IAVEMS: INFRAESTRUTURA DE APOIO À VISUALIZAÇÃO DA EVOLUÇÃO DE MÉTRICAS DE SOFTWARE

Marlon Alves da Silva

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____
Marlon Alves da Silva

Aprovado por:

_____
Profª. Cláudia Maria Lima Werner, D.Sc.
(Presidente)

_____
Prof. Marcelo Schots de Oliveira, B.Sc.
(Co-orientador)

_____
Profª. Adriana Santarosa Vivacqua, D.Sc.

_____
Prof. Leonardo Gresta Paulino Murta, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
JULHO DE 2010

Agradecimentos

A Deus por me guiar, me sustentar e me capacitar.

À minha família, em especial, a minha mãe Vera Lúcia, que não está mais presente, mas participou de tudo o que sou hoje.

À Dayana, pela eterna companhia, carinho e preocupação nesta caminhada.

À Profª Cláudia Werner e ao Marcelo Schots, que acreditaram no meu trabalho e dedicaram seu tempo e esforço junto comigo.

Ao Grupo de Reutilização de Software da COPPE/UFRJ (REUSE), onde recebi todo o apoio necessário para o meu desenvolvimento como pessoa e pesquisador.

A todos os colegas e professores da graduação, que me proporcionaram conhecimento, oportunidades e auxílio nos desafios destes anos.

Aos meus grandes amigos, em geral, que sempre estiveram comigo e que sempre acreditaram em mim.

Aos membros desta banca, Adriana Vivacqua e Leonardo Murta, por terem prontamente aceitado o convite de participar desta defesa de projeto final de graduação.

RESUMO

IAVEMS: INFRAESTRUTURA DE APOIO À VISUALIZAÇÃO DA EVOLUÇÃO DE MÉTRICAS DE SOFTWARE

Marlon Alves da Silva

Orientadores: Cláudia Maria Lima Werner e Marcelo Schots de Oliveira

A tecnologia da informação está sendo disseminada cada vez mais em uma ampla variedade de áreas de aplicação, frequentemente produzindo sistemas maiores em tamanho e complexidade. Ao mesmo tempo, mudanças ocorrem constantemente no desenvolvimento de um software devido a alterações de requisitos; logo, em função de sua penetração no cotidiano das pessoas e das frequentes alterações de especificação, a manutenção e o controle da evolução de software tornam-se vitais para a difusão e permanência desta tecnologia. Desta forma, para que o desenvolvimento de software obtenha sucesso, faz-se necessário que as mudanças ocorridas durante o processo de construção sejam devidamente acompanhadas. Diversas ferramentas apontam métricas para a qualidade de sistemas, porém, benefícios maiores podem ser alcançados por meio da análise da evolução das mesmas e da aplicação de técnicas de visualização que estimulem a capacidade cognitiva humana para uma melhor compreensão do valor informacional destas métricas. Nesse sentido, o presente trabalho apresenta uma infraestrutura de apoio à visualização da evolução de métricas de software, que foi incorporada à ferramenta de evolução de software EvolTrack.

ABSTRACT

IAVEMS: INFRASTRUCTURE TO SUPPORT THE VISUALIZATION OF THE EVOLUTION OF SOFTWARE METRICS

Marlon Alves da Silva

Supervisors: Cláudia Maria Lima Werner and Marcelo Schots de Oliveira

Information technology is increasingly being spread in a wide variety of application areas, often producing larger systems in size and complexity. Simultaneously, changes constantly occur in the software development due to requirement modifications; therefore, due to its broad permeation in daily life and frequent changes in specification, maintenance activities and the software evolution control becomes vital to the spread and permanence of this technology. Thus, for the success of software development, changes that occur during the software construction process must be properly monitored. Several tools present metrics for system quality, however, major benefits can be achieved by analyzing their evolution and applying visualization techniques that stimulate human cognition for a better understanding of the informational value of these metrics. This monograph presents an infrastructure to support the visualization of the evolution of software metrics, which has been incorporated into the EvolTrack software evolution tool.

Sumário

Sumário de Figuras

Sumário de Tabelas

Capítulo 1. Introdução

1.1. Motivação

A tecnologia da informação tem sido utilizada cada vez mais em uma variedade de domínios e, com isso, seu correto funcionamento torna-se essencial para o sucesso do negócio envolvido e para a segurança do ser humano (ISO, 2010). Desta forma, as atividades de manutenção e controle da evolução de software emergem como duas das principais áreas do ciclo de vida de um sistema computacional (PRESSMAN, 2006). Entretanto, a natureza intangível e invisível do software, alinhado com seu crescente grau de complexidade, torna tais atividades uma tarefa de difícil gerenciamento e execução (BALL & EICK, 1996).

A utilização de métricas de software tem o intuito de avaliar a qualidade do mesmo por meio da medição de propriedades e atributos, contribuindo para a atividade de manutenção citada anteriormente. Algumas das principais características para a aceitação de um software (tais como flexibilidade, complexidade e manutenibilidade) podem ser analisadas a partir de métricas de software, segundo alguns estudos (HENDERSON-SELLERS, 1996; SATO *et al.*, 2007), o que motiva o uso das mesmas como meio de auxílio na gestão de projetos de sistemas.

Entretanto, a análise destas métricas, muitas vezes, depende da experiência de especialistas devido à forma como são apresentadas, sendo normalmente valores numéricos isolados. Neste sentido, as técnicas de visualização de software podem ser utilizadas para proporcionar um meio pelo qual os engenheiros de software possam interpretar, compreender e identificar os diferentes artefatos de um software, inclusive métricas, auxiliando o processo de desenvolvimento de todas as fases do ciclo.

Um dos objetivos da visualização é transferir informação do sistema visual humano para o cérebro a partir de imagens. Assim, este processo envolve a interação entre máquinas e seres humanos, explorando nestes últimos a percepção. Segundo DIEHL (2007), 75% de toda informação do mundo real é visualmente percebida, somente 13% é percebido pelo sistema auditivo e os 12% restantes, pelos outros sistemas. Com isso, uma das vantagens do emprego de técnicas de visualização é aproveitar a habilidade do ser humano no reconhecimento de padrões e estruturas em informações visuais (ROBERTSON *et al.*, 1993).

A partir da compreensão da estrutura, do funcionamento e da evolução dos artefatos produzidos no desenvolvimento de software, a atividade de manutenção de software pode se

tornar uma tarefa de menor complexidade. Desta forma, observa-se que a construção de mecanismos que explorem a capacidade perceptiva humana, a partir de técnicas de visualização de software, pode agregar grande valor no contexto do processo de desenvolvimento de software.

Este capítulo de introdução está organizado da seguinte forma: a Seção 1.2 apresenta a ferramenta de evolução de software EvolTrack, contexto do presente trabalho; a Seção 1.3 descreve os objetivos desta monografia; a Seção 1.4 mostra a organização do texto.

1.2. A ferramenta EvolTrack

A ferramenta EvolTrack (CEPEDA, 2008) é um mecanismo capaz de resgatar o ciclo de evolução de um projeto de software, esteja este em desenvolvimento ou já finalizado. A partir das informações obtidas de repositórios de fonte de dados, ela proporciona uma forma de visualização temporal da evolução do projeto, mais especificamente, da sua estrutura. Assim, o modelo do projeto, juntamente com os elementos que o compõem e seus relacionamentos, é visualizado ao longo do ciclo de vida do software.

Basicamente, o EvolTrack extrai periodicamente informações de um projeto a partir de uma fonte de dados configurada e, após o processamento e a transformação destas informações, apresenta a estrutura do projeto de software correspondente àquele período. Para representar esta estrutura do projeto, a ferramenta utiliza o diagrama de classes da UML (OMG, 2010a).

Sua arquitetura (exibida na Figura 1) divide estas tarefas em quatro 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, 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 UML marcado, pois, normalmente aplicam-se estereótipos e valores etiquetados de acordo com um perfil UML especificado. Cada estereótipo ou valor etiquetado representa uma métrica ou medida 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);

- **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;
- **Conector de Visualização** é responsável por transformar as informações de projeto ao longo do tempo em uma ou mais abstrações visuais que facilitem a compreensão por parte do usuário.

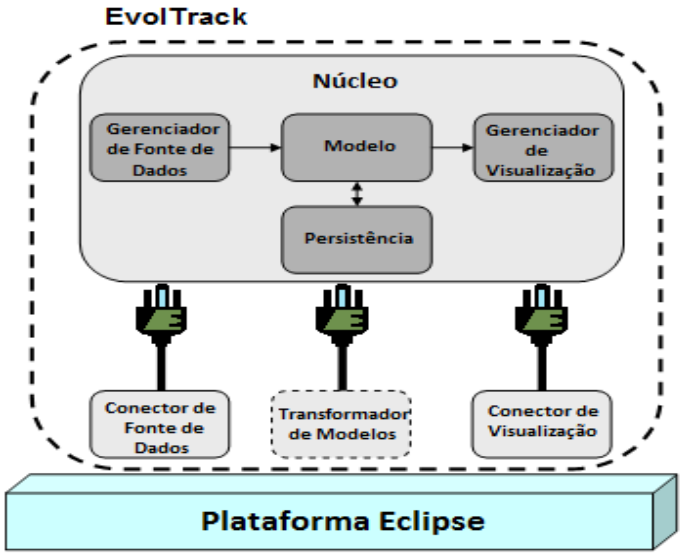


Figura 1. Arquitetura do EvolTrack [Traduzido: (CEPEDA *et al.*, 2010)]

A Figura 2 mostra o fluxo de informação dentro da ferramenta EvolTrack, abrangendo as interações e relacionamentos entre os componentes acima descritos.

Figura 2. Fluxo de Informação [Traduzido: (CEPEDA *et al.*, 2010)]

Observou-se no EvolTrack a necessidade de mecanismos que integrassem a captura de informação dos projetos de software, já que havia a obrigatoriedade da criação de um conector de fonte de dados para cada SCV. Além disso, havia apenas um conector de visualização implementado, que exibia apenas características da estrutura do software ao longo do tempo, não explorando a visualização da evolução de outras informações do projeto. Devido às características mencionadas, especialmente por sua arquitetura flexível e por permitir o acompanhamento da evolução de sistemas de software, percebeu-se que o EvolTrack poderia proporcionar uma melhor estrutura para a análise e compreensão do desenvolvimento de software se houvesse uma infraestrutura voltada para as necessidades mencionadas anteriormente.

1.3. Objetivos

Em face ao exposto na seção 1.1, este trabalho visa incorporar à ferramenta EvolTrack mecanismos que possam primeiramente capturar informações de projetos de software em diferentes sistemas controladores de versão (Fontes de Dados) devido a grande utilização deste tipo de sistema no desenvolvimento de software na atualidade. Além disso, a partir do acréscimo da informação de métricas por meio dos Transformadores de Modelos, estes mecanismos deverão oferecer Visualizadores da evolução destas medidas vislumbrando técnicas de visualização de software que aumentem a percepção e compreensão do processo de desenvolvimento.

Esta infraestrutura visa ser independente da métrica coletada, garantindo a reutilização dos componentes construídos (em especial, os Visualizadores) em variados cenários de utilização.

1.4. Organização do Texto

Esta monografia está organizada em 5 capítulos, além deste capítulo de introdução. O segundo capítulo define métricas de software, suas propriedades, sua participação na qualidade de software e os processos envolvidos. Adicionalmente, uma taxonomia da literatura para métricas é descrita e exemplos de cada classe são dados (em especial aquelas que serão utilizadas como prova de conceito deste trabalho); o terceiro capítulo apresenta a

área de visualização de software, sua história e algumas das técnicas descritas na literatura, com seus benefícios e utilização no processo de desenvolvimento de software, visando atender o objetivo aqui descrito; o quarto capítulo contém a comparação de um conjunto de ferramentas pautadas na visualização e em métricas de software, para que suas contribuições, bem como algumas de suas necessidades e deficiências, sejam identificadas e possam ser tratadas pelo presente trabalho; no quinto capítulo, a abordagem deste trabalho é discutida e apresentada junto aos principais requisitos, sendo descrita a implementação da infraestrutura; neste mesmo capítulo, para demonstrar a aplicação dos mecanismos construídos, é descrito um exemplo de utilização, baseado em um cenário de desenvolvimento de um projeto *open-source*, de forma a apresentar as funcionalidades e propósitos da infraestrutura desenvolvida; no sexto capítulo são delineadas as principais contribuições e limitações deste trabalho, assim como algumas propostas de trabalhos futuros.

Capítulo 2 Métricas de Software

2.1. Introdução

Ao tratar de processos de engenharia, um aspecto chave que surge é a medição. Por sua natureza quantitativa, a engenharia busca utilizar medidas e números para ajudar na projeção e avaliação de produtos. No contexto da engenharia de sistemas, tais medidas são úteis para o planejamento e avaliação de um determinado software, especialmente durante o seu desenvolvimento, no qual decisões podem vir a ser tomadas para sanar deficiências encontradas.

Por ser um elemento abstrato e de grande complexidade, o software é muitas vezes avaliado somente de forma qualitativa, isto é, por meio da experiência dos engenheiros, analistas, desenvolvedores entre outros *stakeholders* (envolvidos nas atividades). Sabe-se que a experiência e o conhecimento dos envolvidos na avaliação de um sistema são importantes, mas tornar esta análise mais objetiva (i. e., por meio de métricas consolidadas, que enunciem as características do produto, a fim de comprovar a qualidade do mesmo) leva a uma maior confiabilidade na avaliação do software. Neste sentido, a utilização de métricas visa esta objetividade na avaliação e no projeto de um software, auxiliando um engenheiro de software a tomar decisões com base em dados quantitativos inferidos de modelos, do código, da execução de processos e de qualquer outro artefato do sistema desenvolvido (GOMES, 2001).

Este capítulo visa realizar um levantamento das métricas existentes (e suas taxonomias) para software. Após o levantamento, é apresentada a abordagem deste trabalho, que se concentra na evolução de métricas em geral, porém, com destaque para as da área de modelos de projetos, devido ao contexto da ferramenta onde o mesmo está inserido.

Este capítulo encontra-se estruturado da seguinte forma: a Seção 2.2 traz termos e conceitos utilizados na área de métricas; a Seção 2.3 contempla a área de Qualidade de Software, que se utiliza de medições e métricas para a avaliação de seus processos e produtos; a Seção 2.4 trata do processo de medição; a Seção 2.5 indica alguns atributos esperados para que as métricas sejam consideradas efetivas, isto é, que agreguem valor ao projeto com a sua utilização; a Seção 2.6 discorre sobre a validação de métricas; a Seção 2.7 exibe algumas taxonomias de métricas existentes; a Seção 2.8 aprofunda o estudo de métricas na área de modelos de projeto, por estarem mais contextualizadas com o foco da ferramenta EvolTrack,

que é a evolução da estrutura do software; por fim, a Seção 2.9 traz considerações finais acerca das ideias propostas neste capítulo.

2.2. Termos e Definições

Esta seção apresenta alguns termos e conceitos que serão mencionados no decorrer deste capítulo. É comum a utilização indiscriminada de alguns termos como medida e métrica, apesar da sutil diferença entre eles. Para uma melhor compreensão e desambiguação, eles são definidos a seguir (REZENDE, 2006):

- **Métrica:** É a medição de um atributo (propriedades ou características) de uma determinada entidade (produto, processo ou recursos).
- **Medida:** Valor quantitativo da extensão, quantidade, dimensões, capacidade ou tamanho de algum atributo do processo ou produto de software.
- **Medida Direta:** É aquela que não envolve a medição de outros atributos além do que se propõe a medir.
- **Medida Indireta:** É aquela que é derivada das medidas de outros atributos além do que se propõe a medir.

2.3. Uso de Métricas em Qualidade de Software

Atualmente, em qualquer setor da indústria, qualidade é considerada um fator crítico para o sucesso. Dessa forma, para se conseguir um setor de software competitivo nacional e internacionalmente, busca-se que os empreendedores coloquem a eficiência e a eficácia em foco nas empresas, almejando à oferta de produtos de software e serviços correlatos conforme padrões internacionais de qualidade (SOFTEX, 2009).

Em virtude do desejo de se aumentar a qualidade nos projetos de software, modelos de qualidade como o MPS.BR (SOFTEX, 2010a) e o CMMI (SEI, 2010), aliados aos estímulos da Associação para Promoção da Excelência do Software Brasileiro (SOFTEX, 2010b), buscam a excelência da produção de software e, para isso, estimulam a medição como processo fundamental para a avaliação dos processos que compõem o ciclo de vida do software.

A qualidade de um sistema de software pode ser entendida de diversas formas e utilizando diversas abordagens, mas todas elas incluem fases de medição e avaliação, demonstrando a importância de estudos e pesquisas no campo de métricas de software. A norma ISO/IEC 9126 (ABNT, 2003), e outras normas como a ISO 12207 (ABNT, 1998), tratam deste assunto estabelecendo um modelo de qualidade com os seguintes componentes:

- **Processo de desenvolvimento**, cuja qualidade afeta a qualidade do produto de software gerado, e é influenciado pela natureza do produto desenvolvido;
- **Produto**, compreendendo os atributos de qualidade do produto (sistema) de software. Estes atributos de qualidade podem ser divididos entre atributos **internos** e **externos**. Estes se diferenciam pela forma como são aferidos (interna ou externamente ao produto de software) e em conjunto compõem a qualidade do produto de software em si;
- **Qualidade em uso**, que consiste na aferição da qualidade do software em cada contexto específico de usuário. Esta é, também, a qualidade percebida pelo usuário.

É evidenciada a necessidade da medição nas atividades de avaliação e melhoria de processos para que, a partir dos dados coletados, sejam obtidas informações que auxiliem em tomadas de decisões para que o desenvolvimento de software venha a se tornar mais eficiente e eficaz. As próximas seções deste capítulo tratam da atividade de medição e apresentam algumas das diversas métricas aplicáveis a software existentes na literatura, selecionadas de forma a exemplificar todas as áreas de métricas da taxonomia utilizada neste trabalho.

2.4. Processo de Medição

Com o amadurecimento da engenharia de software, a medição vem assumindo um papel fundamental na compreensão e controle de práticas e produtos do desenvolvimento de software. Em nível de desenvolvimento, podem ser medidas características do software para, por exemplo, averiguar se os requisitos estão consistentes e completos, se o projeto tem boa qualidade ou se o código está pronto para ser testado. Em nível gerencial, podem ser medidos atributos do processo e do produto que indiquem quando o software estará pronto para entrega ou se o orçamento será ultrapassado. Na manutenção, as pessoas interessadas podem utilizar a medição para avaliar se o produto precisa ser melhorado. Dessa forma, observa-se a importância da medição em diversas fases do ciclo de vida de um software.

Em geral, a medição é o processo pelo qual números ou símbolos são designados a atributos de entidades do mundo real, de forma a descrevê-los de acordo com regras claramente definidas. Uma entidade pode ser um objeto (como um aluno ou uma mesa) ou um evento (como uma aula ou o projeto de desenvolvimento de um software). Um atributo é uma característica ou propriedade de uma entidade. Alguns exemplos de atributos são: a área de uma sala, a duração de uma aula ou o custo de um projeto de desenvolvimento de um software. Portanto, a medição captura informações sobre atributos de entidades.

Segundo ROCHE (1994), um processo de medição pode ser caracterizado por cinco atividades:

- **Formulação:** Derivação de medidas e métricas de software adequadas para a representação do software que está sendo considerado.
- **Coleta:** Mecanismo usado na obtenção dos dados necessários para derivar as métricas formuladas.
- **Análise:** Cálculo de métricas por meio de ferramentas matemáticas.
- **Interpretação:** Avaliação das métricas de forma a se obter profundidade nas características do sistema e respaldo para a tomada de decisão.
- **Realimentação:** Recomendações derivadas da interpretação das métricas de produto, transmitidas à equipe de software.

2.5. Atributos de Métricas

Muitas métricas têm sido propostas para software, porém, nem todas auxiliam de forma prática e simples os engenheiros de software. Algumas possuem um processo de medição complexo, outras são de difícil análise, e ainda existem aquelas que agregam pouca informação. Neste contexto, EJIOGU (1991) define um conjunto de atributos que devem ser contemplados para que métricas de software sejam consideradas efetivas, isto é, que alcancem a meta de auxiliar os participantes do desenvolvimento de software:

- **Simples e computáveis:** A forma de derivar a métrica deve ser de fácil entendimento, e seu cálculo não deve exigir um esforço ou tempo sobremaneira alto.
- **Empíricas e intuitivas:** A métrica deve satisfazer às noções intuitivas do engenheiro sobre o atributo do produto que está sendo considerado, além de ser baseada em experimentos comprovados.

- **Consistentes e objetivas:** A métrica deve sempre produzir resultados não ambíguos. Além disso, o resultado deve agregar valor à análise do software.
- **Consistentes no uso de unidades e dimensões:** O cálculo matemático da métrica deve usar medidas que não levam a combinações de unidades incompatíveis.
- **Independentes da linguagem de programação:** Métricas devem ser baseadas no modelo de análise, modelo de projeto ou na estrutura do programa.

2.6. Validação de Métricas

Métricas de software são úteis apenas se forem caracterizadas efetivamente e validadas de modo que o seu valor seja comprovado. Os seguintes princípios (PRESSMAN, 2006) são representativos de muitos que já foram propostos para a caracterização e validação de métricas:

- **Uma métrica deve ter propriedades matemáticas desejáveis:** O valor da métrica deve estar em um intervalo significativo (por exemplo, no intervalo [0;1], sendo que 0 significa o valor mínimo, 1 indica o máximo e 0,5 representa o valor médio). Uma métrica que foi elaborada em uma escala racional não deve ser composta por atributos que só podem ser medidos em uma escala ordinal.
- **Proporcionalidade:** Quando uma métrica representa uma característica de software que aumenta quando acontecimentos positivos ocorrem (ou diminui quando acontecimentos indesejáveis são encontrados), o valor da métrica deve aumentar (ou diminuir) do mesmo modo.
- **Cada métrica deve ser validada empiricamente em diferentes contextos antes de ser publicada ou utilizada para tomar decisões:** Uma métrica deve medir o fator de interesse, independentemente de outros fatores. Ela deve se adequar a sistemas simples e complexos, além de funcionar em diferentes linguagens de programação e domínios de sistemas.

2.7. Classificações e Exemplos de Métricas

Nesta seção, serão citadas algumas das taxonomias utilizadas para métricas na literatura (PRESSMAN, 2006), e uma breve descrição sobre as mesmas será levantada. O objetivo desta seção é apenas dar uma visão geral acerca das métricas existentes. Na Seção 2.8, um conjunto de métricas voltado para modelos de projetos, em cujo contexto este trabalho está inserido, será descrita com mais detalhes.

2.7.1. Métricas para o modelo de análise

Estas métricas tratam as funcionalidades, os dados e o comportamento de um software por meio de técnicas quantitativas, tais como pontos por função (PRESSMAN, 2006). Algumas das métricas dentro desta classificação são:

- **Funcionalidade entregue:** fornece uma medida indireta da funcionalidade que é empacotada com o software.
- **Tamanho do sistema:** mede o tamanho global do sistema definido em termos de informação disponível como parte do modelo de análise.
- **Qualidade da especificação:** fornece uma indicação da especificidade e completude de uma especificação de requisitos.

2.7.2. Métricas para o modelo de projeto

As métricas para projeto tratam de aspectos da arquitetura, projeto em nível de componentes e projeto de interface. Assim, torna-se possível obter uma avaliação de um projeto de software antes que o mesmo comece a ser desenvolvido, possibilitando uma redução de custos na fase de manutenção do sistema através de um bom projeto criado e avaliado. Por exemplo, existem métricas voltadas para projetos orientados a objetos (detalhadas na subseção 2.8.3), que medem características de classes, suas associações, comunicações e colaborações. A Seção 2.8 traz mais detalhes acerca deste escopo de métricas.

2.7.3. Métricas para código-fonte

Desde o início da produção de software, o código-fonte é sempre tratado como um dos artefatos mais importante do desenvolvimento, por ser normalmente o núcleo do funcionamento de um sistema. Logo, é de se esperar que existam métricas que tratem do mesmo de forma a avaliar a qualidade do produto central de um sistema. Geralmente, as métricas de código são divididas (ZUSE, 1997) em **métricas de complexidade**, que medem a complexidade lógica e a estrutura dos comandos, funções e blocos utilizados, e **métricas de comprimento**, que fornecem uma indicação do tamanho do software, isto é, da quantidade de linhas de código, número de classes, número de operações, número de argumentos, entre outros.

2.7.4. Métricas para teste

Estas métricas visam averiguar a cobertura dos casos de teste utilizados no projeto para verificar comandos e desvios, além de serem utilizadas para a contabilização de erros encontrados (HETZEL, 1993). Normalmente, estas métricas focalizam o processo de teste, não as características técnicas dos testes propriamente ditos.

2.7.5. Outras métricas

Podem ser citadas também métricas de apoio a várias fases do processo de desenvolvimento de software em geral. São enquadradas nessa classe as métricas de gerência de configuração, métricas de auditoria de qualidade, métricas de interface e usabilidade, entre outras (IEEE, 1994).

2.8. Métricas para o Modelo de Projeto

A seguir, serão discutidas algumas métricas de projeto de software, isto é, métricas com o objetivo de fornecer ao projetista uma visão mais aprofundada do software em desenvolvimento e ajudar a aumentar o controle e a qualidade do projeto como um todo.

2.8.1. Métricas de projeto arquitetural

Focalizam as características da arquitetura do programa com ênfase na estrutura arquitetural dos módulos ou componentes dentro da mesma. CARD & GLASS (1990) definem três medidas de complexidade de projeto de software, que serão detalhadas a seguir:

- A **complexidade estrutural** de um módulo i é definida da seguinte maneira: $S(i) = f_{out}{}^2\ (i)$, onde f_{out} é o *fan-out*[1] do módulo i.
- A **complexidade de dados** fornece uma indicação da complexidade na interface interna de um módulo i e é definida como: $D(i) = v(i)\ /\ [f_{out}(i) + 1]$, onde $v(i)$ é o número de variáveis de entrada e saída que são passadas tanto a partir do módulo i quanto para o módulo i.
- A **complexidade de sistema** é definida como a soma das complexidades estrutural e de dados, especificada como: $C(i) = S(i) + D(i)$

À medida que os valores dessas complexidades aumentam, a complexidade arquitetural do sistema também aumenta, levando a um esforço maior em termos de codificação, integração e testes.

2.8.2. Métricas para modelo de projeto orientado a objetos

No desenvolvimento de métricas para sistemas orientados a objetos, WHITEMIRE (1997) descreve nove características distintas e mensuráveis de um projeto OO:

- **Tamanho:** O tamanho de um projeto é definido em termos de quatro perspectivas: população, volume, comprimento e funcionalidade. A **população** é medida pela contagem estática das entidades OO, tais como classes ou operações. Medidas de **volume** são idênticas a medidas de população, mas são coletadas dinamicamente, em um dado instante do tempo. O **comprimento** é a medida de uma cadeia de elementos de projeto interconectados – a profundidade de uma árvore de herança, por exemplo, é uma medida de comprimento. Por fim, apesar de a perspectiva de **funcionalidade** estar relacionada ao modelo de análise, esta fornece uma indicação indireta do valor que deverá ser entregue ao cliente por uma aplicação OO; desta forma, tal perspectiva também impacta no tamanho do projeto.

[1] Fan-out (i): É definido como o número de módulos diretamente subordinados ao módulo i, isto é, o número de módulos que são diretamente acionados pelo módulo i.

- **Complexidade:** Assim como o tamanho, há diferentes perspectivas a serem consideradas no que tange a complexidade do software (ZUSE, 1997). WHITEMIRE (1997) focaliza a complexidade em termos de características estruturais, examinando a forma como as classes de um projeto OO estão inter-relacionadas.
- **Acoplamento:** As conexões físicas entre elementos de um projeto OO (por exemplo, o número de colaborações entre classes ou o número de mensagens passadas entre objetos) representam o acoplamento dentro de um sistema OO.
- **Suficiência:** "É o grau das características exigidas de uma abstração ou o grau das características que um componente de projeto possui na sua abstração, do ponto de vista da aplicação corrente", segundo WHITEMIRE (1997). Em outras palavras, um componente de projeto (por exemplo, uma classe) é suficiente se reflete as propriedades do objeto do domínio de aplicação que o componente em questão está modelando.
- **Completeza:** Enquanto a suficiência compara a abstração do ponto de vista da aplicação corrente, a completeza considera múltiplos pontos de vista, tendo implicação indireta no grau em que a abstração ou componente de projeto pode ser reusado, isto é, avalia a capacidade de o componente se adequar a domínios diferentes do domínio corrente.
- **Coesão:** Um componente OO deve ser projetado de modo a ter todas as operações colaborando de forma a atingir um propósito único e bem definido, segundo o domínio da aplicação. Assim, a coesão é determinada pelo grau em que as propriedades e operações que o componente OO possui fazem parte do problema ou do domínio do projeto.
- **Primitividade:** É o grau de atomicidade de uma operação ou classe, isto é, o quanto uma dada operação é construída a partir de uma sequência de outras operações contidas na classe. Uma classe que possui um alto grau de primitividade encapsula apenas operações primitivas.
- **Similaridade:** É o grau em que duas ou mais classes são semelhantes em termos de estrutura, função, comportamento ou finalidade.
- **Volatilidade:** A volatilidade de um componente OO mede a probabilidade de que uma modificação venha a ocorrer no mesmo (por exemplo, uma adaptação do componente para um novo domínio).

Com base nas características descritas, surgem dois grupos principais de métricas para projetos OO: as métricas orientadas a classe e as métricas orientadas a operações. As primeiras, exemplificadas pelos conjuntos de métricas CK (CHIDAMBER & KEMERER, 1994) e o conjunto de métricas MOOD (HARRISON, 1998), focam nas propriedades da estrutura base de projetos OO (a classe) e buscam medir características de tamanho, acoplamento, coesão e herança nas classes de um projeto de software. Já as segundas, exemplificadas pela complexidade ciclomática (MCCABE, 1976), número médio de parâmetros e tamanho médio de operação (LORENZ & KIDD, 1994), focam principalmente na complexidade, no tamanho, na completeza e coesão.

2.9. Considerações Finais

Neste capítulo buscou-se enaltecer a importância da medição e, por conseguinte, das métricas para software. Neste trabalho, foram abordadas métricas de forma geral, com ênfase nas relacionadas ao modelo de projeto, devido à motivação de auxiliar engenheiros de software e desenvolvedores na análise de um sistema durante a evolução no desenvolvimento do mesmo e a interação do trabalho com uma ferramenta que trata do projeto e arquitetura de software.

Um problema identificado é que a observação destas métricas em instantes isolados do tempo (ou nunca observá-las) dificulta a compreensão do comportamento da evolução do sistema, o que pode acarretar em sistemas cuja complexidade e custo de manutenção são altos, podendo chegar ao ponto de ser proibitivo, sendo preferível a construção de um novo software para a sua substituição. Além dos gastos financeiros, o tempo também é perdido desta forma, o que pode ser muito prejudicial para uma organização.

A análise de métricas de forma tabular e isolada de um contexto, por sua vez, pode não trazer informação útil, já que os valores destas métricas podem passar a ser apenas símbolos sem significado relevante, isto é, tendem a não fazer muita diferença na gestão do desenvolvimento nem na organização. Com isso, a visualização de software, introduzida no próximo capítulo, surge como uma área para fornecer um suporte na compreensão e avaliação das métricas de software, permitindo também uma visão conjunta destas métricas.

Nos capítulos que seguem, mostrar-se-á como foram desenvolvidos alguns mecanismos para a visualização e melhor compreensão de métricas de software, em especial,

no contexto da evolução das mesmas no decorrer do processo de desenvolvimento. Visando à inferência de dados (de um projeto de software, durante sua evolução) que tenham um valor semântico e que venham a contribuir para a tomada de decisões, este estudo serve como motivação para a criação de uma infraestrutura capaz de suportar e visualizar a evolução de diferentes métricas. Mais detalhes desta infraestrutura são discutidos nos próximos capítulos.

Capítulo 3. Visualização de Software

3.1. Introdução

Atualmente, computadores têm se tornado ferramentas importantes para a criação de visualizações, ajudando os usuários a melhor entender fenômenos complexos. Como uma consequência, a visualização se tornou uma disciplina da ciência da computação, definida por GERSHON (1994) da seguinte forma:

"*Visualização é mais que um método de computação. Visualização é um processo de transformar a informação numa forma visual, permitindo que indivíduos observem a informação. A exibição visual resultante permite que cientistas e engenheiros percebam características que estavam escondidas nos dados, todavia, são necessárias para a exploração e análise de dados".*

A visualização executa um papel crucial na utilização da computação para suportar o raciocínio humano; tal campo de estudo é denominado *intelligence amplification*, ou IA (BROOKS, 1996), em contraste com a área de inteligência artificial (*artificial intelligence*, ou AI), cujo objetivo é que o computador se comporte de maneira inteligente sozinho. Dessa forma, técnicas e metodologias da área de visualização podem trazer benefícios a campos do desenvolvimento de software por visarem ampliar o entendimento da informação, que é a natureza principal do software.

Para apresentar a área de visualização de software, este capítulo está organizado da seguinte forma: a Seção 3.2 apresenta um pouco da origem da área de visualização de software; a Seção 3.3 traz a descrição de algumas técnicas de visualização que serão utilizadas no restante deste trabalho; a Seção 3.4 mostra a aplicação da visualização na evolução de métricas de software; a Seção 3.5 faz as considerações finais deste capítulo, sumarizando a importância e objetivos da visualização de software.

3.2. Visualização: Da Informação ao Software

Desde a antiguidade, o homem procura métodos e técnicas para representar melhor a informação, se comunicar com maior eficiência e, com isso, gerar conhecimento para utilizar nas necessidades de sua vida. Como exemplo disto, pode-se citar a escrita, que começou em

paredes de grutas (denominada pintura rupestre), passou por placas de barro cunhadas pelos sumérios (escrita cuneiforme), chegou a hieróglifos e papiros no Egito Antigo, evoluiu para os pergaminhos na Roma Antiga, alcançou o papel e os caracteres que são conhecidos atualmente e que já estão sendo transferidos para o meio digital por meio do uso do computador.

O papel importante que a visualização tem no raciocínio humano, em geral, e no progresso científico, em particular, pode ser sintetizado por frases célebres de filósofos durante séculos:

"...um pensamento é impossível sem uma imagem."

(Aristóteles, 350 A.C.)

"Imaginação ou visualização, e em particular o uso de diagramas, exercem um papel crucial na pesquisa científica."

(René Descartes, 1637)

"O entendimento pode não levar a intuição, os sentidos podem não levar ao pensamento. Somente através da união deles pode surgir o conhecimento."

(Immanuel Kant, 1781)

Neste contexto, 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. 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 é uma tecnologia da informação, considera-se que ele faz parte da visualização da informação.

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 processo de visualização (DIEHL, 2007), independente do nicho, compreende três etapas principais (ilustradas na Figura 3): **Aquisição de dados**, que compreende a captura da informação existente em várias fontes, havendo formas diferentes de extraí-las dependendo do tipo de fonte; **Análise**, que é a atividade tipicamente necessária devido à informação extraída ser muito grande e complexa para que seja imediatamente apresentada a um individuo, devendo ser reduzida com base no foco e contexto do usuário; **Visualização**, etapa na qual os dados resultantes das etapas anteriores são mapeados em um modelo visual, isto é, transformados em informações gráficas para serem renderizados em algum dispositivo de saída.

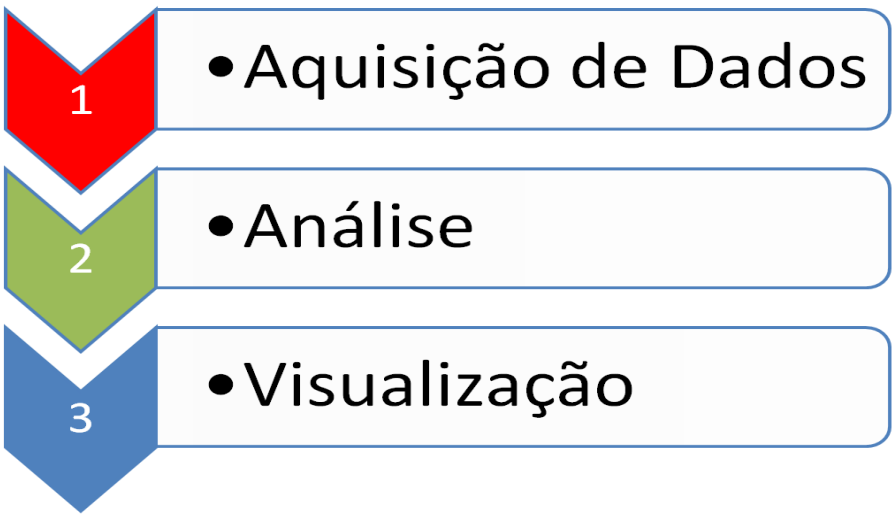


Figura 3. Fluxo de visualização

3.3. Técnicas de Visualização

Como mencionado nas seções anteriores, a visualização da informação pode ser amplamente definida como o processo auxiliado pela computação com o objetivo de revelar

detalhes em fenômenos abstratos por meio de transformações de dados abstratos em formas visuais. A intenção da visualização da informação é otimizar o uso da percepção e raciocínio visual do ser humano ao tratar de fenômenos que não são prontamente compreendidos (CHEN, 2006).

A maior parte das técnicas de visualização segue um dos dois seguintes princípios, ou ambos (DIEHL, 2007):

- **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 Ben Shneiderman (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.

A seguir serão detalhadas algumas técnicas de visualização, seus benefícios e sua aplicabilidade para o desenvolvimento de software.

3.3.1. Agrupamento

Devido à grande concentração de informação no mundo atual, técnicas de agrupamento são necessárias para organizar a quantidade de dados que é exibida para o usuário (DIEHL, 2007). A disposição da informação tem papel fundamental no aspecto da compreensão, pois, em ambientes sobrecarregados de dados, trabalha-se apenas com uma parcela dos mesmos em um dado instante de tempo, normalmente por possuírem alguma propriedade em comum que está sendo analisada.

Assim, a técnica de agrupamento (do inglês *clustering*) busca organizar a informação a ser exibida por meio de grupos construídos com base em uma propriedade em comum. Com isso, norteia-se no princípio de estruturar os dados de acordo com um contexto para que o usuário possa focar no seu interesse, abstraindo toda informação considerada desnecessária para aquele contexto (também conhecida como "ruído"). Em desenvolvimento de software,

especialmente em projetos de larga escala, é interessante a formação de grupos de informação decorrentes da lógica e dos módulos do sistema, pois os desenvolvedores e engenheiros de software normalmente buscam ter uma visão geral do projeto para focar numa área específica. Na Figura 4, pode ser vista uma aplicação típica da referida técnica, onde um modelo UML de diagrama de classes pode ter sua estrutura agrupada por pacotes que contenham arquivos distribuídos logicamente. Assim, a informação da estrutura do sistema é agrupada, não havendo sobrecarrega de dados na representação visual para o usuário.

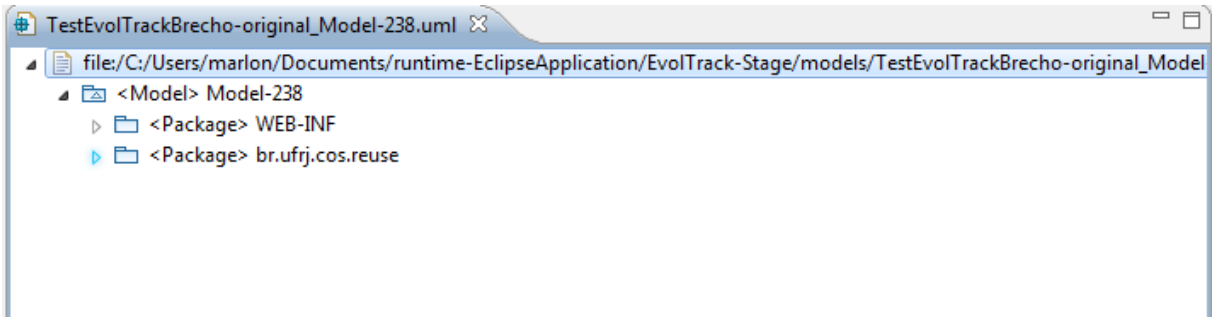


Figura 4. Exemplo de utilização da técnica de agrupamento

3.3.2. Detalhamento

Como forma de complementar as técnicas de agrupamento, foram desenvolvidas técnicas com o objetivo de focar em determinado tipo de informação. Enquanto as primeiras estavam preocupadas em organizar informações diversas segundo determinadas propriedades, estas últimas buscam expandir determinado grupo ou tipo de informação, normalmente a partir da interação com um ser humano, para uma análise mais detalhada (JERN, 1997). Este princípio se norteia na necessidade de um indivíduo, com base em uma visão geral sobre um determinado conjunto de dados, vir a focar em alguns dados específicos. Dessa forma, as técnicas de agrupamento organizam a informação para fornecer a visão geral e as técnicas de detalhamento (do inglês *drill-down)* focam nos dados de interesse do usuário.

No exemplo abaixo (Figura 5), pode ser vista a aplicação da técnica em uma imagem na qual o foco de informações é expandido de forma a fornecer dados adicionais acerca do elemento que está sendo analisado.

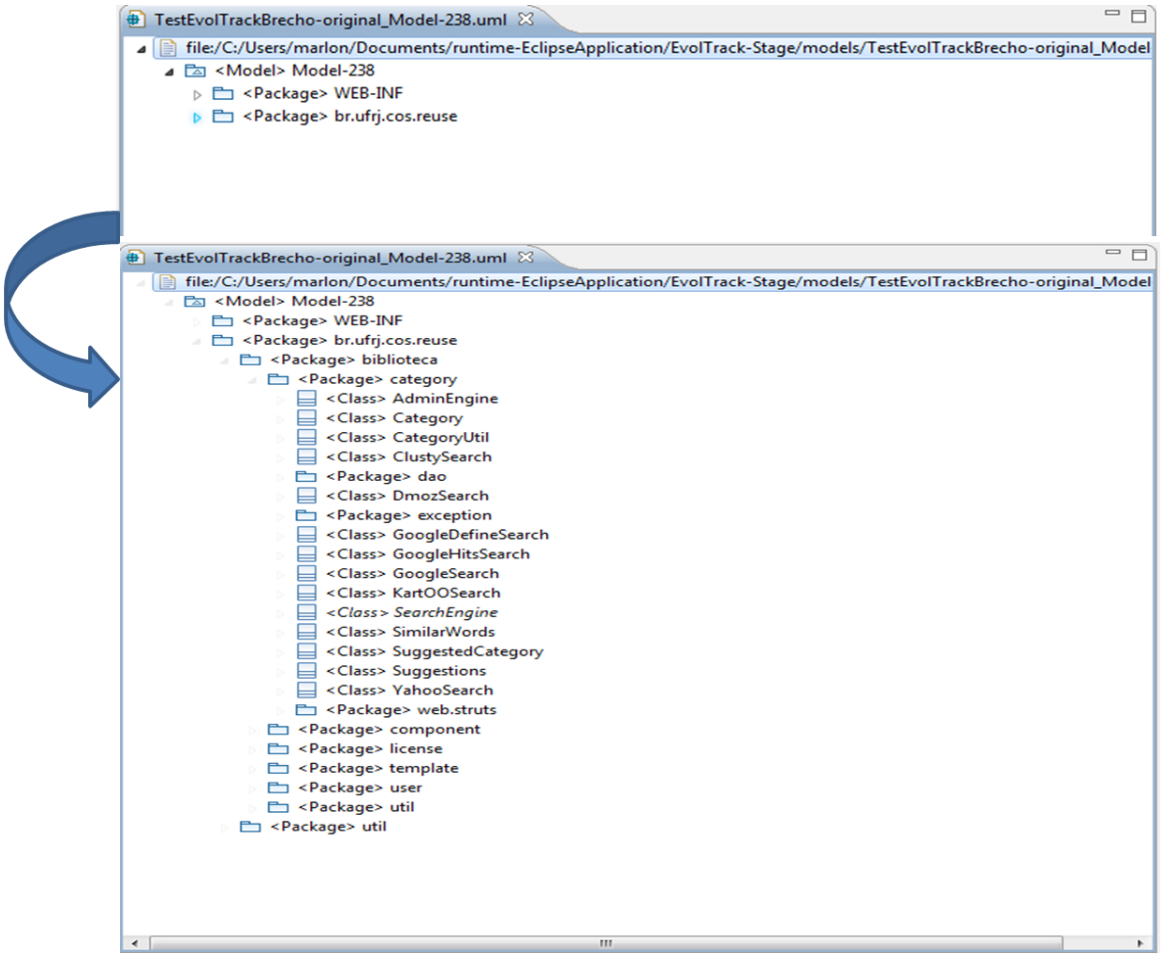


Figura 5. Técnica de detalhamento

3.3.3. Filtragem

Para visualizar grandes volumes de informação é necessário ter meios para reduzir o volume de informação a representar (CARMO & CUNHA, 1998). Entre os métodos existentes, pode ser destacada a criação de filtros que são utilizados para reduzir a quantidade de informação apresentadas ao indivíduo. No desenvolvimento de software, muita informação acaba por residir de forma intrínseca nos artefatos gerados, como código-fonte e documentação. Essa informação pode ser a autoria dos mesmos, a data de criação, os requisitos atendidos, os módulos envolvidos, a data de criação, entre outras.

Assim, projetos de software tendem a possuir muita informação implícita, o que motiva o uso de filtros para auxiliar na recuperação das mesmas, além de outras técnicas de visualização que contribuam com a externalização da informação. Normalmente, o funcionamento de filtros acontece na conjunção de uma ou mais propriedades da informação a ser filtrada (CARMO & CUNHA, 1998).

3.3.4. *Zoom*

Esta técnica, assim como outras baseadas no princípio do foco, busca centrar o contexto e a informação em determinado ponto de interesse do visualizador. HORNBAEK *et al*. (2002) comparam os paradigmas da técnica de foco + contexto com a técnica de *zoom* e concluíram que o grau de satisfação por parte do utilizador é maior no caso da primeira, porém, em casos onde se necessita de navegação e visualização de níveis, a segunda técnica permite interações mais rápidas.

Assim, esta técnica pode ser útil ao desenvolvimento de software, principalmente, pela necessidade de navegação. Na Figura 6, pode ser observado o efeito gerado pela técnica de visualização de *zoom* em uma imagem, focalizando certo ponto de interesse.

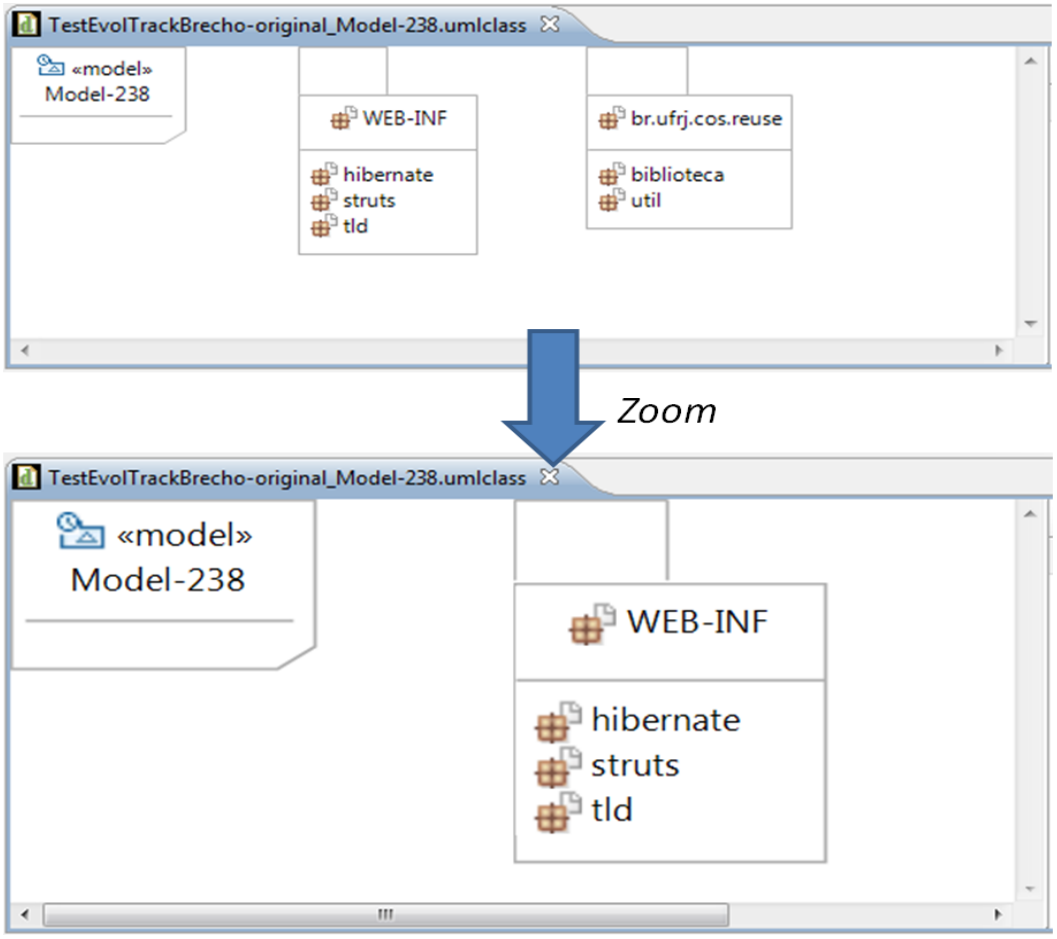


Figura 6. Técnica de zoom

3.3.5. *Minimap*

Minimap (ou *miniview*) é uma técnica para o fornecimento de uma visão geral sobre determinada informação (ROTO *et al.*, 2006). Geralmente, é visto como um mapa em miniatura localizado em um dos cantos de uma tela de exibição, sendo normalmente aplicado em jogos para auxiliar os jogadores a se reorientarem, conhecerem locais importantes, inimigos e aliados. Frequentemente, é utilizado também em sistemas de posicionamento global (GPS) para atualizar dinamicamente a posição de um individuo e situá-lo de forma mais abrangente dentro de uma localidade.

Na Engenharia de Software, existem algumas situações para a utilidade deste tipo de visualização, tais como, a visualização de modelos produzidos no desenvolvimento de software (especialmente, em larga escala e no desenvolvimento colaborativo) e a visualização de diagramas com muitos elementos, situações em que não é possível exibir todos os elementos em uma determinada área. A Figura 7 ilustra a utilização da técnica na visualização de um diagrama de classes no desenvolvimento de software.

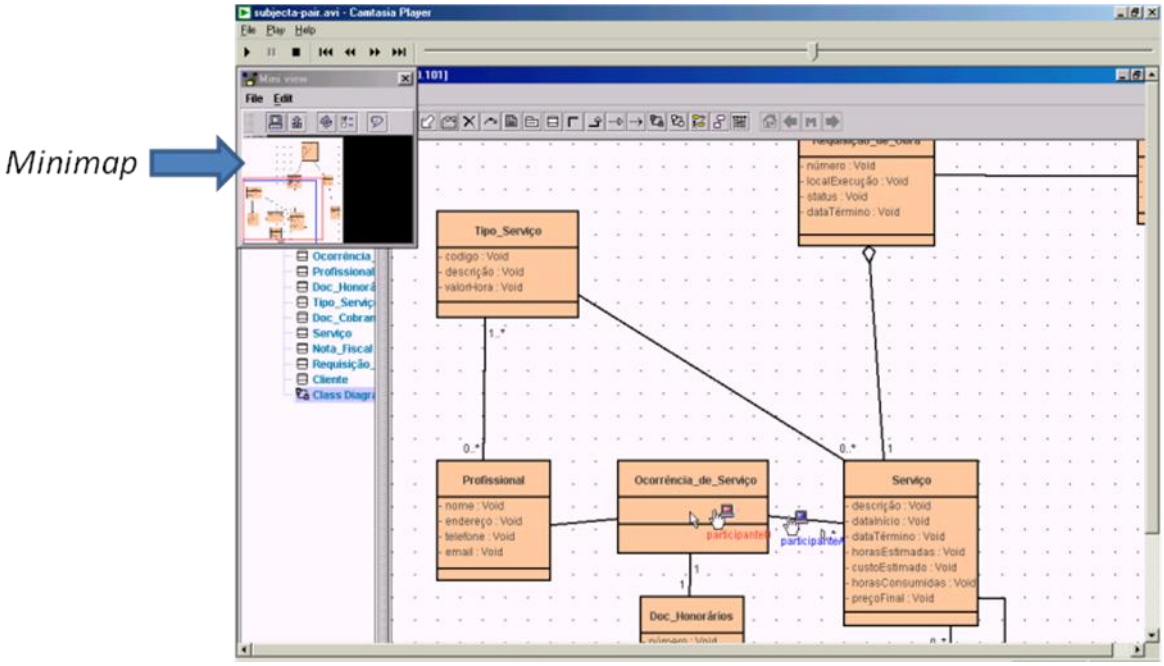


Figura 7. Técnica de *minimap* (REUSE, 2010)

3.3.6. Visualização em Hierarquia

A visualização em hierarquia é uma técnica comum e poderosa para estruturar informações. Tipicamente, hierarquias são árvores desenhadas por nós e arestas, onde cada nó é representado por uma caixa, círculo ou elipse, enquanto as arestas são representadas por linhas.

Por esta definição, diagramas de árvores tendem a perder muito espaço nas regiões superiores, pois, o número de nós por camada da árvore aumenta com a profundidade. Assim, novas técnicas para compactar hierarquias grandes e evitar perdas de espaço têm sido desenvolvidas. Na Figura 8, apresenta-se um exemplo típico de árvore, estrutura clássica de informação na computação.

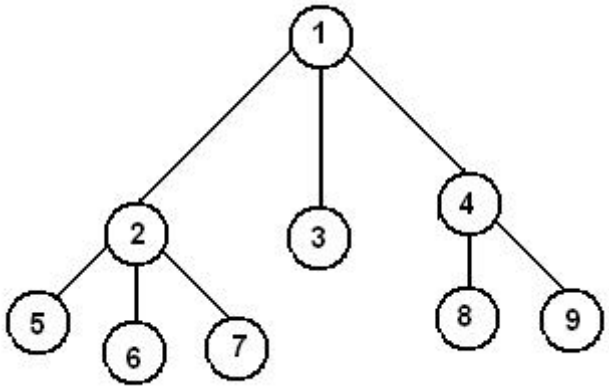


Figura 8. Árvore: Estrutura de informação (DIEHL, 2007)

Treemaps são variações da estrutura de árvore tradicional que codificam visualmente as propriedades dos nós em compartimentos, ao invés de conexões (JOHNSON & SHNEIDERMAN, 1991). A ideia básica dos *treemaps* é, recursivamente, dividir uma área em subáreas não sobrepostas de acordo com uma hierarquia dada. Em muitas aplicações, o tamanho das subáreas é proporcional a algum peso, que na forma mais simples, é a soma de todos os elementos na subárvore correspondente.

Em desenvolvimento de software, *treemaps* podem ser utilizados para a visualização da estrutura de um projeto de software (Figura 9), assim como a relação entre a estrutura de um projeto e os seus desenvolvedores. Para isso, cores diferentes nos retângulos podem representar quão antiga foi a última modificação, enquanto o tamanho dos mesmos pode indicar uma maior quantidade de artefatos. Assim, a visão por meio de *treemaps* pode ser simples e eficiente se, em sua elaboração, forem observados alguns detalhes: (i) hierarquias muito subdivididas ou profundas podem confundir a visão do leitor com a sensação de indiferença do interior, isto é, o leitor não consegue identificar mais pontos interiores a uma área; (ii) as cores selecionadas para os retângulos têm que possuir um significado para a

observação do usuário e devem facilitar a identificação das subáreas; (iii) apesar de fornecerem uma visão geral da hierarquia, existe a dificuldade de se focar em nós que não são folhas (por exemplo, diretórios intermediários).

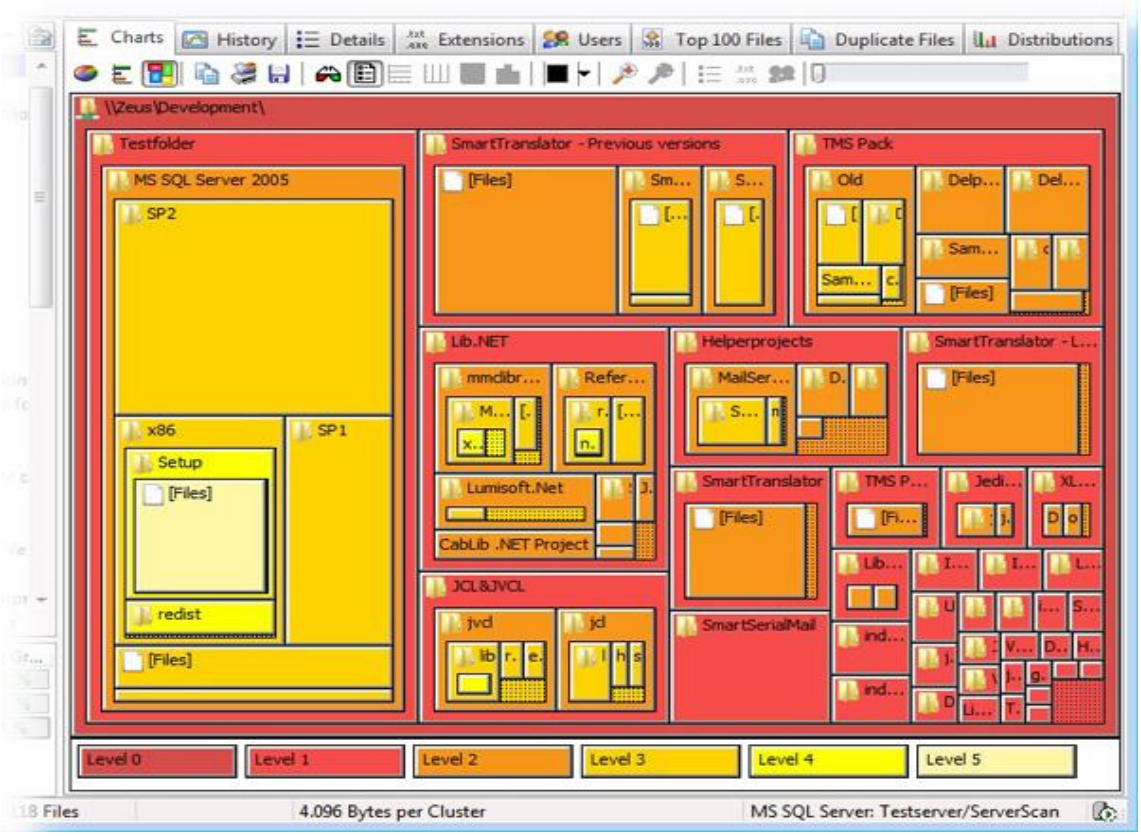


Figura 9. Uso de *Treemap* em Desenvolvimento de Software (JAM, 2010)

Algumas variantes têm sido desenvolvidas, incluindo *treemaps* ordenados (SHNEIDERMAN & WATTENBERG, 2001), *squarified treemaps* (BRULS *et al.*, 2000), *cushion treemaps* (VAN WIJK & VAN DE WETERING, 1999) e *Voronoi treemaps* (BALZER & DEUSSEN, 2005). Existem trabalhos que focam o uso da visualização tridimensional com *treemaps*, formando as pirâmides de informação, onde cada nó é representado por um retângulo, porém, as camadas de retângulos são sobrepostas da maior (formando a base) até a menor (formando o topo). Alguns exemplos podem ser vistos em ANDREWS *et al.* (1997).

3.3.7. Grafo

Grafos são estruturas matemáticas amplamente utilizadas para descrever relacionamentos entre objetos (KAUFMANN & WAGNER, 2001). Normalmente, os objetos

são denominados nós e os relacionamentos, arestas. São frequentemente caracterizados por uma ou mais das seguintes propriedades: grafos podem ser direcionados ou não; podem ser cíclicos ou acíclicos; podem ser conexos ou desconexos; podem ser bipartidos; podem ser planares; entre outras propriedades.

O objetivo de se construir grafos é transmitir a informação subjacente a ele. Para isso, vários critérios estéticos têm sido estudados (PURCHASE *et al.*, 1996), incluindo:

- Minimização de cruzamentos: Se um grafo é não-planar, deve ser desenhado com o menor número de arestas que se cruzem.
- Minimização de dobras: Arestas devem ter o menor número de flexões possíveis. Estudos apontam que a continuidade das arestas é mais importante que o número de flexões.
- Minimização de área: A área do desenho deve ser pequena e deve existir uma distribuição homogênea da densidade do grafo.
- Minimização de comprimento: Arestas menores são mais fáceis para seguir. Assim, é válido minimizar o tamanho das arestas.
- Maximização de ângulo: Ângulos pequenos entre arestas fazem com que seja difícil de distinguir duas arestas.
- Simetria: Grafos simétricos facilitam o entendimento do usuário por estimular a compreensão do desenho.
- Agrupamento: Para grafos grandes, partes fortemente conectadas devem ser desenhadas em separado para facilitar a reação visual do usuário.

A Figura 10 ilustra um grafo (SOUZA, 2010) representando uma rede de desenvolvedores geograficamente distribuídos. Esta visualização pode ser utilizada para se avaliar a interação entre os mesmos e os seus respectivos países, além de possíveis impactos relacionados a indivíduos que deixam a equipe.

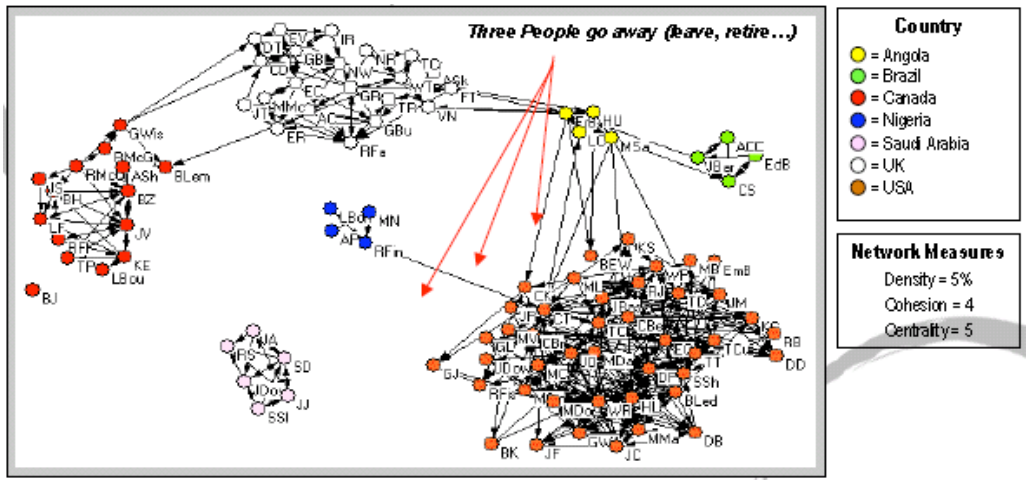


Figura 10. Grafo de desenvolvimento distribuído (SOUZA, 2010)

3.3.8. Visualização 3D

Muitos argumentos têm sido especulados que as visualizações 3D são superiores às bidimensionais. Embora a criação de ferramentas para navegar e visualizar em 3D sejam complexas, há uma grande demanda para a criação destas. Por exemplo, o *National Library of Medicine*, dos Estados Unidos, desenvolveu o projeto *Visible Human* (VISIBLE HUMAN PROJECT, 2010), que é um repositório de imagens sobre o corpo humano. Muitas aplicações têm sido criadas com base nessas imagens, permitindo uma interação total com as visualizações em 3D do corpo humano (como pode ser visto na Figura 11).

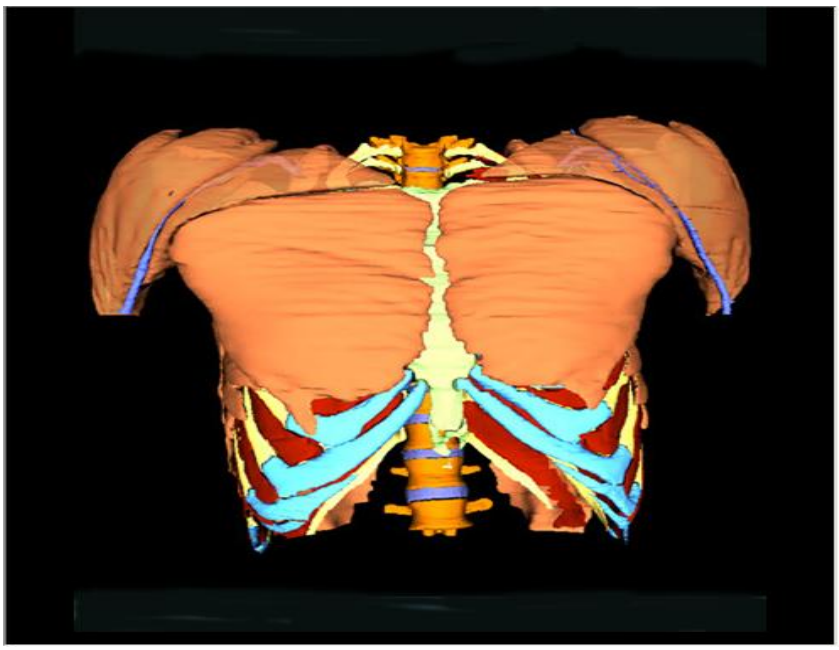


Figura 11. Visualização 3D do pulmão no projeto *Visible Human* (VISIBLE HUMAN PROJECT, 2010)

A visualização volumétrica é a forma mais utilizada em 3D. Ela implica em criar visualizações muito próximas do mundo real, numa tela de exibição 2D de um computador, além da eventual interação em ambientes virtuais. As aplicações atuais desse tipo incluem áreas como a medicina, ensino, meteorologia, arquitetura, desenvolvimento de software, entre outras.

Tem vindo a crescer cada vez mais a utilização de realidade virtual nessas aplicações. O VRML (*Virtual Reality Modeling Language*), por exemplo, é uma linguagem bastante utilizada na criação de universos virtuais, que permite modelar desde lugares históricos até planetas, disponibilizando as visualizações interativas na Internet. O conceito de universo 3D também pode ser ambíguo, existindo basicamente quatro interpretações diferentes (CARVALHO & MARCOS, 2009):

- A primeira interpretação considera que o universo 3D é composto por objetos 3D reais, como os presentes em visualizações nas áreas da medicina, arquitetura ou química. Nesses casos, há sempre o interesse por parte do utilizador em visualizar os objetos no seu interior, pois isto é fundamental para a compreensão da visualização. Navegar para cima, para baixo, para a esquerda, para a direita, para frente e para trás dentro do ambiente apresentado fornece ao utilizador a percepção de contenção espacial.
- A segunda é a de que o universo 3D é artificial e sintético. As visualizações criadas são universos fictícios que se apresentam como bastante reais. Esse tipo de visualização implica em passeios virtuais por ambientes fictícios, e não possui como ideia chave explorar o interior de objetos, como no caso anterior. As aplicações que simulam visitas virtuais a museus ou lugares históricos são exemplos deste caso.
- A terceira categoria de visualização em 3D inclui objetos em que é possível aplicar o sentido do ser humano de 3D para fazê-los parecer tridimensionais. Por exemplo, árvores hierárquicas, redes de comunicação, SIGs (Sistemas de Informação Geográfica), objetos multidimensionais e temporais, podem ser modelados em 3D, mas isto não significa que estes objetos fazem realmente parte de um universo 3D.
- A quarta categoria engloba objetos em que a criação da terceira dimensão é forjada. Alguns exemplos são os gráficos em barras ou pizza construídos a

partir de dados 2D, mas que são visualizados em 3D. Este tipo de visualização, apesar de apelativo, é bastante questionável em termos de qualidade da compreensão que se pode obter.

Segundo WARE & FRANCK (1994), o total de informação que pode ser percebida numa exibição tridimensional excede em três vezes a exibição bidimensional. Tomando este estudo como motivação e os avanços da tecnologia, muitos pesquisadores têm sugerido visualizações 3D para o projeto de software e até o uso de ambientes virtuais para permitir que os indivíduos "caminhem" por um projeto de software (DIEHL, 2007).

GOGOLLA *et al*. (2000) sugeriram alguns cenários nos quais diagramas UML, utilizados normalmente para a construção de arquiteturas de software, podem se beneficiar de um leiaute tridimensional. Eis alguns cenários exemplificados pelos autores:

- Em um diagrama de classes, as classes mais importantes podem ser desenhadas em um plano mais destacado para possuir o foco. Além disso, podem existir diferentes perspectivas do mesmo diagrama, onde cada uma tenha foco num conjunto específico de classes que pode ser modificado conforme comandos do usuário.
- Em diagramas de objetos, vários tipos de formas podem ser utilizados para representar objetos, não estando limitadas às mesmas formas bidimensionais. Assim, objetos de mesmo tipo possuiriam uma mesma forma para uma melhor interpretação.
- Para diagramas de sequência, animações podem mostrar mensagens sendo transmitidas entre objetos. Com o benefício de uma dimensão a mais, representar a interação entre estes se torna mais fácil, principalmente por melhorar a visualização de sobreposição de arestas.
- Vários tipos de diagramas podem ser combinados em uma única visão, por exemplo, os diagramas de classe e sequência.

Desta forma, a utilização da terceira dimensão em projeto de software se mostra de grande utilidade e pesquisas neste âmbito são necessárias para que os avanços tecnológicos no sentido de aplicações gráficas que tenham uma maior participação na engenharia de software. A Figura 12 mostra a aplicação da visualização 3D no contexto da ferramenta Vizz3D (CARLSSON, 2006), onde as esferas coloridas são classes, as esferas transparentes representam pacotes e as arestas mostram a interação entre eles. As diferenças entre as cores servem para a distinção entre entidades de pacotes diferentes.

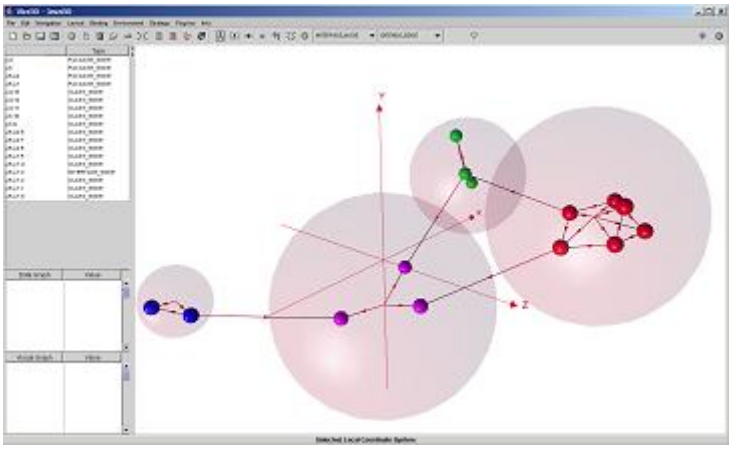


Figura 12. Visualização da interação num projeto de software pelo Vizz3D (CARLSSON, 2006)

3.4. Visualização aplicada à Evolução de Métricas de Software

O termo evolução de software é usualmente definido por muitos autores como a linha do tempo na qual um sistema sofre mudanças (DIEHL, 2007). BROOKS (1987) afirmou em seu trabalho que todos os softwares que obtiveram sucesso sofreram mudanças, que podem ser creditadas à necessidade de um sistema em se adaptar às mudanças ocorridas nos requisitos e no ambiente do cliente. Assim, o papel desempenhado pela mudança num projeto de software é determinante para o sucesso do mesmo, o que valoriza a importância da análise e estudo da evolução de sistemas. Dessa forma, duas áreas surgem na evolução de software (DIEHL, 2007):

- **Projeto para mudança:** O objetivo é desenvolver regras e arquiteturas para facilitar a tarefa de modificar um sistema no decorrer do tempo, abordando questões como reconfiguração, adaptação, extensão, depuração, otimização, avaliação, e gerência de projetos.
- **Análise histórica do software:** Durante o tempo de vida de um sistema de software, muitas versões são produzidas, onde a análise do código-fonte destas versões, assim como a documentação e outros artefatos, pode revelar regularidades e anomalias no processo de desenvolvimento de um sistema.

Engenheiros de software se deparam constantemente com um conjunto de requisitos gerais relacionados à qualidade de software. O sistema deve ser fácil de usar, rápido, correto, confiável, seguro, extensível, entre outras necessidades. Consequentemente, os envolvidos no

desenvolvimento de software buscam métricas de software para quantificar várias propriedades de um sistema e relacioná-las com os aspectos citados.

Assim, a visualização de software pode ser utilizada como um mecanismo de apoio para se conseguir um melhor acompanhamento da evolução do software por meio da evolução de métricas extraídas do mesmo. A partir de técnicas e princípios, como descritos neste capítulo, consegue-se focar em métricas de interesse sem perder o contexto das mesmas, permitindo a análise integrada e aumento da eficácia da métrica. Além disso, é possível agregar informação sem sobrecarregar o usuário, como no caso das técnicas tridimensionais onde se explora a dimensão a mais para que diferentes perspectivas sejam visualizadas em conjunto. Como exemplo, pode-se pensar na evolução de métricas em conjunto com a evolução estrutural de um projeto de software.

Nesse sentido, avaliar a evolução de métricas de software por meio de técnicas e mecanismos advindos da visualização se torna uma ferramenta que contempla as dificuldades no acompanhamento do desenvolvimento de software, agregando valor e qualidade ao mesmo.

3.5. Considerações Finais

Neste capítulo, buscou-se descrever a área da visualização de software através de sua história e algumas de suas técnicas, voltadas para o decorrer deste trabalho (que se encontra focado nos nichos da estrutura e evolução do software), com suas respectivas características e benefícios do seu uso, incluindo exemplos. Além disso, foram relacionados os objetivos da visualização com os problemas encontrados na compreensão da informação em sistemas e no processo de desenvolvimento de software, observando os ganhos agregados com a utilização da mesma.

O próximo capítulo apresenta algumas ferramentas e exemplos de visualização da evolução de métricas de software, visando identificar necessidades para a visualização (para uma melhor compreensão da informação existente) e para a manipulação dos dados de um projeto de software, de forma a obter características a serem contempladas pela abordagem deste trabalho.

Capítulo 4. Trabalhos Relacionados

Este capítulo apresenta um conjunto de ferramentas e abordagens para apoiar o acompanhamento sistematizado durante o ciclo de vida do software. Entretanto, será dado maior enfoque em ferramentas que utilizem visualização e métricas de software como principal recurso no auxílio deste processo. Desta forma, foram analisadas as seguintes ferramentas: SeeSoft (BALL & EICK, 1996), MetricView (TERMEER *et al.*, 2005), Tarantula (JONES, 2002), EPOSpix (WEIßGERBER *et al.*, 2005), GEVOL (COLLBERG *et al.*, 2003), Vizz3D (CARLSSON, 2006), CVSscan (VOINEA *et al.*, 2005), softChange (GERMAN *et al.*, 2006).

O restante do capítulo está organizado da seguinte forma: A seção 4.1 apresenta as funcionalidades da ferramenta SeeSoft e uma breve discussão sobre sua abordagem; a Seção 4.2 discorre sobre o projeto MetricView e suas características; a Seção 4.3 discute a cerca do mecanismo Tarantula; a Seção 4.4 apresenta a ferramenta EPOSpix e suas propriedades; a Seção 4.5 discorre sobra a abordagem GEVOL; a Seção 4.6 discute a ferramenta Vizz3D e seus recursos de visualização; a Seção 4.7 mostra o mecanismo CVSscan e suas funcionalidades; a Seção 4.8 apresenta os recursos da abordagem softChange; a Seção 4.9 conclui mostrando a análise comparativa entre os trabalhos e abordagens descritos.

4.1. SeeSoft

O sistema SeeSoft foi desenvolvido pela *AT&T Bell Laboratories* para visualizar mudança e métricas relacionadas a sistemas complexos e grandes (BALL & EICK, 1996). Ele introduziu a visualização com preenchimento de espaços para métricas relacionadas a linhas de código, de maneira que módulos com até um milhão de linhas de código possam ser exibidos na tela (EICK *et al.*, 1992). O objetivo da visualização com preenchimento de espaços é transmitir a maior quantidade de informação possível com a menor quantidade de *pixels*. A ferramenta provê algumas visualizações codificadas por cores:

- **Representação textual:** Cada linha de texto é mostrada como um texto colorido, de acordo com uma métrica associada. Todos os caracteres da linha são apresentados com a mesma cor.

- **Representação de linha:** Cada linha de texto é representada como uma linha colorida de *pixels*.
- **Representação de pixel:** Cada linha de texto é representada por um pixel (ou conjunto de *pixels*). Os *pixels* podem ser ordenados de acordo com a ordem que aparecem ou com a cor.
- **Representação de sumário de arquivos:** Todo arquivo é representado por uma caixa, existindo quatro tipos diferentes de tamanho.

Assim, os tipos de visualização descritos acima são selecionados ou combinados conforme o tamanho do sistema a ser analisado. Podem ser visualizadas métricas como a idade do código, o número de *bugs*, a profundidade de blocos aninhados, além de dados de perfis (autoria, por exemplo). As escalas de cores também podem ser selecionadas. Por exemplo, usando a escala de cor *heated-object* (vermelho: maior; azul: menor) na métrica de idade do código, pode ser observado em vermelho as linhas recentemente atualizadas, enquanto em azul aparecem linhas que foram atualizadas menos recentemente.

Como mostrado na Figura 13, arquivos são representados por retângulos e, dentro de cada um, linhas coloridas representam linhas de código-fonte. Neste exemplo, utilizou-se a escala de cores *heated-object* citada acima.

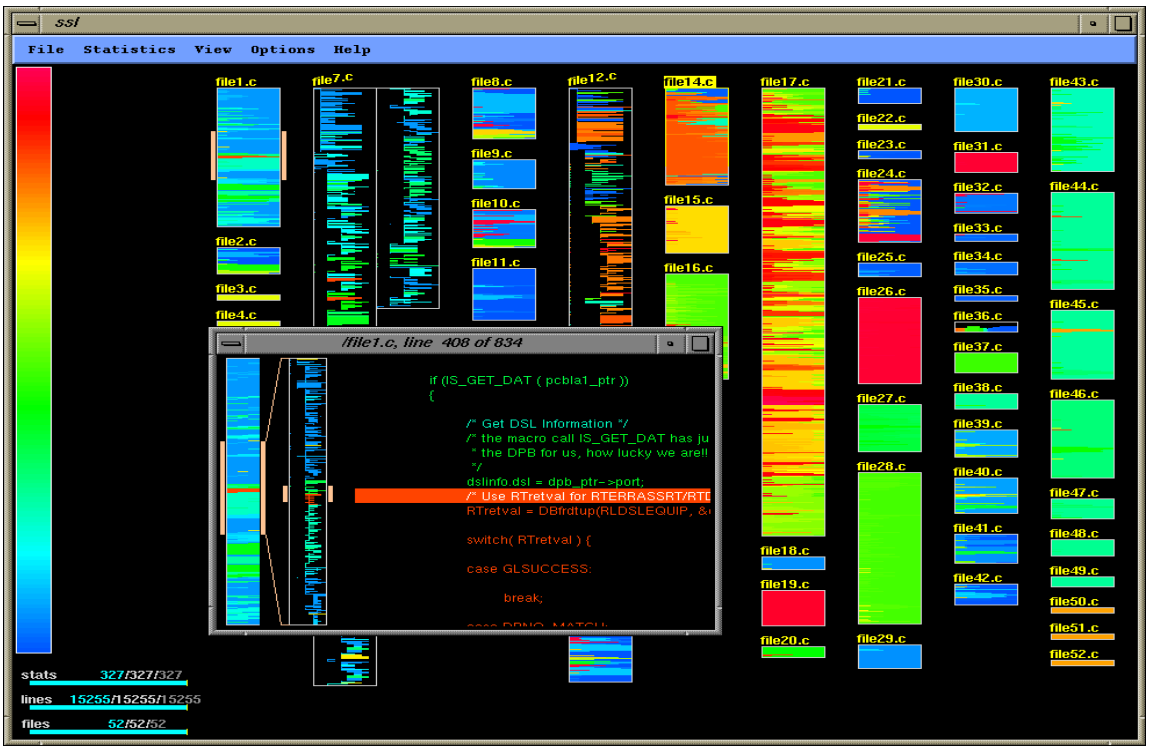


Figura 13. Sistema SeeSoft (BALL & EICK, 1996)

4.2. MetricView

O MetricView é uma ferramenta de exploração e visualização de software que combina tradicionais diagramas UML com a visualização de métricas. Foi desenvolvido pela Universidade de Tecnologia de Eindhoven, no departamento de Arquiteturas e Redes de Sistemas (do inglês SAN), que tem como uma de suas atividades de pesquisa a análise de arquiteturas de sistemas orientados a objetos, usualmente modelados com a UML. Um dos meios que o SAN utiliza para obter maior compreensão nestas arquiteturas é a análise de várias métricas de software extraídas das mesmas (TERMEER *et al.*, 2005).

Diagramas UML representam uma das formas mais difundidas de descrever a arquitetura de software e o projeto da informação. Contudo, enquanto as representações baseadas na UML são focadas na visualização de elementos, as ferramentas que computam métricas fornecem grandes tabelas de números como saídas. Estas, por sua vez, podem ser de difícil compreensão para engenheiros de software, dificultando o entendimento do sistema observado. A solução para este problema foi o objetivo do projeto desta ferramenta: visualizar métricas de software em projetos existentes, representados por modelos UML, de maneira que mais conhecimento possa ser extraído da arquitetura, se comparado com a análise de métricas da forma tabular. Desta forma, dado um diagrama UML (Figura 14a) e um conjunto de valores de métricas (Figura 14b), MetricView produz o resultado mostrado na Figura 14c.

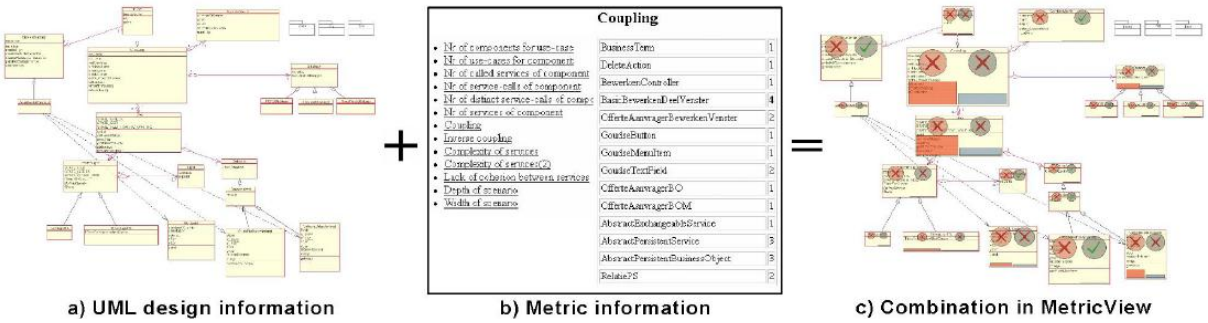


Figura 14. Visão geral do MetricView (TERMEER *et al.*, 2005)

O MetricView pode visualizar os seguintes diagramas da UML: classe, sequência, estado, caso de uso e colaboração. Eles são importados a partir de arquivos XMI (*XML Metadata Interchange*), conforme a versão 1.2 descrita pela OMG (2010b). Os dados da UML são representados usando o metamodelo da UML 1.3 (OMG, 2010c).

A ferramenta suporta tanto métricas globais, isto é, definidas para o modelo UML como um todo, e métricas de elementos, isto é, definidas para um elemento ou relacionamento de um modelo. Uma métrica é modelada com um par (chave, valor), onde a chave é o nome único da métrica. Atualmente, o MetricView suporta métricas booleanas e numéricas e cada elemento pode ter qualquer número de métricas.

A Figura 15 mostra uma típica sessão de visualização do MetricView. A área de exibição, identificada pela letra A, exibe um diagrama de classes, combinado com seis métricas. Os usuários podem selecionar o diagrama desejado a partir de um conjunto de diagramas, utilizando o navegador identificado pela letra B. O diagrama é desenhado utilizando as informações estruturais armazenadas no arquivo XMI. A letra C apresenta um seletor de cores para as métricas a serem visualizadas. A lista de métricas, identificada pela letra D, mostra uma lista textual de todas as métricas disponíveis no arquivo de entrada, onde são exibidos seus nomes e tipos. Por fim, a letra E mostra a área de controle da GUI (*Graphical User Interface*) onde são realizadas as configurações da ferramenta.

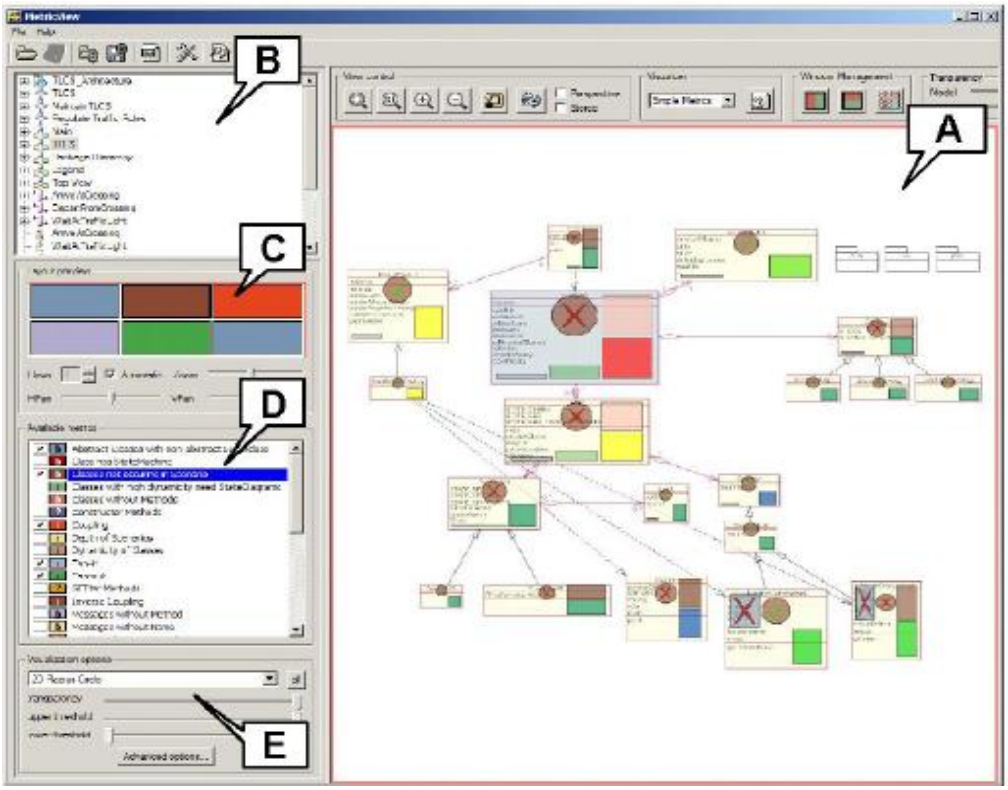


Figura 15. Visualização do MetricView (TERMEER *et al.*, 2005)

4.3. Tarantula

Uma das atividades mais caras e que consomem mais tempo do processo de depuração de software é a localização de defeitos ou falhas (COLLOFELLO & WOODFIELD, 1989). Para localizar estes defeitos, os desenvolvedores devem identificar os pontos envolvidos em falhas e selecionar os comandos suspeitos de conterem defeitos. Esta ferramenta apresenta uma técnica que utiliza a visualização para assistir estas tarefas. A técnica utiliza a cor para mapear visualmente a participação de cada comando do programa, na execução do mesmo em um conjunto de testes (composto por casos de teste que obtiveram sucesso e falha). Baseados neste mapeamento visual, o usuário pode inspecionar os comandos no programa, identificando aqueles envolvidos em falhas, e localizar os que representam potencialmente um defeito.

O sistema de visualização Tarantula foi escrito em Java e toma como entrada o código-fonte de um sistema e o resultado do conjunto de testes executados no mesmo. A partir disto, é exibida uma visão interativa do sistema de acordo com uma variedade de mapeamentos visuais. A Figura 16 apresenta uma ilustração da execução do sistema, onde o código-fonte aparece no centro e o canto superior esquerdo possui um menu para a seleção do modo de exibição, havendo seis possibilidades: O modo mais simples apresenta apenas o código-fonte sem coloração; o modo discreto utiliza uma abordagem de mapeamento em três cores (verde, amarelo e vermelho); o modo contínuo é o mais complexo, por utilizar equações de cor e brilho; outros três modos adicionais permitem que o usuário foque em comandos executados em casos de testes com sucesso, falha ou uma combinação dos dois.

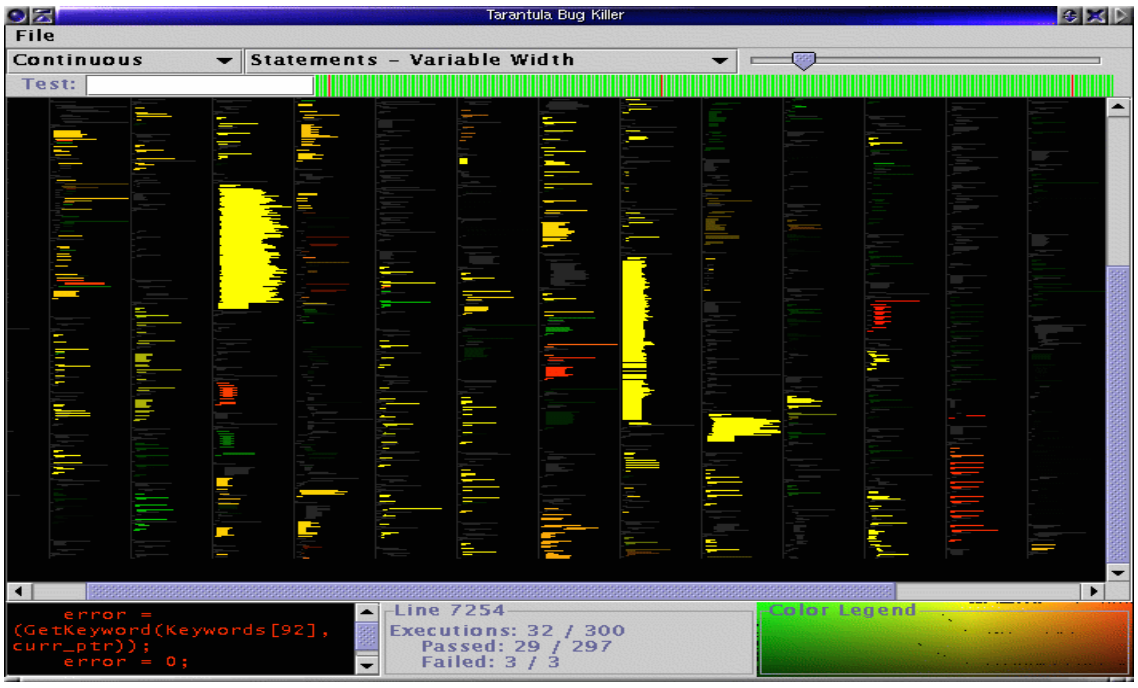


Figura 16. Sistema Tarantula em modo contínuo (JONES, 2002)

4.4. EPOSpix

O EPOSpix é um *plugin* desenvolvido para a plataforma Eclipse que integra a visualização de mapas de *pixels* e permite explorar interativamente a evolução do acoplamento entre diferentes arquivos de um projeto. Buscando uma nova forma de explorar a informação armazenada em arquivos, o mecanismo busca extrair automaticamente a informação do arquivo do software e visualizá-lo num mapa de *pixels*, onde cada um destes representa um acoplamento.

Neste mecanismo, acoplamento significa quão frequentemente dois arquivos são modificados simultaneamente. A ferramenta oferece os seguintes recursos de interatividade:

- ***Zoom*:** Na maior resolução, cada acoplamento é representado por um único *pixel* na tela. A resolução pode ser reduzida, de forma que quadrados de *pixels* representem um acoplamento. Como os mapas de *pixels*, em particular os de resolução baixa, tendem a se tornarem grandes, recursos de *scroll* (rolagem) são suportados.
- **Detalhamento:** Quando o usuário move o cursor do *mouse* pelo mapa de *pixels*, os detalhes sobre o acoplamento corrente são exibidos na parte inferior da IDE Eclipse.
- ***Crosshairs:*** Duas linhas de cor verde são exibidas de forma que a interseção entre elas esteja sobre o acoplamento corrente, relacionando os arquivos envolvidos diretamente na visão do explorador de pacotes.
- **Opções:** São dadas opções para a escolha das cores utilizadas tanto para o valor do acoplamento como para a confiança do mesmo. Além disso, filtros podem ser utilizados e o mapa pode ser exportado.

A Figura 17 apresenta a interface do *plugin*.

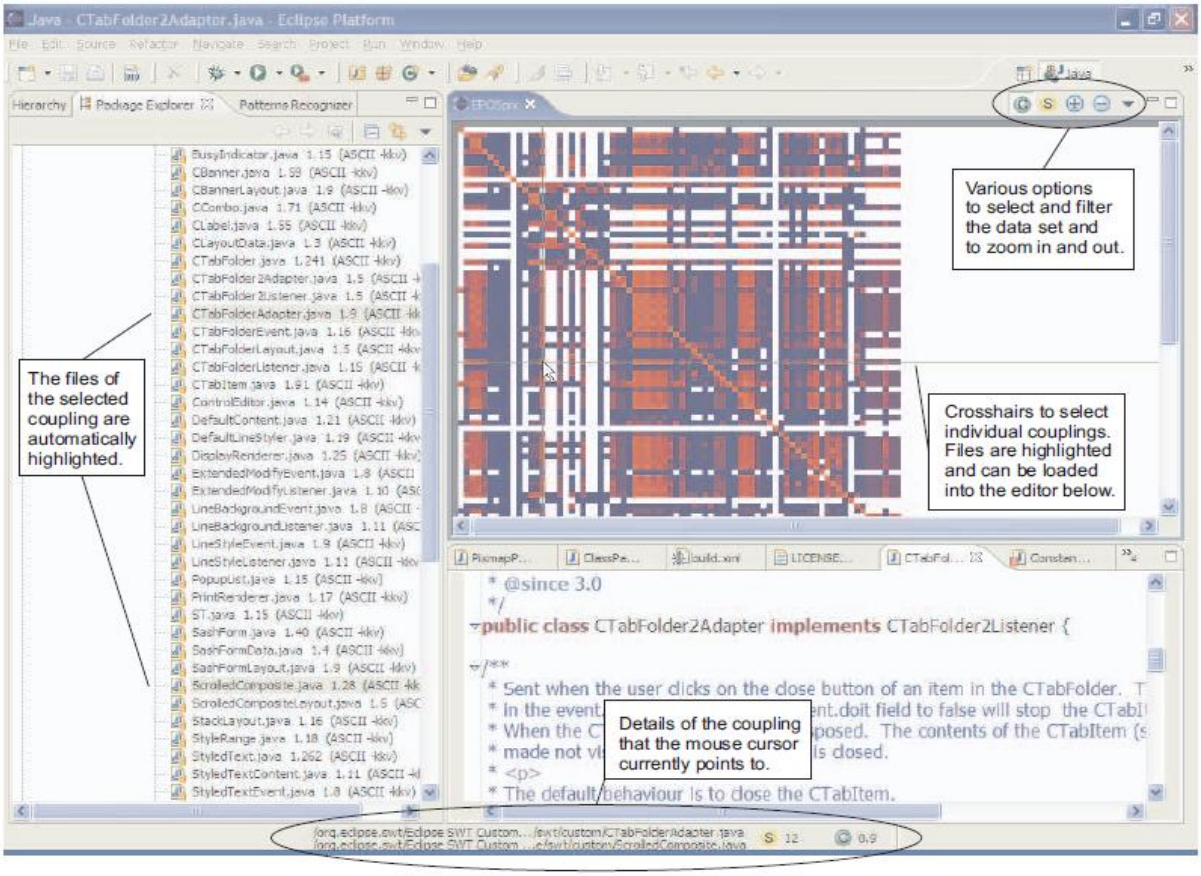


Figura 17. *Plugin* EPOSpix (WEIßGERBER *et al.*, 2005)

4.5. GEVOL

GEVOL é um sistema que visualiza a evolução de software utilizando uma técnica de desenho de grafos para a visualização ao longo do tempo. Ele extrai informações de programas escritos em Java, armazenados em repositórios CVS, e mostra o software utilizando o visualizador de grafos temporal. Esta informação pode ser utilizada por desenvolvedores para o entendimento de um sistema legado, respondendo a perguntas como: Como o programa está estruturado? Quem foram os responsáveis por quais partes do sistema em quais períodos de tempo? Que partes do software parecem instáveis durante longos períodos de tempo?

Como muitos sistemas não possuem documentação e nem sempre é possível comunicar-se com os desenvolvedores iniciais do projeto, atividades de manutenção se tornam mais complexas devido à falta de compreensão sobre o sistema a ser tratado. Por isso,

as funções providas pelo GEVOL podem ser consideradas de grande utilidade. A Figura 18 apresenta uma visão geral da ferramenta.

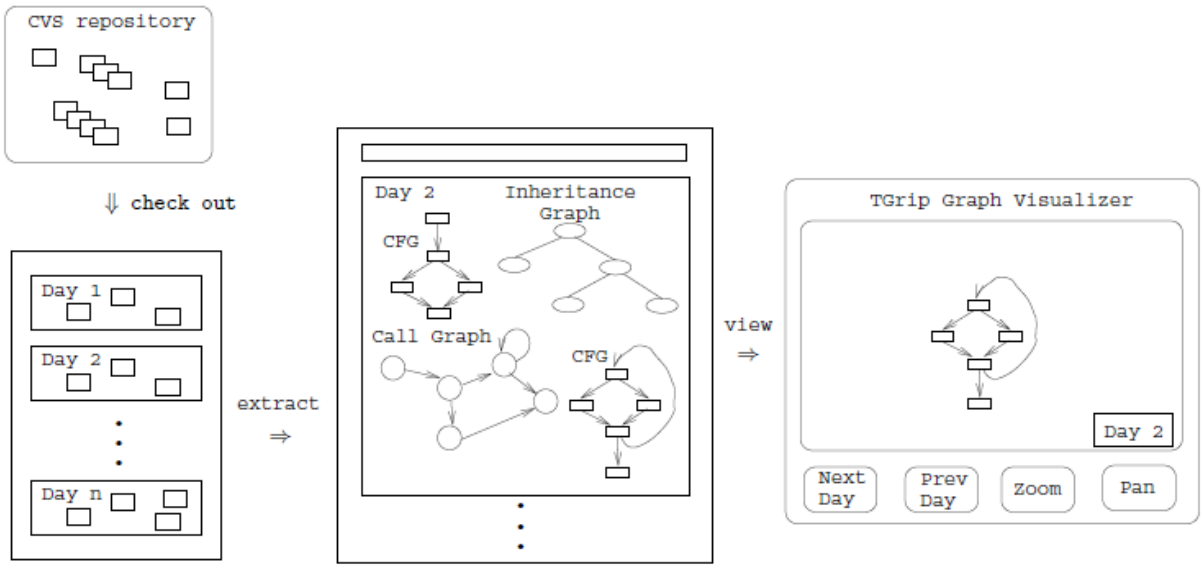


Figura 18. Visão geral do GEVOL (COLLBERG *et al.*, 2003)

Na construção do GEVOL, buscou-se não apenas a geração de fotos instantâneas do código, mas também a capacidade de visualizar toda a história do processo de desenvolvimento. Isso pode levar a ideias interessantes que não poderiam ser observadas por meio da análise de apenas um instante do tempo. Desta forma, o objetivo foi desenvolver um sistema que permitisse a visualização de todos os aspectos evolutivos do programa, expressando-os como grafo. São extraídas as seguintes informações:

- O autor de cada mudança em cada arquivo.
- Os grafos de controle de fluxo para cada método do programa.
- A mudança em cada bloco básico nos grafos de controle de fluxo.
- O grafo de herança do programa.
- O grafo de chamada de métodos do programa.
- O tempo de cada mudança em cada arquivo.

Cada parte da informação é coletada em cada intervalo de tempo. A granularidade temporal é configurável, porém, na versão analisada do sistema, este tamanho equivale a um dia. Com estas informações, o sistema possibilita alguns tipos de visualização:

- Os nós são coloridos de acordo com o tempo em que eles não foram modificados. Todos começam com a cor vermelha e vão perdendo a coloração conforme a ausência de alterações no decorrer do tempo, chegando até o tom azul.
- Quando o usuário se interessa por um determinado trecho do grafo, ele pode detalhá-lo mais, observando os autores que atuaram naquela parte, por meio de um clique no grafo.
- Ao observar que uma área se mantém constantemente vermelha, porém não cresce, é possível perceber que é uma área candidata a ser um foco de erros e, por isso, pode necessitar de mais testes e/ou atenção.

A Figura 19 ilustra o uso da ferramenta, por meio da visualização do grafo de herança, onde as classes foram nomeadas no grafo.

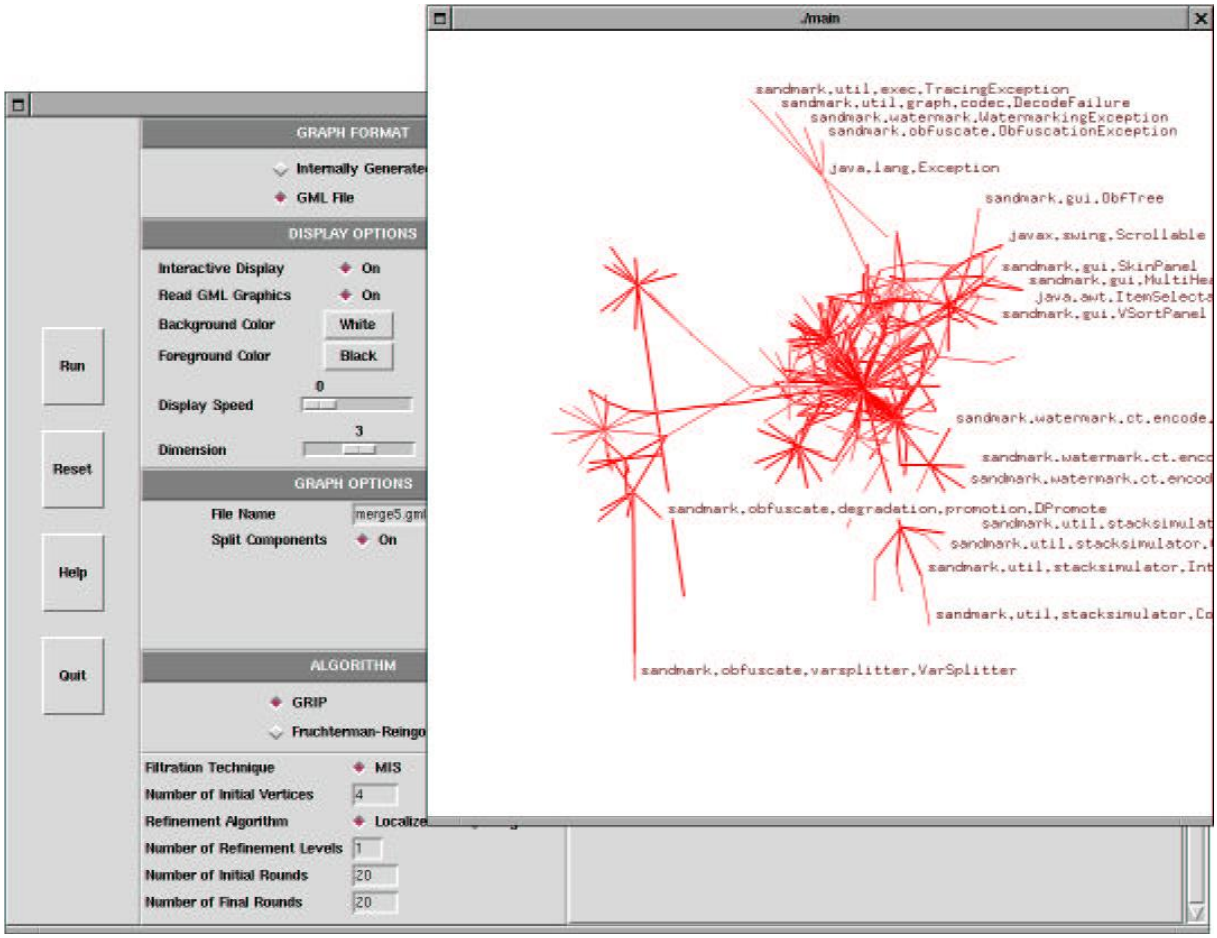


Figura 19. Grafo de herança exibido pelo GEVOL (COLLBERG *et al.*, 2003)

4.6. Vizz3D

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 renderização 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.

O sistema é organizado segundo a estrutura, exibida na Figura 20, onde as classes principais (VizzJ3D e VizzGL) são derivadas de uma classe denominada *PlugIn*. Todo *plugin* utilizado deve herdar esta classe. As classes principais iniciam o *MainFrame* (janela principal do programa), criam um *FileLoader* (que carrega o grafo) e um *GraphicsHandler* (que por sua vez, cria o cenário e constrói a estrutura do grafo e o adiciona ao *canvas*, área de exibição do grafo).

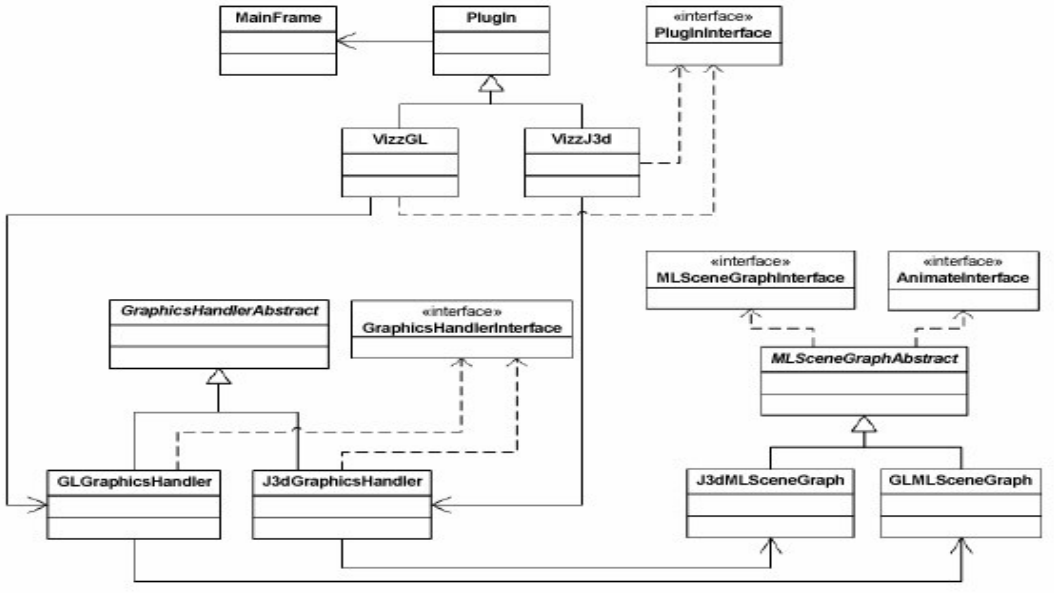


Figura 20. Estrutura do Vizz3D (CARLSSON, 2006)

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 no *MainFrame* que contém menus, ícones, painéis e o *canvas*, conforme ilustra a Figura 21.

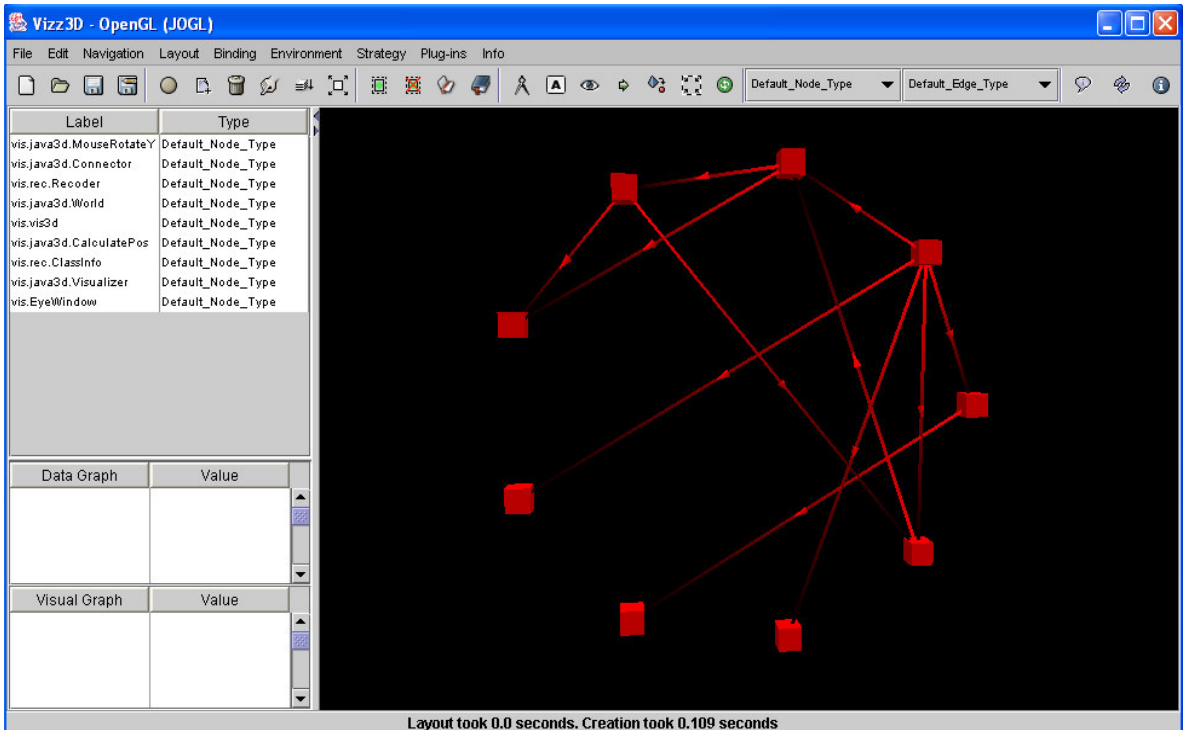


Figura 21. *MainFrame* do Vizz3D (CARLSSON, 2006)

4.7. CVSscan

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 22 ilustra algumas colorações utilizadas para cada perspectiva.

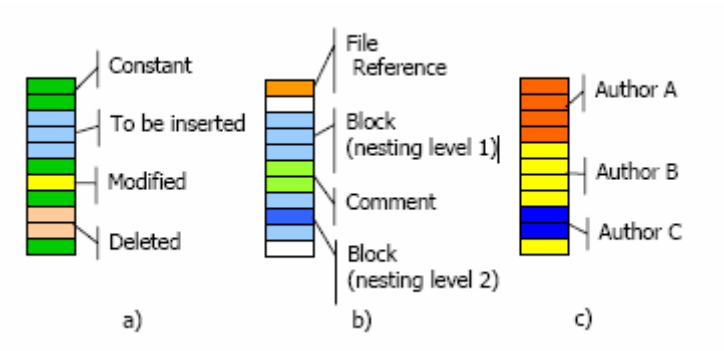


Figura 22. Colorações utilizadas: status da linha (a), tipo de construção (b) e autor (c) (VOINEA *et al.*, 2005)

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 da tupla (identificação da versão, autor, data, código). Então, para comparar versões consecutivas de um determinado arquivo, a ferramenta utiliza uma aplicação externa nos moldes de um *diff* (HUNT & MCILROY, 1976) (HUNT & SZYMANSKI, 1977) do sistema operacional UNIX. 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 23 ilustra como ocorre o processo de visualização na ferramenta. É interessante notar que a ferramenta não utiliza endentação 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.

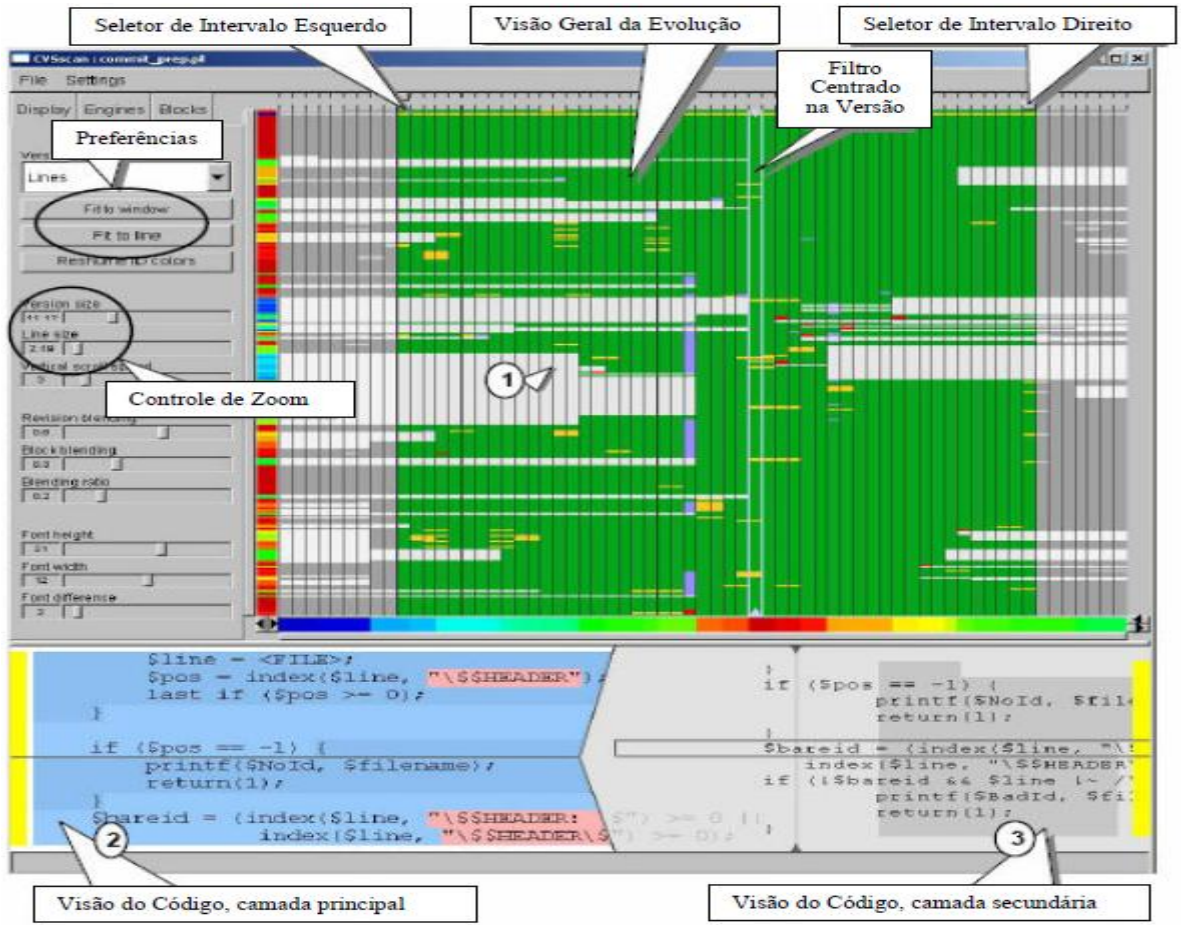


Figura 23. Visão geral da ferramenta CVSscan (VOINEA *et al.*, 2005)

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 23. 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 "*Seletor de Intervalo Esquerdo"* e do "*Seletor de Intervalo Direito"*), 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 como suas linhas foram acrescidas, removidas ou alteradas ao longo da execução do projeto, o que limita o potencial de visualização de um conjunto de informações.

4.8. softChange

Tomando como ponto de partida as limitações estabelecidas pelo sistema de controle de versão CVS no que tange ao entendimento da evolução dos projetos sob seu controle, foi criada a ferramenta softChange. Esta tem por objetivo aproveitar as informações armazenadas do projeto e, a partir destas, apresentar diferentes características do software ao longo do tempo.

A abordagem da ferramenta define como **rastros de software** as informações deixadas pelos contribuidores de um determinado projeto ao longo do processo de desenvolvimento. Alguns exemplos destas informações são listas de e-mail, *web sites*, registros do sistema de controle de versão, *releases* de software, documentação e código fonte, dentre outros. O objetivo da ferramenta é, a partir de repositórios CVS, transformar estes rastros de software, utilizando determinadas heurísticas, em informações de mais alto nível que serão posteriormente apresentadas para o usuário final. Por exemplo, pode ser apresentado, ao longo da linha do tempo, um reagrupamento do conjunto de revisões dos arquivos do projeto para um dado evento de *check-in* no repositório. Isto se torna útil neste cenário, uma vez que o sistema CVS não é orientado a transação, ou seja, o mesmo não mantém qualquer tipo de rastro ou ligação entre os diferentes arquivos modificados em um dado *check-in*.

A partir deste processo de extração dos rastros de software e da reconstrução de todos os eventos de *check-in* do projeto, uma etapa de análise é realizada. Nesta, classifica-se o evento como sendo uma adição de nova funcionalidade, uma reorganização de código fonte, um simples comentário, etc. Por fim, após estas atividades de extração e análise, o softChange provê uma representação gráfica destas informações. A ferramenta também disponibiliza uma

representação destas informações utilizando documentos que contenham hipertextos para serem visualizados em navegadores *web*.

A arquitetura da ferramenta pode ser observada na Figura 24. Nesta, nota-se a existência de quatro componentes básicos: repositório de rastros de software, extrator de rastros de software, analisador de rastros de software e um visualizador.

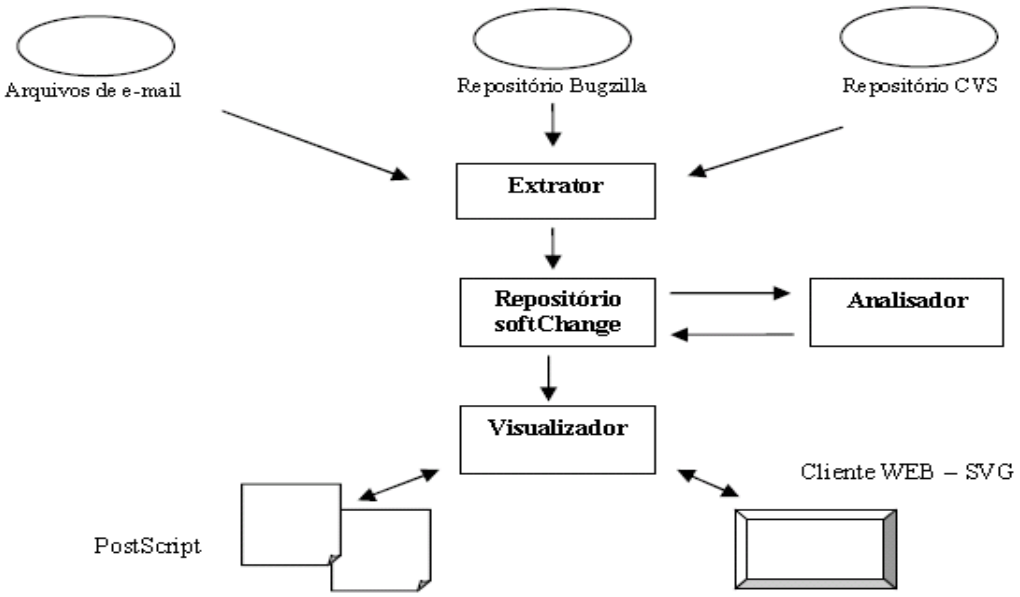


Figura 24. Arquitetura da ferramenta softChange (GERMAN *et al.*, 2006)

O repositório de rastros de software é um banco de dados relacional responsável por armazenar todas as informações históricas extraídas sobre o projeto. O extrator de *rastros de software* é o componente responsável por resgatar as informações de projeto das fontes de dados suportadas. Nesta implementação, as fontes de dados suportadas são o repositório CVS, arquivos do tipo *ChangeLog*, *releases* do software (através de arquivos compactados distribuídos pelos membros da equipe) e o sistema Bugzilla (BUGZILLA, 2010). O componente analisador de *rastros de software* é responsável por aplicar as heurísticas definidas pela abordagem e, a partir das informações previamente coletadas, inferir novas informações e características de software. Por exemplo, este componente possui a inteligência de correlacionar eventos de *check-in* realizados ao longo do tempo com problemas cadastrados na ferramenta Bugzilla.

O último componente é o visualizador. Este é divido em duas partes: um interpretador de hipertexto (i.e. *browser*) e um visualizador gráfico. O primeiro é utilizado pelo usuário para navegar através dos diversos *check-ins* realizados no projeto ao longo do tempo. Esta navegação pode ser realizada por data, autor ou arquivo. Em uma navegação por data, por

exemplo, o softChange provê informações acerca de quais revisões dos arquivos pertencem a cada *check-in*, além de outros metadados associados à modificação.

O visualizador gráfico também pode ser subdividido em duas partes principais. A primeira utiliza arquivos Postscript para gerar gráficos estáticos dos *rastros de software*. Um exemplo de gráfico gerado em Postscript é ilustrado na Figura 25 (neste caso, o autor chama de MR o que está sendo chamado de *check-in* neste texto). A segunda utiliza os recursos de SVG – *Scalable Vector Graphics* (SVG, 2010), – para apresentar tais informações de forma interativa.

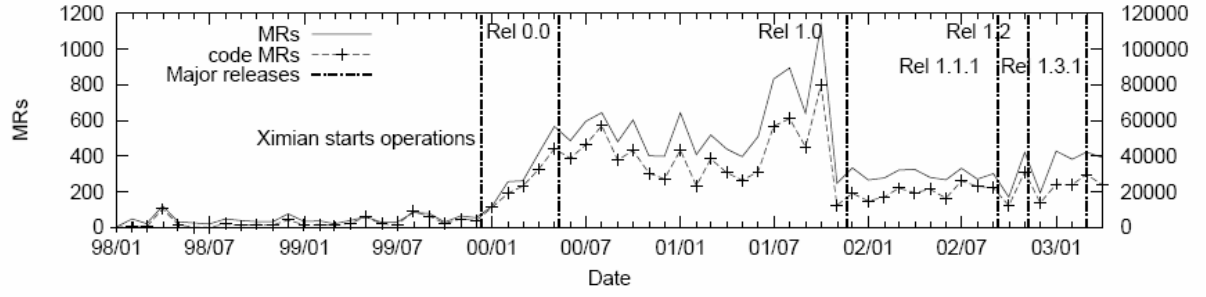


Figura 25. Exemplo de um gráfico gerado pelo softChange para o projeto *open source* Ximian (GERMAN *et al*., 2006)

A ferramenta softChange apresenta como principais contribuições a integração com mais de uma fonte de dados para a coleta de informações acerca do projeto e uma arquitetura modular na qual papéis e responsabilidades foram bem definidos, aumentando a facilidade de sua extensão. Além disto, proporciona um método para a visualização gráfica de diferentes propriedades de um dado projeto de software, como: crescimento de linhas de código ao longo do tempo, número de *check-ins* realizados ao longo do tempo, número de arquivos criados ao longo do tempo e número de arquivos por evento de *check-in*, dentre outros.

4.9. Análise dos trabalhos relacionados

Os critérios abaixo descritos foram definidos de acordo com as características comuns encontradas em todas as ferramentas pesquisadas e de acordo com o conjunto de características necessárias para uma abordagem reutilizável e extensível para a visualização da evolução de métricas de software. Desta forma, as ferramentas descritas neste capítulo foram comparadas de acordo com os seguintes critérios:

- **C1. Tipo de fonte de dados utilizada:** Indica que outra ferramenta externa armazena os dados utilizados sobre o sistema em análise.
- **C2. Granularidade das informações apresentadas:** Indica o nível de granularidade das informações extraídas do sistema em análise que são apresentadas.
- **C3. Integração com algum ambiente de desenvolvimento de software:** Indica se a ferramenta possui algum tipo de integração com ambientes de desenvolvimento de software, podendo extrair, alterar ou apresentar informações destes/nestes ambientes.
- **C4. Tipo de métricas analisadas:** Indica que tipo de propriedade é analisado pelo software.
- **C5. Tipo de visualização utilizada:** Indica que tipo de recurso visual foi adotado para representar as características do sistema em análise.
- **C6. Análise Temporal:** Indica se a abordagem em questão proporciona ao seu usuário a análise de informações da evolução do sistema em espaços discretos do tempo, em tempo real, ou ambas as estratégias.

A Tabela 1 classifica e caracteriza as ferramentas de acordo com os critérios enunciados. A classificação dos critérios C1, C4, C5 e C6 corresponde a uma descrição por extenso do nome de sistemas, tecnologias ou metodologias utilizadas, que atendam ao critério em questão. A classificação do critério C2 é dada como alta ou baixa. Informações de alto nível do sistema em análise, como módulos, pacotes, classes e subsistemas, são classificadas como granularidade alta. Em contrapartida, a granularidade baixa está associada a informações do sistema em análise que estejam em um nível mais baixo, como métodos, atributos ou até mesmo linhas de código-fonte sem qualquer tipo de classificação. Adicionalmente, o critério C3 é respondido apenas como "sim" ou "não".

	FERRAMENTAS							
CRITERIOS	**SeeSoft**	**MetricView**	**Tarantula**	**EPOSpix**	**GEVOL**	**Vizz3D**	**CVSscan**	**softChange**
C1	Usuário	Usuário	Usuário	eROSE	CVS	VizzAnalyzer	CVS	CVS
C2	Baixa	Alta	Baixa	Alta	Baixa	Alta	Baixa	Baixa
C3	Não	Não	Não	Sim	Não	Não	Não	Não
C4	Código e Tamanho	Variadas	Código e Teste	Acoplamento	Código	Variadas	Código	Tamanho
C5	Formas e Escala de Cores	Gráficos de barra, pizza, 2D e 3D	Formas e Escala de Cores	Mapa de *pixels*	Grafos	Grafos e Formas 3D	Gráfico Baseado em Linhas	Gráfico Baseado em *Check-in*
C6	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto

Tabela 1. Quadro comparativo entre as ferramentas analisadas

A partir desta comparação, conclui-se que as ferramentas pesquisadas, de um modo geral, estão vinculadas a determinados sistemas de controle de versão ou dependentes do usuário para captar as informações do sistema que deverá ser analisado (salvo aquelas ferramentas que utilizam outra como fonte externa de dados, como no caso da EPOSpix e da Vizz3D). Assim, observando a dependência do esforço do usuário e a constante evolução tecnológica, bem como possíveis mudanças no ferramental utilizado na gerência de configuração, estas características se tornam um fator de risco para a utilização destas ferramentas.

Desta forma, faz-se necessária a criação de uma abordagem alternativa que realize a etapa de extração de informações de projetos de software de forma automática e que possibilite a utilização de outros tipos de fonte de dados, com pouco ou até mesmo sem nenhum impacto na arquitetura da ferramenta. Assim, a análise comparativa das ferramentas serviu como base para a construção dos requisitos que este trabalho contempla. O próximo capítulo apresenta a abordagem proposta por esta monografia (denominada IAVEMS), que envolve a utilização de técnicas de visualização almejando explorar a capacidade de cognição e raciocínio do ser humano para facilitar a compreensão de informações advindas da evolução de sistemas de software, em especial, de métricas de software.

Capítulo 5. Abordagem Proposta: IAVEMS

5.1. Introdução

Neste capítulo, a abordagem proposta é apresentada por meio de uma descrição de alto nível, incluindo um diagrama de componentes UML e um modelo conceitual, contendo os principais elementos da abordagem. Adicionalmente, são apresentados os requisitos de uma abordagem que atenda as necessidades anteriormente discutidas e, posteriormente, como foram construídos os mecanismos que compõem a infraestrutura desenvolvida.

O restante do capítulo apresenta com mais detalhes a abordagem proposta. A Seção 5.2 apresenta uma discussão preliminar sobre a abordagem, ilustrando seus principais elementos, como requisitos e arquitetura; a Seção 5.3 mostra em detalhes como a presente abordagem foi desenvolvida; a Seção 5.4 apresenta um exemplo de utilização; a Seção 5.5 discute os principais pontos apresentados no capítulo, enfatizando as principais características da abordagem.

5.2. Abordagem IAVEMS

A abordagem IAVEMS (Infraestrutura de Apoio à Visualização da Evolução de Métricas de Software) visa apoiar a compreensão da evolução de métricas de software. Desta forma, o acompanhamento da evolução do próprio software pode ser realizado com maior consistência e exatidão. Como a maior parte dos projetos de software encontra-se armazenada em repositórios de controle de versão, a abordagem IAVEMS, proposta neste trabalho, busca, por meio de um único conector de fonte de dados genérico, uma forma de se conectar com estes repositórios independentemente de sua implementação (contanto que as atividades básicas de controle de versão providas por eles estejam bem definidas). Assim, um dos objetivos desta abordagem foi desenvolver um conector que realizasse uma interface com estes repositórios, de forma que esta fosse extensível para reduzir o impacto da introdução de novas tecnologias nesta área.

Também é possível notar que as ferramentas analisadas usualmente se concentram em determinadas propriedades do software, ficando limitadas a certos tipos de métricas. Para contornar isso, procurou-se o desenvolvimento de mecanismos que fossem aplicados a

diversos tipos de métricas, privilegiando o reúso da abordagem em mais de um contexto. A seguir, serão detalhados os requisitos e a arquitetura especificados para este trabalho.

5.2.1. Requisitos da abordagem

Em face do que foi apresentado nos capítulos anteriores, um determinado conjunto de requisitos funcionais, representando características desejadas da abordagem, foi elaborado de forma a contemplar os objetivos deste trabalho. Estes foram divididos em requisitos de integração, fonte de dados e visualização.

Requisito de Integração

- **R1:** Manter compatibilidade com a ferramenta EvolTrack;

Requisitos de Fonte de Dados

- **R2:** Prover uma interface externa para que outros componentes possam utilizar os recursos de acesso a repositórios de software;

- **R3:** Prover uma interface para a conexão com diferentes Sistemas de Controle de Versão (SCVs) de forma a contemplar uma maior diversidade de projetos de software;

- **R4:** Para uma captura automática da informação de projetos de software, realizar a Engenharia Reversa de código, evitando a necessidade de interferência do usuário nesta fase de coleta. Como isto tornaria o escopo da ferramenta muito abrangente, será utilizada somente a linguagem Java para este processo, podendo haver expansões posteriores;

- **R5:** Identificar a estrutura de pacotes para o agrupamento da informação estrutural[2];

[2]No contexto deste trabalho, entende-se por visualização estrutural a visualização do conjunto de classes e interfaces do sistema.

Requisitos de Visualização

- **R6:** Visualizar a estrutura de um projeto de software com base no agrupamento e detalhamento da informação;

- **R7:** Visualizar a evolução de métricas de software com base no princípio de foco + contexto;

- **R8:** Perimtir, por meio da visualização integrada, uma análise comparativa da evolução de métricas de software com base na interação com o usuário;

- **R9:** Visualizar a evolução de métricas de software aliada à estrutura do mesmo, por meio de técnicas tridimensionais de visualização.

É importante notar que o requisito R1 cria uma dependência da integração com o ambiente de desenvolvimento de software Eclipse (ECLIPSE FOUNDATION, 2010a), pois este é o ambiente nativo da ferramenta que abrigará os mecanismos da IAVEMS. Por estar integrada a um ambiente utilizado em uma grande quantidade de projetos de desenvolvimento de software, esta abordagem se torna passível de utilização por parte de qualquer desenvolvedor que tenha algum conhecimento na plataforma Eclipse ou ferramentas similares. Desta forma, é aumentada a compatibilidade com boa parte dos projetos de software orientados a objetos desenvolvidos atualmente.

Além disto, buscando a reutilização, a abordagem proposta deve fornecer uma interface externa de acesso aos repositórios para que outros componentes também aproveitem o mecanismo construído. A IAVEMS também contemplará a utilização de qualquer tipo de sistema de controle de versão (conforme o requisito R2), provendo simultaneamente um controle centralizado e uma flexibilidade do mecanismo, aumentando o potencial de utilização em outros cenários, isto é, permitindo a integração com outros SCVs.

Vale ressaltar que, nos requisitos em que é mencionado o uso de métricas de software, não há restrição a métricas específicas; desta forma, a estrutura deve suportar qualquer métrica marcada pelo Transformador de Modelos.

5.2.2. Arquitetura da infraestrutura

Como necessidade para suportar o requisito R1, a infraestrutura desenvolvida precisou seguir a arquitetura da ferramenta onde ela está inserida, o EvolTrack. Assim, a IAVEMS se divide em dois mecanismos:

- **EvolTrack-VCS:** Este componente tem a função do Conector de Fonte de Dados citado na arquitetura do Evoltrack. Sua responsabilidade é a captura de informações de diferentes projetos de software armazenados em diferentes tipos de sistemas controladores de versão. Deve atender aos requisitos R2, R3, R4 e R5, além de ser desenvolvido como um *plugin* para a plataforma Eclipse.

- **EvolTrack-MetricEView:** Este componente tem a função do Conector de Visualização, cabendo a este transformar as informações a cerca da evolução de medidas do projeto de software em uma representação visual que facilite a compreensão do desenvolvimento do mesmo. Deve cobrir os requisitos R6, R7, R8 e R9, além de também ser desenvolvido como um *plugin* para a plataforma Eclipse.

Desta forma, uma visão geral da abordagem proposta é apresentada na Figura 26, onde os principais elementos e suas interações são ilustrados. De maneira geral, o EvolTrack-VCS efetua as seguintes tarefas: (i) captura as informações de projeto de um sistema controlador de versão; (ii) realiza a engenharia reversa, gerando como saída modelos UML; (iii) envia cada modelo para o Transformador de Modelos, que transforma o modelo agregando, por meio de marcações (estereótipos UML) algumas métricas e seus respectivos valores; (iv) registra cada modelo marcado no Núcleo do EvolTrack; (v) transmite o modelo para o Evoltrack-MetricEView, que, por sua vez, analisa o modelo e suas marcações, gerando representações visuais de forma a atender a diferentes pontos de vista.

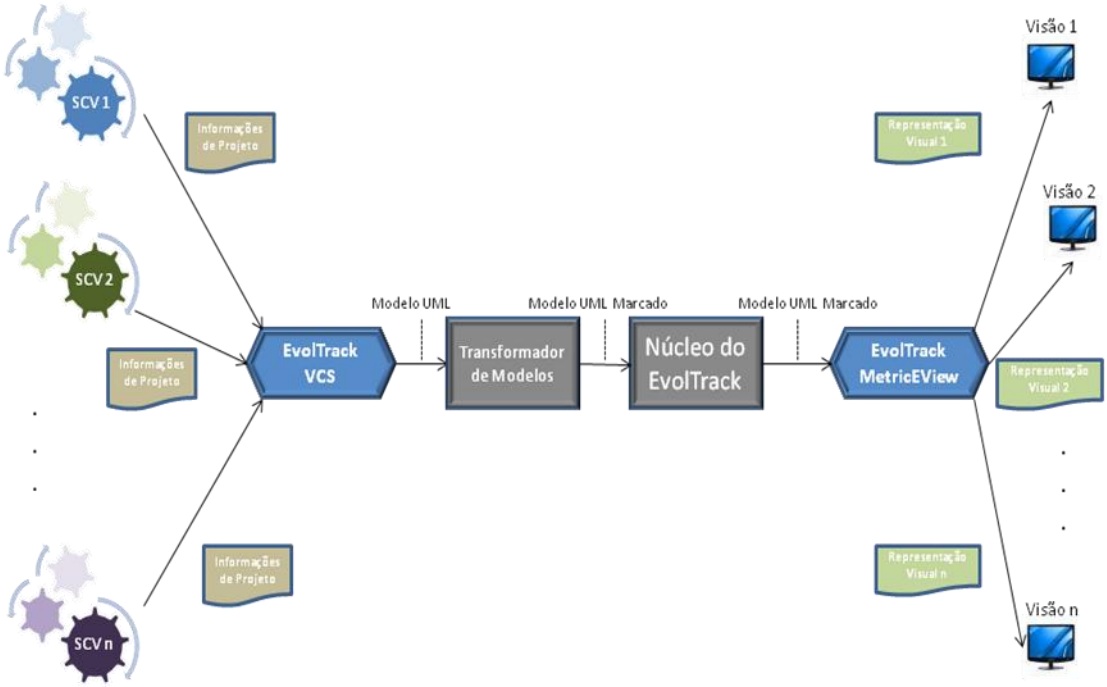


Figura 26. Visão Geral da Abordagem IAVEMS

5.3. Implementação

O início deste capítulo apresentou de forma funcional a abordagem proposta por este trabalho e os requisitos a serem atendidos. Esta seção tem por objetivo apresentar sob a perspectiva técnica o projeto e a implementação dos pontos discutidos anteriormente. Isto é, são abordados os detalhes de como cada conceito discutido e definido pela IAVEMS foi implementado através de um sistema de software.

Por estar inserida na ferramenta EvolTrack, a abordagem proposta também se insere como *plugin* para a plataforma Eclipse. Este ambiente integrado de desenvolvimento não corresponde a um programa monolítico, mas sim, a um núcleo reduzido, também conhecido como *Plugin Loader*, cercado por centenas de outros *plugins* (CLAYBERG & RUBEL, 2006). A Figura 27 apresenta resumidamente esta arquitetura. Observe que a plataforma corresponde a um pequeno conjunto de componentes que cooperam entre si para formar a funcionalidade básica da mesma e que, adicionalmente, montam um *framework* para a conexão de novas funcionalidades na forma de *plugins*.

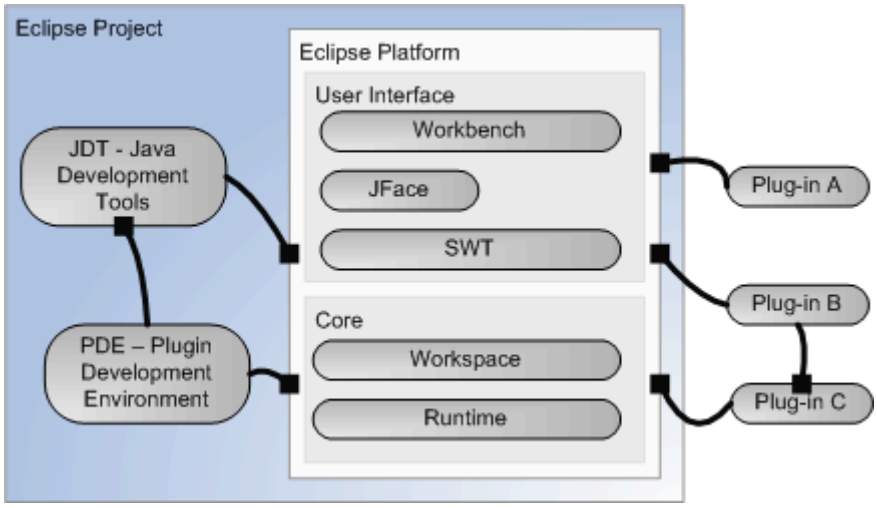


Figura 27. Arquitetura simplificada da Plataforma Eclipse (ECLIPSE ARCHITECTURE, 2010)

Neste contexto, para viabilizar a conexão de novas funcionalidades (ou *plugins*) desenvolvidas na IAVEMS, a seguinte estratégia foi adotada: dois arquivos, manifest.mf (vide Figura 28) e plugin.xml (vide Figura 29), devem ser criados e disponibilizados para a plataforma. No primeiro, o desenvolvedor do *plugin* especifica informações como a versão do mesmo, as relações de dependência com outros *plugins*, informações de visibilidade (i.e. que partes do *plugin* poderão ser acessadas por outros *plugins* da plataforma), informações de carga, dentre outras que serão utilizadas tanto para a execução como para um possível site de atualização (*Update Site*) do Eclipse. Por exemplo, na Figura 28 podem ser observados, através do atributo "Export-Package", todos os pacotes que este *plugin* permitirá acesso externo. Desta forma, pacotes que não estejam listados neste atributo não poderão ter seus recursos acessados por outros *plugins* instalados no Eclipse, facilitando, portanto, a utilização de técnicas de encapsulamento e ocultação da informação pelos desenvolvedores destes sistemas.

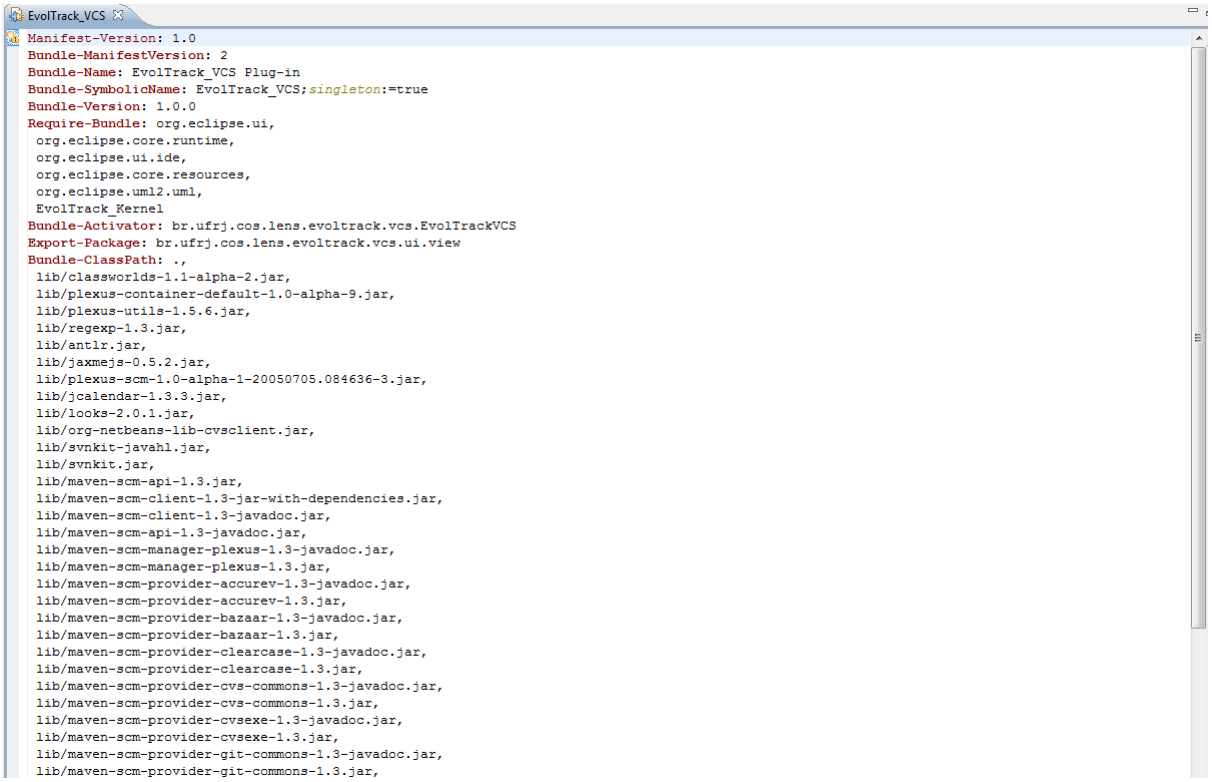


Figura 28. Arquivo manifest.mf utilizado pelo *plugin* Evoltrack-VCS

Já o segundo arquivo, plugin.xml, contém a descrição propriamente dita do *plugin*. A partir deste arquivo, a plataforma obtém o ponto de entrada do *plugin* e o conjunto de funcionalidades que este deverá fornecer para o restante da plataforma. No exemplo apresentado pela Figura 29, observa-se que o ponto de entrada do *plugin* em questão é representado pela classe "br.ufrj.cos.lens.evoltrack.vcs.connector.VCSConnector". Para tal, esta classe deve obrigatoriamente estender a classe abstrata Plugin ou alguma subclasse da mesma, como por exemplo, a classe abstrata AbstractUIPlugin, utilizada por *plugins* que de alguma forma estenderão as funcionalidades de interface com o usuário da plataforma Eclipse.

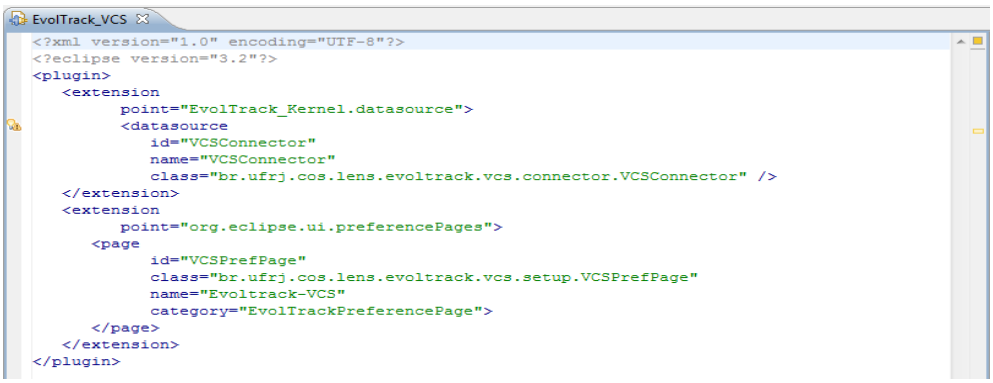


Figura 29. Arquivo plugin.xml utilizado pelo *plugin* EvolTrack-VCS

Dois novos *plugins* para a plataforma Eclipse foram construídos e incorporados ao EvolTrack, cada um correspondendo a um módulo funcional descrito na seção 5.2.2 – ou seja, um *plugin* para representar o módulo Conector de Fonte de Dados (Evoltrack-VCS), outro para o módulo Visualizador (EvolTrack-MetricEView). Uma visão geral da abordagem, representada por um diagrama de componentes, pode ser vista na Figura 30.

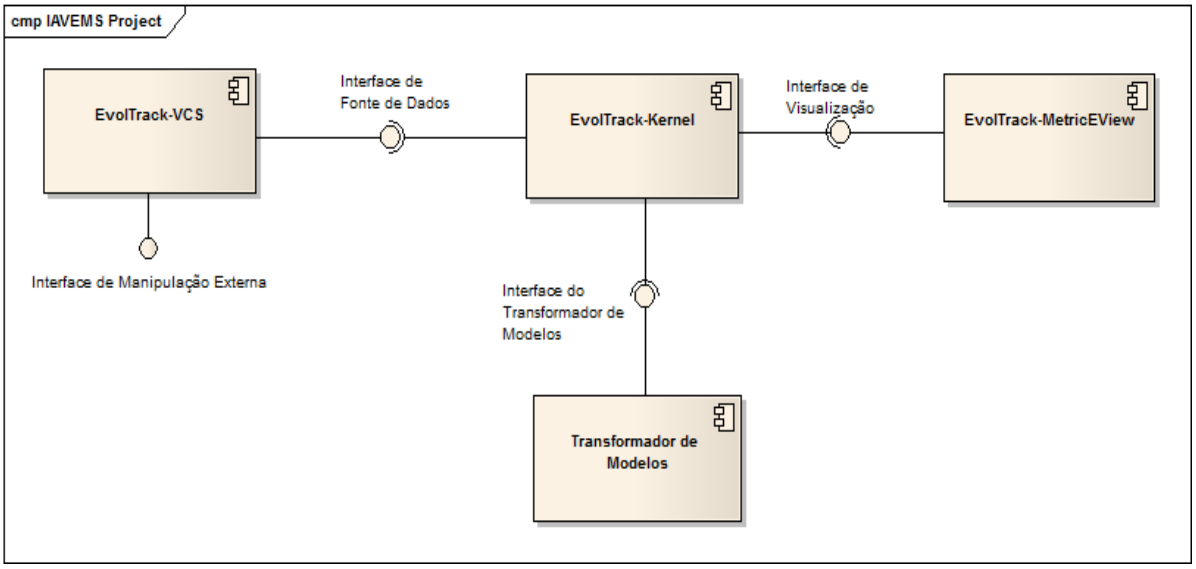


Figura 30. Diagrama de componentes da abordagem

5.3.1. EvolTrack-VCS

Motivado pelos benefícios de um novo conector que não fosse acoplado a um sistema de controle de versões específico (dificuldade encontrada pela ferramenta EvolTrack e pelas ferramentas analisadas no capítulo 4), o EvolTrack-VCS foi construído a partir da utilização da API (*Application Program Interface*) denominada Maven SCM (MAVEN SCM, 2010), um software livre que provê mecanismos para o uso e integração de ferramentas de gerência de configuração por meio de interfaces genéricas. Atualmente, o conector fornece interface para doze SCVs: ACCUREV (2010), BAZAAR (2010), CLEARCASE (2010), CVS (2010), GIT (2010), MERCURIAL (2010), PERFORCE (2010), STARTEAM (2010), SUBVERSION (2010), SYNERGY (2010), VISUAL SOURCE SAFE (VSS, 2010) e TEAM FOUNDATION SERVER (TFS, 2010). O projeto do EvolTrack-VCS, auxiliado pela estrutura da API utilizada, visou a extensibilidade para que a arquitetura da ferramenta permitisse que outros SCVs pudessem ser posteriormente adicionados com o passar do tempo, bastando que a interface requerida da API fosse implementada. É importante enfatizar que

alguns desses SCVs são proprietários; desta forma, apesar de a API Maven SCM ter sido utilizada para integração com estes SCVs, não foi possível testar seu funcionamento no contexto do EvolTrack, sendo testados somente os SCVs fundamentados em projetos de software livre.

Este componente pode operar em dois modos de extração: (i) **opção de tempo real ativada**, na qual o conector busca, em determinados intervalos de tempo, por uma versão mais nova do software no repositório (caso haja) para adicioná-la imediatamente ao fluxo de evolução. Esta estratégia funciona buscando novas versões no SCV (*pooling*), podendo ser melhorada para que o SCV informe automaticamente sobre as mudanças (*hooks*), e (ii) **opção de tempo real desativada**, que trabalha apenas com as versões selecionadas previamente. A Figura 31 mostra a tela de configuração do conector, com os dados requeridos para o seu funcionamento, incluindo a seleção do SCV (a), a opção de extrator de tempo real (b) e o navegador para escolha das versões (c).

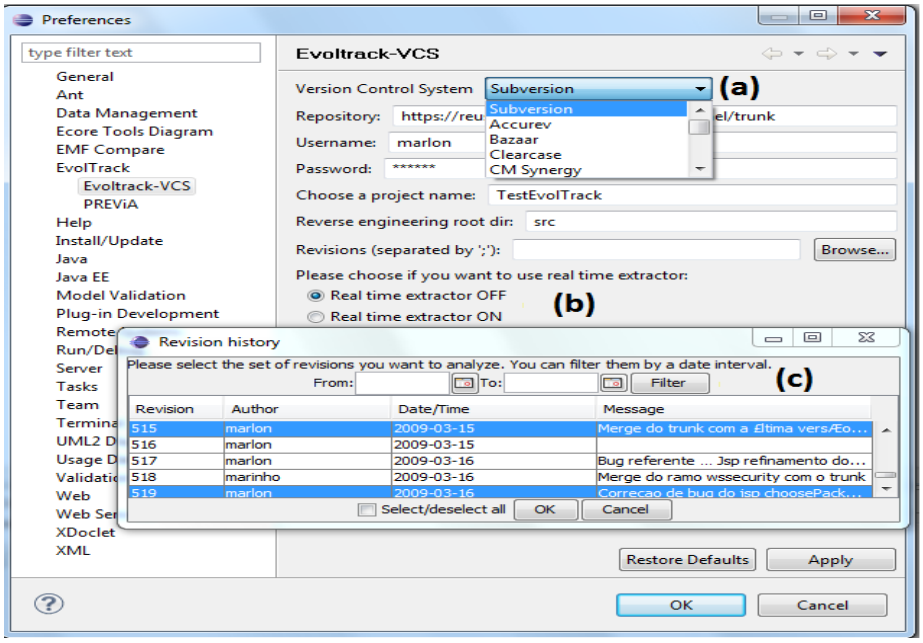


Figura 31. Tela de configuração do Evoltrack-VCS

Este navegador auxilia o usuário na escolha das revisões que serão consideradas pela ferramenta para análise. Isto permite que pontos específicos do ciclo de desenvolvimento possam ser focados, selecionando a quantidade de informação adequada a ser processada (por exemplo, nas situações em que operações de *check-in* são demasiadamente frequentes, enviando alterações muito pequenas ao repositório). Algumas funcionalidades para melhor utilização do navegador são oferecidas, como filtros por data e a opção de marcar/desmarcar todas as revisões relacionadas ao endereço recebido por parâmetro.

Após a captura destas informações, o conector realiza a engenharia reversa do código-fonte obtido através do framework JaxMeJS (*JaxMe JavaSource generation framework*) (APACHE SOFTWARE FOUNDATION, 2010), um *parser* de código Java baseado no gerador de *parser* ANTLR, que pode abranger outras linguagens de programação, e constrói o diagrama de classes da versão do projeto, estruturando-o de acordo com a organização interna do mesmo (estrutura de pacotes), suas classes, interfaces e relacionamentos. A Figura 32 mostra a estrutura do modelo UML gerado pelo conector, com as marcações de métricas na forma de estereótipos (na imagem, aparecem os estereótipos *DifferenceKind*, *Precision e Recall*), realizadas por um Transformador de Modelos (SCHOTS *et al.*, 2010).

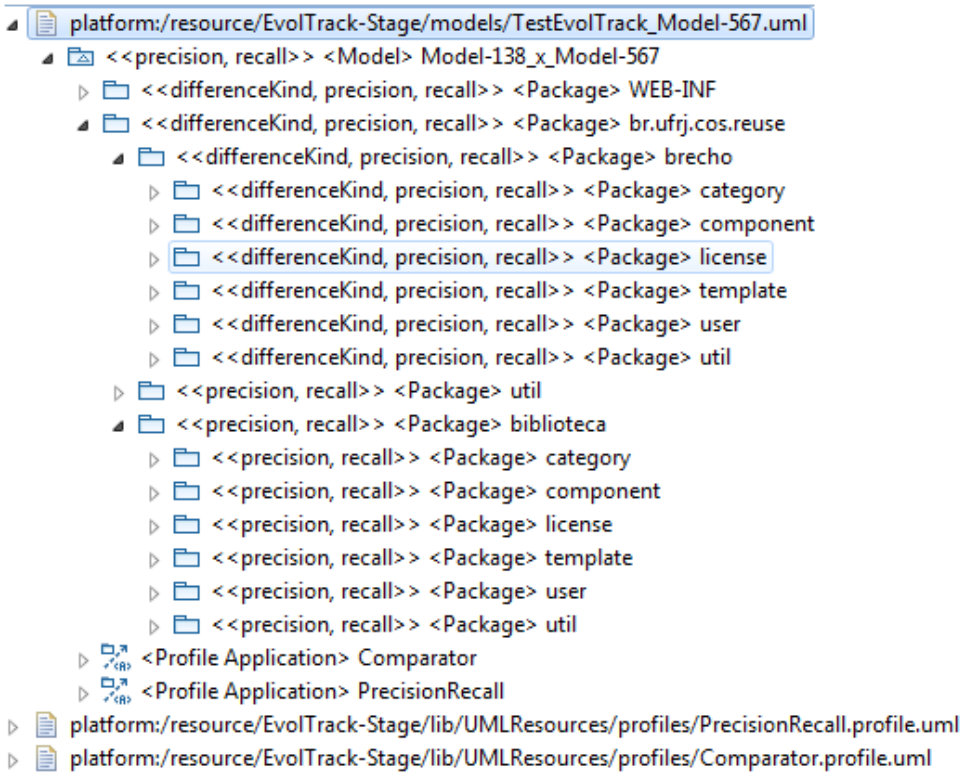


Figura 32. Modelo UML gerado pelo EvolTrack-VCS e marcado por um Transformador de Modelos

A arquitetura deste componente (Figura 33) foi dividida em alguns pacotes, a saber:

- O pacote *VCS* contém a classe de configuração do *plugin*, ou seja, aquela que, junto como o arquivo plugin.xml, fornece todas as informações necessárias para a execução no ambiente Eclipse. Esta classe estende a classe AbstractUIPlugin (Plataforma Eclipse) para executar esta tarefa; o pacote *Connector* contém a classe principal do *plugin*, isto é, a classe responsável por fazer a interface com a ferramenta EvolTrack, acionando o extrator de

informações de projeto de software e o transformador destas informações em modelos;

- O pacote *Setup* mantém o controle da tela de preferências do *plugin*, que é exibida no menu *Window* > *Preferences* do ambiente e utiliza recursos gráficos providos no pacote *UI*;
- O pacote *Extractor* é responsável pela extração (*checkout*) do código-fonte dos projetos de software e é nele que estão os métodos disponíveis como interface externa para a reutilização dos recursos de controle de versão;
- O pacote *Transformer*, por sua vez, trata de realizar a engenharia reversa do código-fonte, por meio do *parser* Java JaxMeJs. Após a transformação, o modelo UML é gerado e transmitido para o núcleo (*kernel*) do EvolTrack onde é registrado.

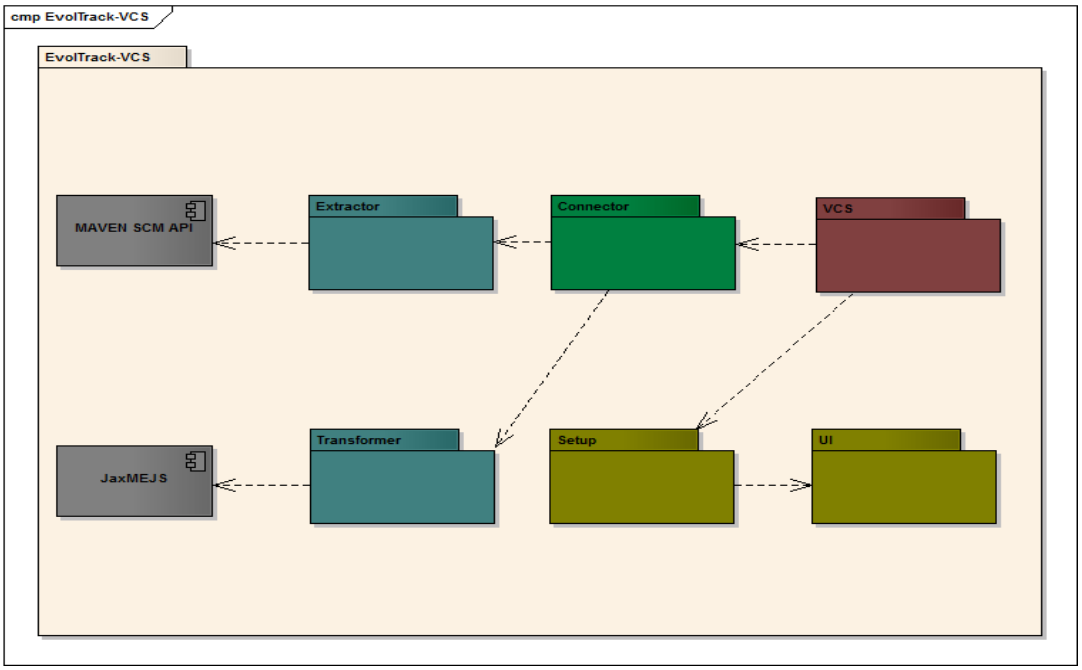


Figura 33. Arquitetura do EvolTrack-VCS

O componente Maven SCM API é internamente utilizado pelo pacote do extrator como recurso para os métodos de controle de versão requeridos, oferecendo a possibilidade de extensão para novos SCVs. Para isto, são necessárias as seguintes etapas:

- Criar uma classe que estenda org.apache.maven.scm.provider.ScmProviderRepository ou org.apache.maven.scm.provider.ScmProviderRepositoryWithHost, onde será definido o padrão de url e o tipo de sistema de controle de versão que será adicionado.

- Criar uma classe que estenda org.apache.maven.scm.provider.AbstractScmProvider. Esta irá realizar a análise da url e também a ligação dos comandos do novo SCV com as suas implementações.
- Implementar os comandos específicos do SCV e relacioná-los com os descritos na classe criada no passo anterior.

O EvolTrack-VCS também provê algumas funcionalidades na forma de uma interface externa, para que outros componentes reutilizem os mecanismos construídos, tais como:

- Realizar o *check-out* da versão HEAD (versão mais recente).
- Realizar o *check-out* de uma versão específica.
- Recuperar o histórico de modificações do SCV.
- Recuperar o histórico de informações de um determinado arquivo.

5.3.2. EvolTrack-MetricEView

Como infraestrutura para a atividade de apresentação da evolução de métricas no decorrer de um projeto de software, desenvolveu-se o EvolTrack-MetricEView. A proposta deste componente é apoiar o engenheiro de software (e outros interessados) no entendimento do processo de evolução de um determinado software. O projeto deste componente pode ser descrito pela arquitetura simplificada exibida na Figura 34.

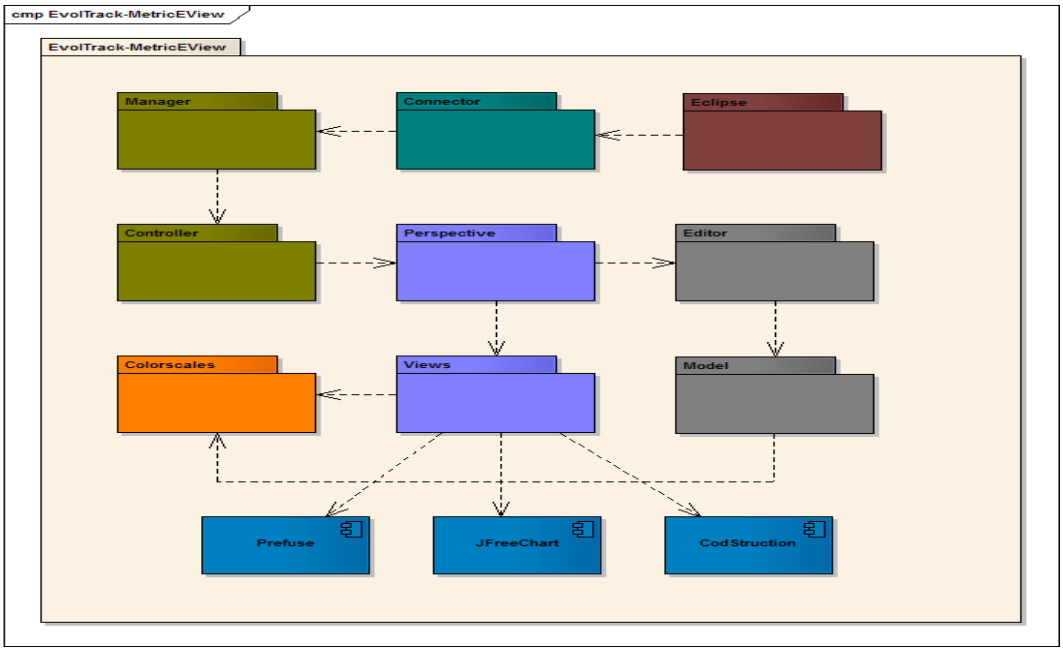


Figura 34. Arquitetura simplificada do EvolTrack-MetricEView

A arquitetura deste componente foi dividida em alguns pacotes, a saber:

- O pacote *Eclipse* contém a classe de configuração do *plugin*, ou seja, aquela que, junto como o arquivo plugin.xml, fornece todas as informações necessárias para a execução no ambiente Eclipse. Esta classe estende a classe AbstractUIPlugin (Plataforma Eclipse) para executar esta tarefa; o pacote *Connector* contém a classe principal do *plugin*, isto é, a classe responsável por fazer a interface com a ferramenta EvolTrack, acompanhando a evolução e o rastro do software analisado;
- O pacote *Manager*, que é responsável pelo gerenciamento do rastro de evolução, mantendo sob controle todos os diagramas criados;
- O pacote *Controller*, que é responsável por gerenciar os eventos ocorridos na visão gerada no ambiente Eclipse;
- O pacote *Perspective* contém a classe que descreve para a plataforma um conjunto de visões que aparecerão em conjunto para o engenheiro, formando uma perspectiva (APÊNDICE A);
- O pacote *Views* possui as diferentes visões desenvolvidas a partir de técnicas de visualização; o pacote *Editor* é o responsável pela exibição dos diagramas UML para o usuário, sendo utilizados para isso recursos do *plugin* UML2Tools (ECLIPSE FOUNDATION, 2010b);
- O pacote *Model* contém utilitários para a manipulação de entidades e elementos do domínio (modelos e diagramas UML);
- O pacote *Colorscales* trata de diversas escalas de cores que ficam à disposição do usuário para uso nos mecanismos de visualização.

A partir da arquitetura acima, foram desenvolvidos mecanismos de visualização baseados em algumas das técnicas descritas no capítulo 3, podendo ser divididas em quatro frentes:

Agrupamento da informação estrutural

Como a ferramenta EvolTrack busca visualizar a evolução da estrutura de um projeto de software, observou-se que o excesso de informação dificulta o entendimento do indivíduo.

Desta forma, aproveitando o modelo estruturado gerado pelo Evoltrack-VCS, foram utilizadas técnicas de agrupamento e detalhamento (descritas no capítulo 3) para melhorar a apresentação da estrutura de um projeto de software. Estas técnicas também serão úteis para a análise da evolução das métricas do projeto.

Assim, na visualização principal do EvolTrack após a inclusão da IAVEMS, são primeiramente apresentados os pacotes do projeto, com uma indicação do seu conteúdo para que, a partir do interesse e foco do analista, seja detalhado o conteúdo do pacote lhe convier (bastando um duplo clique sobre a representação visual do pacote). Anteriormente, todas as classes de uma determinada versão do projeto eram exibidas de uma só vez no EvolTrack (isto é, não existia a estrutura de pacote). A Figura 35 ilustra o processo de agrupamento e detalhamento desenvolvido. Para este processo, foi utilizado o projeto Brechó (WERNER *et al.*, 2007) para análise. Observa-se o mecanismo em questão ao selecionar com um duplo clique o pacote br.ufrj.cos.reuse (circulado), que imediatamente expande o conteúdo do mesmo, sendo possível observar os pacotes que compõem o pacote selecionado.

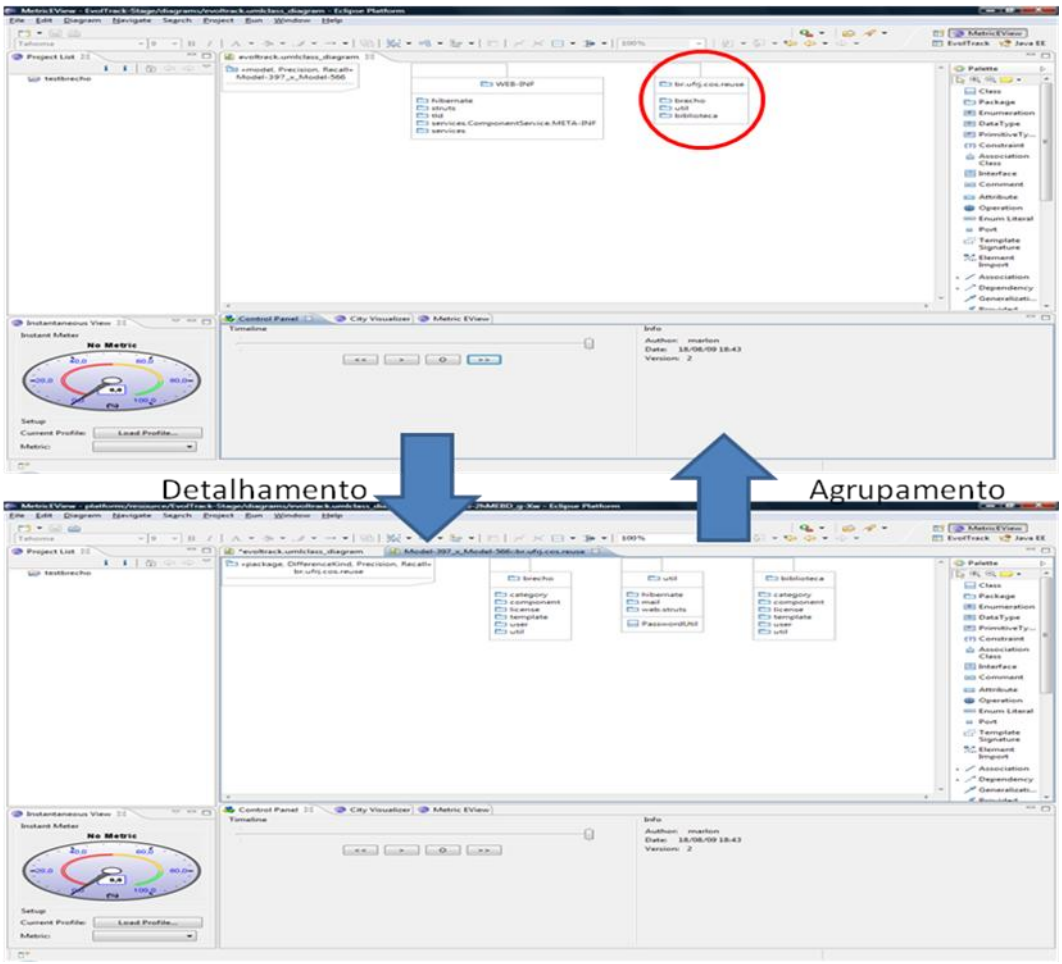


Figura 35. Ilustração do agrupamento e detalhamento da estrutura

Foco em métricas específicas

Dependendo do contexto desejado pelo engenheiro de software, o foco pode estar em uma medida ou em outra. Logo, buscou-se um mecanismo que possibilite uma visualização detalhada de uma parte da informação – o foco – que é incorporado dentro de uma visualização do contexto, isto é, a perspectiva inteira de visão do usuário. Com isso, optou-se pela biblioteca gráfica JFreeChart (JFREECHART, 2010) pela facilidade de construção de gráficos. Na utilização de métricas específicas pela IAVEMS, o foco é uma métrica (a ser escolhida pelo usuário) que já esteja marcada no modelo a partir de estereótipos da UML (tarefa realizada por um Transformador de Modelos). A Figura 36 apresenta a escolha de perfis da UML (exemplos no ANEXO I), nos quais as métricas estão descritas por meio dos estereótipos que compõem este perfil.

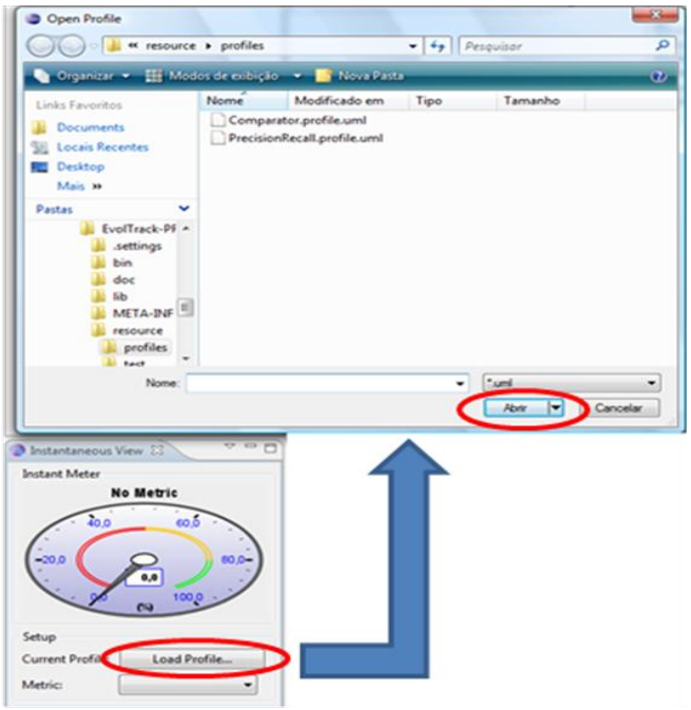


Figura 36. Seleção do perfil UML que contenha as métricas a serem utilizadas

A partir da seleção do perfil, escolhe-se a métrica desejada, cuja evolução é visualizada por meio de um velocímetro, onde o ponteiro caminha pelos valores conforme a métrica varia na evolução do software. Além disto, a ferramenta permite a instanciação de várias visões deste tipo, conforme a demanda do usuário, permitindo a observação de várias métricas em paralelo (o recurso de múltiplas visões é suportado pelo ambiente Eclipse),

caracterizando a visualização integrada. A Figura 37 demonstra a visualização após a seleção do perfil e da métrica (a área circulada abre um menu com a opção de gerar uma nova instância da visualização).

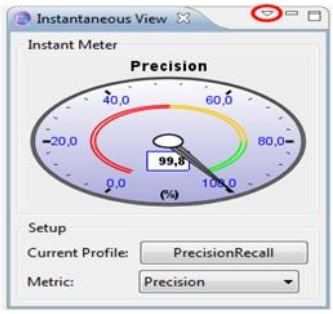


Figura 37. Visualização focada na métrica

Análise conjunta de métricas

Buscando fornecer recursos para que o usuário tenha uma visão mais ampla da evolução das métricas, podendo realizar comparações e observar tendências nos valores das mesmas, foi desenvolvido este mecanismo de visualização, pautado na interação e visão de dados históricos, no qual o usuário pode escolher uma sequência de métricas, configurando suas formas e cores. Tal visualização torna possível avaliar a influência de determinadas métricas em outras, o que pode ser determinante para escolhas de políticas corretivas ou adaptativas em alguns contextos.

Para isto, optou-se por utilizar a biblioteca gráfica de visualização da informação Prefuse (PREFUSE, 2010), devido à infraestrutura disponibilizada pela mesma para a manipulação de dados e construção de abstrações visuais sobre os mesmos. Isto agrega maior interatividade na visualização, um dos requisitos da IAVEMS.

A Figura 38 ilustra a tela de configuração do mecanismo que, similar ao anterior, parte da seleção de um perfil UML que contenha as métricas a serem visualizadas (e que devem estar marcadas no modelo). Para o exemplo, foram escolhidas as formas “estrela” e “hexágono” e as cores “oliva” e “azul” para as métricas de precisão e cobertura, respectivamente (que são detalhadas na seção 5.4).

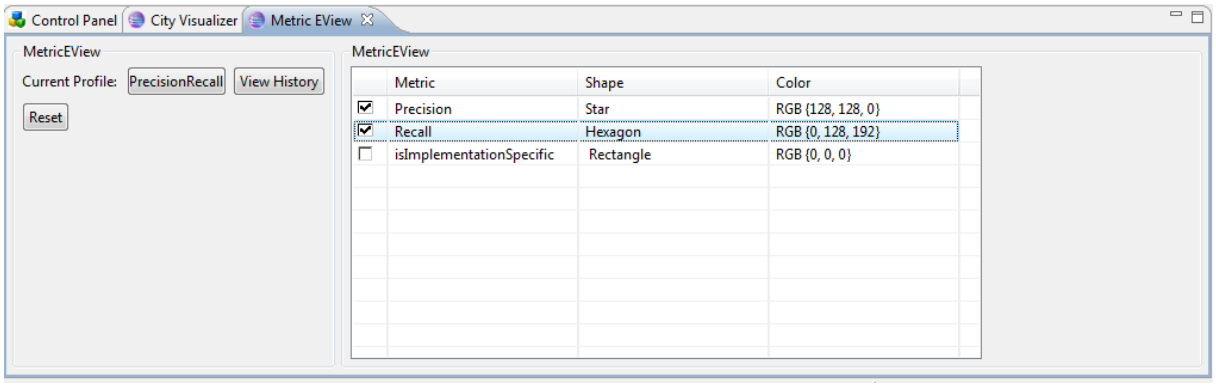


Figura 38. Tela de configuração da análise comparativa

Estas métricas são visualizadas numa janela interativa, onde é possível aplicar filtros utilizando diferentes critérios, a saber: métricas, versões e autores. Adicionalmente, recursos de *zoom* e de movimentação do gráfico (*drag*), direcionando a visão para um ponto em específico, são disponibilizados para que o usuário interaja com a informação, como é detalhado na Figura 39.

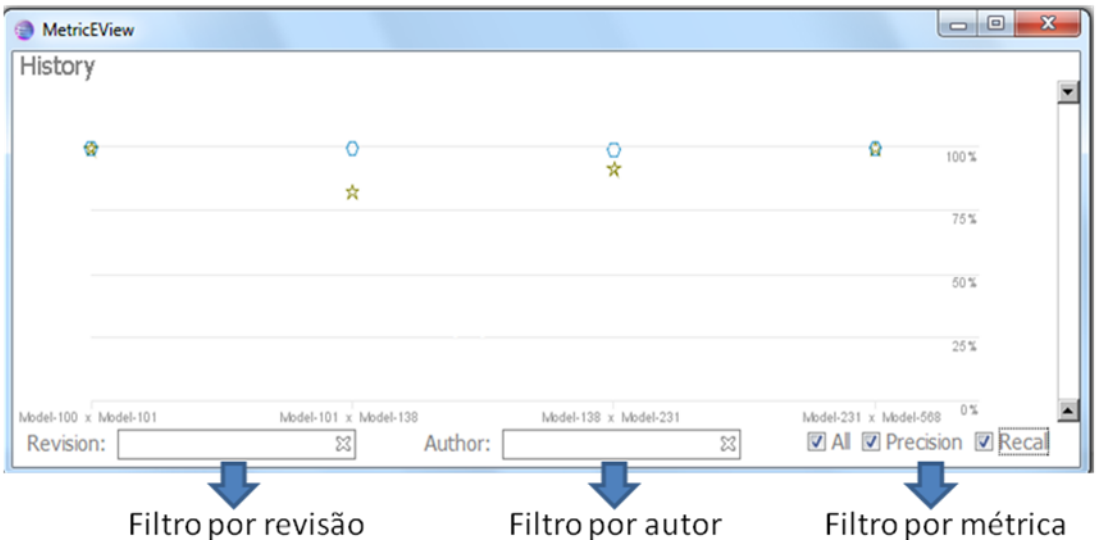


Figura 39. Visualização interativa da análise comparativa

Análise conjunta de métricas com estrutura do projeto

Para explorar uma análise envolvendo tanto métricas como a estrutura do projeto, optou-se por desenvolver um mecanismo de visualização 3D, almejando aumentar a capacidade de informação transmitida como descrito no capítulo 3. Inspirando-se no trabalho de LANGE *et al.* (2007) e estendendo o *plugin* CODSTRUCTION (2010) para Eclipse, foi desenvolvida uma visualização da organização do projeto na qual as classes e as interfaces

são construções (prédios e pirâmides, respectivamente). A visualização também permite que duas métricas (escolhidas pelo usuário) possam ser visualizadas como o tamanho e a cor da construção (cada métrica representa uma abstração).

Na Figura 40, observa-se a tela de configuração onde são selecionadas as métricas (por meio do perfil UML), tanto para representarem o tamanho como a coloração das construções. O usuário pode escolher a escala de cor a ser utilizada, sendo que esta pode ser operada em dois modos: **modo variável**, no qual, para cada valor de uma métrica, esta será interpolada na escala de cores escolhida (entre o valor mínimo e máximo da escala); **modo específico**, no qual o usuário define a cor para cada valor de métrica de interesse. Existem nove escalas de cores que foram reutilizadas do EvolTrack: BTC, BTY, Gray, Heated Object, Linear Gray, LOCS, Magenta, OCS e Rainbow.

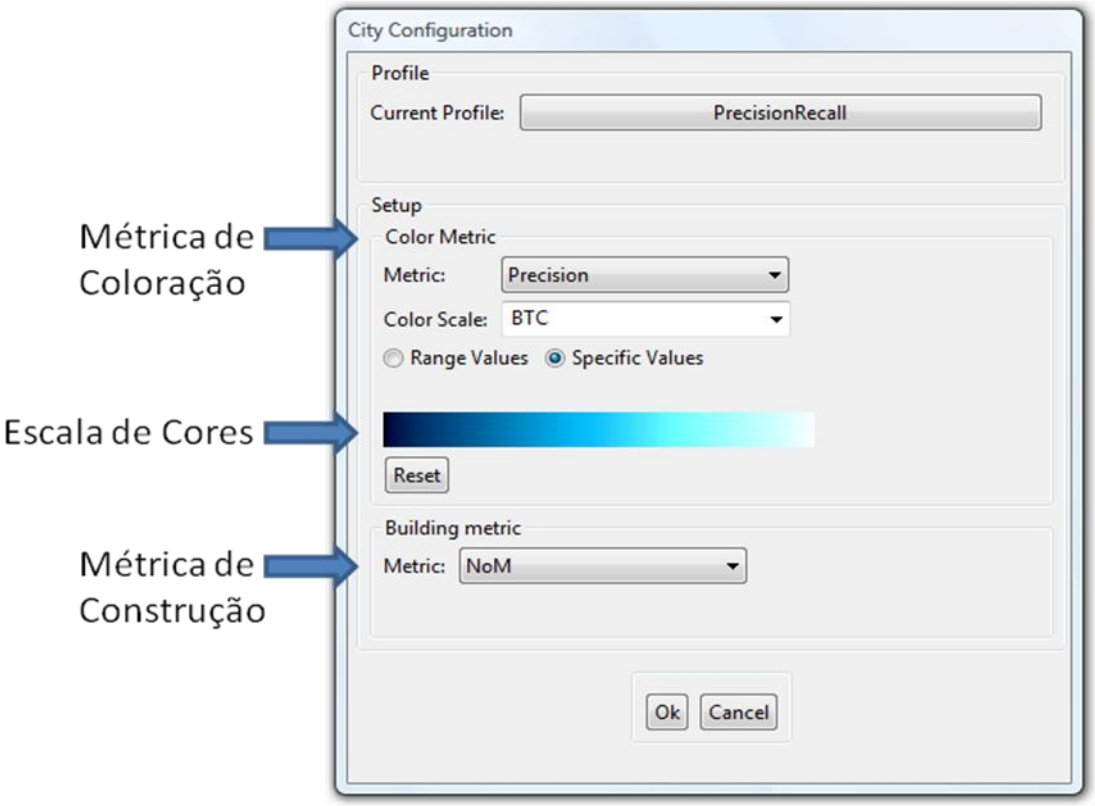


Figura 40. Tela de configuração da visualização da cidade UML

Na Figura 41, é possível observar a estrutura de um projeto utilizando a métrica de precisão para a cor (os valores mais baixos possuem coloração mais escura, enquanto os valores mais altos possuem coloração mais clara) e o número de métodos (NoM – *Number of Methods*) para a altura das construções (o foco do *mouse* mostra o nome da classe, interface ou pacote).

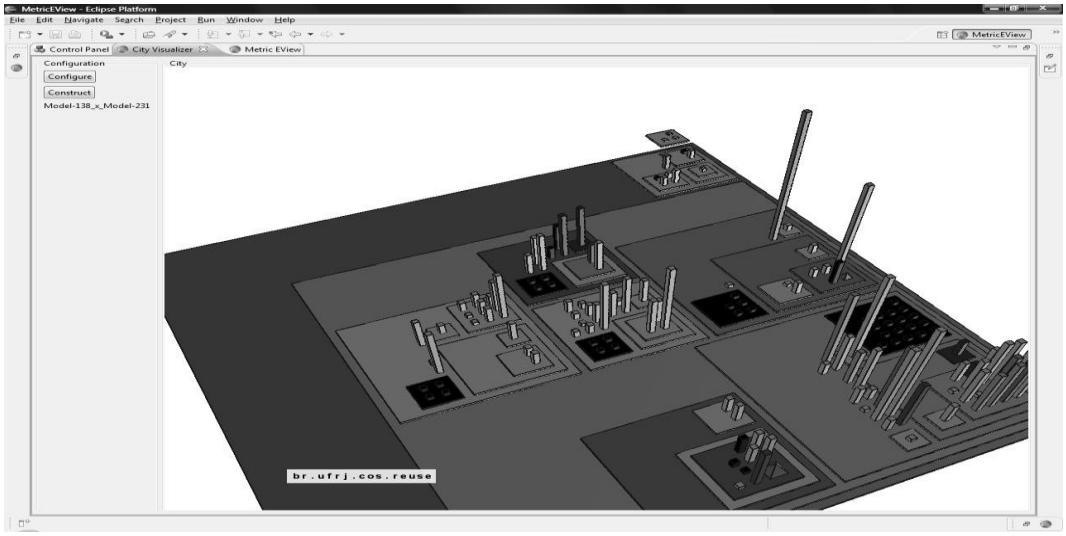


Figura 41. Exemplo de visualização da cidade UML – Projeto Brechó

Em outro exemplo (Figura 42), a mesma técnica pode ser utilizada em outro contexto: ao analisar a versão 9658 do projeto *open-source* Adempière (ADEMPIÈRE, 2010) comparada à versão 8829 do mesmo projeto, pode-se observar as estruturas que foram alteradas, adicionadas ou removidas. Neste caso, a classe GraphBuilder foi adicionada (em verde escuro), as duas classes em azul permaneceram inalteradas e todas as classes restantes (verde oliva) foram modificadas.

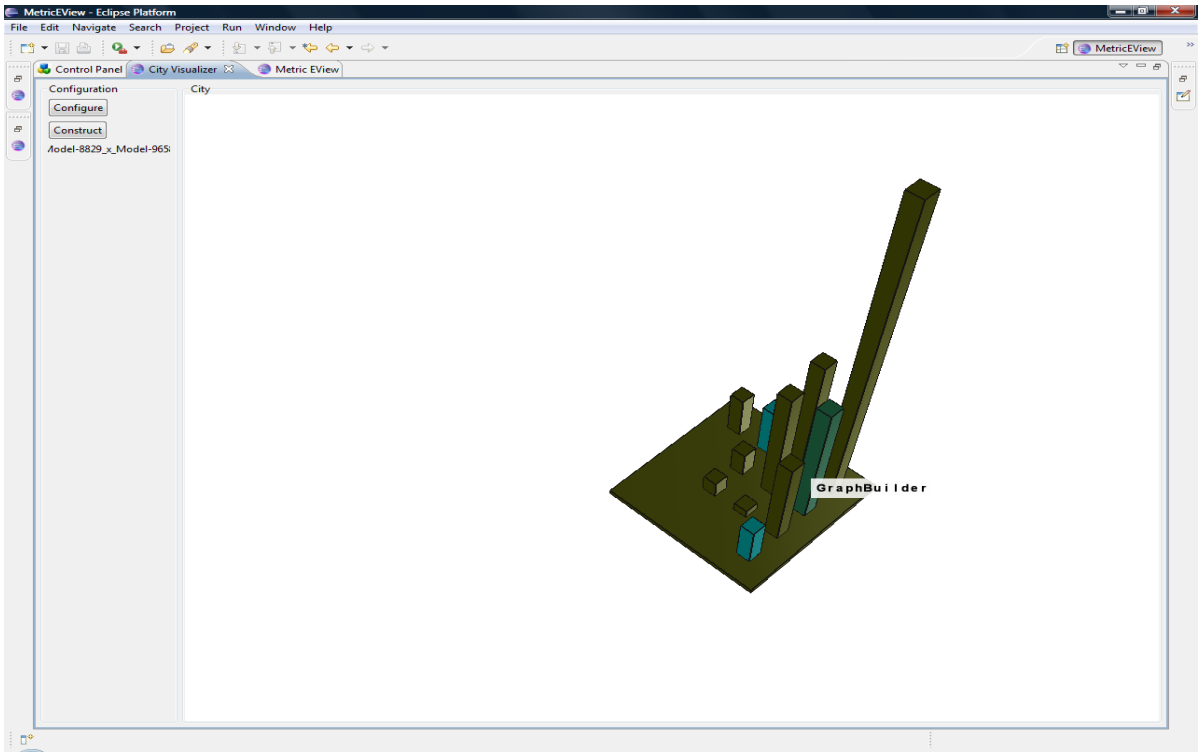


Figura 42. Exemplo de visualização da cidade UML – Projeto Adempière

5.4. Exemplo de Utilização

Esta seção visa ilustrar a utilização da infraestrutura desenvolvida por meio de um exemplo. Para isso, será utilizado como objeto de análise o projeto *open-source* Adempière (ADEMPIÈRE, 2010) cujo código-fonte é disponibilizado para qualquer interessado. Para a coleta de métricas, será utilizada a abordagem PREViA (SCHOTS *et al.*, 2010) cuja função é agregar os valores calculados nos modelos UML gerados para posterior visualização e análise da evolução. Assim, a Seção 5.4.1 apresenta em mais detalhes a abordagem PREViA; a Seção 5.4.2 descreve as métricas que são coletadas e analisadas; a Seção 5.4.3 mostra o exemplo de análise da evolução destas métricas no projeto Adempière.

5.4.1. Abordagem PREViA

Durante o desenvolvimento de software, muitos engenheiros de software acabam se deparando com diferenças entre o modelo conceitual, ou seja, o modelo planejado na fase de projeto do software, e o modelo emergente, que é o modelo efetivamente implementado em código-fonte. Essa divergência pode representar, por exemplo, elementos do modelo conceitual que ainda não foram implementados, ou mesmo elementos implementados – presentes no modelo emergente – que não foram descritos no modelo conceitual (SCHOTS *et al.*, 2010).

A abordagem PREViA (Procedimento para Representar a Evolução por meio da Visualização de Arquiteturas) fornece um meio de se compreender a evolução de modelos arquiteturais de software a partir da comparação de versões de modelo(s) armazenados em fontes de dados versionados (e.g., repositórios de gerência de configuração), utilizando-se de técnicas de visualização de software para apresentação do resultado da análise e comparação. Esta abordagem visa duas perspectivas de comparação (SCHOTS *et al.*, 2010): (i) a aderência do modelo emergente, aquele que varia com o desenvolvimento, ao modelo conceitual, aquele planejado na fase de projeto; e (ii) as diferenças encontradas entre duas versões distintas no processo de evolução arquitetural.

No contexto do EvolTrack, a PREViA assume o papel de Transformador de Modelos, adicionando informações das métricas de precisão e cobertura (entre outras, como a diferença detectada entre modelos) ao modelo UML de diagrama de classes gerado.

5.4.2. Métricas de Precisão e Cobertura

Com o intuito de auxiliar a quantificação da aderência entre as arquiteturas conceitual e emergente (foco na aderência), o trabalho de SCHOTS *et al.* (2010) propõe a utilização dos conceitos de **precisão** (*precision*) e **cobertura** (*recall*), advindos da área de recuperação de informação (OARD & DORR, 1996; BAEZA-YATES & RIBEIRO-NETO, 1999), adaptados para modelos arquiteturais de software. A Figura 43 exibe um paralelo da adaptação destas duas métricas com relação às originais.

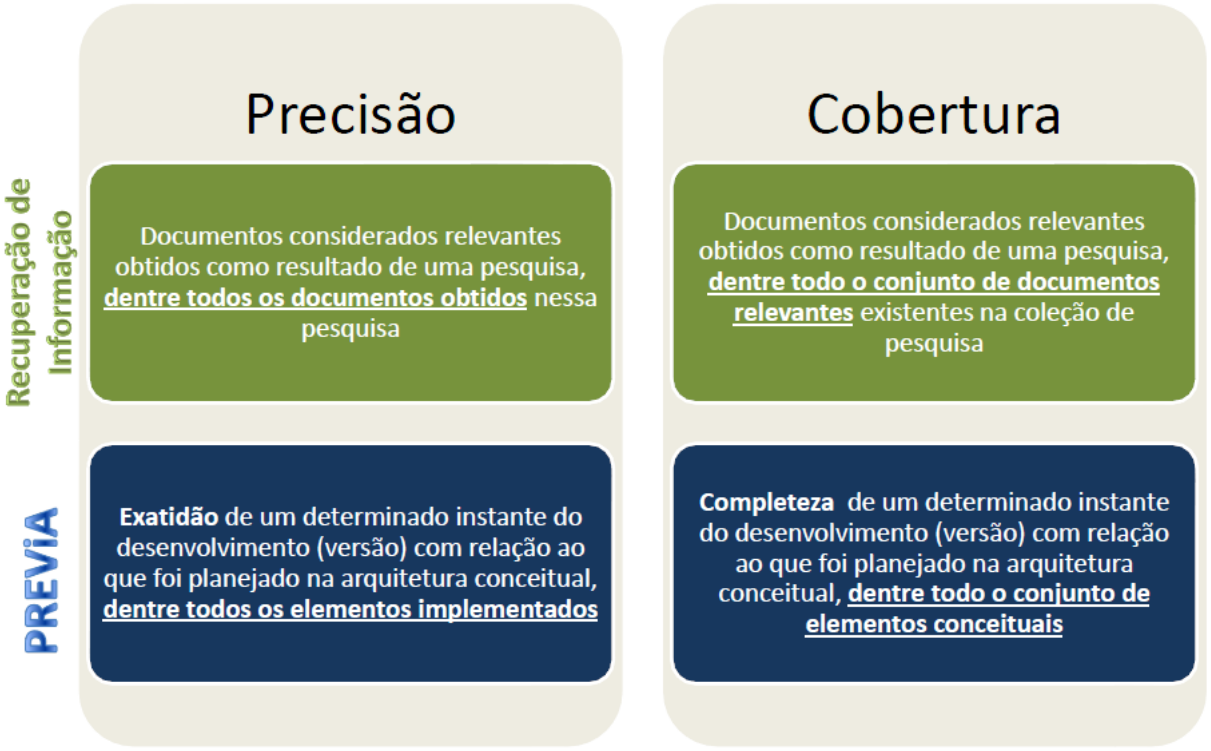


Figura 43. Paralelo da adaptação das métricas de precisão e cobertura (SCHOTS *et al.*, 2010)

As medidas de precisão e cobertura podem indicar, respectivamente, a exatidão e a completeza de um determinado instante do desenvolvimento (ou seja, uma dada versão do software) com relação ao que foi planejado na arquitetura conceitual. Os cálculos são efetuados da seguinte forma:

$$\textbf{Precisão} = \frac{n_{ci}}{n_{i-}\,n_{ii}} \quad \textbf{(i)} \qquad\qquad \textbf{Cobertura} = \frac{n_{ci}}{n_c} \quad \textbf{(ii)}$$

Nas fórmulas (i) e (ii), n_{ci} é o número de elementos conceituais implementados, n_i é o número total de elementos implementados, n_{ii} é o número de elementos implementados

identificados como específicos de implementação (isto é, que não foram incluídos na comparação e por isso devem ser subtraídos de forma a não interferir no cálculo da precisão) e n_c é o número total de elementos conceituais. Os valores de precisão e de cobertura variam entre 0 e 1. A Figura 44 ilustra como estes conceitos se relacionam.

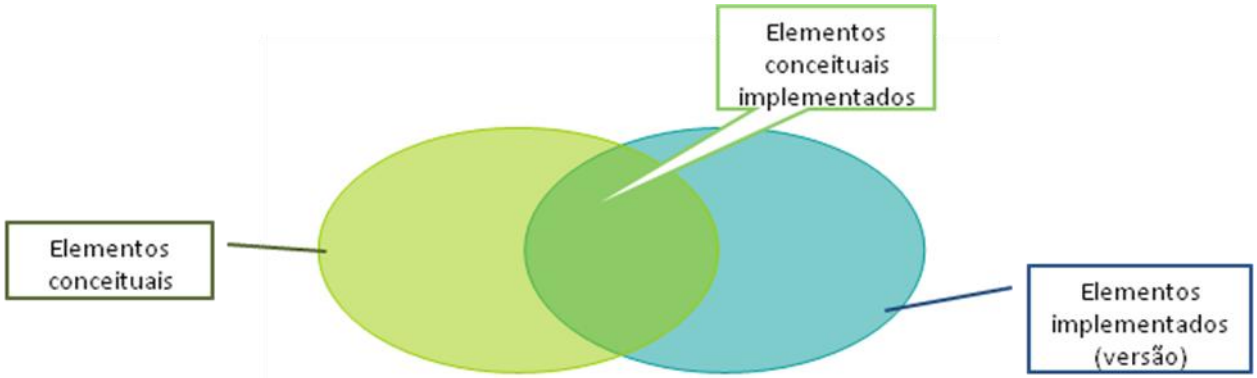


Figura 44. **Conceitos de precisão e cobertura** (SCHOTS *et al.*, 2010)

5.4.3. Exemplo

Por não ter sido encontrado nenhum modelo conceitual para o projeto Adempière, foi utilizada a revisão 8113 para desempenhar este papel apenas para fins de demonstração das métricas de precisão e cobertura da abordagem PREViA, provando o funcionamento da infraestrutura desenvolvida. Primeiramente, no menu de preferências do ambiente Eclipse, é configurado o EvolTrack-VCS. Neste exemplo, a conexão para o projeto Adempière é feita conforme visualizado na Figura 45a. Os campos de usuário e senha aparecem em branco, pois no repositório em questão essas informações não são necessárias. Em seguida, precisa-se decidir o modo de funcionamento do EvolTrack-VCS: opção de tempo real ativada ou desativada. Como a análise do projeto será feita em pontos discretos do tempo (neste caso, em revisões do referido projeto), será selecionada a segunda opção. Para a escolha destes pontos, a ferramenta fornece o recurso de um navegador acionado pelo botão "*Browse...*", conforme visto na Figura 45b.

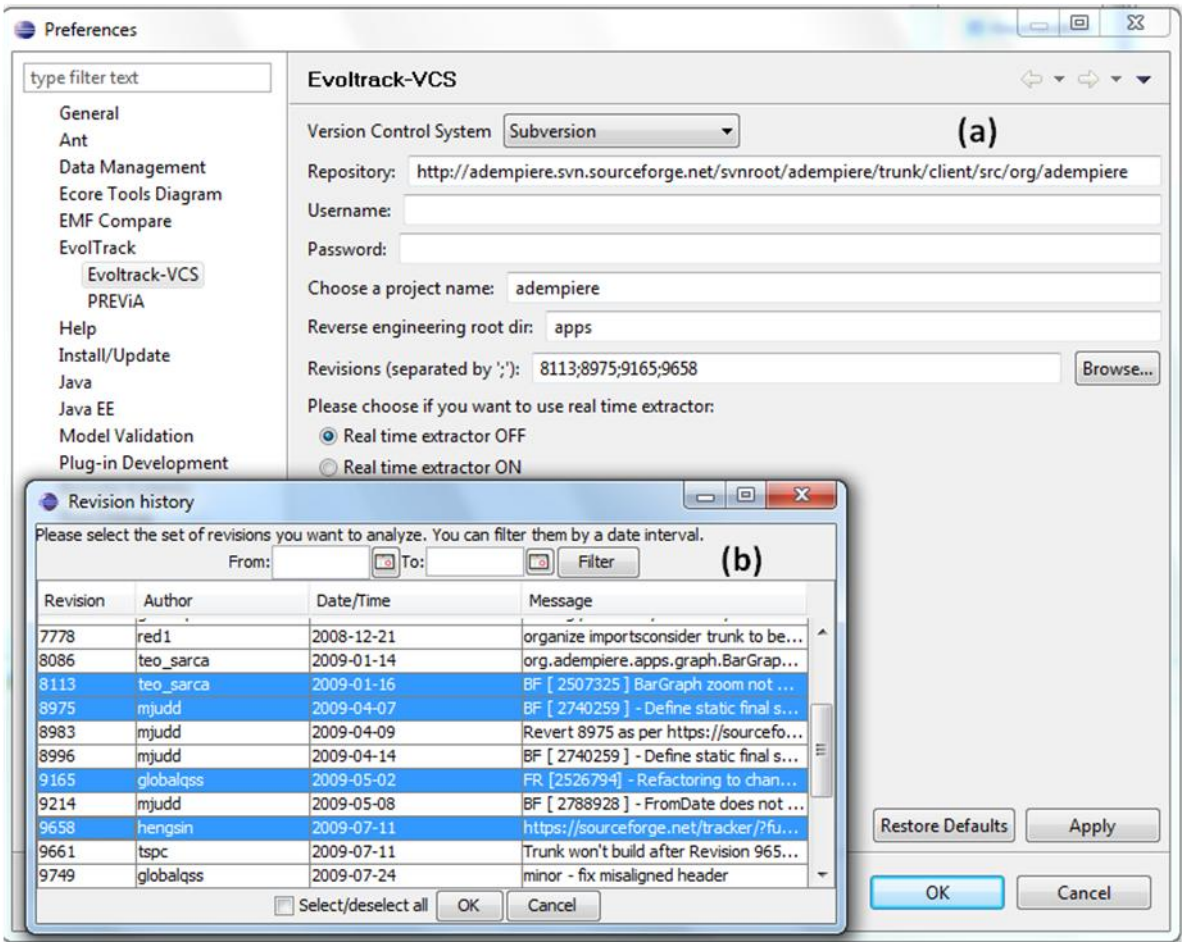


Figura 45. Configuração do EvolTrack-VCS

Após a seleção das revisões desejadas para análise (8113; 8975; 9165; 9658), aciona-se o EvolTrack-VCS no menu de preferências da ferramenta EvolTrack (Figura 46) para que a extração das informações do projeto de software seja iniciada

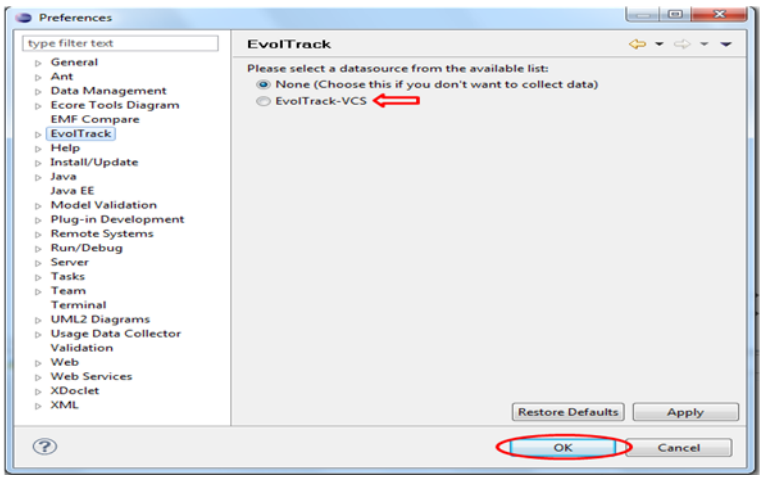


Figura 46. Acionamento do EvolTrack-VCS

Após a etapa de extração e construção dos modelos baseados nas informações do projeto de software, o processo de análise da evolução de métricas se inicia com o acionamento da perspectiva de visualização do EvolTrack-MetricEView (Figura 47).

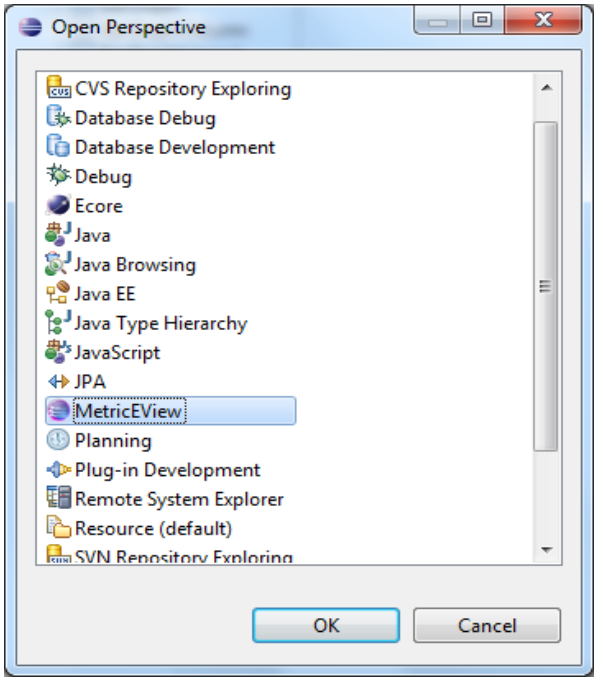


Figura 47. Seleção da perspectiva MetricEView

A partir deste momento, o engenheiro de software pode optar pela visualização que lhe convier, dependendo do tipo de análise que deseje realizar na evolução das métricas. Ao querer observar a evolução de métricas em conjunto, a visualização em foco pode ser utilizada, inclusive, por meio de múltiplas instâncias (sendo que cada instância pode ser utilizada para uma métrica distinta). A Figura 48 apresenta duas instâncias deste tipo para as métricas de precisão e cobertura, respectivamente, onde é possível navegar pela linha do tempo através do painel de controle. Nela, é possível observar que, ao comparar o modelo "conceitual" e a revisão 8975, é obtido o valor de 95,7 % de precisão e 100 % de cobertura. Utilizando os botões de navegação presentes no painel de controle, movimenta-se a linha do tempo do software, atualizando automaticamente o diagrama de classes apresentado, além das visualizações com foco em precisão e cobertura.

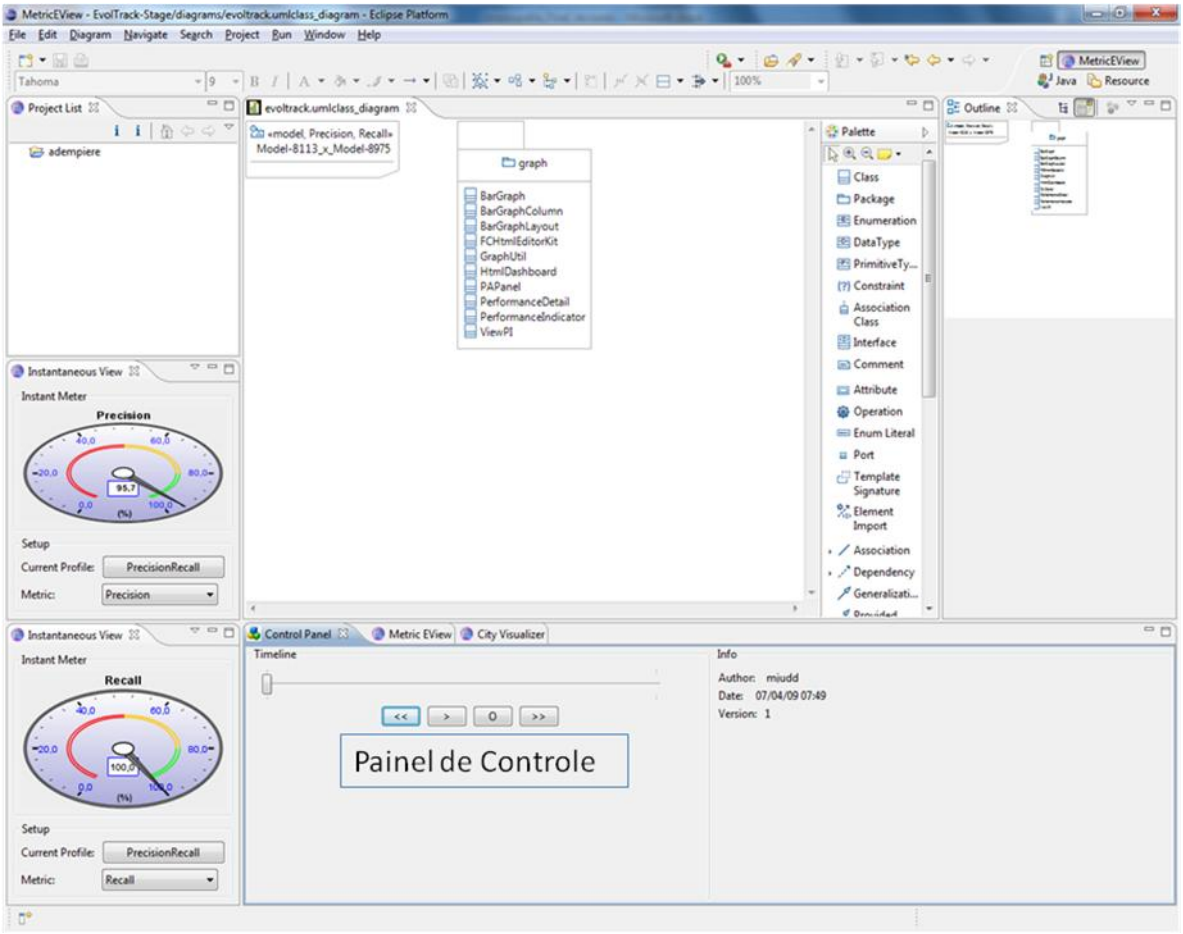


Figura 48. Foco em precisão e cobertura

Caso o interesse seja a comparação evolutiva de diversas métricas, a visualização comportamental conjunta de métricas pode ser mais indicada. Na Figura 49, configura-se este tipo de visualização para a comparação das métricas de precisão e cobertura, foram escolhidas as formas "estrela" e "hexágono" e as cores "oliva" e "azul" para elas, respectivamente.

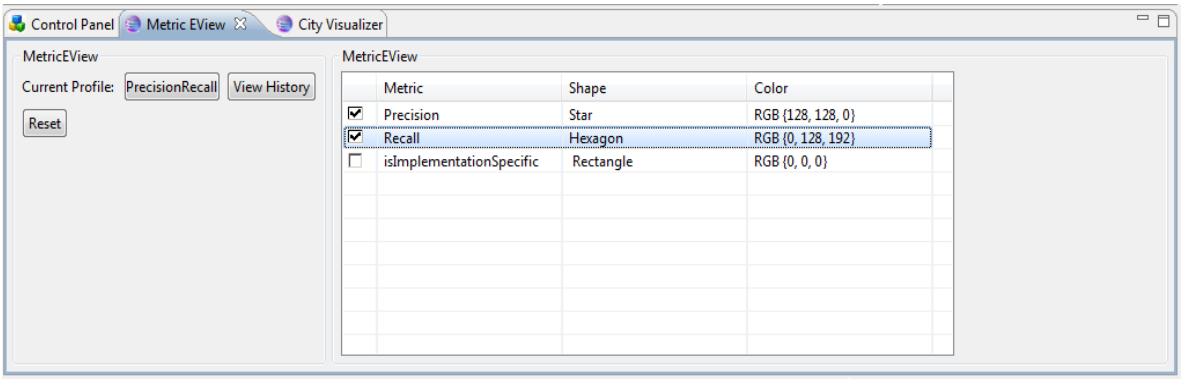


Figura 49. Configuração da análise comparativa da evolução de métricas

Com a configuração descrita, o resultado exposto na Figura 50 mostra que a evolução das métricas deste exemplo traça um perfil cônico na linha do tempo, onde se inicia no cenário ideal (duas métricas no valor máximo, pois se trata da comparação do modelo "conceitual" com ele mesmo), porém, acontece um distanciamento das mesmas com a evolução do software. Neste exemplo, o valor de cobertura se mantém em 100 % para todas as revisões, enquanto o valor de precisão se mantém no intervalo entre a 8975 e 9165, caindo para 83,1 %, na revisão 9658. Como o modelo que é utilizado como conceitual não o é de fato, nada se pode afirmar acerca deste comportamento. No entanto, se o modelo utilizado fosse um modelo conceitual, isto poderia mostrar que certo distanciamento do modelo planejado começa a aparecer (devido à introdução de entidades não planejadas).

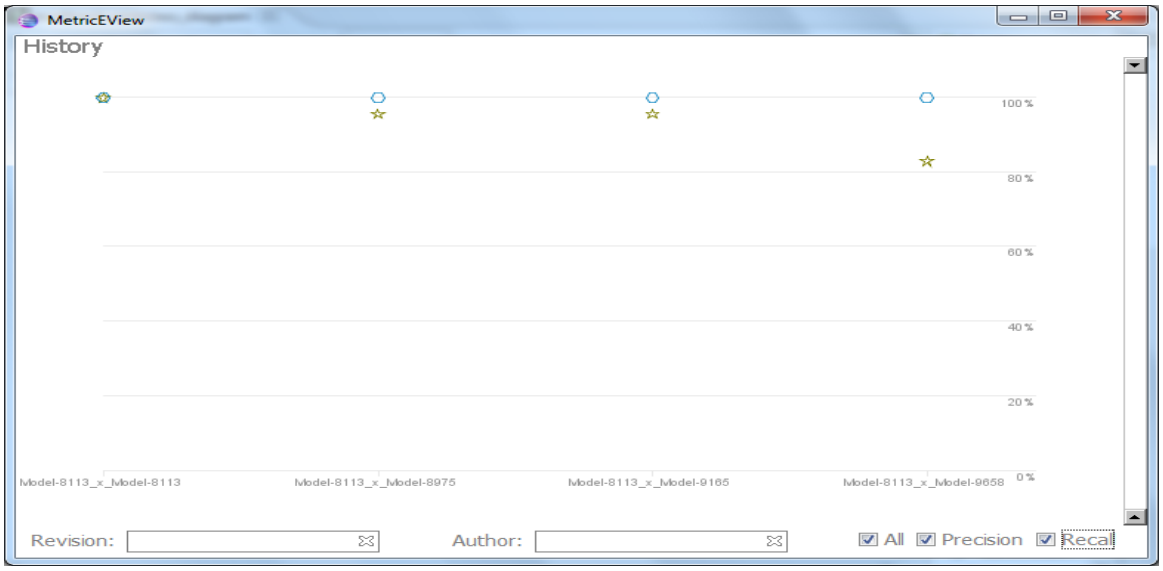


Figura 50. Evolução comparativa das métricas de precisão e cobertura

Por fim, é possível descobrir informações importantes como fenômenos e impactos em um software através da análise conjunta da estrutura com as métricas. Para que não haja um excesso de dados que prejudiquem a compreensão, optou-se pela utilização de uma visualização tridimensional. Sua configuração envolve a escolha de uma métrica que represente o tamanho das estruturas e outra, que representa a coloração das mesmas. Na Figura 51 é descrita a seleção da métrica *DifferenceKind* (que representa se determinado elemento foi alterado – apontando também os casos em que um subelemento contém alguma diferença, como por exemplo um atributo em uma classe –, removido ou adicionado) para a coloração e o número de métodos do objeto para o tamanho da construção, resultando na visualização exposta pela Figura 52.

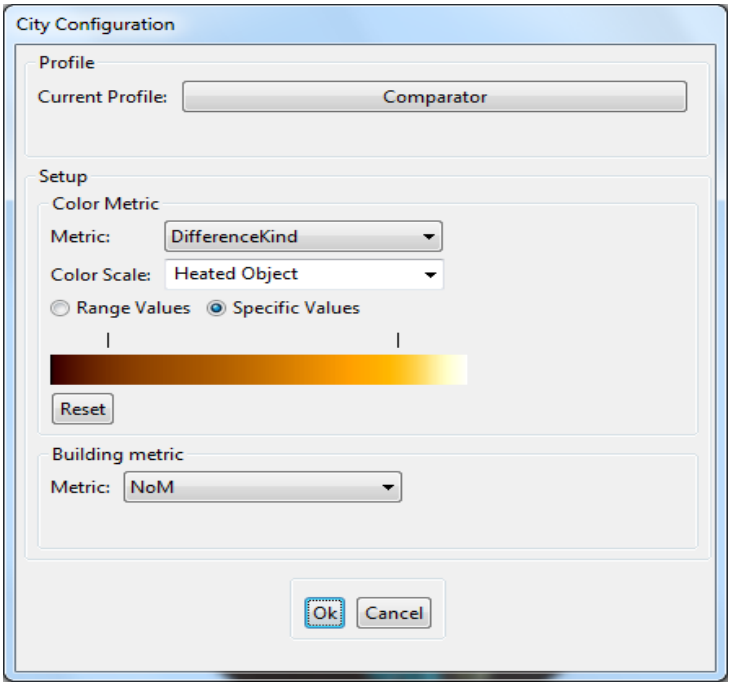


Figura 51. Configuração das métricas para a análise conjunta de métricas e estrutura

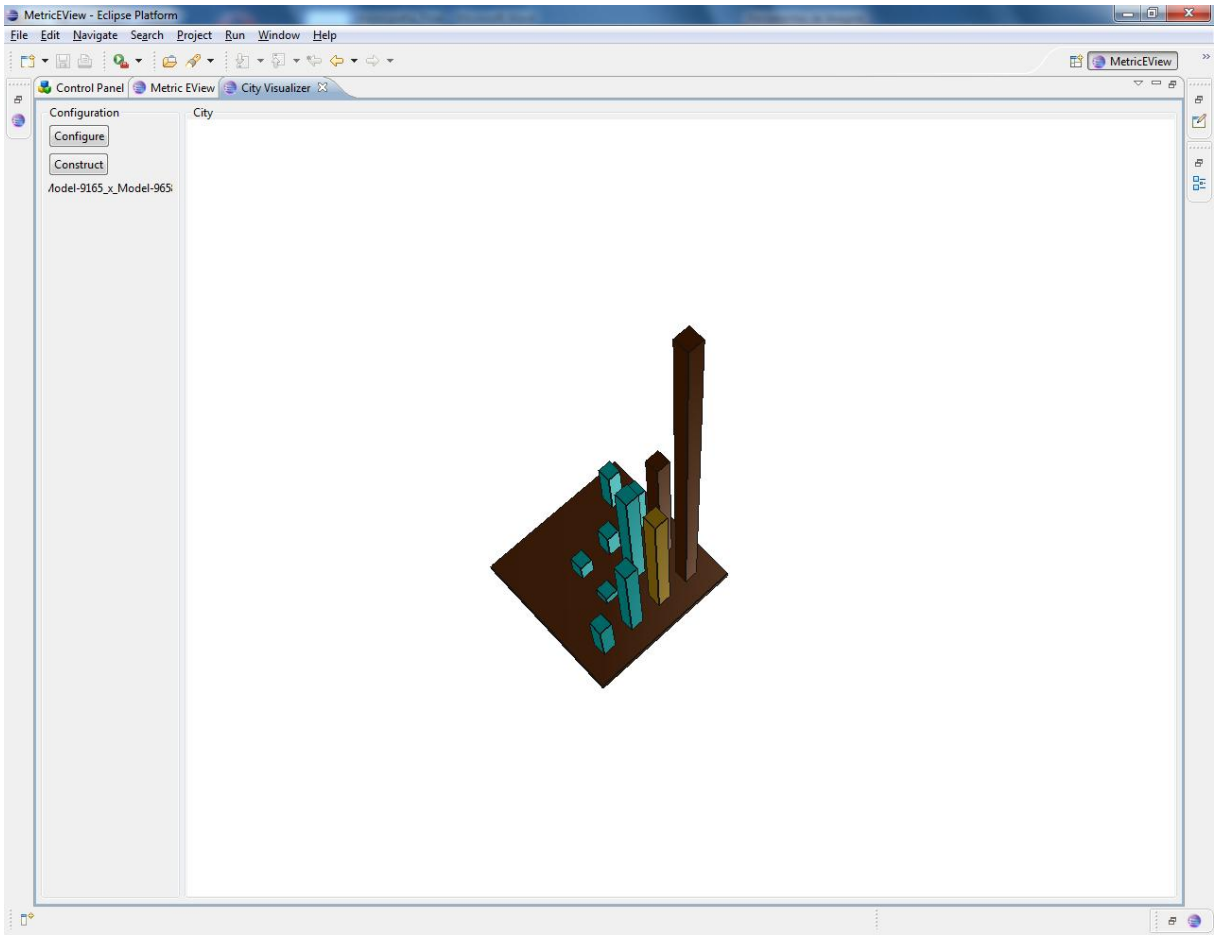


Figura 52. Visualização 3D: Sem modificação (Azul); modificado (Marrom); adicionado (Amarelo)

5.5. Considerações Finais

Este capítulo apresentou detalhes do projeto e implementação de toda a infraestrutura IAVEMS para suportar a evolução de métricas em conjunto com a estrutura de um determinado projeto. Desta forma, foram apresentadas as instâncias criadas para os elementos funcionais descritos na Seção 5.2 (i.e. Fonte de Dados e Visualizador). Neste contexto, com o intuito de concretizar a abordagem proposta, dois *plugins* para plataforma Eclipse foram desenvolvidos. O *plugin* EvolTrack-VCS foi desenvolvido para suportar a interação entre diferentes SCVs, sendo capaz de capturar informações de projetos de software durante o seu desenvolvimento e transformá-las em modelos UML.

Estes modelos são armazenados e gerenciados pelo *plugin* EvolTrack-Kernel (presente na ferramenta EvolTrack). Este é responsável por notificar todas as alterações no projeto para os Visualizadores possivelmente interessados em receber atualizações deste projeto. Por último, construiu-se um Visualizador denominado EvolTrack-MetricEView, pautado em técnicas de visualização de software para ampliar o entendimento durante o processo de desenvolvimento, conforme descrito anteriormente.

Capítulo 6. Conclusão

6.1. Contribuições

No capítulo 4 foi realizado um comparativo entre as ferramentas pesquisadas. Frente a um conjunto de critérios estabelecidos, verificou-se que nenhuma das abordagens atendia por completo aos requisitos apresentados no Capítulo 4. Visando incluir a ferramenta IAVEMS na comparação, os critérios são reapresentados:

- **C1. Tipo de fonte de dados utilizada:** Indica que outra ferramenta externa armazena os dados utilizados sobre o sistema em análise.
- **C2. Granularidade das informações apresentadas:** Indica o nível de granularidade das informações extraídas do sistema em análise que são apresentadas.
- **C3. Integração com algum ambiente de desenvolvimento de software:** Indica se a ferramenta possui algum tipo de integração com ambientes de desenvolvimento de software, podendo extrair, alterar ou apresentar informações destes ambientes.
- **C4. Tipo de métricas analisadas:** Indica que tipo de propriedade é analisada pelo software.
- **C5. Tipo de visualização utilizada:** Indica que tipo de recurso visual foi adotado para representar as características do sistema em análise.
- **C6. Análise Temporal:** Indica se a abordagem em questão proporciona ao seu usuário analisar informações da evolução do sistema em espaços discretos do tempo, em tempo real, ou ambas as estratégias.

A Tabela 2 exibe novamente o comparativo, incluindo a abordagem IAVEMS.

	FERRAMENTAS								
CRITÉRIOS	**SeeSoft**	**MetricView**	**Tarantula**	**EPOSpix**	**GEVOL**	**Vizz3D**	**CVSscan**	**softChange**	**IAVEMS**
C1	Usuário	Usuário	Usuário	eROSE	CVS	VizzAnalyzer	CVS	CVS	SCVs
C2	Baixa	Alta	Baixa	Alta	Baixa	Alta	Baixa	Baixa	Ambos
C3	Não	Não	Não	Sim	Não	Não	Não	Não	Sim
C4	Código e Tamanho	Variadas	Código e Teste	Acoplamento	Código	Variadas	Código	Tamanho	Variadas
C5	Formas e Escala de Cores	Gráficos de barra, pizza, 2D e 3D	Formas e Escala de Cores	Mapa de *pixels*	Grafos	Grafos e Formas 3D	Gráfico Baseado em Linhas	Gráfico Baseado em *Check-in*	Velocímetro, Gráficos interativos e Formas 3D
C6	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto	Ambos

Tabela 2. Quadro comparativo entre as ferramentas pesquisadas e a abordagem IAVEMS

A abordagem IAVEMS proporciona ao usuário um método de visualização da evolução de métricas juntamente com a estrutura do determinado projeto de software, o que acrescentou novas possibilidades à ferramenta EvolTrack que só tratava da estrutura do software, podendo alternar informações de granularidade mais alta (como no modelo de projeto) com outras de granularidade mais baixa (como métricas de código-fonte) (**C2**). Adicionalmente, vale ressaltar que a abordagem possibilita a utilização de diferentes tipos de sistemas de controle de versão (**C1**). O sistema de visualização (**C5**) não se resumiu a uma única técnica de visualização, mas buscou a combinação de algumas de forma a usufruir das vantagens que cada uma oferece. Toda a ferramenta foi desenvolvida como *plugin* para o Eclipse, o que cumpre a integração com um ambiente de desenvolvimento (**C3**) e trataram-se as questões referentes às métricas de maneira genérica para que a abordagem não ficasse restrita ao tipo de métrica, característica comum na maioria das ferramentas analisadas (**C4**). Quanto à perspectiva temporal, o conector de fonte de dados pode ser configurado tanto para trabalhar com momentos pré-definidos como para ficar observando a existência de novidades (**C6)**. A ferramenta possui também alguns pontos de extensão que facilitam a manutenção e adaptação da mesma. São eles: Adição de SCVs, novas linguagens interpretadas pelo Parser, acréscimo de escalas de cores suportadas pelas visualizações e inclusão de outras visões de software.

Com isso, os protótipos desenvolvidos atenderam a todos os requisitos estabelecidos para alcançar os objetivos deste trabalho e, ao serem incorporados à ferramenta EvolTrack e auxiliados pela abordagem PREViA, representam a prova de conceito desta abordagem. A ferramenta encontra-se disponível no site http://reuse.cos.ufrj.br/evoltrack/ (onde também se encontram o manual de uso e guia de instalação), porém, existe também a possibilidade de instalação diretamente pela IDE Eclipse por meio do site de atualização (*update site*): http://reuse.cos.ufrj.br/evoltrack/updates.

6.2. Limitações

O trabalho proposto, apesar da generalidade das visualizações para o suporte a diferentes tipos de métricas, só possui atualmente a abordagem PREViA como coletora e marcadora de métricas, restringindo-se a medidas de comparação e as métricas de precisão e cobertura. Existe também a abordagem Orion (PRUDÊNCIO, 2009), tratando da

transformação de modelos para o cálculo de métricas de controle de concorrência, porém, carece de adaptações para integração com o EvolTrack. Desta forma, não foi possível avaliar a utilização da abordagem com outros Transformadores, apesar de a utilização da PREViA fornecer indícios da viabilidade do uso de modelos marcados por outros Transformadores. No entanto, existe um trabalho em andamento no grupo que visa trabalhar com a evolução de métricas de redes sociais; após a conclusão deste trabalho, será feito um estudo de utilização da abordagem IAVEMS neste contexto.

As visualizações referentes ao foco na métrica e na interatividade, por sua natureza, se restringem ao uso de métricas numéricas já que o uso de métricas qualitativas não faz sentido no contexto das mesmas. Além disso, o *parser* utilizado para a engenharia reversa é, atualmente, limitado à linguagem de programação Java.

6.3. Trabalhos Futuros

Como trabalho futuro, almeja-se o planejamento e execução de uma avaliação da presente abordagem, em especial, dos potenciais benefícios obtidos a partir do uso dos mecanismos de visualização construídos num projeto de software real.

Pretende-se também ampliar o uso de técnicas 3D para a análise da evolução de outras visões do desenvolvimento de software, não se restringindo a estrutura ou a métricas. Alguns exemplos destas outras visões são: (i) a análise das redes sociais que se desenvolvem ao longo de um projeto de software; (ii) ferramentas que apoiem a colaboração durante o desenvolvimento, o que pode trazer benefícios tais como um menor número de conflitos gerados num desenvolvimento distribuído. Neste sentido, no grupo de pesquisa Reuse (em que este trabalho está inserido), existem dois trabalhos relacionados: uma tese de Doutorado em andamento, envolvendo a análise dos aspectos sociais, técnicos e sociotécnicos de redes sociais em desenvolvimento de software, e uma dissertação de mestrado envolvendo a prevenção de conflitos e o estímulo de colaboração por meio da visualização da estrutura junto com o fator humano de um projeto de software.

Referências Bibliográficas

ABNT, 2003, *NBR ISO/IEC9126-1: Engenharia de software - Qualidade de produto - Parte 1 - Modelo de qualidade*, 35p, Rio de Janeiro.

ABNT, 1998, *NBR ISO/IEC 12207 – Tecnologia de informação - Processos de ciclo de vida de software*, 35p, Rio de Janeiro.

ACCUREV, 2010, Em: http://www.accurev.com/, Acesso em Maio.

ADEMPIERE, 2010, Em: http://www.adempiere.com/index.php/ADempiere, Acesso em Maio.

ANDREWS, K.; WOLTE, J.; PICHLER, M., 1997, *Information pyramids: A new approach to visualizing large hierarchies*, In: Proceedings of IEEE Visualization97, Phoenix, Arizona, pp. 49–52, October.

APACHE SOFTWARE FOUNDATION, 2010, Em: http://ws.apache.org/jaxme/js/index.html, Acesso em Maio.

BAEZA-YATES, R. A.; RIBEIRO-NETO, B., 1999, *Modern Information Retrieval*, Addison-Wesley Longman Publishing Co., Inc.

BALL, A.T; EICK, S.G., 1996, *Software Visualization in the Large*, IEEE Computer, vol. 29, n. 4, pp. 33-43, April.

BALZER, M.; DEUSSEN, O., 2005, *Voronoi treemaps*, In: Proceedings of the IEEE Symposium on Information Visualization, Minneapolis, MN, pp. 7-14 , 23 – 25 October, IEEE Computer Society Press.

BAZAAR, 2010, Em: http://bazaar.canonical.com/en/, Acesso em Maio.

BROOKS, F. P. Jr., 1996, The Computer Scientist as Toolsmith II, Comunications of the ACM, 39(3), pp. 61-68.

BROOKS, F. P., 1987, No silver bullet: Essence and accidents of software engineering, Computer, 20(4), pp. 10-19, April.

BRULS, M.; HUIZING, K.; VAN WIJK, J. J., 2000, *Squarified treemaps*, In: Proceedings of Joint IEEE TCVG Symposium on Visualization, pp. 33 – 42, Vienna, IEEE Computer Society Press.

BUGZILLA, 2010, Em: www.bugzilla.org, Acesso em Maio.

CARD, D. N.; GLASS, R. L., 1990, Measuring Software Design Quality, Prentice-Hall.

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

CARMO, M. B.; CUNHA, J. D., 1998, *Filtragem e Escolha de Representação na Visualização de Informação*, Atas do VIII Encontro Português de Computação Gráfica, Coimbra, pp. 81-94, Fevereiro.

CARVALHO, E. S.; MARCOS, A. F., 2009, *Visualização da Informação*, Centro de Computação Gráfica, Guimarães, 61p.

CEPEDA, R. S. V.; MAGDALENO, A. M.; MURTA, L. G. P.; WERNER, 2010, *EvolTrack: improving design evolution awareness in software development*, Journal of the Brazilian Computer Society, Springer London, June (to appear).

CEPEDA, R. S. V., 2008, *Visualizando a Evolução de Software com EvolTrack*, Trabalho de Conclusão de Curso (Graduação em Bacharel Em Matemática Mod Informática), pp. 83, Universidade Federal do Rio de Janeiro, Orientadores: Cláudia Maria Lima Werner e Leonardo Gresta Paulino Murta.

CHEN, C., 2006, Information Visualization, 316p, Springer.

CHIDAMBER, S. R.; KEMERER, C. F., 1994, *A Metrics Suite for Object-Oriented Design*, IEEE Trans. Software Engineering, vol. SE-20, n. 6, June, pp. 476 – 493.

CLAYBERG, E., RUBEL, D., 2006, *Eclipse Infrastructure*, In: Gamma, E., Nackman, L., Wiegand, J. (eds), *Eclipse: Building Commercial-Quality Plug-ins*, 2a Edição, Cap. 3, Boston, MA, EUA, Addison Wesley Professional.

CLEARCASE, 2010, Em: http://www-01.ibm.com/software/awdtools/clearcase/, Acesso em Maio.

CLUSTERING, 2010, Em: http://fotos.sapo.pt/aY5hiir3RrodW07Y1mzf/, Acesso em Maio.

CODSTRUCTION, 2010, Em: http://codstruction.wordpress.com/, Acesso em Maio.

COLLBERG, C.; KOBOUROV, S.; NAGRA, J.; PITTS, J.; WAMPLER, K., 2003, *A system for graph-based visualization of the evolution of software*, In: Proceedings of the 2003 ACM symposium on Software visualization, pp. 77 – 86, San Diego, California, June.

COLLOFELLO, J. S.; WOODFIELD, S. N., 1989, *Evaluating the efectiveness of reliability-assurance techniques*, Journal of Systems and Software, 9(3):pp. 191-195.

CVS, 2010, Em: http://www.nongnu.org/cvs/, Acesso em Maio.

DIEHL, S.; 2007, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*, Springer, 185p.

ECLIPSE ARCHITECTURE, 2010, Em: http://www.developer.com/open/article.php/10936_3316241_2/A-First-Look-at-Eclipse-Plug-In-Programming.htm, Acesso em Maio.

ECLIPSE FOUNDATION, 2010a, Em: http://eclipse.org, Acesso em Maio.

ECLIPSE FOUNDATION, 2010b, Em: http://www.eclipse.org/modeling/mdt/?project=uml2tools, Acesso em Maio.

EICK, S. G.; STEFFEN, J. L.; SUMMER, E. E. Jr., 1992, *SeeSoft TM – a tool for visualizing line oriented software statistics*, IEEE Transactions on Software Engineering, 18(11):957 – 968.

EJIOGU, L., 1991, *Software Engineering with Formal Metrics*, QED Publishing.

GERMAN, D.M.; HINDLE, A.; JORDAN, N., 2006, *Visualizing the Evolution of Software using softChange*, International Journal of Software Engineering and Knowledge (IJSEKE), vol. 16, n. 1, pp. 5 - 21, February.

GERSHON, N. D., 1994, *From perception to visualization*, In Scientific Visualization: Advances and Challenges, Academic Press.

GIT, 2010, Em: http://git-scm.com/, Acesso em Maio.

GOGOLLA, M.; RADFELDER, O., 2000, *On better understanding UML diagrams through three-dimensional visualization and animation*, In: Proceedings of Advanced Visual Interfaces (AVI'2000), Palermo, Italy, pp. 292-295, New York, NY, ACM Press.

GOMES, A.; OLIVEIRA, K.; ROCHA, A. R., 2001, *Avaliação de Processos de Software Baseada em Medições*, XV Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, pp. 84 – 99, Outubro.

HARRISON, R., COUNSELIi, S. J., e NITHI, R. V., 1998, *An Evaluation of the MOOD Set of Object-Oriented Software Metrics*, IEEE Trans. Software Engineering, vol. SE-24, n. 6, June, pp. 491 – 496.

HENDERSON-SELLERS, B., 1996, *Object-oriented metrics: measures of complexity*, Prentice-Hall, pp.142-147.

HETZEL, B., 1993, *Making Software Measurement Work*, QED Publishing.

HORNBAEK, K.; BEDERSON, B. B.; PLAISANT, C., 2002, *Navigation Patterns and Usability of Overview+Detail and Zoomable User Interfaces for Maps*, ACM Transaction on Computer-Human Interaction, volume 9, December, pp. 362-389.

HUNT, W.J., MCILROY, D.M., 1976, *An algorithm for differential file comparison*, In: Relatório Técnico 41, AT&T Bell Laboratories, Inc., Murray Hill, NJ, EUA.

HUNT, W.J., SZYMANSKI, G.T., 1977, *A fast algorithm for computing longest common subsequences*, Communications of the ACM, v. 20, n. 5, pp. 350-353, May.

IEEE, 1994, *Software Engineering Standards*, 1994 edition.

ISO/IEC 25000 (SQuaRE), 2010, *Software product Quality Requirements and Evaluation – Guide to SQuaRE*, Em: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35683, Acesso em Maio.

JAM SOFTWARE, 2010, Em: http://www.jam-software.com/spaceobserver/screenshots/, Acesso em Maio.

JERN, M., 1997, *Information Drill-down using Web tools*, In: Proceedings of IEEE Conference on Information Visualization, pp. 118, IEEE Computer Society Press.

JFREECHART, 2010, Em: http://www.jfree.org/jfreechart/, Acesso em Maio.

JOHNSON, B.; SHNEIDERMAN, B., 1991, *Tree-maps: A space-filling approach to the visualization of hierarchical information structures.* In Proceedings of IEEE Visualization Conference, pp. 284-291, San Diego, CA.

JONES, J. A.; HARROLD, M. J.; STASKO, J., 2002, *Visualization of Test Information to Assist Fault Localization*, In: Proceedings of the International Conference on Software Engineering, Orlando, Florida, U.S.A., May.

KAUFMANN, M.; WAGNER, D., 2001, *Drawing Graphs – Methods and Models*, Springer, Berlin, Heidelberg.

LORENZ, M. e KIDD, J., 1994, *Object-Oriented Software Metrics*, Prentice-Hall.

MAVEN SCM, 2010, Em: http://maven.apache.org/scm/maven-scm-api/index.html, Acesso em Maio.

MCCABE, T. J., 1976, *A Complexity Measure*, IEE Transactions on Software Engineering, vol. SE-2, no. 4.

MERCURIAL, 2010, Em: http://mercurial.selenic.com/, Acesso em Maio.

OARD, D. W.; DORR, B. J., 1996, *A Survey of Multilingual Text Retrieval*, University of Maryland, College Park, MD20742, April.

OMG, 2010a, Em: http://www.omg.org/uml, Acesso em Maio.

OMG, 2010b, Em: http://www.omg.org/spec/XMI/, Acesso em Maio.

OMG, 2010c, Em: http://www.omg.org/spec/UML/1.3/, Acesso em Maio.

PERFORCE, 2010, http://www.perforce.com/, Acesso em Maio.

PREFUSE, 2010, Em: http://prefuse.org/, Acesso em Maio.

PRESSMAN, R.S., 2006, *Software e Engenharia de Software*, In: *Engenharia de Software*, 6ª Edição, Capítulo 1, McGraw-Hill.

PRUDÊNCIO, J. G.; MURTA, L. G. P.; WERNER, C. M. L., 2009, *On the Selection of Concurrency Control Policies for Configuration Management*, sbes, XXIII Brazilian Symposium on Software Engineering, pp.155-164, Fortaleza, CE, October.

PURCHASE, H. C.; COHEN, R. F.; JAMES, M., 1996; *Validating graph drawing aesthetics*, In Franz J. Brandenburg, editor, Graph Drawing, vol. 1027 of Lecture Notes Computer Science, pp. 435-446, Berling, Heidelberg, New York, Springer.

REUSE, 2010, Em: http://reuse.cos.ufrj.br, Acesso em Maio.

REZENDE, D. A., 2006, *Engenharia de software e sistemas de informação*, Brasport Livros e Multimídia Ltda.

ROBERTSON, G.; CARD, S., MACKINLAY, J., 1993, *Information visualization using 3-D interactive animation*, Communications of the ACM, vol. 36, n. 4, pp. 57-71, Abril.

ROCHE, J. M., 1994, *Software Metrics and Measurement Principles*, Software Engineering Notes, ACM, vol. 19, n. 1, January, pp. 76 – 85.

ROTO, V.; POPESCU, A.; KOIVISTO, A.; VARTIAINEN, E., 2006, *Minimap: a web page visualization method for mobile phones*, In Proceedings of the SIGCHI conference on Human Factors in computing systems, Montréal, pp. 35 – 44, April.

SATO, D.; GOLDMAN, A.; KON, F., 2007, *Tracking the evolution of object-oriented quality metrics on agile projects*, In: Proceedings of the 8th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2007), Como, pp. 84–92, June.

SCHOTS, M.; SILVA, M. A.; MURTA, L. G. P.; WERNER, C. M. L., 2010, *Verificação de aderência entre arquiteturas conceituais e emergentes utilizando a abordagem PREViA*, VII Workshop de Manutenção de Software Moderna, pp. 1-8, Belém, PA, June.

SEI, 2010, Em: http://www.sei.cmu.edu/cmmi/, Acesso em Maio.

SEPT, 2010, Em: http://www.12207.com/, Acesso em Maio.

SHNEIDERMAN, B., 1996, *The eyes have it: A task by data type taxonomy for information visualizations*, In Proceedings of IEEE Conference on Visual Languages, Boulder, CO, pp. 336-343, Washington, DC, IEEE Computer Society Press.

SHNEIDERMAN, B.; WATTENBERG, M., 2001, *Ordered treemaps layouts,* In: Proceedings of IEEE Symposium on Information Visualization, pp. 73 – 78, Washington, DC, IEEE Computer Society Press.

SOFTEX, 2010a, Em: http://www.softex.br/mpsbr/_guias/default.asp, Acesso em Maio.

SOFTEX, 2010b, Em: http://www.softex.br/, Acesso em Maio.

SOUZA, C., 2010, Em: http://www.ufpa.br/cdesouza/teaching/methods/8-Social-Networks.pdf, Acesso em Maio.

SVG, 2010, Em: http://www.w3.org/Graphics/SVG/, Acesso em Maio.

STARTEAM, 2010, Em: http://www.borland.com/br/products/starteam/, Acesso em Maio.

SUBVERSION, 2010, Em: http://subversion.tigris.org/, Acesso em Maio.

SYNERGY, 2010, Em: http://www-01.ibm.com/software/awdtools/synergy/, Acesso em Maio.

TERMEER, M.; LANGE, C.F.J.; TELEA, A.; CHAUDRON, M.R.V., 2005, *Visual Exploration of Combined Architectural and Metric Information*, 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, Budapest, pp.21 - 26, October.

TFS, 2010, Em: http://msdn.microsoft.com/en-us/library/ms364061.aspx, Acesso em Maio.

USMEPCOM, 2010, Em: http://www.mepcom.army.mil/organization/vips_index.html, Acesso em Maio.

VAN WIJK, J. J.; VAN DE WETERING, H., 1999, *Cushion treemaps: Visualization of hierarchical information*, In: Proceedings of IEEE Symposium on Information Visualization, pp. 73 – 78, San Francisco, October.

VISIBLE HUMAN PROJECT, 2010, Em: http://www.nlm.nih.gov/research/visible/, Acesso em Junho.

VOINEA, L.; TELEA, A., 2006, *CVSGrab: Mining the History of Large Software Projects*, In: Proceedings of Eurographics/IEEE-VGTC Symposium on Visualization, IEEE Press, pp. 187-194, Lisboa, Portugal, May.

VSS, 2010, Em: http://msdn.microsoft.com/pt-br/library/ms181038%28VS.80%29.aspx, Acesso em Maio.

WALL, L.; CHRISTIANSEN, T.; ORWANT, J., 2000, *Programming Perl*, 3a Ed., Stanford, CA, EUA, O'Reilly.

WARE, C.; FRANCK, G., 1994, *Viewing a graph in a virtual reality display is three times as good as a 2D diagram*, In: proceedings of the IEEE Symposium on Visual languages (VL'94), St. Louis, MO, pp.182-183, Washington, DC, IEEE Computer Society Press.

WERNER, C. M. L.; MURTA, L. G. P.; LOPES, M.; DANTAS, A.; LOPES, L. G.; FERNANDES, P.; PRUDENCIO, J. G.; MARINHO, A.; RAPOSO, R., 2007, *Brechó: Catálogo de Componentes e Serviços de Software*, In: XXI Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, pp. 24-30 João Pessoa, outubro.

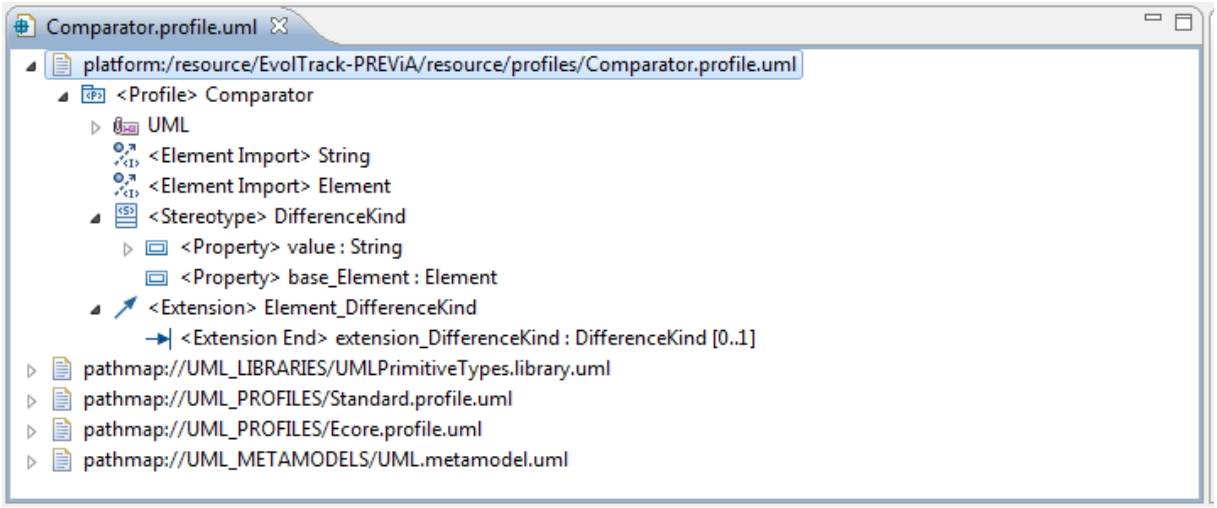
WEIßGERBER, P.; VON KLENZE, L.; BURCH, M.; DIEHL, S., 2005, *Exploring evolutionary coupling in Eclipse*, In: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, pp. 31 – 34, San Diego, California, October.

WHITEMIRE, S., 1997, *Object-Oriented Design Measurement*, Wiley.
ZUSE, H., 1997, *A Framework of Software Measurement*, DeGruyter.

Anexo I

Perfis UML extraídos da Abordagem PREViA (SCHOTS *et al.*, 2010):

- Perfil de comparação disponibilizando métricas para a indicação de elementos que foram removidos, alterados ou adicionados.



- Perfil de precisão/cobertura disponibilizando métricas para a indicação, respectivamente, da exatidão e da completeza de um determinado instante do desenvolvimento (ou seja, uma dada versão do software) com relação ao que foi planejado na fase de projeto. A métrica *isImplementationSpecific* não foi utilizada neste trabalho.