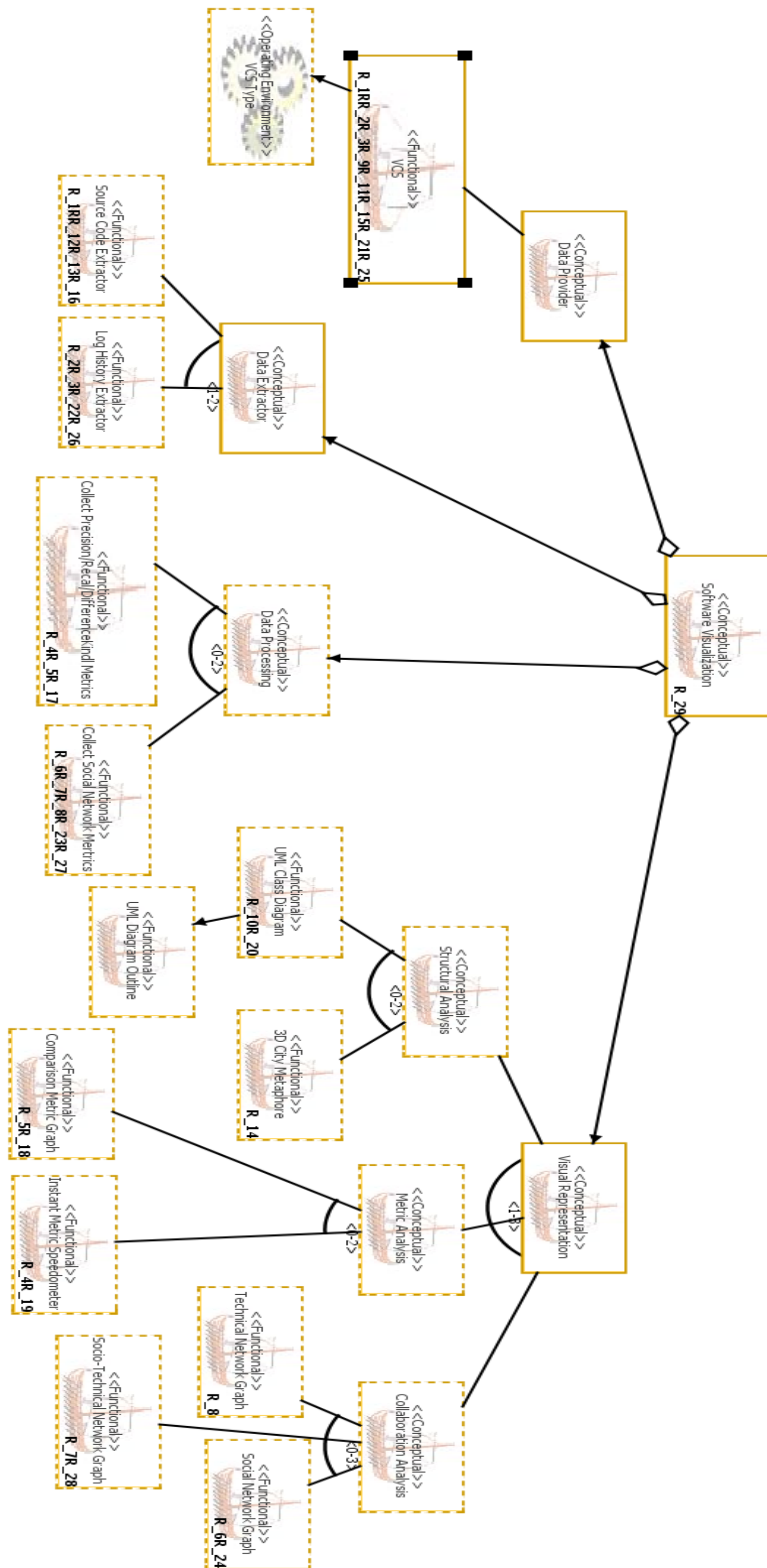


Figura 3.4. Modelo de características da LPVS - diagrama específico

Estas características podem estar interrelacionadas entre si por regras de composição (por exemplo, relações de inclusão e exclusão) definidas no ambiente do Odyssey. Estas regras, como citado anteriormente, aparecem no canto inferior das características relacionadas com uma sequência alfanumérica. A Figura 3.5 mostra a área de criação destas regras (marcação A), assim como a área de associação das características com seus componentes de implementação (marcação B). Lembre-se que é neste diagrama específico onde as características estão ligadas efetivamente aos artefatos de implementação, atentando para o fato de que as características do diagrama sumarizado são associadas a elas.

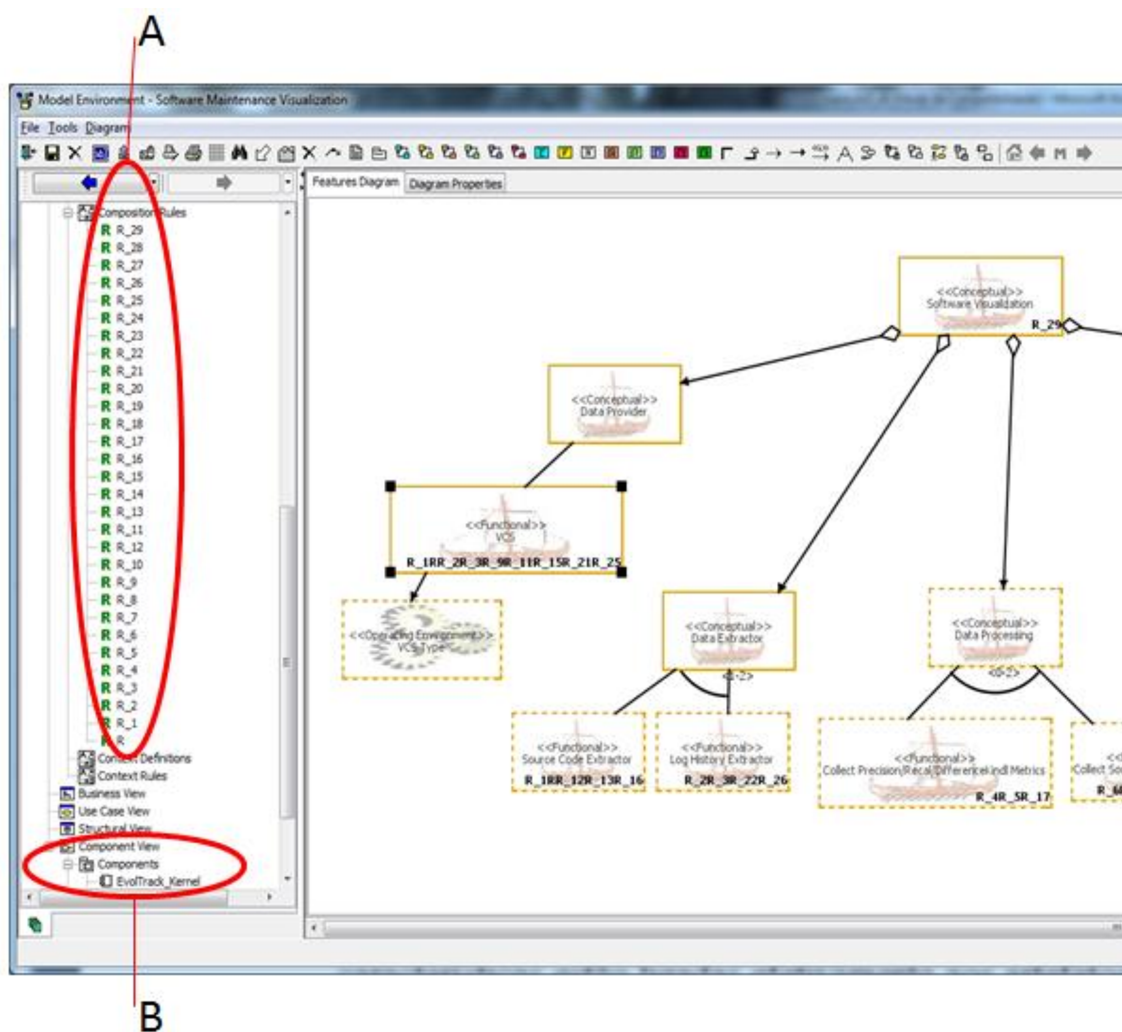


Figura 3.5. Áreas de criação de regras de composição (A) e associação de características com componentes (B)

Como o modelo de características construído é limitado e, portanto, passível de evolução através da ferramenta Odyssey, houve a necessidade de intercambiar estas informações com os demais módulos da LPVS. Sendo assim, expandiu-se um *plugin* do

Odyssey, denominado Odyssey-XMI, para que ele suportasse a exportação do modelo de características no formato XMI (amplamente utilizado para integração e comunicação entre sistemas), servindo de entrada para os módulos seguintes da LPVS. A Figura 3.6 mostra esta nova funcionalidade do *plugin* Odyssey-XMI.

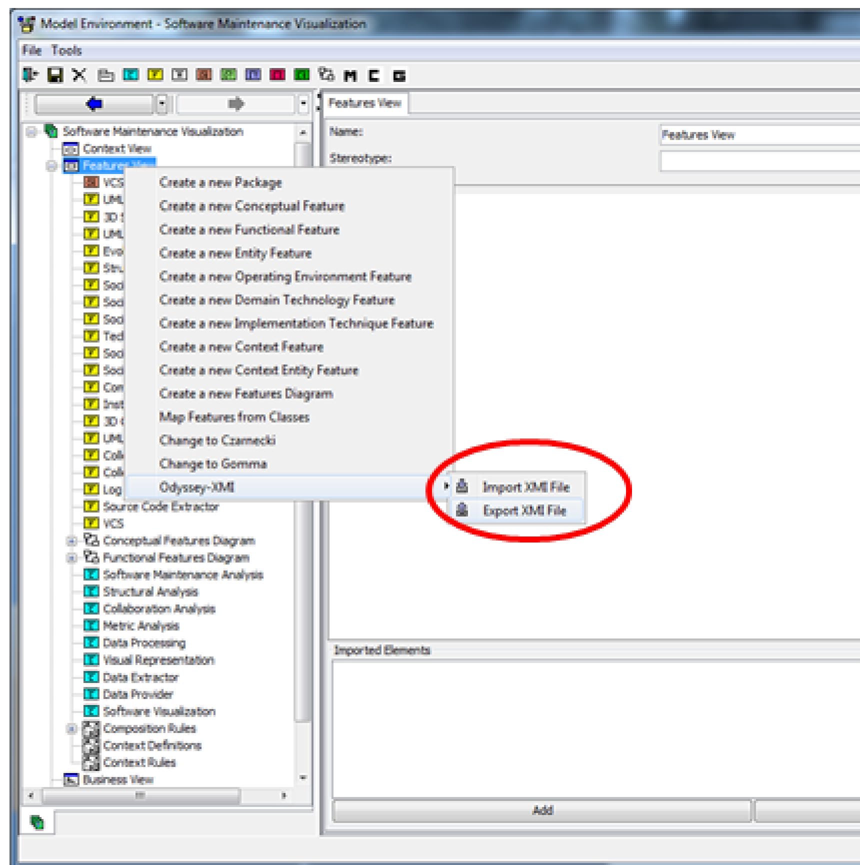


Figura 3.6. Exportação do modelo de características no formato XMI para uso pelos demais módulos da LPVS

3.3.2 Gerador de Componentes

Sendo uma LPS, a LPVS precisa de um núcleo de componentes que implemente as funcionalidades projetadas no modelo de características. Para este fim, é necessária uma arquitetura que aborde e seja compatível com os componentes construídos, integrando-os de forma que o seu trabalho conjunto reflita nas necessidades de visualização de software do *stakeholder*.

Neste sentido, estes requisitos são atendidos pela arquitetura da abordagem EvolTrack (Werner *et al.*, 2011), citada na Seção 3.2, um mecanismo de visualização

baseado na plataforma Eclipse[1] com uma arquitetura flexível, capaz de trabalhar com alguns tipos diferentes de componentes. A Figura 3.7 detalha a arquitetura do EvolTrack.

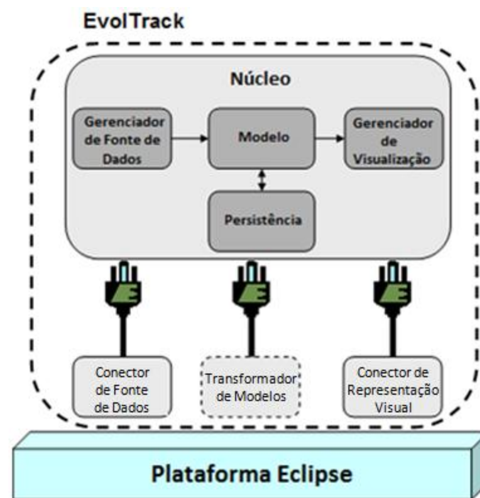


Figura 3.7. Arquitetura do EvolTrack (Werner *et al.*, 2011)

Esta arquitetura (exibida na Figura 3.7) divide as tarefas de visualização em quatro tipos de componentes:

- **Conector de Fonte de Dados**, responsável por prover informações sobre o histórico do projeto ao qual o sistema de software sob análise pertence. Exemplos destas fontes de dados podem ser sistemas de controle de versão (SCV), como Subversion ou CVS, listas de emails, bases de teste, entre outros;
- **Transformador de Modelos**, um componente opcional que pode adicionar informações ao modelo gerado pelo Conector de Fonte de Dados, por exemplo, o valor de métricas. Este novo modelo é chamado de modelo marcado, pois, normalmente aplicam-se estereótipos e valores etiquetados de acordo com um perfil especificado (tanto o modelo como o perfil utilizado pela arquitetura EvolTrack são baseados na UML pela ampla disseminação e aceitação deste formato). Cada estereótipo ou valor etiquetado representa uma métrica ou medida ou qualquer outra informação associada ao projeto (por exemplo, um transformador pode ser implementado para o cálculo do acoplamento entre classes, onde o modelo de cada versão do projeto receberia a marcação de estereótipos em suas classes com o respectivo valor de acoplamento);

[1] http://www.eclipse.org/

- **Núcleo**, cujo principal objetivo é gerenciar as informações de projeto extraídas da fonte de dados, mantendo sua rastreabilidade e orquestrando o fluxo de informação a ser apresentado; e
- **Conector de Representação Visual**, responsável por mapear as informações de projeto ao longo do tempo em uma ou mais abstrações visuais que facilitem a compreensão por parte do usuário.

Basicamente, o fluxo de informação na arquitetura do EvolTrack começa com a extração periódica de dados de um projeto de software a partir de uma fonte de dados configurada e, após o processamento e a transformação destas informações (opcional), as mapeia em uma ou mais abstrações visuais que facilitem a compreensão de determinada informação por um público-alvo.

Dado que a arquitetura detalhada anteriormente apresenta as características necessárias para suportar mecanismos de visualização com poder de configuração e reúso, um dos focos deste trabalho, o passo seguinte está na concretização de componentes aderentes a esta arquitetura para comporem o núcleo inicial da LPVS e, assim, gerarem os mecanismos de visualização desejados.

Para tal, esta abordagem desenvolveu um **gerador de componentes**, cujo principal objetivo é estabelecer um padrão arquitetural e de codificação dos componentes, visando à aderência arquitetural e maior agilidade na construção dos mesmos. Além disso, com o gerador é possível incorporar conjuntos de testes, documentação e pontos pré-estabelecidos de implementação, permitindo que os produtores de componentes se concentrem apenas em ajustes nas funções relativas à visualização de software. Isto busca acelerar o processo e aumentar a variabilidade e qualidade dos produtos da LPVS.

O gerador de componentes foi desenvolvido utilizando a tecnologia Acceleo[2], fortemente baseada em Engenharia dirigida por Modelos (do inglês, *Model Driven Engineering*), onde o desenvolvedor de software pode reduzir a necessidade de codificação por sua parte em razão de uma modelagem consistente das suas necessidades, que são transformadas no devido código segundo regras estabelecidas.

Dessa forma, o gerador de componentes da LPVS funciona a partir de um modelo XMI baseado em UML, construído com base nos requisitos do produtor de

[2] http://www.acceleo.org/pages/home/en

componentes. Relembrando a arquitetura destes componentes, explicitada anteriormente, eles podem ser de três tipos (fonte de dados, transformador e representação visual), além do componente núcleo que gerencia todos os outros e é único por mecanismo de visualização.

No contexto deste gerador, este trabalho limitou-se a englobar, no processo de geração, os componentes do tipo de fonte de dados e de representação visual, já que estes são obrigatórios a todo mecanismo de visualização de software. Futuras versões do gerador são planejadas para abrangerem o transformador de modelos. Atenta-se para o fato que o componente núcleo não entra neste contexto dado que ele é único e imutável para todos os mecanismos de visualização, enquanto os outros tipos de componentes agregam poder de flexibilidade e variabilidade para os produtos da LPVS.

O funcionamento do gerador se dá pela diagramação de um modelo UML a partir de qualquer ferramenta de modelagem que exporte seus modelos no formato XMI (versão 2.0). Entre algumas possibilidades, podem ser citados: RSA, RAD, Rose, Enterprise Architect, ArgoUML, Papyrus, TopCased, GMF, Magic Draw, Umbrello, entre outros.

A criação de um componente do tipo fonte de dados acontece com a modelagem de um diagrama de classes, como mostra a Figura 3.8. Neste exemplo, a modelagem foi feita com a ferramenta Papyrus[3] e apresenta um componente do tipo de dados (do inglês, *datasource*). O componente é modelado como uma classe com o estereótipo *component* e possui três propriedades do tipo texto: **id**, identificador do componente utilizado internamente, especialmente, para nomenclatura da estrutura interna (classes e pacotes); **name**, descrição utilizada para exibição dentro do mecanismo de visualização; **type**, propriedade que representa o tipo de componente a ser gerado (fonte de dados ou representação visual).

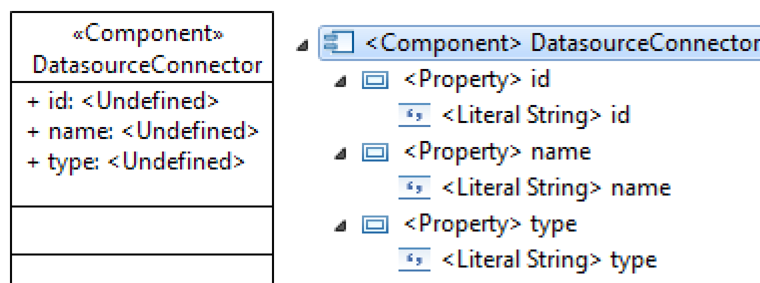


Figura 3.8. Modelo usado no gerador para componente de fonte de dados

Após o desenho e preenchimento do modelo retratado, o gerador se encarrega de transformar essa informação na estrutura de um componente para a referida finalidade. A Figura 3.9 exibe o resultado da geração para o componente de fonte de dados. Nele, observa-se a estrutura construída em conformidade com a arquitetura do EvolTrack, onde a propriedade **id** do componente é utilizada na estrutura de pacote e nos nomes das classes (marcação A); a propriedade **name** descreve o nome do projeto do componente e é utilizada no código que exibe a descrição do componente; a propriedade **type** faz com que toda a configuração (marcação B) seja realizada para o funcionamento de um conector de fonte de dados. Além disso, a marcação C apresenta um típico documento explicativo do conteúdo que cada pasta deve conter, auxiliando os desenvolvedores na construção de forma adequada do componente.

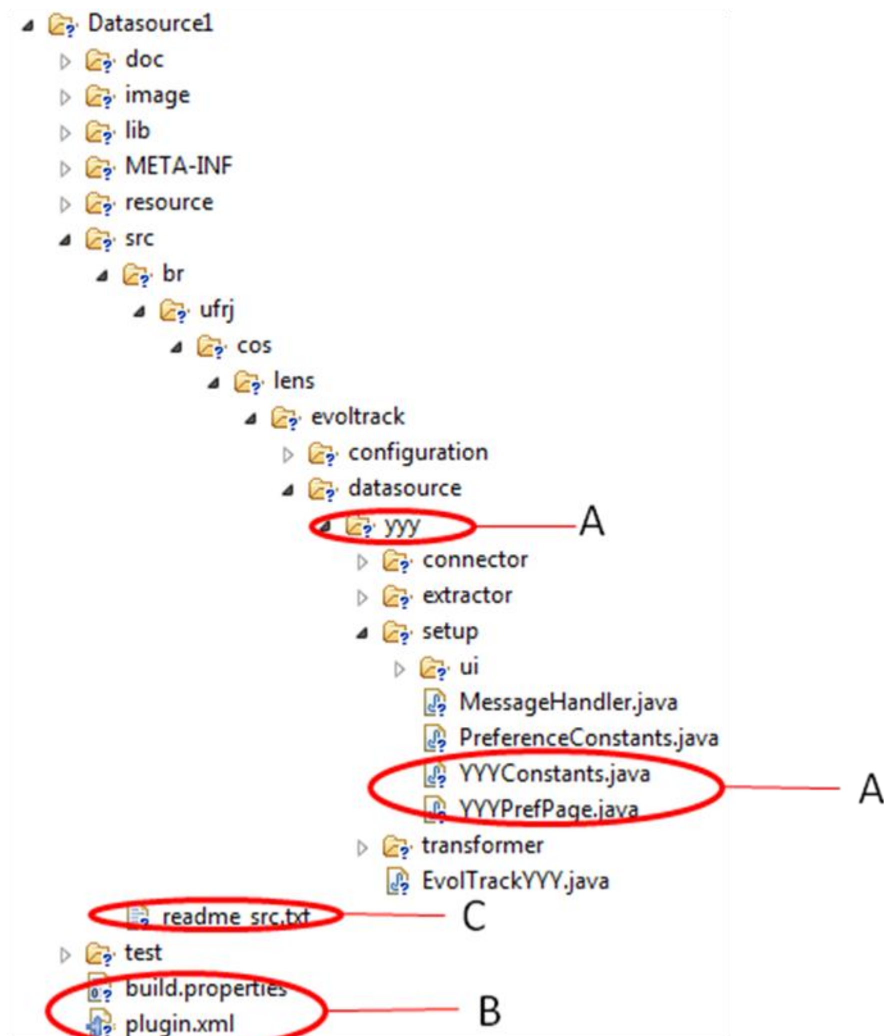


Figura 3.9. Resultado da geração do componente de fonte de dados

[3] http://www.papyrusuml.org/

Na Figura 3.10, é possível visualizar uma classe gerada (YYYExtractor.java) com todo o código estruturado para o funcionamento do componente e um ponto de implementação (circulado) para que o produtor possa especificar e detalhar o processo de extração de dados da fonte que o componente se propõe a atender.

```java
package br.ufrj.cos.lens.evoltrack.datasource.yyy.extractor;

import br.ufrj.cos.lens.evoltrack.connectors.datasource.complex.Extractor;

public class YYYExtractor implements Extractor {

    private static YYYComm comm = null;
    private static String currentRevision = "No Revisions";
    private static boolean newStuff = false;
    private static String currentProject = null;

    public YYYExtractor() {
        comm = new YYYComm();
        loadState();
    }

    @Override
    public void extract() {

        currentRevision = PlatformUI.getPreferenceStore().getString(
                PreferenceConstants.P_CURRENTREVISION);
        System.out.println("Current Revision: " + currentRevision);

        String project = PlatformUI.getPreferenceStore().getString(
                PreferenceConstants.P_PROJECT);
        if (currentProject == null || currentProject.equals(project) == false) {
            currentProject = project;
        }
        /*TODO - Implement a algorithm to get revision data and, in each iteraction, update
        * the variable currentRevision until a break case (may use newStuff to stop).
        */

        PlatformUI.getPreferenceStore().setValue(PreferenceConstants.P_CURRENTREVISION, currentRevision);

    }

    public void saveState() {

    }

    public void loadState() {
        currentRevision = "";
    }

    public static boolean hasNewStuff() {
        return newStuff;
    }

    public static void setTransformed() {
        newStuff = false;
    }
```

Figura 3.10. Código da classe Extractor gerado com ponto de implementação em destaque

No cenário de componentes de representação visual, o componente também é modelado como uma classe com o estereótipo *component*, porém, este possui seis propriedades do tipo texto: **id**, identificador do componente utilizado internamente, especialmente, para nomenclatura da estrutura interna (classes e pacotes); **name**, descrição utilizada para exibição dentro do mecanismo de visualização; **type**, propriedade que representa o tipo de componente a ser gerado (representação visual,

neste caso); **perspectiveId**, identificador da perspectiva de visualização, isto é, do conjunto de visões do mecanismo de visualização; **perspectiveTitle**, título descritivo da perspectiva de visualização; **perspectiveImage**, endereço da imagem ícone da perspectiva de visualização.

Além delas, também são modeladas classes com o estereótipo *component,* representando as visões que o conector de representação visual vai possuir (esta ligação é feita por meio de associações em UML). Cada visão possui propriedades análogas ao do conector, como identificador, tipo, nome, e imagem, porém, possui outras para descrever a possibilidade de múltiplas instanciações daquela visão. A Figura 3.11 mostra o modelo utilizado para a geração de componente de representação visual com dois tipos de visões.

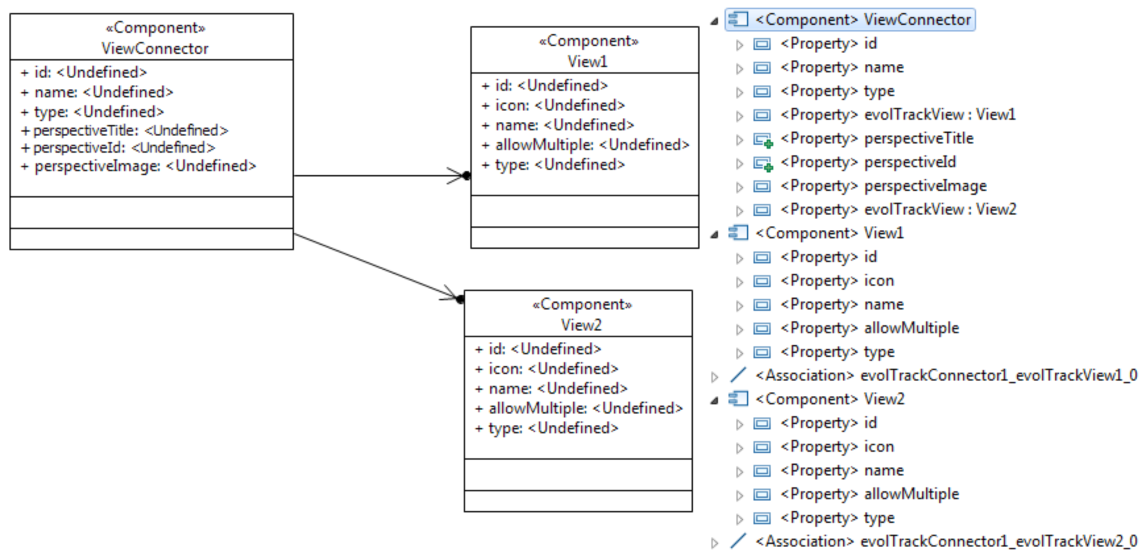


Figura 3.11. Modelo usado no gerador para componente de representação visual

De forma análoga ao componente de fonte de dados, a Figura 3.12 mostra a estrutura do projeto do componente para representação visual (A) e, em destaque (B), o código-fonte de uma classe (View1View.java) com o respectivo ponto de implementação (circulado).

Figura 3.12. Resultado da geração do componente de representação visual

Quando os componentes estão prontos para uso, após a geração inicial e posterior implementação, eles precisam ser armazenados numa base onde possam ser recuperados pelo módulo seguinte da LPVS, para a composição do mecanismo de visualização de software. Inspirado por repositórios de dependências de projetos de software, como o projeto Maven[4], foi utilizado o projeto *open-source* Artifactory[5], que possui uma interface web de administração e gerenciamento de bases de dependências construído com base no Maven, para manter o núcleo de componentes e artefatos da LPVS. Sua escolha se deve a sua interface amigável através da web junto com sua capacidade de compor projetos de software por meio de componentes. A Figura 3.13 apresenta a interface principal da base da LPVS (em destaque) com alguns componentes cadastrados.

---

[4] http://maven.apache.org/
[5] http://www.jfrog.com/home/v_artifactory_opensource_overview

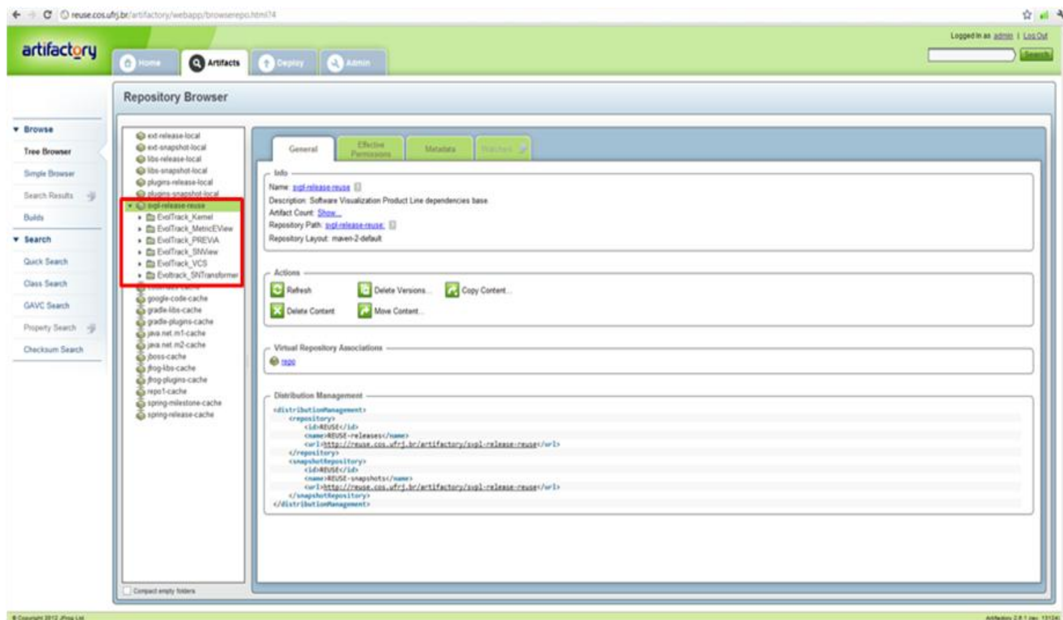


Figura 3.13. Interface web da base de componentes da LPVS

Alguns dos componentes utilizados por esta dissertação foram desenvolvidos por abordagens anteriores do nosso grupo de pesquisa - sumarizadas em (Werner *et al.*, 2011) -, como a PREViA (Oliveira, 2011) (para análise estática e co-evolução através da coleta de métricas e comparação de modelos), EvolTrack-SocialNetwork (Magdaleno *et al.*, 2010) (para análise de redes sociais) e EvolTrack-VCS (para extração de sistemas controladores de versão) (Schots *et al.*, 2010). Outro componente para análise de reúso através de métricas está em desenvolvimento por integrantes do grupo de Reutilização de Software da COPPE/UFRJ.

3.3.3 *Wizard* de Visualização

Este último módulo da LPVS lida com os aspectos técnicos da EA, buscando prover um mecanismo que ofereça uma forma de seleção e configuração das funcionalidades do mecanismo de visualização a ser gerado, a partir do modelo de características advindos da AD, transparente ao *stakeholder*. Atuando como uma interface com o *stakeholder*, o *wizard* de visualização possibilita a seleção de características, levando em consideração as restrições e regras de consistências (mapeadas no modelo), para prevenir a geração de produtos inconsistentes. Devido a este alto nível de abstração, o *wizard* substitui a necessidade dos *stakeholders* conhecerem os componentes responsáveis pela implementação das funcionalidades e suas regras de composição e dependências. Assim, o *wizard* se torna uma camada entre

o modelo de características, o núcleo de componentes e o *stakeholder*, isto é, ele recupera os componentes necessários baseado na seleção feita pelo *stakeholder*, fazendo a sua composição e entregando um pacote de visualização.

O *wizard* pode funcionar em dois modos: a **configuração expressa**, onde espera-se que usuários menos experientes e com *know-how* menos aprofundado possam ter maior comodidade em realizar a escolha de características com base em funcionalidades descritas num nível maior de abstração; e a **configuração detalhada**, onde almeja-se oferecer para usuários com maior experiência a possibilidade de detalhar mais o mecanismo que lhe atenda, conseguindo maior flexibilidade e poder de customização. A Figura 3.14 ilustra a tela principal do *wizard*, apresentando os dois modos para a geração de produtos na LPVS.

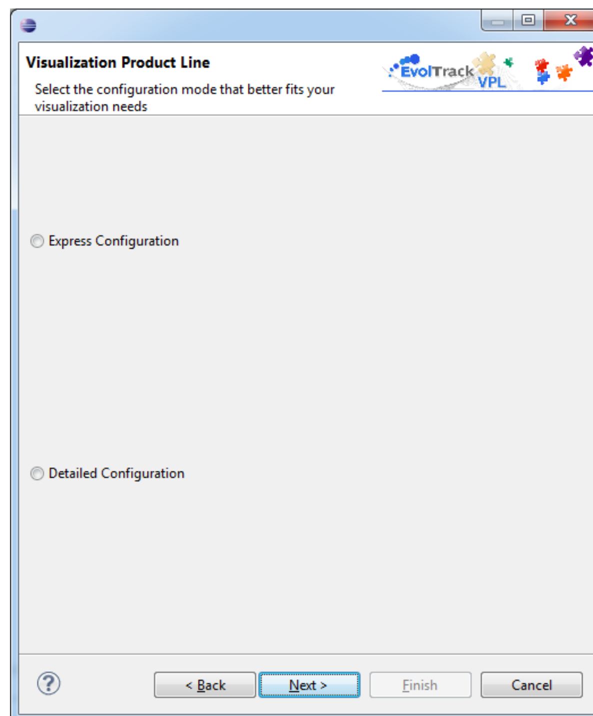


Figura 3.14. Modos de funcionamento do *wizard* de visualização

Vale ressaltar que por estar sendo adotada a arquitetura da abordagem EvolTrack, os componentes construídos e os mecanismos de visualização gerados estão inseridos na plataforma Eclipse, o que levou a criação do *wizard* também no ambiente desta IDE. Nota-se que os dois modos de configuração descritos anteriormente estão diretamente associados ao conteúdo dos diagramas sumarizado e específico, respectivamente, detalhados no projeto de domínio.

O fluxo de trabalho do *wizard* começa recebendo como entrada o modelo de características, de onde captura o diagrama sumarizado e específico, além das regras de composição das características intra e inter diagramas. A partir da escolha do modo de configuração do produto de visualização da LPVS, o *wizard* é capaz de exibir uma árvore com as características do nível de abstração desejado e verificar a consistência da seleção das mesmas por parte do *stakeholder*, conforme mostra a Figura 3.15. Dessa forma, é informada a obrigatoriedade de determinada característica e/ou a impossibilidade de seleção de outra.

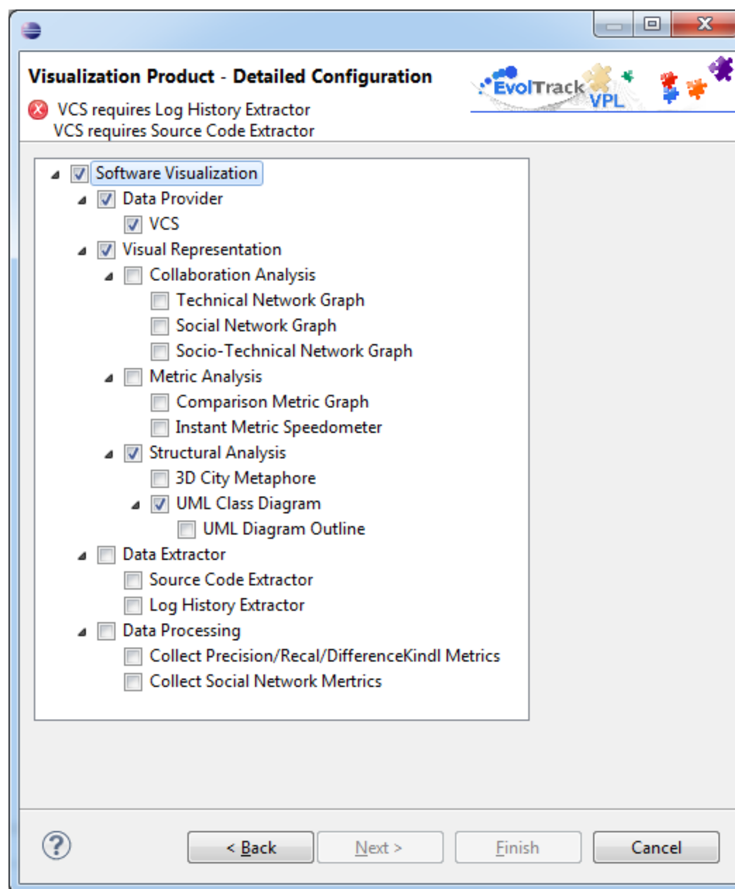


Figura 3.15. Verificação de consistência da seleção de características

Com a seleção verificada e validada, o fluxo pode ser dividido em dois: caso esteja no modo de configuração expresso, o *wizard* segue para a geração do produto; caso esteja no modo de configuração detalhado, o *wizard* possibilita a parametrização de características do tipo de ambiente operacional. Neste último caso, estas características já estão no modelo e possibilitam um nível de configuração mais detalhado, onde o *stakeholder* pode especificar valores para determinada propriedade. A Figura 3.16 mostra este tipo de parametrização para a característica de tipo de sistema

de controle de versão onde, neste exemplo, está sendo parametrizada para atuar com a propriedade SVN.

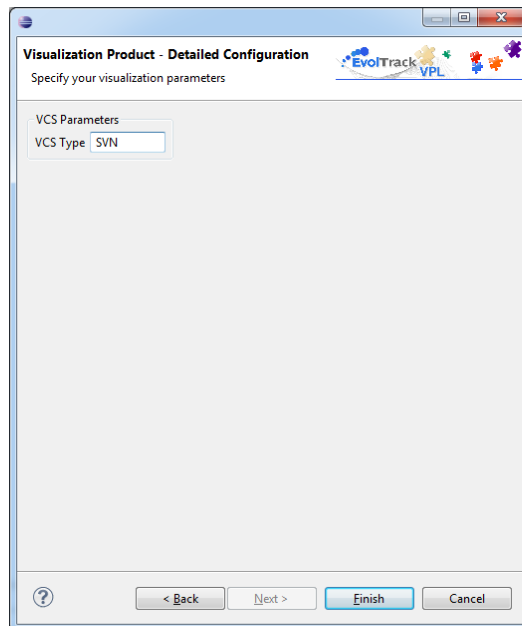


Figura 3.16. Parametrização de características de ambiente operacional no modo de configuração detalhado

Após a seleção e parametrização de todas as características desejadas, para alcançar um nível maior de customização do produto, o *wizard* busca no núcleo de componentes (configurado na tela de preferências do *wizard*, conforme Figura 3.17) todos aqueles envolvidos na implementação das funcionalidades, suas documentações e os empacota, entregando um produto pronto para uso pelo *stakeholder* de visualização. Caso nenhuma parametrização seja feita, são utilizados valores padrões pelos componentes que implementam as funcionalidades selecionadas (isto é previsto na geração do componente, visto na Seção 3.3.2).

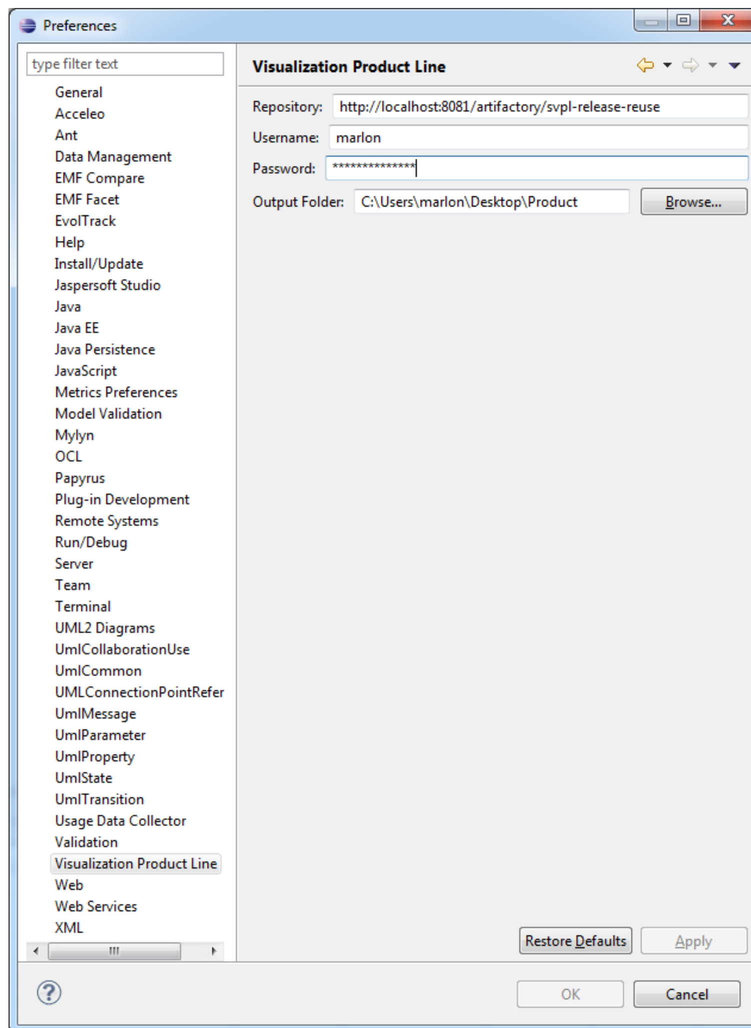


Figura 3.17. Tela de preferências do *wizard* de visualização

A Figura 3.18 apresenta as escolhas feitas por um *stakeholder* que deseja acompanhar e monitorar o crescimento dos módulos de seu sistema. Para isto, ele utilizou o modo detalhado do *wizard* e planejou um produto de visualização capaz de absorver dados de um projeto de software no sistema de controle de versão Subversion (SVN), além de exibir sua estrutura em 3D de forma que fosse possível avaliar o crescimento das classes, interfaces e pacotes em relação ao número de operações.

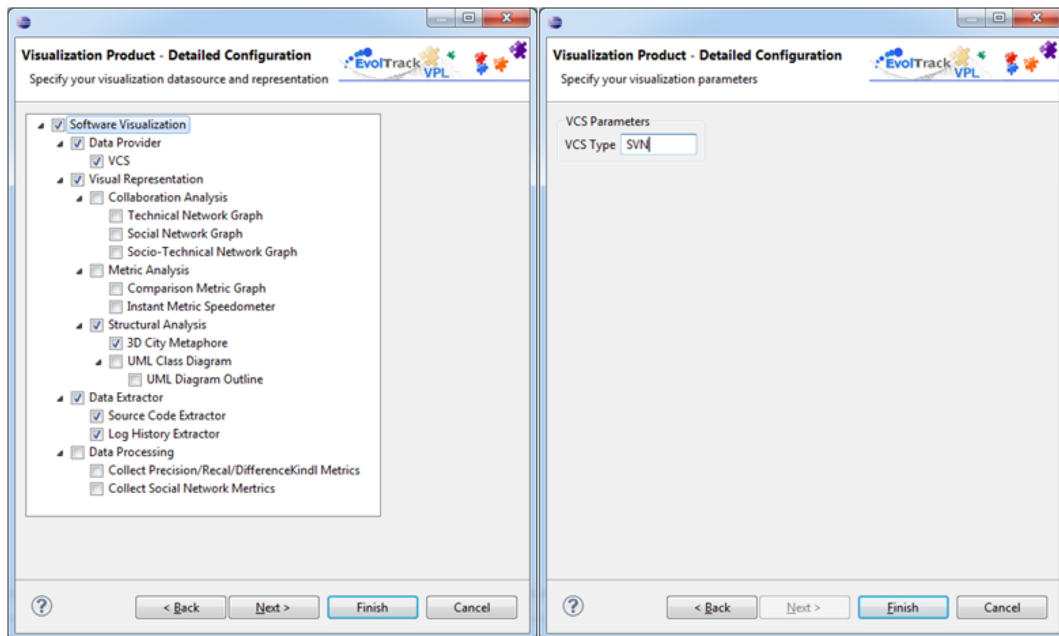


Figura 3.18. Seleção e configuração do produto de visualização

O produto gerado pela LPVS, com manual de instalação e guia de uso (artefatos presentes também no repositório), segundo as escolhas feitas anteriormente, é mostrado na Figura 3.19. Enquanto isso, a Figura 3.20 exibe o resultado da execução do referido produto, após a instalação, auxiliando o *stakeholder* a avaliar o crescimento do software, podendo verificar algum sinal de erosão arquitetural através da detecção de "classes deuses" (do inglês *god class*), má modularização, entre outros fatores.

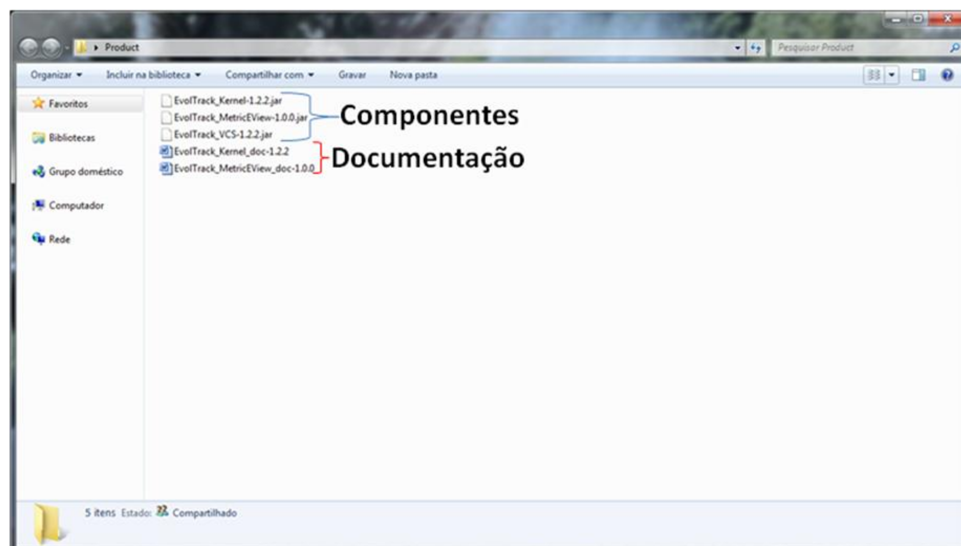


Figura 3.19. Produto gerado pela LPVS (visão dos componentes e documentação)

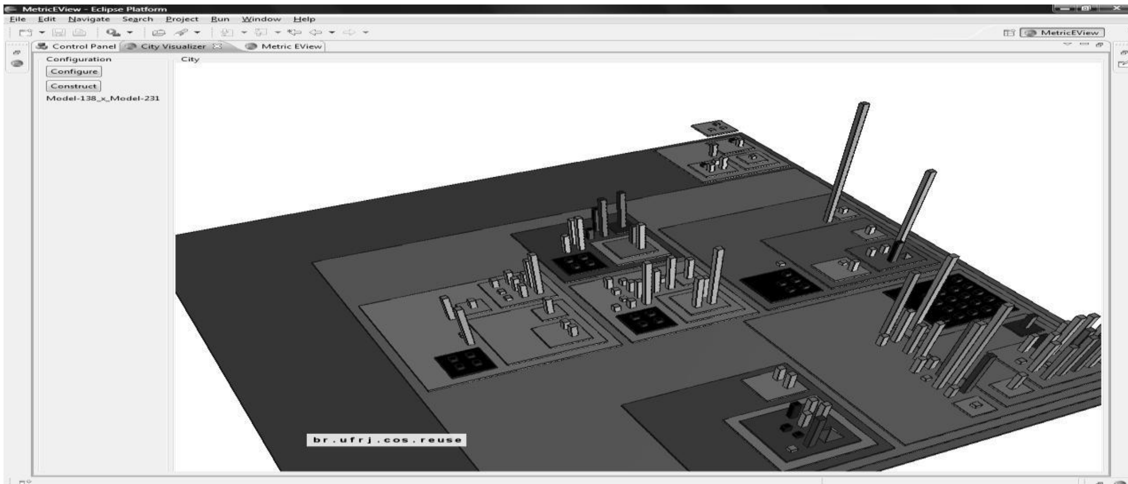


Figura 3.20. Resultado da execução do produto da LPVS (análise estrutural de um sistema)

3.4 Cenário de Utilização

Esta seção apresenta um cenário de utilização de toda a infraestrutura provida pela LPVS, mostrando um passo a passo do uso em uma situação de aplicação. Conforme foi relatado na Seção 3.3.1, uma das áreas de apoio de visualização em manutenção de software é no campo de teste de sistemas. Assim, este cenário de utilização foca em mostrar desde como incluir esta nova funcionalidade na LPVS até o uso da mesma por parte do *stakeholder*.

Esta seção está dividida em: Subseção 3.4.1, descrevendo a necessidade; Subseção 3.4.2, apresentando a criação do novo componente; Subseção 3.4.3, mostrando a inclusão do mesmo na LPVS; Subseção 3.4.4, apresentando o uso da nova funcionalidade.

3.4.1 Análise da necessidade

No contexto deste cenário de utilização, há a necessidade de apoio na atividade de teste de software (explicitada na Seção 3.3.1). Neste campo, existem diversas frentes atuantes como a análise e interpretação de testes, a mineração de *bugs* e bases de erros, além do apoio na descoberta de erros em tempo de execução do software. Este cenário se concentra na primeira frente relacionada aos testes.

Assim, devido ao impacto da análise dos testes na qualidade dos projetos de software, estabeleceu-se como objetivo o apoio na compreensão do resultado de execução dos testes unitários de um sistema, já que estes são utilizados com frequência em projetos de software. Neste exemplo, limita-se a projetos de software construídos na

linguagem de programação Java, portanto, utilizando o framework JUnit para testes unitários.

Logo, a demanda existente é a de construção de um componente que capture os dados de resultados de teste do JUnit, para que estes venham a ser mapeados numa representação visual que auxilie na interpretação dos mesmos de acordo com a estrutura do sistema. Para a parte visual, reutilizará o componente já existente de visualização em 3D da arquitetura de sistemas, mostrando a flexibilidade e capacidade de reutilização da LPVS.

3.4.2 Criação do novo componente

Nesta subseção, é apresentado o passo a passo da criação do novo componente que fará a extração de resultados de teste do JUnit. Este novo componente será denominado EvolTrack-JUNIT e a Figura 3.21 apresenta o modelo UML utilizado no gerador de componentes da LPVS para a sua criação. Atenta-se para o fato de ele ser um componente do tipo de fonte de dados, já que realiza a extração de resultados de teste unitário. Os outros tipos de componentes necessários para o funcionamento dos mecanismos serão reutilizados dos já existentes na base. O resultado do gerador é o projeto de software estruturado como na Figura 3.22.

Figura 3.21. Modelo UML utilizado no gerador de componentes da LPVS para a criação do componente EvolTrack-JUnit

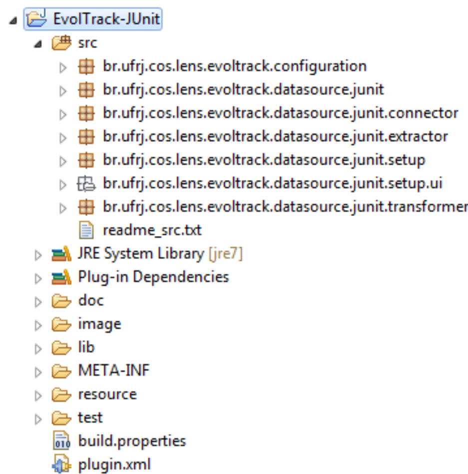


Figura 3.22. Projeto do componente gerado

Neste projeto, existem duas classes de constantes importantes para a reutilização e melhor qualidade do código: JUNITConstants.java, cujo propósito é conter as constantes de uso geral do projeto; e PreferenceConstants.java, que, por sua vez, manipula as constantes da página de preferências (configurações) do componente. A Figura 3.23 ilustra a primeira enquanto a Figura 3.24, a segunda. Nelas foram adicionadas constantes para uso neste exemplo de aplicação, isto é, constantes que representam tanto o status dos testes como o *label* dos campos de preferência.

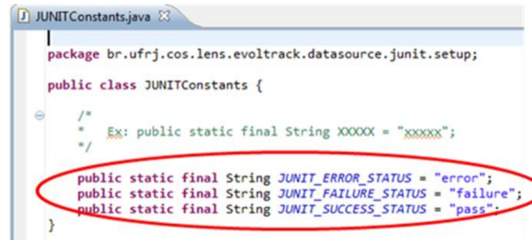


Figura 3.23. Classe JUNITConstants (constantes adicionadas em destaque)

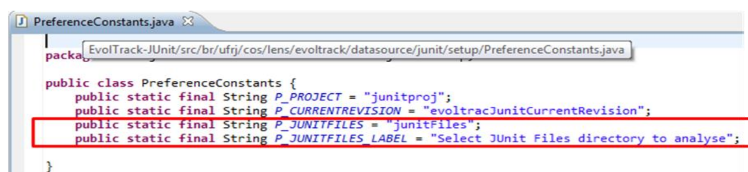


Figura 3.24. Classe PreferenceConstants (constantes adicionadas em destaque)

Para a análise de testes unitários, é necessário lidar com os arquivos de resultados dos testes executados pelo framework JUnit (utilizado neste cenário). Portanto, precisam-se receber estes arquivos como entrada para o processamento dos dados. Assim, edita-se a classe JUNITPrefPage.java para se criar uma página de

preferências e configurações, onde os referidos dados possam ser inseridos. A Figura 3.25 mostra esta classe com destaque para os campos adicionados (A), suas construções (B) e a ação a ser executada após a submissão dos dados por meio dos campos (C).

```
JUNITPrefPage.java
public class JUNITPrefPage extends FieldEditorPreferencePage implements
        IWorkbenchPreferencePage {

    private static List<FieldEditor> fields;

    private DirectoryFieldEditor directoryField;     — A
    private StringFieldEditor proj;

    public JUNITPrefPage() {
        super();
        setPreferenceStore(PlatformUI.getPreferenceStore());
    }

    @Override
    protected void createFieldEditors() {
        fields = new ArrayList<FieldEditor>();

        /*Create new fields to your preferences page*/
        directoryField = new DirectoryFieldEditor(PreferenceConstants.P_JUNITFILES,
                PreferenceConstants.P_JUNITFILES_LABEL, getFieldEditorParent());
        directoryField.setEmptyStringAllowed(false);
        fields.add(directoryField);
                                                         — B
        proj = new StringFieldEditor(PreferenceConstants.P_PROJECT,
                "Choose a project name:", getFieldEditorParent());
        proj.setEmptyStringAllowed(false);
        fields.add(proj);

        // adding fields to preferences page
        for (FieldEditor field : fields) {
            addField(field);
        }
    }

    public static void updatePrefPage() {
        for (FieldEditor field : fields) {
            field.store();
        }
    }

    @Override
    public boolean performOk() {
        JUNITPrefPage.updatePrefPage();

        if (DatasourceManager.instance.getDatasources().isEmpty()){
            DatasourceManager.instance.loadDatasources();
        }

        JUNITComm comm = JUNITExtractor.getComm();        — C
        JUNITExtractor.resetExtractor();
        if (comm != null) {
            comm.setJunitFilesDirectory(directoryField.getStringValue());
        }

        return true;
    }
```

Figura 3.25. Classe JUNITPrefPage e código da tela de preferência adicionados (em destaque)

A partir da definição da entrada dos dados, o fluxo segue na extração da informação destas fontes. Nesse contexto, na classe JUNITExtractor.java, define-se o algoritmo de extração das informações. Neste exemplo, isto se dá pela análise de cada arquivo com resultado de testes unitários até que se esgote o conjunto dos mesmos. Na Figura 3.26, observa-se o referido algoritmo definido no corpo do método *extract()* da classe de extração.

```java
/*TODO - Implement a algorithm to get revision data and, in each iteraction, update
* the variable currentRevision until a break case (may use newStuff to stop).
*/
// Initializing vector of revisions to be checked out
if (revisions == null && comm.getJunitFilesDirectory() != null) {
    revisions = new ArrayList<String>();
    File junitDir = new File(comm.getJunitFilesDirectory());
    if(junitDir.exists()){
        for(File junit : junitDir.listFiles()){
            revisions.add(junit.getName());
        }
    }
}

if (revisions != null) {

    if(currentRevision.equals("No Revisions") && index == 0){
        currentRevision = "";
    }

    // Initializing current revision with index variable
    if (!currentRevision.equals("No Revisions")) {

        if (currentRevision.equals("")) {
            currentRevision = revisions.get(0);
        }

        System.out.println("Extracting from JUnit... Revision: "
                + currentRevision);

        index++;
        if (index < revisions.size()) {
            currentRevision = revisions.get(index);
            newStuff = true;
        }
        else  {
            currentRevision = "No Revisions";
            newStuff = true;
        }

    } else{
        newStuff = false;
    }


    PlatformUI.getPreferenceStore().setValue(PreferenceConstants.P_CURRENTREVISION, currentRevision);
}
```

Figura 3.26. Algoritmo de extração de dados

A última etapa deste fluxo é a transformação das informações extraídas em um modelo canônico para que os demais componentes da abordagem possam fazer uso do mesmo. Para tal, a classe JUNITTransformer.java se encarrega deste mapeamento através do método *transform(),* onde cada elemento capturado é representado num modelo UML para uso futuro. Neste cenário, as suítes de teste são mapeadas em pacotes por agregarem diversas classes de teste; as classes de teste, em classes UML; os testes, em operações de classes. Vale atentar que, neste componente, são acrescentadas informações do status do teste (sucesso, falha e erro), além da agregação e contagem dos mesmos na forma de valores etiquetados da UML. A Figura 3.27 apresenta a transformação feita no cenário deste conector de fonte de dados.

```
public Package transform() {
    model = null;
    if (JUNITExtractor.hasNewStuff()) {
        System.out.println("Transforming junit data...");

        String projectName = PlatformUI.getPreferenceStore().getString(
                PreferenceConstants.P_PROJECT);
        if (project != null
                && !projectName.equalsIgnoreCase(project.getName())) {
            // Project changed

        }
        project = DatasourceManager.instance.createProject(projectName);

        /*Implementing Transformer*/
        packages = new HashMap<String, Package>();
        classes = new HashMap<String, Class>();
        nTestsclasses = new HashMap<String, Integer>();
        nErrorsclasses = new HashMap<String, Integer>();
        nFailuresclasses = new HashMap<String, Integer>();
        currentRevision = JUNITExtractor.getLastRevision();
        rootDirectory = comm.getJunitFilesDirectory();

        File junit = new File(rootDirectory + File.separatorChar + currentRevision);

        List testsuiteList = readJUnitFile(junit);

        model = project.createNewModel("Model-" + currentRevision, "System",  new SysDate(
                new SimpleDateFormat().format(junit.lastModified())),currentRevision);

        processTestsuites(testsuiteList);

    }
    return model;
}
```

Figura 3.27. Método para transformação das informações extraídas num modelo UML

Neste ponto, o novo componente encontra-se pronto, entretanto, ele ainda precisa ser armazenado num local onde os demais mecanismos consigam recuperá-lo para a composição do produto de visualização. Logo, o componente é exportado no formato JAR e carregado no núcleo de artefatos da LPVS. A Figura 3.28 ilustra esse processo de carga do componente na base, lembrando que, junto com o mesmo, pode ser carregado um documento de instruções (acrescentando o sufixo "_doc" ao nome do componente).

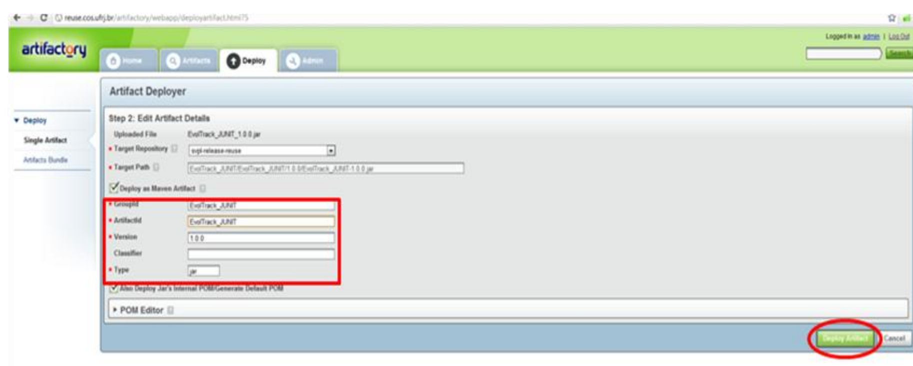


Figura 3.28. Carga do novo componente no núcleo de artefatos

Dessa forma, o processo envolvendo a criação de um novo componente para a realização de uma nova funcionalidade é concluído com um total de 168 linhas de código escritas num conjunto de 6 classes diferentes, passando, então, para a atualização

do modelo de características que, por sua vez, refletirá a nova funcionalidade para o *stakeholder*.

3.4.3 Evolução do modelo de características

Com o novo componente pronto para a execução da nova funcionalidade, faz-se necessário atualizar o modelo de característica com esta informação. Para tal, utiliza-se a ferramenta Odyssey como gerenciadora do modelo para realizar estas mudanças evolutivas. A Figura 3.29 retrata a inclusão de duas novas características no diagrama detalhado da LPVS, onde a primeira (*JUnit*) representa um novo provedor de dados para os mecanismos de visualização gerados pela LPVS e a segunda (*Test Result Extractor*) representa a parte da informação que está sendo capturada.

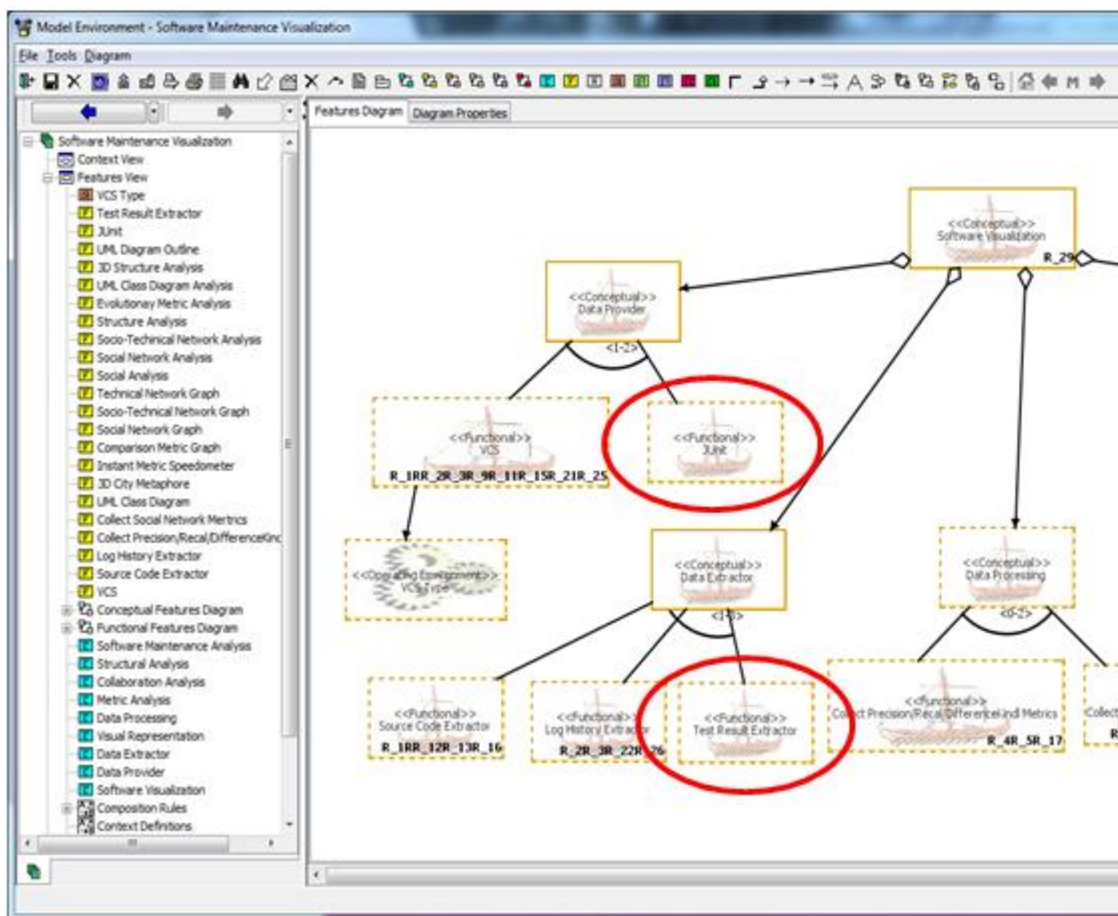


Figura 3.29. Inclusão de novas características para a representação da nova funcionalidade

Dado que neste cenário, apenas dados do framework JUnit são obtidos e, mais especificamente, informações dos resultados dos testes, é colocada uma regra de composição que associe a inclusão de uma característica, caso a outra seja selecionada (Figura 3.30).

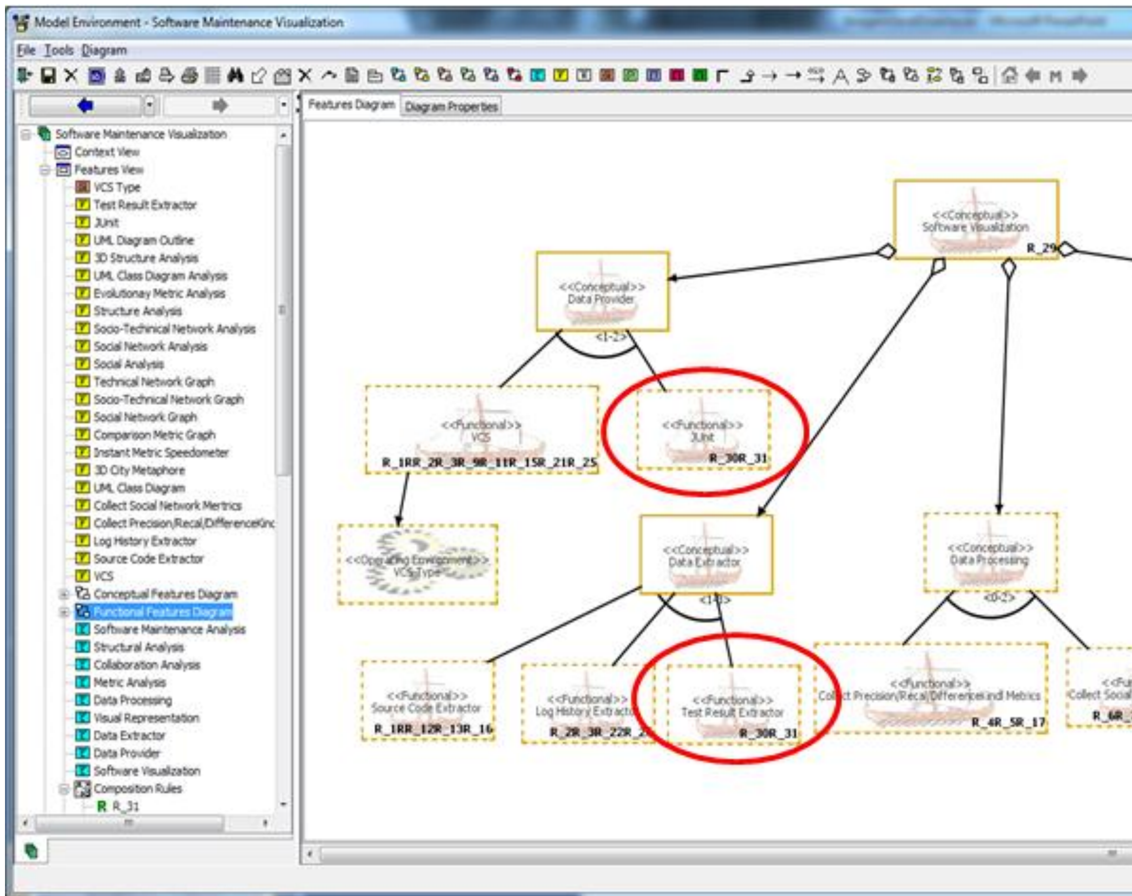


Figura 3.30. Inclusão de regra de composição entre características

Refletindo no uso da nova funcionalidade por usuários menos experientes, acrescenta-se uma nova característica no diagrama sumarizado que resume o projeto deste exemplo de aplicação. De forma análoga, é associada por meio de regras de composição às características incluídas no diagrama detalhado anteriormente. A Figura 3.31 mostra o resultado desta modificação.

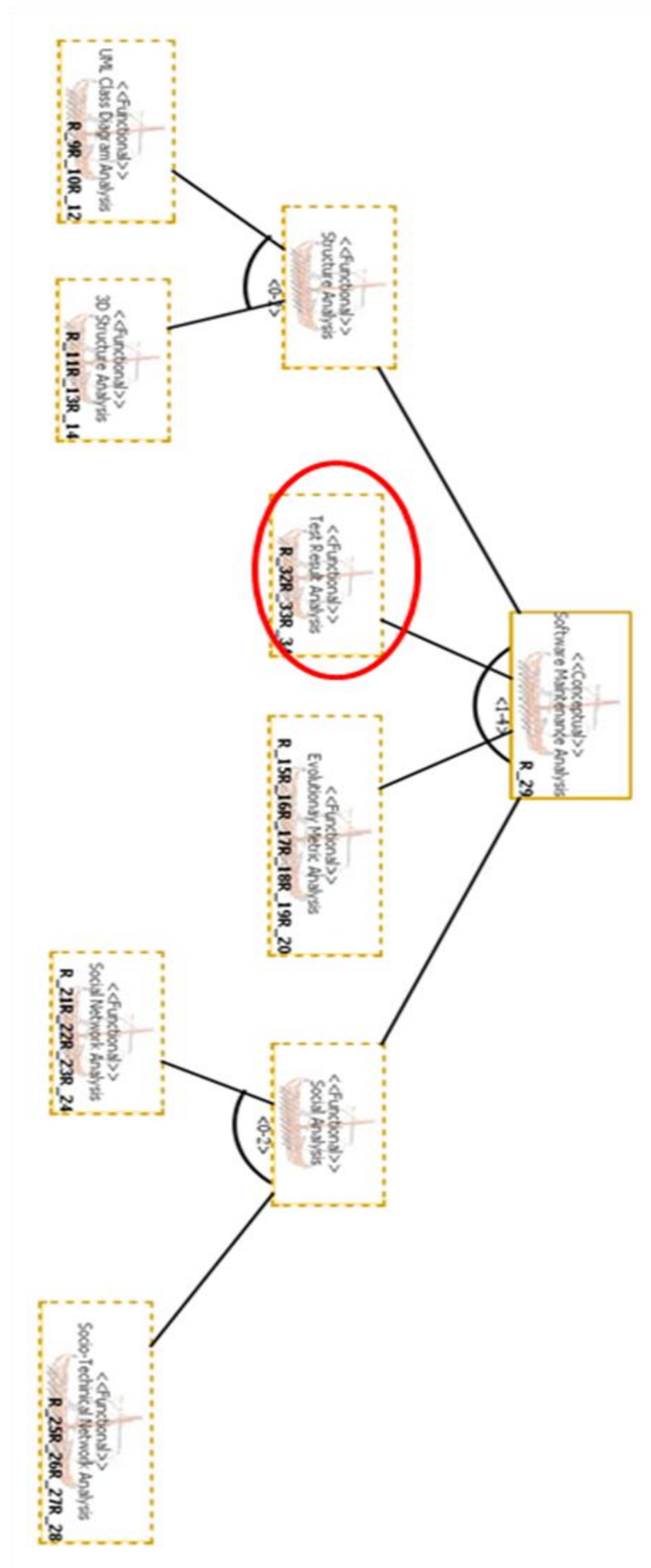


Figura 3.31. Inclusão de nova característica no diagrama sumarizado

Por fim, inclui-se o novo componente no modelo (Figura 3.32) e associa-se as novas características ao mesmo (Figura 3.33) encerrando esta fase de evolução do modelo de características com a exportação do mesmo no formato XML (Figura 3.34), para ser utilizado pelo *wizard*.

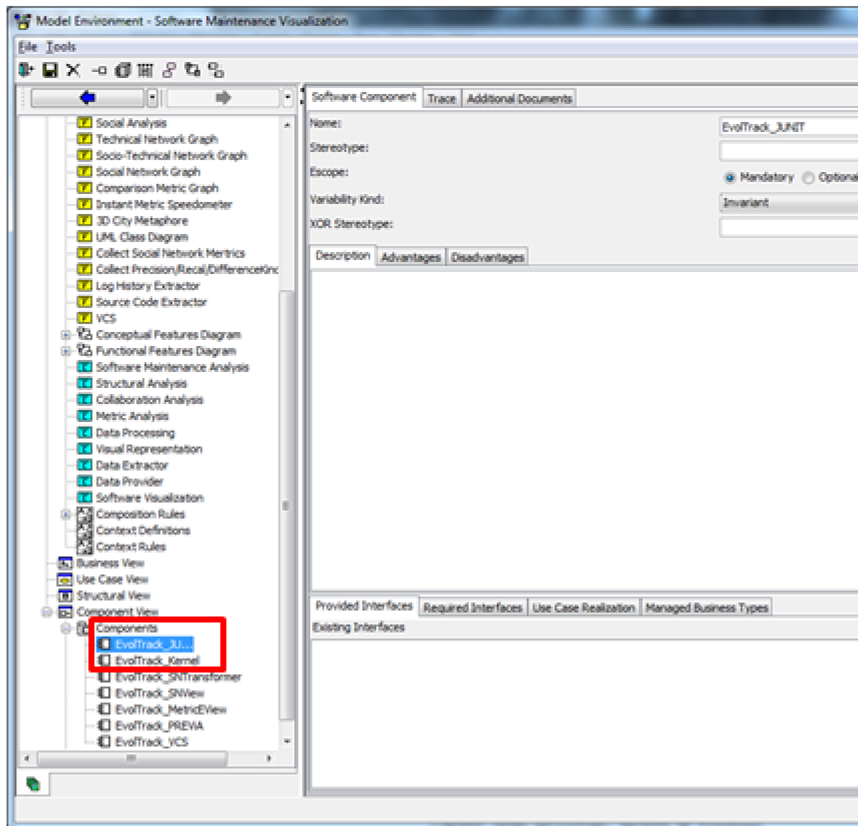


Figura 3.32. Inclusão do novo componente no modelo

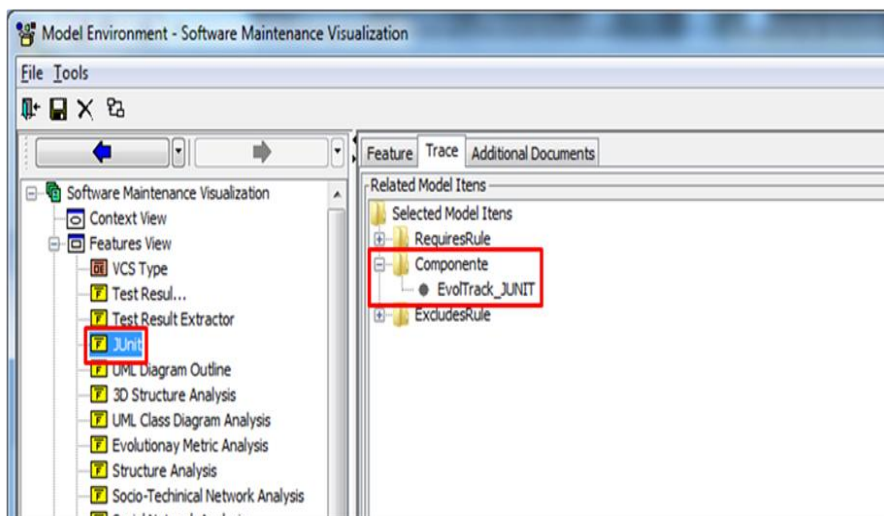


Figura 3.33. Associação das novas características com o novo componente

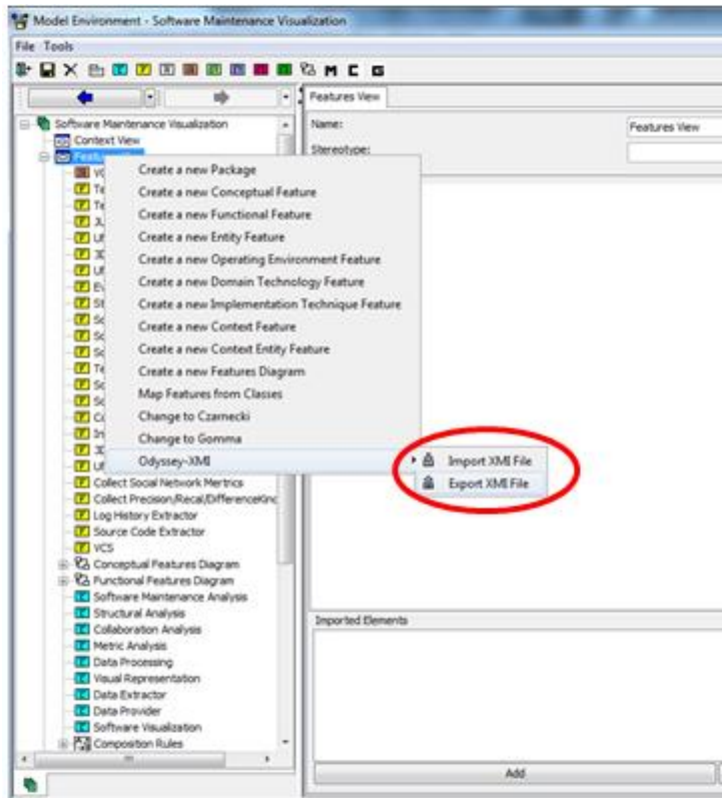


Figura 3.34. Exportação do modelo atualizado no formato XML

3.4.4 Utilização da nova funcionalidade

Este exemplo de aplicação é encerrado pela execução da geração de um novo produto de visualização com a nova funcionalidade descrita no início deste cenário. Tomando como base o modelo de características evoluído na seção anterior, ele é incluído dentro da *release* do *wizard* de visualização que, ao ser iniciado no modo de configuração detalhado (Figura 3.35), já apresenta as novas funcionalidade conforme mostra a Figura 3.36 (novas características no detalhe).

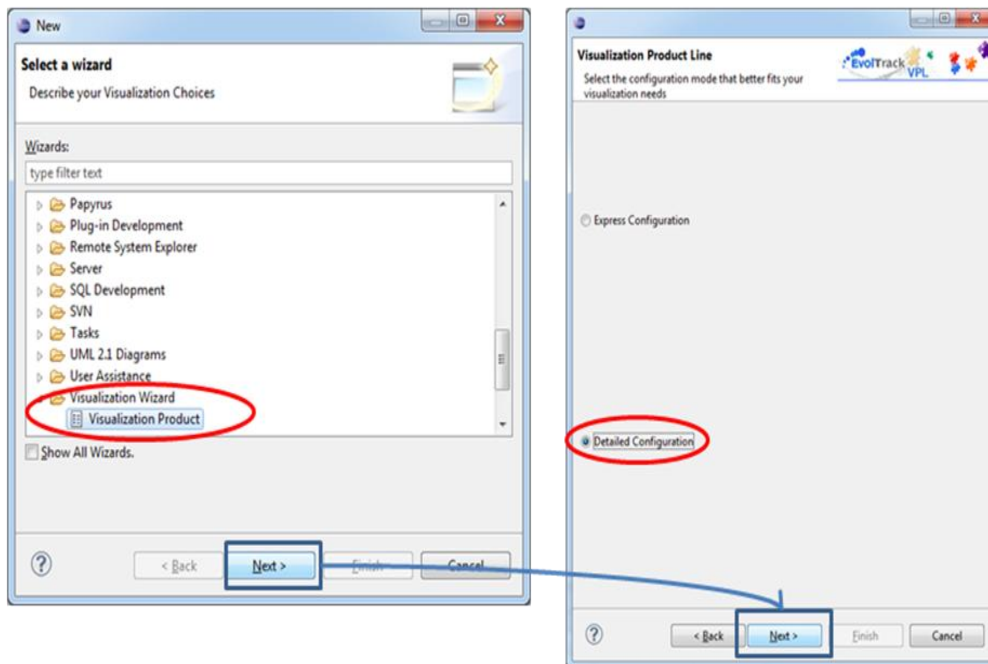


Figura 3.35. Iniciação do Wizard em modo detalhado

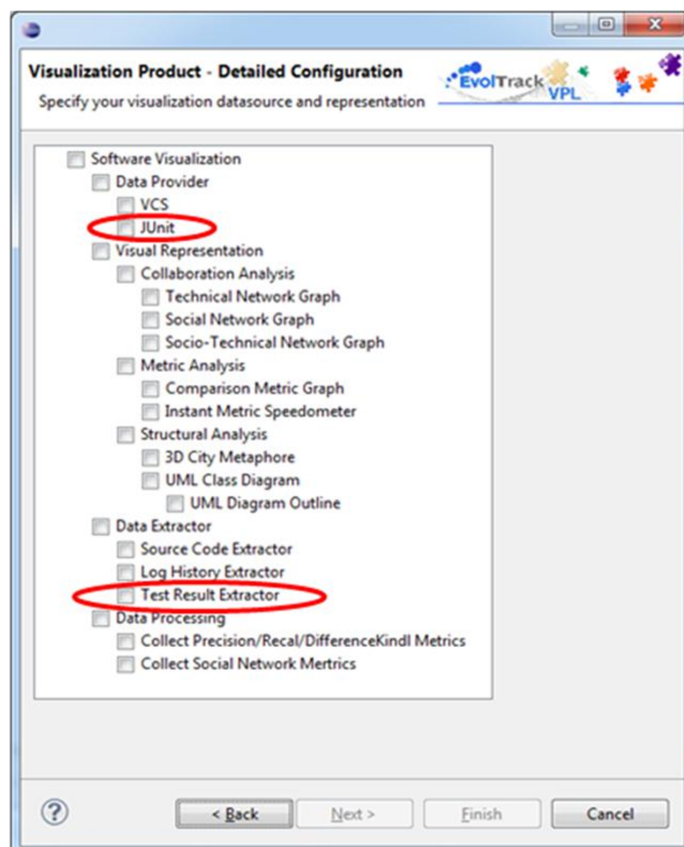


Figura 3.36. Novas características apresentadas no Wizard

Neste caso, estamos interessados em analisar os resultados dos testes unitários de uma forma que enalteça como se deu a evolução dos testes. Para isso, aproveitaremos a característica já existente da metáfora 3D, para que os resultados dos testes unitários sejam vistos desta forma. A Figura 3.37 mostra a simples seleção das características para a geração deste produto de visualização.

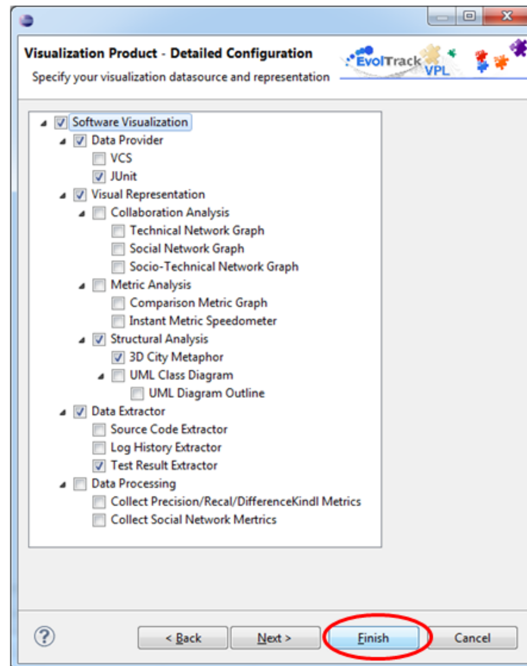


Figura 3.37. Seleção das características para geração de novo produto

Após esta etapa, o *stakeholder* realiza a instalação do mecanismo de visualização pela simples colocação do produto gerado dentro da pasta *dropins* da sua IDE Eclipse (Figura 3.38).

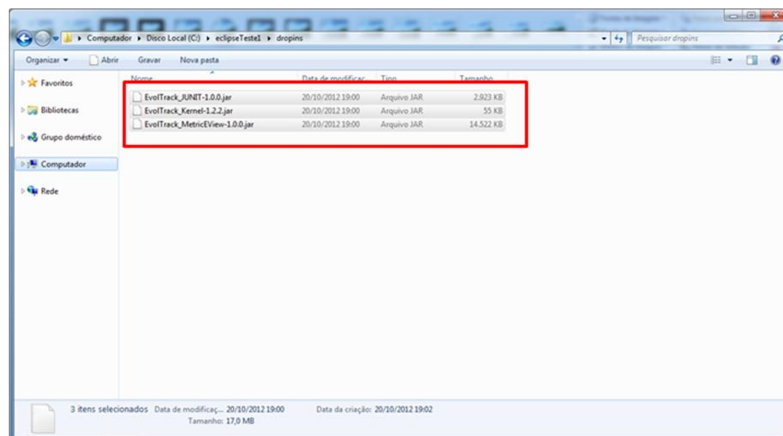


Figura 3.38. Instalação do mecanismo de visualização

Ao executar a IDE, um novo conjunto de opções já estará disponível para uso, como mostra a Figura 3.39, onde o *stakeholder* faz referência aos arquivos de resultados de teste a serem analisados.

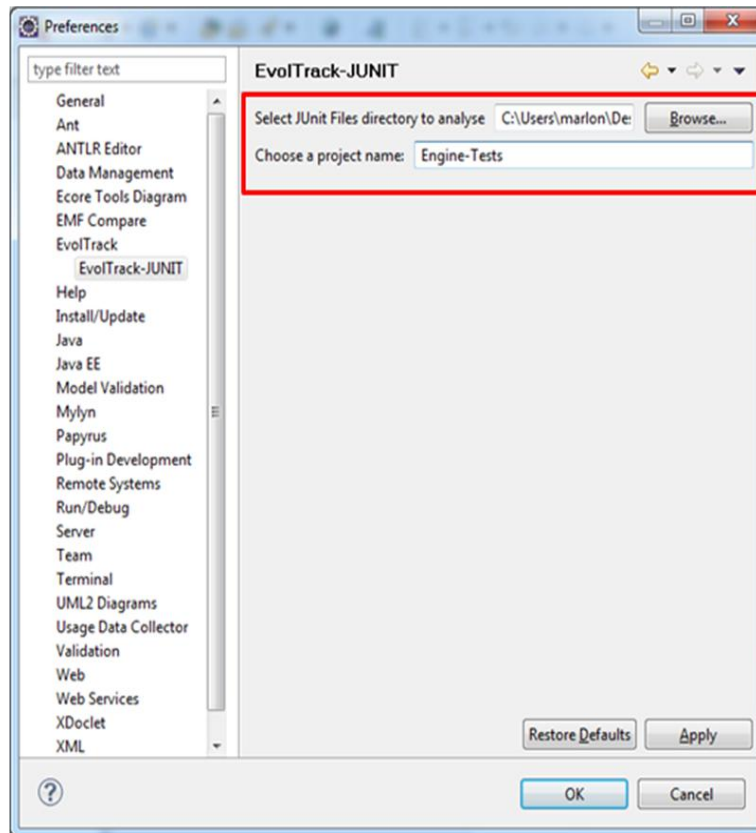


Figura 3.39. Novas opções para análise de resultados de testes unitários

A partir da execução da análise, o mecanismo de visualização é capaz de reportar para o *stakeholder* o resultado dos testes utilizando uma metáfora 3D (Figura 3.40), onde cada suíte de teste é um pacote (metáfora de região); uma classe de teste, uma classe (metáfora de prédio); e um teste, uma operação. Nesta execução, as cores representam o status: verde, para classe que todos os testes passaram; amarelo, para os testes que falharam; violeta, para os testes que obtiveram erros. O tamanho dos prédios foi configurado para representar a quantidade de testes existente na respectiva classe. Além disso, é possível acompanhar estes resultados conforme a evolução dos testes acontece, por meio do painel de controle.

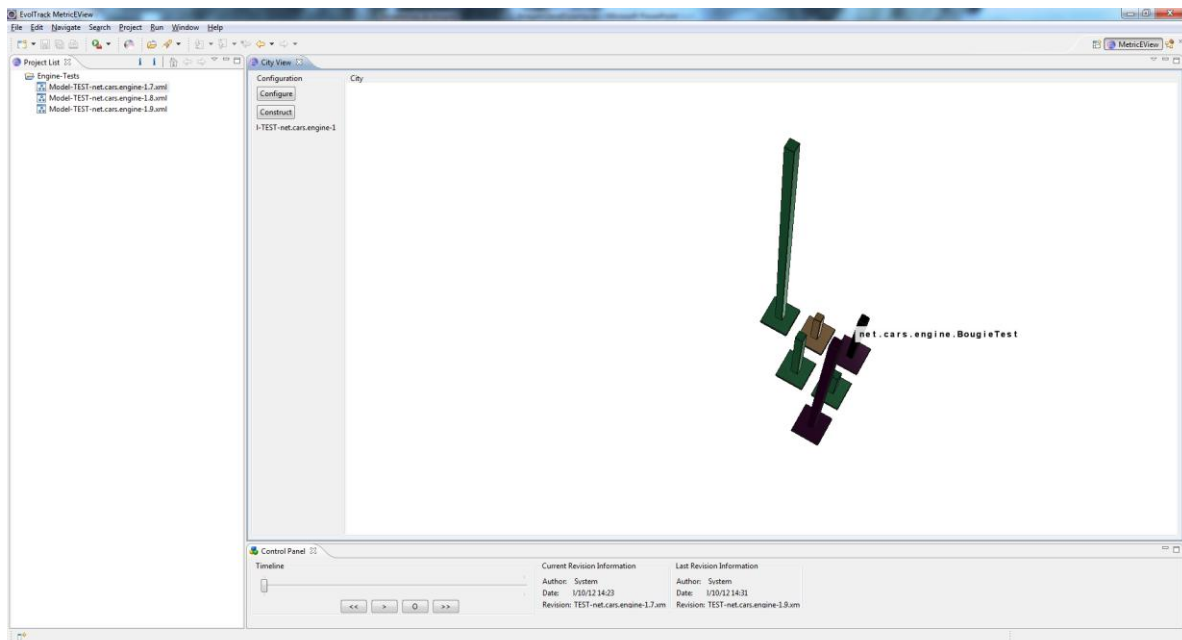


Figura 3.40. Resultados dos testes na forma de metáfora 3D

Com isso, o exemplo de aplicação é finalizado mostrando como foi possível, a partir de uma necessidade, incluir uma nova funcionalidade na LPVS, distribuindo as atividades necessárias de forma que o principal interessado no uso do mecanismo de visualização tenha a maior facilidade, rapidez e flexibilidade na construção do mesmo.

3.5 Considerações Finais

Considerando os problemas destacados no Capítulo 2, sobre a construção de mecanismos de visualização, a abordagem LPVS, apresentada neste capítulo, criou uma infraestrutura capaz de reutilizar componentes de mecanismos de visualização em mais de um cenário, oferecendo um mecanismo que, a partir da seleção de características, seja capaz de validar e compor corretamente os devidos componentes para a execução do que foi especificado oferecendo rapidez e flexibilidade de configuração dos mesmos.

Além disto, também distribui-se as atividades inerentes à construção de mecanismos de visualização de forma que o principal interessado na utilização ficasse apenas com o cargo decisório acerca de sua necessidade, enquanto temas tecnológicos e de organização ficassem com especialistas. Isto busca otimizar o processo de criação, dado que algumas abordagens detalhadas no capítulo anterior passavam parte desse ônus para o interessado no uso.

Por fim, este capítulo apresentou um exemplo de aplicação da abordagem detalhando todo o fluxo, processo e ferramentas envolvidas na LPVS para apoiar as

necessidades mencionadas. Neste cenário de utilização, pode-se observar a rapidez e reusabilidade que os componentes ofereceram na geração dos mecanismos de visualização.

CAPÍTULO 4 - ESTUDO DE OBSERVAÇÃO DA LPVS

4.1 Introdução

Como definido no Capítulo 1, o foco da pesquisa desse trabalho é fornecer uma infraestrutura para a geração rápida de mecanismos de visualização, focados predominantemente em reutilização e em padrões que tragam como benefícios a redução do seu custo de desenvolvimento e melhorem as condições de manutenção dos mesmos. Assim, almeja-se oferecer para usuários de visualizações de software (testadores, programadores, analistas, etc.) uma forma de construí-las rapidamente, não exigindo profundo conhecimento acerca de como são implementadas, entregando-as prontas para uso.

Para avaliar a viabilidade de utilização da abordagem proposta neste trabalho de pesquisa, diversos aspectos precisam ser avaliados, dentre eles: aspectos de desempenho; aspectos de usabilidade; aspectos relacionados a sua real eficácia, verificando se de fato esta abordagem proporciona uma forma intuitiva e rápida para a criação de mecanismos de visualização por meio de reúso.

Este capítulo apresenta um estudo que procurou observar e avaliar aspectos ligados ao desempenho, forma de interação e à eficácia da ferramenta desenvolvida, considerando variáveis como experiência em desenvolvimento de software e em visualização, buscando saber a influência destas na utilização da abordagem.

Baseado nisto, o protótipo da infraestrutura da LPVS descrito no Capítulo 3 foi utilizado em um estudo de observação para avaliar a abordagem LPVS. Participaram deste estudo alunos, mestrandos e graduandos da UFRJ, utilizando a infraestrutura da LPVS para a criação de mecanismos de visualização para o apoio no entendimento de propriedades do desenvolvimento de software, tais como a arquitetura, métricas, processos, testes, etc. Espera-se utilizar o resultado deste estudo para identificar as limitações atuais da abordagem proposta, além de determinar os próximos passos do desenvolvimento da pesquisa.

Nas próximas seções deste capítulo, o estudo realizado é detalhado, descrevendo seus objetivos, discutindo seu planejamento e apresentando os resultados obtidos. A Seção 4.2 apresenta os objetivos gerais do estudo; a Seção 4.3 define com mais detalhes o estudo realizado; a Seção 4.4 explica o procedimento de execução do estudo; a Seção 4.5 apresenta os resultados e as observações obtidas sobre a execução do estudo; a

Seção 4.6 sumariza as observações e comentários feitos pelos participantes sobre o estudo; a Seção 4.7 discute a validade do estudo; e por fim, a Seção 4.8 apresenta as considerações finais da avaliação realizada.

4.2 Objetivos

Visando analisar se o uso da infraestrutura desenvolvida pela abordagem LPVS é capaz de construir visualizações de software com rapidez e eficácia, tanto para usuários experientes como novatos, foi realizado um estudo de observação. Neste estudo, o participante realiza uma tarefa enquanto é observado por um experimentador. Este tipo de estudo tem a finalidade de coletar dados sobre como determinada tarefa é realizada (Shull *et al.*, 2001). Por meio destas informações, pode-se obter uma compreensão de como um novo processo é utilizado.

O objetivo deste estudo de observação segundo a estrutura do GQM (Basili *et al.*, 1994) é **analisar** o uso da infraestrutura da LPVS **com o propósito de** caracterizar, **com respeito à** viabilidade da mesma na criação de mecanismos de visualização de software **do ponto de vista** dos *stakeholders* de visualização **no contexto de** compreensão de atividades ligadas à manutenção de software.

4.3 Definição do Estudo

Este estudo foi realizado em um ambiente acadêmico, no Laboratório de Realidade Virtual e Aumentada (Lab3D) da COPPE/UFRJ. Para participar no estudo, os indivíduos precisavam possuir pelo menos algum conhecimento ou experiência em desenvolvimento de software, uma vez que as tarefas executadas são baseadas no domínio de manutenção de software, dentro do escopo de desenvolvimento de software. Neste contexto, alunos de graduação e pós-graduação se voluntariaram para participar do estudo.

Através desse estudo, esperava-se observar como *stakeholders* de visualização de software interagiriam com a infraestrutura da LPVS para a resolução de tarefas no contexto de suas áreas de atuação. Mais especificamente, tentou-se verificar, qualitativamente se, com a introdução da LPVS, os participantes conseguiriam construir mecanismos de visualização de forma rápida e intuitiva para atender suas necessidades em compreender determinadas propriedades sobre um projeto (é observado apenas o processo de geração do mecanismo de visualização e não a eficácia do mesmo no apoio

da atividade, já observado por outros estudos). Desta forma, com o objetivo de realizar esta avaliação, os seguintes pontos principais foram observados:

- O nível de experiência, tanto no domínio de desenvolvimento como em visualização de software, do participante influi no uso da ferramenta? De qual forma?
- Os participantes foram capazes de gerar visualizações de software que atendessem as tarefas especificadas?
- Quanto tempo levou para que os participantes conseguissem executar as tarefas?
- Quais dificuldades os participantes tiveram e/ou observaram no uso da ferramenta durante as tarefas?

Para poder fazer essas observações, o participante deveria se encontrar no contexto de uma empresa de desenvolvimento de software (a empresa XYZ) e se enquadrar no papel de consultor técnico, apoiando as diversas atividades da empresa que necessitassem de mecanismos de visualização. Para facilitar a imersão do participante no contexto do experimento, em cada tarefa, lhe foi apresentado todo o contexto e a necessidade daquele cenário.

Como o estudo pretendia analisar a execução de cada participante individualmente, os sete participantes foram separados em sessões distintas. Assim, em todas as sessões, foi distribuído um conjunto igual de tarefas que envolviam necessidades de compreensão de alguma propriedade ou processo no desenvolvimento de software. Este conjunto foi dividido em quatro tarefas: “Tarefa 1”, “Tarefa 2", "Tarefa 3" e "Tarefa 4" (Apêndice C). Todo o material utilizado no estudo (inclusive os formulários) foi revisado por especialistas antes da aplicação no experimento.

4.4 Procedimento de Execução

Cada sessão do estudo foi feita individualmente e durou cerca de 50 minutos (não sendo estabelecido um tempo máximo para a mesma). Em cada sessão, inicialmente, cada participante foi informado sobre o experimento através do Formulário de Consentimento (Apêndice A). Caso concordasse em participar, o participante preenchia o Questionário de Caracterização (Apêndice B). Este questionário tem como objetivo poder avaliar o nível de conhecimento e experiência do participante em diferentes temas relacionados ao estudo. Esses dados foram utilizados

para garantir que os participantes estavam aptos a executar o estudo e também para interpretar os resultados obtidos por cada um dos participantes segundo o perfil dos mesmos. O preenchimento dos formulários iniciais levou cerca de 10 minutos.

Em seguida, o participante recebeu um treinamento de aproximadamente 15 minutos sobre os principais conceitos envolvidos e funcionamento geral da abordagem da LPVS. A descrição detalhada das tarefas que deveriam ser executadas durante o estudo foram entregues logo depois (e podem ser acessadas no Apêndice C). O participante recebia uma tarefa por vez, tendo acesso somente a seguinte após a conclusão da anterior. Toda a infraestrutura da LPVS já estava na área de trabalho dos participantes, devidamente instalada, pronta para uso. Foi pedido para que, a cada tarefa, o participante mudasse o local de geração do mecanismo de visualização de forma a facilitar a análise posterior realizada em cima dos produtos gerados.

Foram propostas 4 tarefas, realizadas na mesma ordem e sob as mesmas condições para todos os participantes. Em todas, no final da execução das mesmas, um questionário era respondido. Este questionário (Apêndice D) tem como objetivo poder avaliar a forma que o participante utilizou a ferramenta, se foi possível identificar outras soluções para a tarefa (critério para verificação da flexibilidade) e quais foram as dificuldades e impressões durante o uso da LPVS.

4.5 Resultados e Observações

O estudo foi realizado com 7 participantes, sendo 5 alunos de mestrado e 2 de graduação, 5 alunos de Engenharia de Computação da COPPE e 2 da Escola Politécnica da UFRJ, selecionados por conveniência. Como pode ser observado na Tabela 4.1, todos os participantes possuem um perfil semelhante quanto à experiência em visualização de software, sendo quase nenhum conhecimento até algum tipo de prática básica. Entretanto, quanto à experiência em desenvolvimento de software, a maior parte se encontra numa categoria classificada como "júnior", dado que possuem menos de 2 anos de experiência na indústria, enquanto uma outra parte se encontra na categoria "sênior", com mais de 4 anos de experiência na indústria. No meio destes dois grupos houve um participante com 3 anos de experiência, o que o levou a uma classificação de "pleno".

Tabela 4.1. Questionário de Caracterização, por participante.

	Participantes						
	P1	P2	P3	P4	P5	P6	P7
Formação	Mestrando	Mestrando	Mestrando	Mestrando	Mestrando	Graduando	Graduando
Eclipse	2	3	4	4	4	2	4
Visualização	1	1	0	1	2	0	2
Experiência - Tempo (anos)	3	1,5	5	6	2	1,5	1
Experiência - Ambiente	Indústria	Indústria	Indústria	Indústria	Indústria	Academia	Indústria
Experiência - Classificação	Pleno	Júnior	Sênior	Sênior	Júnior	Júnior	Júnior

A escala de valores para os itens Eclipse e Visualização varia dos níveis 0 até 4: **zero**, representa nenhum conhecimento; **um**, significa ter estudado em curso, aula ou livro; **dois**, descreve a prática em projetos de sala de aula/curso; **três**, o uso em projetos pessoais; **quatro**, uso em projetos na indústria. O gráfico da Figura 4.1 ilustra o perfil dos participantes quanto à experiência em desenvolvimento de software e em visualização de software.

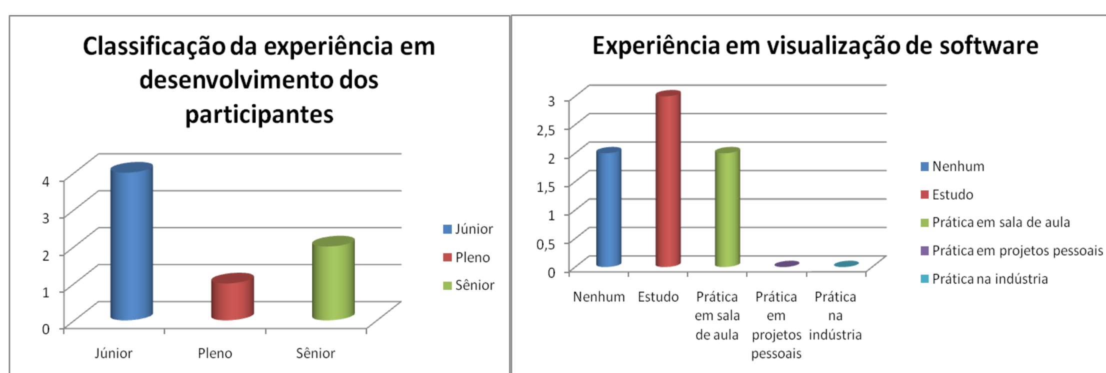


Figura 4.1. Experiência dos participantes em desenvolvimento de software (gráfico da esquerda) e em visualização de software (gráfico da direita).

A caracterização do perfil destes participantes é importante para responder o questionamento acerca da influência destas variáveis durante o uso da LPVS. Por fim, o gráfico da Figura 4.2 mostra que quase a totalidade dos participantes possui uma experiência mesmo que mínima na indústria.

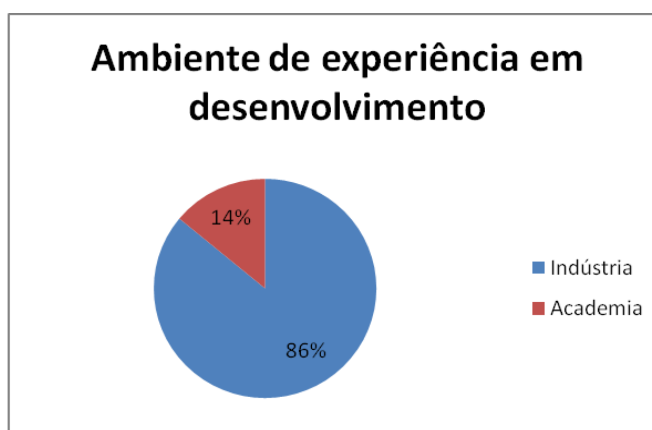


Figura 4.2. Ambiente de experiência em desenvolvimento de software dos participantes.

Após a execução das tarefas solicitadas, cada participante teve que responder a um questionário de observação acerca do uso da LPVS durante a tarefa. Os questionamentos realizados podem ser consultados no Apêndice D. A seguir, uma breve descrição de cada pergunta realizada e os resultados obtidos durante a aplicação do estudo são detalhados.

Primeiramente, foi solicitada a marcação do horário de início e término do participante, além do modo de execução da LPVS utilizado pelo mesmo (expresso ou detalhado). A partir disto, a questão 1 buscava verificar subjetivamente as dificuldades que os participantes tiveram durante a execução da tarefa, desde a compreensão de cada funcionalidade disponibilizada pela LPVS, passando pelo entendimento da tarefa pedida, até o fluxo de ações na ferramenta.

A questão 2 tinha o intuito de inferir se os participantes conseguiram identificar mais de uma solução para a tarefa proposta, dentre as funcionalidades disponibilizadas pela ferramenta. Juntamente com a análise dos produtos gerados, esta questão busca a flexibilidade na forma de se criar mecanismos de visualização na LPVS.

Por fim, houve um espaço para sugestões e críticas gerais dos participantes a cerca da ferramenta e do estudo. Para compreender melhor os resultados divulgados na sequência, é importante notar o conceito de divergência utilizado: (i) divergência positiva: quando o participante selecionou características esperadas para uma possível solução para a tarefa, entretanto, marcou outras a mais que não prejudicavam ou influenciavam na solução da tarefa; divergência negativa: quando faltou a seleção de alguma característica ou parâmetro para a correta resolução da tarefa. Na sequência, são discutidos os resultados e observações sobre cada tarefa aplicada.

4.5.1 Tarefa 1

A tarefa 1 apresentava a necessidade da compreensão, por parte de uma empresa de desenvolvimento de software, da organização do código-fonte de projetos de forma a evitar desvios, erosões e vulnerabilidades nos sistemas. Para esta tarefa, existiam 6 soluções possíveis (denominadas SN, onde N é o índice da solução) e a Tabela 4.2 mostra o resultado obtido após a execução da mesma nos 7 participantes. As soluções apresentadas como gabarito podem ser verificadas no Apêndice E.

Tabela 4.2. Resultado da execução da tarefa 1.

	Participantes						
	P1	P2	P3	P4	P5	P6	P7
Tempo (minutos)	02:44	03:25	03:44	03:00	05:22	03:10	03:23
Houve dificuldades?	N	S	N	N	S	N	N
Identificou mais de 1 solução	S	S	S	N	N	N	N
Modo de Execução	E	E	D	D	D	D	E
Solução Executada	S3	S3	S4	S4	N	S5	S1
Soluções Encontradas	S3;S1;S2	S3;S1;S2;S4;S5;S6	S4;S5;S6	S4	N	S5	S1

Nesta tarefa notou-se a realização de 4 soluções diferentes das 6 possíveis, ao passo que todas as possíveis foram identificadas durante a execução. Observou-se também um tempo médio de execução de 03:32 minutos. Na questão de eficácia, a Tabela 4.3 ilustra as divergências notadas durante esta tarefa (N, nenhuma; DP, divergência positiva; DN, divergência negativa).

Tabela 4.3. Divergências encontradas na tarefa 1.

	Divergências encontradas	
Participantes	**Descrição**	**Tipo de Divergência**
P1	Marcação da característica de Evolução de Métricas além da solução	DP
P2	A solução convergiu exatamente como o gabarito	N
P3	Marcação da característica de Evolução de Métricas além da solução	DP
P4	A solução convergiu exatamente como o gabarito	N
P5	O participante confundiu a hierarquia das características e marcou incorretamente	DN
P6	Marcação da característica de coleta de métricas além da solução	DP
P7	Marcação da característica de Evolução de Métricas além da solução	DP

Ao analisar as divergências encontradas, houve apenas um participante que não conseguiu chegar a uma solução satisfatória (a citar, P5), justificando o seu insucesso pela não percepção da hierarquia entre as diferentes representações visuais, marcando incorretamente. Entre os demais, houve o caso de 4 dos participantes (P1, P3, P6 E P7) selecionarem uma característica a mais relativa a coleta de métricas de software, entendendo que poderia ser útil para a solução da tarefa. Como a mesma só relacionava a necessidade de observação da organização estrutural em pacotes, classes e métodos, acordou-se que a coleta de métricas não se fazia necessário, porém, é justificável a forma de pensar dos participantes que selecionaram a mesma. Os dois participantes restantes (P2 e P4) executaram suas tarefas em total conformidade com o gabarito.

4.5.2 Tarefa 2

A tarefa 2 norteava um cenário de uma equipe responsável por coletar e analisar métricas de software dos projetos da empresa, comparando os valores historicamente,

além de analisá-los instantaneamente no momento da sua ocorrência. Havia a clara especificação que as métricas a serem trabalhadas seriam as de Precisão e Cobertura de arquiteturas de software (Schots *et al.*, 2010), além de que os projetos-alvo da análise seriam obrigatoriamente advindos de repositórios Mercurial. Para esta tarefa, havia apenas uma solução completa e a Tabela 4.4 apresenta o resultado obtido na execução da mesma. O Apêndice E descreve a solução planejada como gabarito. A notação Sn', onde n representa o número da solução, denota uma variação da solução Sn original.

Tabela 4.4. Resultado da execução da tarefa 2.

	Participantes						
	P1	P2	P3	P4	P5	P6	P7
Tempo (minutos)	03:15	02:18	00:50	01:11	01:40	01:55	01:56
Houve dificuldades?	N	N	N	N	N	N	N
Identificou mais de 1 solução	N	N	N	N	N	N	N
Modo de Execução	D	D	D	D	D	D	D
Solução Executada	S1	N	S1	S1	N	S1	S1
Soluções Encontradas	S1	N	S1	S1	N	S1	S1

Nesta tarefa, notou-se a realização da única solução possível. Observou-se também um tempo médio de execução de 01:52 minutos. Na questão de eficácia, a Tabela 4.5 ilustra as divergências notadas durante esta tarefa (N, nenhuma; DP, divergência positiva; DN, divergência negativa).

Tabela 4.5. Divergências encontradas na tarefa 2.

	Divergências encontradas	
Participantes	**Descrição**	**Tipo de Divergência**
P1	Convergiu exatamente para a solução do gabarito	N
P2	Faltou especificar o tipo de Repositório Mercurial requerido pela questão	DN
P3	Convergiu exatamente para a solução do gabarito	N
P4	Convergiu exatamente para a solução do gabarito	N
P5	Faltou marcar a característica para visualização instantânea de métricas	DN
P6	O participante marcou a mais a visualização de redes sociais	DP
P7	Convergiu exatamente para a solução do gabarito	N

Nesta tarefa, dois participantes não conseguiram chegar a uma solução satisfatória (P2 e P5), dado que o primeiro esqueceu-se de parametrizar o repositório a ser utilizado para Mercurial e o segundo de marcar a funcionalidade de visão instantânea de uma métrica. As duas faltas podem ser justificadas por um desvio de atenção durante a leitura da necessidade envolvida na tarefa. Para o participante P6, houve um desvio positivo justificado pelo esquecimento de uma característica a mais marcada, enquanto ainda estava procurando as melhores alternativas. Os outros participantes executaram suas tarefas em conformidade com o gabarito.

4.5.3 Tarefa 3

A tarefa 3 descreveu um cenário relativamente diferente para os participantes, onde envolvia uma demanda de entendimento das relações da forma de trabalho dentro da equipe de desenvolvimento de software, buscando informações para melhoria no processo de trabalho e nas ferramentas de comunicação da equipe. Este típico cenário envolve o conhecido conceito de redes sociais. Para esta tarefa, havia duas soluções possíveis e a Tabela 4.6 apresenta o resultado obtido na execução da mesma.

Tabela 4.6. Resultado da execução da tarefa 3.

	Participantes						
	P1	P2	P3	P4	P5	P6	P7
Tempo (minutos)	02:55	02:10	02:55	02:00	00:32	02:59	01:46
Houve dificuldades?	S	S	N	S	N	N	S
Identificou mais de 1 solução	N	S	N	N	N	N	N
Modo de Execução	D	E	D	D	E	D	E
Solução Executada	S2	S1	S2	S2	S1	S2	S1
Soluções Encontradas	S2	S1;S2	S2	S2	S1	S2	S1

Nesta tarefa, notou-se a realização das duas soluções possíveis. Observou-se também um tempo médio de execução de 02:11 minutos. Na questão de eficácia, a Tabela 4.7 ilustra as divergências notadas durante esta tarefa (N, nenhuma; DP, divergência positiva; DN, divergência negativa).

Tabela 4.7. Divergências encontradas na tarefa 3.

	Divergências encontradas	
Participantes	**Descrição**	**Tipo de Divergência**
P1	Além da característica necessária, o participante adicionou 2 novas visões relacionadas à redes sociais	DP
P2	Além da característica necessária, o participante adicionou 1 nova visão relacionada às redes sociais	DP
P3	Convergiu exatamente para a solução do gabarito	N
P4	Além da característica necessária, o participante adicionou 2 novas visões relacionadas à redes sociais	DP
P5	Além da característica necessária, o participante adicionou 1 nova visão relacionada às redes sociais	DP
P6	Além da característica necessária, o participante adicionou 1 nova visão relacionada às redes sociais e a diagrama de classes	DP
P7	Além da característica necessária, o participante adicionou 1 nova visão relacionada à diagrama de classes	DP

Além disso, todos os participantes conseguiram chegar a uma solução satisfatória, entretanto apenas um (P3) convergiu exatamente para a solução do gabarito. Todos os outros participantes divergiram positivamente, marcando outras funcionalidades relativas a redes sociais que não eram necessárias para a tarefa. Por ser um domínio bem distinto, justifica-se a marcação além do necessário como uma forma de suprir as necessidades quando não se tem um total conhecimento do assunto (e a ferramenta oferece fácil suporte para isso).

4.5.4 Tarefa 4

A tarefa 4 abordou um ambiente de testes como cenário para uma demanda de interpretação dos resultados de testes realizados em projetos de software. Especificando o framework JUnit como o adotado pela empresa em seus projetos, a tarefa explicitou a necessidade da interpretação destes testes de acordo com a organização do software, isto é, sua estrutura de classes e pacotes. Para esta tarefa, existiam quatro soluções possíveis e a Tabela 4.8 apresenta o resultado obtido na execução da mesma.

Tabela 4.8. Resultado da execução da tarefa 4.

	Participantes						
	P1	P2	P3	P4	P5	P6	P7
Tempo (minutos)	02:00	02:23	01:40	00:52	01:55	01:30	01:34
Houve dificuldades?	N	N	N	N	N	N	N
Identificou mais de 1 solução	N	S	S	N	N	N	N
Modo de Execução	D	D	D	D	D	D	D
Solução Executada	S2	S4	S4	N	S2	N	S2
Soluções Encontradas	S2	S4;S3;S2	S4;S3;S2	N	S2	N	S2

Nesta tarefa, notou-se a realização de duas soluções das quatro possíveis, apesar da identificação de três delas (um indicador que a quarta solução possivelmente precisa ficar mais clara para o usuário). Observou-se também um tempo médio de execução de 01:42 minutos. Na questão de eficácia, a Tabela 4.9 ilustra as divergências notadas durante esta tarefa (N, nenhuma; DP, divergência positiva; DN, divergência negativa).

Tabela 4.9. Divergências encontradas na tarefa 4.

	Divergências encontradas	
Participantes	**Descrição**	**Tipo de Divergência**
P1	O participante marcou a característica VCS além da solução	DP
P2	Convergiu exatamente para a solução do gabarito	N
P3	Convergiu exatamente para a solução do gabarito	N
P4	O participante selecionou a visualização de métricas que também é compatível com a análise de testes, porém, não resolveria a questão de aparecer a localização em forma de pacotes e classes	DN
P5	Convergiu exatamente para a solução do gabarito	N
P6	O participante selecionou a visualização de métricas que também é compatível com a análise de testes, porém, não resolveria a questão de aparecer a localização em forma de pacotes e classes	DN
P7	O participante selecionou também a visualização instântanea de métricas, que é compatível com análise de testes, mas não era requerida para a tarefa	DP

Nesta tarefa, dois dos participantes (P4 e P6) não conseguiram chegar a uma solução satisfatória, selecionando a representação visual de métricas como solução. Apesar desta forma visual ser aplicável a testes, bastando imaginar que o número de erros, falhas, sucessos como métricas, não satisfazia a tarefa no sentido de não apresentar segundo a organização estrutural do sistema. Em contrapartida, os participantes P1 e P7, além da solução, acrescentaram essas funcionalidades, cumprindo a necessidade da tarefa. O restante dos participantes seguiu o especificado no gabarito.

4.5.5 Observações Gerais

De forma geral, o uso da infraestrutura da LPVS nestas 4 tarefas, pelos 7 participantes, gerou um resultado satisfatório relatado na Tabela 4.10.

Tabela 4.10. Resultado geral da execução das tarefas.

Razão Sucesso/Falha na execução das tarefas - Geral	Valor(abs)	Valor(%)
Nº Tarefas Executadas	28	100
Sucesso	23	82
Falha	5	18

O percentual de sucesso nas tarefas utilizando a abordagem da LPVS girou em torno de 80% de eficácia (considerando as tarefas que convergiram e divergiram positivamente). Destas tarefas, registrou-se também que a maior parte foi feita com o modo detalhado da LPVS (cerca de 79%). O tempo médio geral da execução das tarefas ficou estabelecido em 02:19 minutos, o que pode ser considerado extremamente rápido

se for considerar abordagens semelhantes as relatadas no Capítulo 2, que envolvem o aprendizado de frameworks ou linguagens específicas além do tempo e destreza para desenvolvimento a partir das mesmas.

Ao se incluir a variável de experiência de desenvolvimento na análise, atenta-se para o relevante fato de que a mesma não teve um grande impacto no tempo de execução das tarefas, isto é, a velocidade que a ferramenta ofereceu para a criação dos mecanismos de visualização foi tão significativa que a diferença de experiência em desenvolvimento não impactou tanto a dimensão tempo. Os participantes do grupo sênior ficaram com um tempo médio de 02:01 minutos enquanto os juniores, com 02:22 minutos.

Em contrapartida, o mesmo tipo de análise, sob o escopo da eficácia, gerou impactos mais significativos elevando a taxa de sucesso para quase 90%, no caso dos seniores, enquanto os juniores tiveram uma redução para 75%. Essa diferença é justificada dado que pelo domínio das tarefas estarem intimamente ligados ao desenvolvimento de software (manutenção de software abrange este escopo), a maior experiência destes participantes fez com que os mesmos tivessem maior certeza e precisão na escolha das características necessárias para atender as tarefas. Isto pode ser um indício de que quanto mais familiar com o domínio que se quer compreender melhor, maior corretude se terá na criação dos mecanismos.

Outro fato observado, ainda em relação a estes perfis de experiência em desenvolvimento de software, foi de que quanto mais experiente, maior tendência o participante tinha a utilizar direto o modo detalhado da LPVS, enquanto os menos experientes dividiam suas atenções entre os modos expresso e detalhado. Cerca de 30% das tarefas executadas pelos juniores foram realizadas por meio do modo expresso, enquanto o grupo dos seniores realizou 100% das tarefas com o modo detalhado. Isto embasa a decisão da criação deste suporte na LPVS para apoiar os usuários menos experientes.

Analisando os mesmos resultados com relação à variável de conhecimento em visualização, não foi possível descobrir algum indício de que a maior experiência neste campo fornece algum ganho ou diferença no uso da ferramenta. Isto aconteceu pela pouca experiência na área citada por todos os participantes.

Um último dado de destaque neste experimento foi o número de divergências. Estas demonstraram que a forma de pensar e a experiência adquirida de cada indivíduo refletem na forma como fazem suas escolhas. Por um lado, a Tabela 4.11 apresenta o

percentual das tarefas segundo a ocorrência de divergências. Cerca de quase 50% das tarefas divergiram positivamente, mostrando, por um lado, certa insegurança quanto ao domínio das tarefas e fazendo com que os participantes aumentassem a sua cobertura criando mecanismos de visualização com mais funcionalidades do que as requeridas.

Tabela 4.11. Percentual de tarefas segundo divergências.

Atributo - Geral	Valor(abs)	Valor(%)
Nº Tarefas Executadas	28	100
Nº Tarefas divergentes Negativamente	5	18
Nº Tarefas divergentes Positivamente	13	46
Nº Tarefas Convergentes ao Gabarito	10	36

Este fato fornece indícios de que a ferramenta se mostra útil para usuários que não possuem certeza sobre como resolver determinada demanda de compreensão, dado que permite a rápida criação de mecanismos englobando inclusive funcionalidades a mais que o necessário, sem impactar no tempo e custo de desenvolvimento para tal. Meyer (2006) considera este tipo de característica como muito importante na criação de visualizações, dado que, normalmente, é na execução do mecanismo que se afere a eficácia do mesmo, portanto, formas simples de se construir rápido estes mecanismos são vitais nesta área. A próxima seção trata dos comentários e sugestões dos participantes sobre a abordagem, revelando possíveis pontos de melhoria e dificuldades que tiveram no uso da ferramenta.

4.6 Avaliação realizada pelos participantes

Depois de concluir cada tarefa do estudo, os participantes preencheram um questionário de avaliação com perguntas referentes a dificuldades, sugestões e críticas durante execução do estudo. Com relação à abordagem LPVS, os participantes se mostraram satisfeitos com o uso da ferramenta como um todo, especialmente, com a facilidade em se gerar os mecanismos de visualização com algumas seleções, tomando como base suas experiências de desenvolvimento, não seriam facilmente desenvolvidos. Alguns comentários desses participantes foram:

- "A Ferramenta é de fácil utilização e as opções são variadas."
- "As dicas (*tooltips*) foram muito importantes no uso da ferramenta. Talvez, a mesma aplicação na tela dos modos expresso e detalhado fosse importante!"

- "O console de erros do wizard guiou muito bem na especificação das características da tarefa."

Entretanto, algumas dificuldades com relação ao domínio de aplicação dos mecanismos também foram levantadas, justificando em parte algumas das divergências encontradas:

- "Fiquei em dúvida com relação a que parâmetro utilizar para o repositório VCS. Não houve especificação na questão, acabando por deixar em branco."
- "Não soube identificar qual das *Collaboration Analysis* atenderia ao problema. Na dúvida, selecionei todas."
- "Dificuldade em definir a análise da estrutura a ser utilizada."

Do ponto de vista de sugestões e críticas à ferramenta e forma de interação, surgiram comentários interessantes, detalhados a seguir:

- "Julguei que utilizando o modo expresso não consegui especificar todas as necessidades". *A ideia de utilizar o modo expresso é justamente simplificar a montagem do produto, deixando para o modo detalhado o poder de especificar mais profundamente*;
- "Fiquei na dúvida sobre as vantagens que a visualização 3D teria sobre a normal e se ofereceria as mesmas oportunidades/funcionalidades que a mesma". *Talvez, uma melhor descrição das duas funcionalidades, inclusive, com imagens que ilustrassem o resultado, fosse mais representativa da diferença*;
- "Foi difícil e tedioso selecionar as características no modo detalhado, preferindo o modo Expresso". *Para demandas que não carecem de especificações tão detalhadas, não é recomendado o uso do mesmo justamente pelo maior esforço;*
- "Poderia colocar mais de um parâmetro em VCSTYPE". *Este é um tipo de demanda que acaba ocorrendo numa Linha de Produtos de Software e, os especialistas nos componentes se responsabilizariam por introduzirem este novo parâmetro;*
- "As características poderiam ter suas dependências marcadas automaticamente". *Esta funcionalidade foi planejada inicialmente, porém, ao se imaginar que poder-se-ia ter características que excluíssem outras, pensou-se que ficaria*

mais claro para o usuário um aviso sobre tal ocorrência do que a simples transformação automática.

4.7 Validade

Este estudo preliminar foi executado a fim de caracterizar o uso controlado da abordagem desenvolvida, verificando a viabilidade da mesma no uso para geração de mecanismos de visualização de software. Os resultados obtidos a partir das observações e da avaliação dos participantes nos ajudaram a entender as limitações da abordagem e os pontos que precisam ser melhorados. No entanto, devido às restrições deste estudo, os resultados obtidos não podem ser generalizados.

A seleção dos participantes foi feita através da solicitação de voluntários dentro de um grupo de alunos que compartilham um mesmo laboratório de pesquisa do qual também fazem parte os experimentadores. Isto foi necessário devido a restrições de tempo e de pessoal disponível. Como consequência, o grupo escolhido pode não ser representativo para a população que se deseja testar e pode ser influenciado pela sua relação com os experimentadores.

Além disso, houve um número reduzido de participantes neste estudo. É possível que os resultados sejam influenciados pelo tamanho e pelas características específicas do grupo. O uso de alunos de pós-graduação, variando de não tão experientes à consideravelmente experientes na indústria, também restringe a generalização das observações obtidas.

Para realizar algumas justificativas na análise dos dados, as informações de caracterização que cada participante forneceu de si mesmo foram utilizadas. Entretanto, não é possível confirmar que tais informações fornecidas estejam corretas. O entendimento dos participantes sobre as questões dos formulários é diretamente influenciado pela forma como as questões foram elaboradas; se a questão tiver sido mal formulada, o estudo pode ser afetado negativamente (Wohlin *et al.*, 1999). A análise dos instrumentos utilizados no estudo (inclusive os formulários) por especialistas visou reduzir esta interferência. No entanto, a partir das análises das respostas, pode-se observar que algumas das tarefas foram interpretadas de maneiras diferentes pelos participantes.

4.8 Considerações Finais

Neste capítulo, foi descrito o estudo realizado para avaliar preliminarmente uma parte da abordagem LPVS. Neste estudo, cada participante executou algumas tarefas pré-determinadas de interação com o Wizard de Visualização, simulando um consultor de uma empresa de desenvolvimento de software que provisiona soluções para as diversas equipes existentes.

Para conseguir simular este cenário, os participantes foram dispostos isoladamente em um ambiente onde não poderiam se comunicar com mais ninguém. A partir desse estudo, foi possível ter indícios de que a abordagem LPVS é uma alternativa real para auxiliar *stakeholders* de visualização de software no processo de construção destes mecanismos. Além disto, com as respostas obtidas e através da própria observação dos participantes ao longo do estudo, foi possível identificar diversos pontos de melhoria e algumas limitações da abordagem.

Em relação aos quatro pontos apresentados no início deste capítulo, obtivemos relativo sucesso na observação do uso LPVS em seu propósito:

- O nível de experiência, tanto no domínio de desenvolvimento como em visualização de software, do participante influi no uso da ferramenta? De qual forma? A experiência em desenvolvimento influenciou no sentido da eficácia dos mecanismos gerados, ao passo que não houve influência significativa no tempo de criação. Com relação à experiência em visualização de software, não houve indícios de impactos significativos;
- Os participantes foram capazes de gerar visualizações de software que atendessem as tarefas especificadas? Na maior parte das tarefas, cerca de 82%, foram criados mecanismos que atendiam às tarefas;
- Quanto tempo levou para que os participantes conseguissem executar as tarefas? O tempo médio para a execução das tarefas foi considerado baixo, levando cerca de 02:19 minutos;
- Quais dificuldades os participantes tiveram e/ou observaram no uso da ferramenta durante as tarefas? Boa parte das dificuldades residira em interpretações das necessidades de cada tarefa, enquanto outras residiram em melhorias na interface do wizard para melhor uso por parte do participante.

Este estudo foi um primeiro passo para a avaliação da abordagem proposta neste trabalho de pesquisa. O estudo é limitado em diferentes aspectos, restringindo a generalização dos resultados obtidos, como discutido na seção anterior. Serão necessários, ainda, estudos adicionais para avaliar plenamente o que foi proposto, especialmente, gerar evidências comparativas com outras abordagens.

CAPÍTULO 5 - CONCLUSÃO

5.1 Epílogo

A compreensão de software, por sua natureza, é uma atividade não trivial. Através do rápido desenvolvimento de novas tecnologias e processos, a construção de sistemas têm envolvido cada vez mais indivíduos e informações de diferentes fontes. Desta forma, para se obter um grau aceitável de qualidade sobre o produto desenvolvido, necessita-se de controle sobre o mesmo e, para isto, precisa-se entendê-lo.

Em geral, o entendimento acaba por se dar através de horas de pesquisa, entrevistas, estudo etc. O problema é que no mundo atual, este precioso tempo nem sempre existe para que haja compreensão por estas formas. Para isto, um campo de pesquisa foi desenvolvido focado em descobrir atributos visuais que, alinhadas a tecnologia, pudessem ampliar a capacidade de raciocínio humano, amplificando o seu entendimento sobre determinado assunto. A este campo, denominou-se visualização de informação.

A partir desta grande área, especializações foram surgindo para tipos distintos de informação. Entre elas, a peculiaridade do software fez nascer a área de visualização de software, voltada para o entendimento das propriedades e processos envolvidos no seu ciclo de vida (Diehl, 2007). Apesar de estudos mostrarem benefícios com o uso de mecanismos desta área, alguns problemas foram identificados, tais como:

- Alta complexidade e custo de desenvolvimento, dado que estes mecanismos envolvem diversas áreas como processamento de dados e computação gráfica;
- Pouca flexibilidade e reúso nos mecanismos existentes, dificultando o reaproveitamento integral/parcial dos mesmos em diferentes cenários;
- Baixa manutenibilidade nos mecanismos existentes na atualidade, o que dificulta a disseminação de uso (Petrillo *et al.*, 2012).

Desta forma, os problemas de falta de reúso, flexibilidade e alto tempo e custo de desenvolvimento emergem como principais fatores de fracasso na construção de mecanismos de visualização de software. Como foi observado, inúmeras abordagens foram criadas com o objetivo de modificar este quadro, fornecendo novas formas de

geração destas ferramentas que minimizam os problemas citados. Porém, muitas dessas abordagens acabam por reduzir a flexibilidade no momento da construção para facilitar o uso da ferramenta, enquanto outras ampliam o poder de configuração e acabam por onerar o usuário final destes mecanismos com as especificidades de implementação e desenvolvimento que não deveriam possuir.

A abordagem LPVS, apresentada nesta dissertação, busca uma solução intermediária, isto é, que tenta unir sob uma única forma tanto a facilidade de uso como o poder de configuração e flexibilidade na geração. A forma como essa união é realizada e representada é o que diferencia este trabalho dos demais. Neste sentido, aplicou-se uma técnica da área de reutilização denominada Linha de Produtos de Software.

Ao combinar o poder de reúso e configuração de LPS com componentes de visualização, a abordagem LPVS se mostrou capaz de prover uma forma intuitiva e rápida para a criação de mecanismos de visualização que atendessem atividades dentro do cenário de manutenção de software.

A seguir, são apresentadas as principais contribuições (Seção 5.2) e limitações (Seção 5.3) deste trabalho de pesquisa, e são também discutidos os principais trabalhos futuros (Seção 5.4).

5.2 Contribuições

Esta dissertação apresentou os resultados de um trabalho de pesquisa que visou propor uma abordagem para a geração de mecanismos de visualização de software e, com isso, uma infraestrutura reutilizável para apoiar *stakeholders* de visualização na criação de mecanismos que auxiliem em atividades de compreensão, apresentando potenciais soluções para os problemas mais comuns na construção desses mecanismos. Entre as principais contribuições deste trabalho, podemos destacar:

- O estudo e a comparação de diferentes abordagens para a construção de mecanismos de visualização de software descritos na literatura, destacando os pontos positivos e negativos mais comuns destas abordagens;
- A definição de uma arquitetura padrão capaz de ser reutilizada e expandida para outros domínios de visualização que não seja o de manutenção de software;
- A análise de domínio de manutenção de software (Silva *et al.*, 2012), levantando e estudando trabalhos relacionados para a modelagem de características passíveis de auxiliarem nas atividades intrínsecas a este domínio;

- A implementação de uma infraestrutura protótipo de um gerador de componentes, baseada na arquitetura definida (EvolTrack), capaz de gerar componentes homogêneos seguindo o padrão definido, acelerando o desenvolvimento e aumentando o reúso;
- A implementação de uma infraestrutura protótipo de um *wizard* de visualização, baseada no ambiente Eclipse, para a seleção, verificação, busca e composição de das diversas características selecionadas pelos usuários (e seus respectivos componentes de implementação);
- A prova de conceito da abordagem, ilustrando desde a fase inicial do surgimento de uma demanda para compreensão de testes unitários, passando pela criação e armazenamento de um novo componente que extraísse dados de resultados de teste do framework JUnit, até o efetivo uso do mesmo durante a criação de um mecanismo de visualização envolvendo a interpretação de resultados testes do JUnit;
- A avaliação preliminar da abordagem, através de um estudo de observação, com alunos de pós-graduação e graduação, analisando a execução de determinadas tarefas de manutenção de software carentes da criação de novos mecanismos de visualização de software para apoiá-las.

5.3 Limitações

Fazendo uma análise crítica da abordagem proposta e da infraestrutura, é possível perceber as limitações deste trabalho. As principais são listadas a seguir:

- Todos a infraestrutura da LPVS é dependente do ambiente de desenvolvimento Eclipse. Desde o gerador de componentes, passando pelos componentes gerados, até o wizard de visualização, são projetados especialmente para funcionarem dentro da IDE;
- Atualmente, o gerador de componentes trabalha somente com componentes para a representação visual e a coleta de dados. Uma futura evolução para componentes de processamento faz-se necessária para a completa realização da arquitetura;
- A avaliação da abordagem foi limitada, sendo restrita a apenas um estudo de observação de *stakeholders* de desenvolvimento de software. Para que exista uma evidência significativa do real ganho no uso desse tipo de abordagem e se a mesma possui escalabilidade para tratar de cenários reais, estudos adicionais

deverão ser elaborados tanto para efeitos de comparação como para validação em cenários mais complexos.

5.4 Trabalhos Futuros

A partir do que foi estudado e desenvolvido ao longo deste trabalho, é possível identificar possibilidades de melhoria e novas opções para se expandir a abordagem e a infraestrutura apresentada. Entre esses possíveis trabalhos futuros, encontram-se:

- Um estudo experimental com uma população relevante para validar o ganho real do uso da LPVS em relação às outras abordagens citadas no Capítulo 2;
- A extensão da infraestrutura por meio da criação de novos componentes que implementem novas funcionalidades, abrangendo novos domínios como a área de requisitos, gerenciamento de projetos etc.;
- Implementar itens que ofereçam maior clareza, para os usuários da LPVS, a cerca das características oferecidas, tais como imagens ilustrativas das visualizações disponíveis;
- No *wizard* de visualização, evoluir as formas de interação do usuário com a infraestrutura de forma a facilitar e tornar mais intuitivo o seu uso, por exemplo, criando uma opção de marcação de dependências automática para os usuários que quisessem assumir tal risco;
- Avaliar a abordagem proposta em um ambiente de aprendizado, como, por exemplo, em uma disciplina em que trabalhos de criação de mecanismos de visualização fossem realizados. Desta forma, poderia ser avaliado o potencial de disseminação dos conceitos de visualização de software através de uma infraestrutura que facilite a construção destes mecanismos;
- Estudar uma nova arquitetura dos mecanismos de visualização de forma a separar os recursos de interatividade dos demais componentes;
- Buscar formas e estratégias para auxiliar o *stakeholder* de visualização no momento da seleção das funcionalidades do mecanismos que ele deseja;
- Estudar os benefícios do uso de diferentes visões e metáforas simultaneamente para a execução de uma determinada atividade.

REFERÊNCIAS

Ali Babar, M., L. Chen, e F. Shu. “Managing variability in software product lines.” *IEEE Software*, 2010: 27:89–91, 94.

Anslow, C., J., J. Noble, S. Marshall, e E. Tempero. “Towards end-user web software visualization.” *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2008).* Pittsburgh, PA, USA, 2008, Sept. pp.256-257.

Bachmann, F., e P.C. Clements. *Variability in software product lines.* Pittsburgh, USA: Software Engineering Institute, 2005, (September):46.

Basili, V., G. Caldiera, e H. Rombach. “Goal Question Metric Paradigm.” *Encyclopedia of Software Engineering.* edited by John J. Marciniak, John Wiley & Sons, 1994. v. 1, pp. 528-532.

Bassil, S., e R. K. Keller. “Software visualization tools: survey and analysis.” *In Proceedings of 9th International Workshop on Program Comprehension (IWPC 2001).* Toronto, Canada: IEEE Computer Society, 2001. pp. 7-17.

Blois, A., R Oliveira, N. Maia, C.M.L. Werner, e K. Becker. “Variability Modeling in a Component-based Domain Engineering Process.” *9th International Conference on Software Reuse, Lecture Notes in Computer Science.* Turin, Italy: Springer, Heidelberg, Germany, 2006. pp. 395 - 398.

Bull, R.I. *Model Driven Visualization: Towards a Model Driven Engineering Approach for Information Visualization.* Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy, Victoria, Canada: Department of Computer Science of the University of Victoria, 2008.

Campo, M.R., e R.T. Price. “Luthier - A Framework for Building Program Analysis Tools.” In: *Object-Oriented Aplication Frameworks*, por Mohamed Fayad and Ralph Johnson. Wiley, Usa, 1998.

Carlsson, J. *Optimisation of a Graph Visualization Tool: Vizz3D.* Reports from MSI, Växjö University, SE-351 95 VÄXJÖ,: School of Mathematics and Systems Engineering, 2006, 50p.

Chen, C. *Information Visualization.* 316p, Springer, 2006.

Clements, P., e L. Northrop. *Software Product Lines: Practices and Patterns.* Boston, MA: Addison-Wesley in the SEI Series, 2002.

Cottam, J.A., J. Hursey, e A. Lumsda. “Representing unit test data for large scale software development.” *In Proceedings of the 4th ACM symposium on Software visualization (SoftVis '08).* New York, NY, USA: ACM, 2008. pp. 57 - 66.

Czarnecki, K., e U. W. Eisenecker. *Generative programming: methods,tools, and applications.* New York,NY, USA: ACM Press/Addison-Wesley Publishing Co, 2000.

D'Ambros, M., M. Lanza, e M Pinzger. “"A Bug's Life" Visualizing a Bug Database.” *Visualizing Software for Understanding and Analysis, VISSOFT 2007. 4th IEEE International Workshop on.* 24-25 June, 2007. pp.113-120.

Diehl, S. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software.* 1 ed. Springer, 2007.

Fischer, M., M. Pinzger, e H. Gall. “Analyzing and relating bug report data for feature tracking.” *In Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003).* 13-16 Nov, 2003. pp. 90- 99.

Gershon, N. D. “From perception to visualization.” In: *Scientific Visualization: Advances and Challenges*, por L. Rosenblum, et al. Academic Press, 1994.

IEEE. “Ieee recommended practice for architectural description of software intensive systems.” Technical Report, 2000.

Koschke, R. “Software Visualization for Reverse Engineering.” *Revised Lectures on Software Visualization, International Seminar*, 2002: pp. 138-150.

Magdaleno, A. M., C. Werner, e R. Araújo. “Analyzing Collaboration in Software Development Processes through Social Networks.” *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA).* Heraklion, Crete, Greece, 2010. p. 435-446.

Martin, R. K., C. A. Scheraga, J. R. Moran, e S. Nagrath. “Global information systems: factors or fiction,.” *Innovation in Technology Management - The Key to Global Leadership. PICMET '97: Portland International Conference on Management and Technology.* Portland, OR, 1997. 739p.

Mayrhauser, A. Von, e A. M. Vans. “Program comprehension during software maintenance and evolution.” *Compute*, 1995, Aug: vol.28, no.8, pp.44-55.

Meyer, M. *Scripting Interactive Visualizations.* Masterarbeit der Philosophisch naturwissenschaftlichen Fakult¨at, Institut f¨ur Informatik und angewandte Mathematik, 2006.

Oliveira, M. S. *PREViA: Uma Abordagem para a Visualização da Evolução de Modelos de Software.* Dissertação de M. Sc., COPPE/UFRJ, 2011.

Pereira, A., P. Silveira, V Garcia, e P. Muniz. “Linhas de Produtos de Software: Uma tendência da indústria.” In: *ERCEMAPI 2011 - Livro Texto dos Minicursos*, pp. 7-31. Teresina, 07 e 08 de novembro de 2011: Sociedade Brasileira de Computação, 2011.

Petrillo, F., M. Pimenta, e C. Dal Sasso. “O Estado-da-Arte das Ferramentas de Visualização de Software.” *5ª edição da Conferência Iberoamericana de Engenharia de Software (CIbSE 2012).* Buenos Aires, 2012. 14p.

Pohl, K., G. Bockle, e F.J. van der Linden. *Software Product Line Engineering. Foundations, Principles and Techniques.* 2005.

Reuse. *Odyssey Project.* http://reuse.cos.ufrj.br/odyssey. (acesso em 29 de setembro de 2012).

Schafer, T., e M. Mezini. “Towards More Flexibility in Software Visualization Tools.” *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2005).* Williamsburg, VA, USA, 2005. pp.1-6.

Schmid, K., F. J. v. d. Linden, e E. Rommes. *Software Product Lines in Action:TheBest Industrial Practice in Product Line Engineering.* Springer-Verlag, 2007.

Schots, M., M. Silva, L. Murta, e C. Werner. “Verificação de aderência entre arquiteturas conceituais e emergentes utilizando a abordagem PREViA.” *VII Workshop de Manutenção de Software Moderna.* Belém, Brasil., 2010.

Sharafi, Z. “A Systematic Analysis of Software Architecture Visualization Techniques.” *IEEE 19th International Conference on Program Comprehension (ICPC 2011).* Kingston, Ontario, Canada, 2011. pp. 254-257.

Shull, F., J. Carver, e G. H. Travassos. “An empirical methodology for introducing software processes.” *ACM SIGSOFT Software Engineering Aote.* New York, NY, USA, 2001. pp. 288 - 296.

Silva, M., M. Schots, e C. Werner. “ Supporting Software Maintenance Activities through a Software Visualization Product Line Infrastructure.” *IX Workshop de Manutenção de Software Moderna.* Fortaleza, Brasil, 2012. 5p.

Storey, M.-A., C. Bennett, R.I. Bull, e D. German. “Remixing visualization to support collaboration in software maintenance.” *Frontiers of Software Maintenance. FoSM 2008.* Sept. 28 - Oct. 4, 2008. pp.139-148.

Trumper, J., J. Bohnet, S. Voigt, e J. Dollner. “Visualization of Multithreaded Behavior to Facilitate Maintenance of Complex Software Systems.” *In Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology (QUATIC '10).* Porto, Portugal: IEEE Computer Society, 2010. pp. 325-330.

Voinea, L., A. Telea, e J. J. Van Wijk. “CVSscan: visualization of code evolution.” *In: Proceedings of the 2005 ACM symposium on Software visualization.* New York, NY, USA, 2005. p. 47–56.

Voinea, L., e A. Telea. “Visualizing Debugging Activity in Source Code Repositories.” *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007).* 24-25 June, 2007. pp.156-157.

Wall, L., T. Christiansen, e J. Orwant. *Programming Perl.* 3 ed. O’ReillyMedia, Inc, 2000.

Weiss, D.M., e C.T.R. Lai. *Software Product-Line Engineering – A Family-Based Software Development Process.* Reading, Massachusetts: Addison-Wesley, 1999.

Werner, C., et al. “EvolTrack: A Plug-in-Based Infrastructure for Visualizing Software Evolution.” *In 1st Brazilian Workshop on Software Visualization.* Brazil, 2011. pp. 1-8.

Wohlin, C., P. Runeson, e M. Host. “Experimentation in Software Engineering: An Introduction.” 1 ed. Springer, 1999. 236p.

Xiong, C.J., Y.F. Li, M. Xie, S.H. Ng, e T.N. Goh. “A model of open source software maintenance activities.” *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 2009).* 8-11 Dec, 2009. pp.267-271.

APÊNDICE A – FORMULÁRIO DE CONSENTIMENTO

Formulário de Consentimento

Estudo

Este estudo visa a caracterizar a viabilidade do uso da infraestrutura LPVS em gerar mecanismos de visualização de software para *stakeholders* de manutenção de software.

Idade

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por Marlon Alves da Silva na Universidade Federal do Rio de Janeiro.

Procedimento

Este estudo acontecerá em uma única sessão, composta de quatro tarefas. Em todas as tarefas, os participantes deverão executar um conjunto de passos e responder a uma série de questionamentos relacionados as atividades realizadas. Eu entendo que, uma vez o experimento tenha terminado, os trabalhos que desenvolvi serão estudados visando a entender a eficiência dos procedimentos e as técnicas propostas.

Confidencialidade

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e apresentado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e ferramentas para a visualização de software.

Pesquisador responsável

Marlon Alves da Silva
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Pesquisador colaborador

Marcelo Schots de Oliveira
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Professor responsável

Profª. Cláudia Maria Lima Werner
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Nome (em letra de forma):____
Assinatura:____________________________________ **Data:**_______________

APÊNDICE B - QUESTIONÁRIO DE CARACTERIZAÇÃO

Questionário de Caracterização

1) Formação acadêmica

( ) Doutorado ( ) Doutorando
( ) Mestrado ( ) Mestrando
( ) Graduação ( ) Graduando

Ano de ingresso: _________ Ano de conclusão (ou previsão de conclusão): ________

2) Formação geral

2.1) Por favor, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:

0 = nenhum
1 = estudei em aula ou em livro
2 = pratiquei em projetos em sala de aula
3 = usei em projetos pessoais
4 = usei em projetos na indústria

2.1.1) Ambiente de desenvolvimento Eclipse	0	1	2	3	4
2.1.2) Visualização de Software	0	1	2	3	4

2.2) Qual é a sua experiência com desenvolvimento de software?
(marque aquele item que melhor se aplica)
( ) já li material sobre desenvolvimento de software.
( ) já participei de um curso sobre desenvolvimento de software.
( ) nunca desenvolvi software.
( ) tenho desenvolvido software para uso próprio.
( ) tenho desenvolvido software como parte de uma equipe, relacionada a um curso.
( ) tenho desenvolvido software como parte de uma equipe, na indústria.

2.3) Por favor, explique sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em desenvolvimento de software. (Ex.: “Eu trabalhei por 2 anos como programador de software na indústria”)

____
____
____

Obrigado por sua colaboração!

APÊNDICE C – TAREFAS DO ESTUDO

Tarefa 1

Descrição:

A empresa XYZ, especialista em desenvolvimento de sistemas, busca melhorar a qualidade dos softwares por ela construídos. Para tal, criou um programa de melhoria da qualidade onde uma das atividades seria designar um profissional para acompanhar a evolução do código-fonte do projeto, de forma a evitar desvios, erosões e vulnerabilidades no novo produto.

Neste sentido, este profissional precisa de apoio ferramental para facilitar o seu entendimento da estrutura dos novos sistemas. Sabe-se que todos estes sistemas ficam em repositórios de controle de versão como SVN e Mercurial. Você foi escolhido para assessorar este novo profissional nesta nova atividade do programa de melhoria de qualidade, por meio da construção de um mecanismo de visualização de software que apóie a atividade de acompanhamento da organização do código-fonte, especialmente, classes e pacotes do novo software. Vale atentar que o profissional que utilizará este novo mecanismo necessita de informações das classes, métodos e pacotes para o seu trabalho.

Utilizando a LPVS, pede-se que construa um mecanismo para atender a necessidade exposta acima e responda as perguntas a seguir. (Existe mais de uma possibilidade de produto para ser gerado na LPVS que atenda o problema)

Tarefa 2

Descrição:

A empresa XYZ, especialista em desenvolvimento de sistemas, busca melhorar a qualidade dos softwares por ela construídos. Para tal, criou um programa de melhoria da qualidade onde uma das atividades seria designar uma equipe para a coleta e monitoramento de métricas de software.

Atualmente, esta equipe trabalhará apenas em cima de duas métricas: Precisão (***Precision***) e Cobertura (***Recall***) (Schots *et al.*, 2010). Ambas são métricas numéricas que avaliam a aderência arquitetural de um projeto de software à um modelo de referência. Neste sentido, este profissional precisa de apoio ferramental para facilitar a sua compreensão da evolução destas métricas tanto de forma comparativa, ao longo da história, como de forma instantânea, isto é, em valores absolutos num determinado momento.

Além disto, estas métricas devem ser coletadas diretamente dos repositórios de controle de versão da empresa, em particular, deverá ser específico aos repositórios **Mercurial** dado que a diretoria de empresa não autorizou este tipo de atividade nos outros repositórios por razão de segurança da informação. Você foi escolhido para assessorar esta equipe nesta nova atividade do programa de melhoria de qualidade, por meio da construção de um mecanismo de visualização de software que apóie a atividade de coleta e monitoramento das métricas citadas.

Utilizando a LPVS, pede-se que construa um mecanismo para atender a necessidade exposta acima e responda as perguntas a seguir.

Schots, M., M. Silva, L. Murta, e C. Werner. ". Verificação de aderência entre arquiteturas conceituais e emergentes utilizando a abordagem PREViA." *VII Workshop de Manutenção de Software Moderna (WMSWM).* Belém, Brasil, 2010.

Tarefa 3

Descrição:

A empresa XYZ, especialista em desenvolvimento de sistemas, busca melhorar a qualidade dos softwares por ela construídos. Para tal, criou um programa de melhoria da qualidade onde uma das atividades seria designar uma equipe para o estudo da rede social gerada na equipe de desenvolvimento durante a construção do sistema.

O intuito desta atividade é absorver as formas de interação e colaboração entre as pessoas e o trabalho técnico vislumbrando melhorias no processo e nas ferramentas utilizadas. Neste sentido, este profissional precisa de apoio ferramental para facilitar o levantamento e compreensão destas redes geradas durante o desenvolvimento.

Vale ressaltar que, para a equipe, o interesse reside em redes que associam as pessoas e os artefatos de desenvolvimento em conjunto. Você foi escolhido para assessorar esta equipe nesta nova atividade do programa de melhoria de qualidade, por meio da construção de um mecanismo de visualização de software que apóie a atividade de estudo das redes sociais no desenvolvimento.

Utilizando a LPVS, pede-se que construa um mecanismo para atender a necessidade exposta acima e responda as perguntas a seguir. (Existe mais de uma possibilidade de produto para ser gerado na LPVS que atenda o problema)

Tarefa 4

Descrição:

A empresa XYZ, especialista em desenvolvimento de sistemas, busca melhorar a qualidade dos softwares por ela construídos. Para tal, criou um programa de melhoria da qualidade onde uma das atividades seria designar uma equipe para a análise dos resultados dos testes unitários executados durante o desenvolvimento.

O objetivo desta atividade é identificar e interpretar os erros e falhas ao longo do desenvolvimento para dirigir os esforços de manutenção corretamente. Neste sentido, este profissional precisa de apoio ferramental para facilitar a análise e interpretação do resultado destes testes durante o desenvolvimento.

Vale ressaltar que esta atividade estará voltada para os resultados de teste unitários (disseminados em todas as equipes de desenvolvimento) gerados pelo framework JUnit (padrão da empresa). Além disso, é importante que estes resultados sejam interpretados segundo a organização do próprio software, isto é, as classes e pacotes, para facilitar a localização do ponto crítico.Você foi escolhido para assessorar esta equipe nesta nova atividade do programa de melhoria de qualidade, por meio da construção de um mecanismo de visualização de software que apóie a atividade de análise dos resultados dos testes unitários executados durante o desenvolvimento.

Utilizando a LPVS, pede-se que construa um mecanismo para atender a necessidade exposta acima e responda as perguntas a seguir. (Existe mais de uma possibilidade de produto para ser gerado na LPVS que atenda o problema)

APÊNDICE D – QUESTIONÁRIO DE OBSERVAÇÃO

Horário de Início:
Horário de Término:

Qual o Modo de Funcionamento do Wizard utilizado?
() Expresso () Detalhado

1) Você sentiu dificuldades em utilizar a LPVS nesta tarefa? Especifique.

____
____
____

2) Você identificou mais de uma solução possível que atendesse ao problema? Especifique.

____
____
____

Observações gerais (sugestões e críticas sobre o uso da ferramenta):

____
____
____
____

APÊNDICE E – GABARITO DAS TAREFAS

Tarefa 1:

Solução 1 (Modo expresso)
Hierarquia que Leva a UML Class Diagram Analysis

Solução 2 (Modo expresso)
Hierarquia que Leva a 3D Structure Analysis

Solução 3 (Mode expresso)
Combinação das soluções 1 e 2

Solução 4 (Modo detalhado)
VCS - Source Code Extractor - Log History Extractor - UML Class Diagram

Solução 5 (Modo detalhado)
VCS - Source Code Extractor - Log History Extractor - 3D City Metaphore

Solução 6 (Modo detalhado)
Combinação das soluções 4 e 5

Tarefa 2:

Solução 1 (Modo detalhado)
VCS - Source Code Extractor - Log History Extractor - Collect Precision/Recall/DifferenceKing metrics - Comparinson Metric Graph - Instant Speedometer Graph - Parametrização de VCS Type com valor = "mercurial"

Tarefa 3:

Solução 1 (Modo Expresso)
Hierarquia que Leva a Social Technical network Analysis

Solução 2 (Modo detalhado)
VCS - Source Code Extractor - Log History Extractor - Collect Social Network metrics - Socio-Techincal Network Graph

Tarefa 4:

Solução 1 (Modo Expresso)
Hierarquia que Leva a Test Result Analysis

Solução 2 (Modo detalhado)
JUnit - Test Result extractor - UML Class Diagram

Solução 3(Modo detalhado)
JUnit - Test Result extractor - 3D City Metaphore

Solução 4 (Modo detalhado)
Combinação das soluções 2 e 3