Visminer DEV: Um Ambiente de Visualização Web com Suporte a Múltiplas Estratégias de Análise Visual

Rafael Bispo*
Instituto Federal de Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N Barbalho
Salvador-Ba, Brasil
rafaelbispo@ifba.edu.br

Renato Novais†
Instituto Federal de Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N Barbalho
Salvador-Ba, Brasil
renato@ifba.edu.br

RESUMO

A visualização de software é um tema de destaque na Engenharia de Software. Ela tem ajudado a melhorar a compreensão do desenvolvimento e evolução de software. Analisar como softwares evoluem ao longo do tempo pode ser bastante difícil, devido a quantidade e complexidade que os projetos ao longo da sua história podem adquirir. Apesar de já existirem diversas aplicações que abordam tipos de visualização de evolução de software, elas geralmente não se preocupam em permitir uma interação fácil e intuitiva para os usuários. Apesar de muitos ambientes de visualização de evolução de software conterem detalhamentos acerca das métricas, eles não possuem em alguns casos, mais do que uma estratégia de análise visual implementada. Este trabalho teve por finalidade desenvolver um ambiente de visualização que por meio de uma aplicação web mostra dados em diferentes perspectivas de visualização. Essas visualizações podem ser configuradas dinamicamente e são exibidas de forma global e detalhada. Esse ambiente se fez necessário para implementar as estratégias de análise visual em uma aplicação web, pois outros ambientes não implementam determinadas estratégias de análise visual em uma aplicação web de forma combinada. Por fim, o trabalho foi submetido a testes utilizando o repositório do JUnit4 como objeto de análise. O Junit4 é um repositório bastante extenso e serviu como validação do ambiente de visualização, testando se realmente as visualizações melhoram significantemente a visualização de softwares. Como resultados, foi obtido uma aplicação que pode ajudar os profissionais de engenharia de software a analisarem de uma forma mais fácil as informações, dados e métricas de softwares.

Palavras-chaves

Visualização de Software; Evolução de Software; Estratégias de Análise Visual

1. INTRODUÇÃO

A Visualização de Software pode ajudar engenheiros de software a lidar com a complexidade do software enquanto torna os programadores mais efetivos [1]. Grande parte dos gastos com software são oriundos de processos de manutenção e evolução de software [16]. Porém, a evolução de software é inevitável, uma vez que o software precisa ser melhorado e adaptado ao longo do tempo. Desta forma, a visualização de software é uma ferramenta importante no processo de desenvolvimento e evolução de software.

Engenheiros de software, seja na academia ou no contexto industrial, podem utilizar técnicas de visualização de informação como uma das fontes para compreender as diferentes etapas do ciclo de vida de um software em evolução [22]. A visualização de software é também importante nas etapas de manutenção, pois ela ajuda na compreensão do software permitindo identificar problemas que puramente com a observação do código-fonte talvez não fosse possível.

Muitas vezes, identificar os dados referentes a evolução de um software sem utilizar técnicas de visualização de software, pode ser uma tarefa árdua e por muitas vezes difícil [15], tornando a visualização de software bastante importante para análises de evolução. Porém, existem desafios para visualizar a evolução do software em níveis detalhados. Identificar diferenças em níveis de versões, por exemplo, não é uma atividade trivial de ser realizada [22]. Os softwares atuais que trabalham com análise visual, apesar de muitas vezes proverem tipos de visualizações com um nível de detalhes razoável, apresentam algumas limitações relacionadas à customização de visualizações e, em alguns casos, dificuldades de interação com o usuário.

Visualizar a evolução de um software é de extrema importância para extrair comportamentos e informações acerca dele. Existem diversas formas de visualizar esta evolução. Essas formas são denominadas estratégias de análise visual [22], as quais definem como se dá a construção de visualizações de evolução de software. No trabalho de Novais et al. [22] são apresentadas cinco estratégias diferentes, classificadas em duas grandes categorias: Estratégias Diferenciais e Temporais. O autor destaca a importância de se combinar diferentes estratégias de análises. Vários estudos foram conduzidos no sentido de mostrar os benefícios da combinação. Os estudos conduzidos podem ser vistos em Novais et al. [23] e em Novais et al. [25]. Entretanto, ainda está em aberto possibilidades de combinações diferentes [24].

Para facilitar essas combinações é importante construir ambientes extensíveis que permitam a utilização de diferentes visualizações com diferentes estratégias de análise. Nesse sentido, está sendo desenvolvido um novo sistema, chamado Visminer. O Visminer é um sistema Web de visualização que tem como objetivo ser um ambiente de multiperspectivas e multiestratégias [6]. Através de diferentes tipos de métricas e visualizações, ele auxilia na compreensão da evolução de

*Graduando em Análise e Desenvolvimento de Sistemas
†Prof. Doutor em Ciência da Computação

sistemas, possibilitando que os usuários possam ter uma visão mais esclarecedora de como está o estado dos softwares produzidos e que estão em constante evolução.

Este trabalho contribui com a construção de duas visualizações integradas ao Visminer, que fazem uso de duas estratégias de análise visual. A primeira visualização é o *Treemap*, que além de permitir a visualização de uma única versão do software, dá suporte a estratégia diferencial relativa [21]. Com esta visualização é possível realizar comparações entre elementos duas a duas versões considerando a hierarquia pacote-classe-métodos. A segunda visualização é a visão *parallel coordinates* que dá suporte a estratégia *Temporal Overview* [5]. Através dela tem-se uma visão mais global de determinado elemento selecionado a partir da visão anterior. É possível visualizar em uma visão temporal todos o valores das métricas em cada versão do software. Este módulo, desenvolvido neste trabalho, é denominado de *Visminer Development Evolution Visualization* (Visminer DEV). É proposto uma solução que traz tipos de visualizações configuráveis em diferentes visões das informações do projeto analisado em uma aplicação Web.

Como resultado deste trabalho, pretendeu-se obter uma solução que agregue ao Visminer um módulo Web que possa ser utilizado por profissionais da área de engenharia de software. Tais profissionais podem realizar análises detalhadas dos dados do projeto, a fim de ajudar nas tomadas de decisão acerca da evolução do software.

Para avaliar o ambiente desenvolvido, foi conduzido um estudo exploratório utilizando o repositório do software JUnit4. Foram identificadas algumas informações que provavelmente somente com as ferramentas proporcionadas pela visualização de software poderiam ser identificadas. Uma delas são os métodos que com a evolução do JUnit4 sofreram muitas variações dos valores de suas métricas. Outro ponto foi a identificação de um padrão de crescimento e decrescimento dos valores das métricas com relação a determinado elemento. As ferramentas do Visminer DEV ajudaram bastante na identificação dessas informações. Uma das ferramentas que ajudou foi a possibilidade de rapidamente alterar as métricas do repositório analisado. A navegação entre níveis no *Treemap* e os atributos visuais ajudaram também a encontrar rapidamente determinados padrões.

Além desta Introdução, este trabalho está organizado como se segue. A Seção 2 apresenta os assuntos relacionados a este trabalho. A seção 3 apresenta os trabalhos que de alguma forma se assemelham a este. A Seção 4 apresenta a solução desenvolvida neste trabalho juntamente com os métodos e materiais que mostra o passo a passo que foi seguido para atingir os resultados esperados. A Seção 5 mostra como o projeto foi avaliado e validado. Por fim, a Seção 6 conclui este trabalho, destacando as suas limitações e trabalhos futuros.

2. REFERENCIAL BIBLIOGRÁFICO

Esta seção apresenta os principais assuntos relacionados a este trabalho. A subseção 2.1 apresenta conceitos relacionados a Evolução de Software. A subseção 2.2 discute o tópico Visualização de Software. Por fim, na subseção 2.3 é abordado o assunto Estratégias de Análise Visual.

2.1 Evolução de Software

O termo Evolução de software não possui uma definição padrão [4]. Entretanto, pode-se dizer que evolução de software são todas as fases de desenvolvimento pertencentes a um software.

Evolução de software não é somente mudanças de código que possam ocorrer durante o ciclo de vida de um projeto [20]. Esse conceito está muito relacionado também com características desejáveis que um software deve ter, tais como: performance, capacidade de ser modificado e qualidade em geral [16].

A sociedade passou a depender bastante dos recursos tecnológicos e com eles temos certamente os softwares que passaram a controlar e automatizar diversas áreas críticas da sociedade, como por exemplo bancos, que possuem uma grande dependência com softwares para controle e gerência de capital [16]. A evolução de software serve justamente para que tais sistemas e aplicações possam ser modificados e atualizados de modo que sua qualidade esteja sempre melhorando. Além disso, apesar de o conceito de evolução de software ser muitas vezes confundido com manutenção de software, a evolução de software é uma atividade mais ampla que parte desde o início do desenvolvimento do software e prossegue por todo o seu ciclo de vida [22]. Portanto, pode-se dizer que a evolução de software está mais relacionada com o porquê, do que como o software mudou [22], pois a evolução de software se preocupa com os benefícios que determinada alteração e/ou manutenção pode trazer para o projeto do que como essa modificação será feita.

Apesar de a evolução de software não ser manutenção de software, podemos dizer que a manutenção de software faz parte da evolução de software. Partindo desse pressuposto, existem vários tipos de manutenção que influenciam diretamente na evolução de software [20], definidas por:

- ***Perfective maintenance*:** Este tipo de manutenção engloba todo tipo de modificação que o software sofre após ter sido implantado que englobe melhora de performance ou manutenibilidade;
- ***Corrective maintenance*:** É o tipo de manutenção que ocorre quando é necessário a correção de erros que apareceram após a implantação do software;
- ***Adaptive maintenance*:** É a modificação do software para adaptação de ambiente, por exemplo mudança de sistema operacional em que o software irá funcionar;
- ***Preventive maintenance*:** É a manutenção que ocorre com o intuito de prevenir que ocorram possíveis erros antes que possam acontecer.

Além da manutenção, existem outros mecanismos utilizados para evoluir um software [13]. Um deles é engenharia reversa. A engenharia reversa é importante principalmente quando é necessário entender a arquitetura ou comportamento de um software [20]. Ela faz parte da evolução de software, pois é de grande importância para a evolução e crescimento de sistemas que são de difícil manutenção. Por fim, mais um exemplo de técnica de evolução de software é a *Incremental change*. Essa técnica é utilizada na fase de planejamento do software, em que é necessário identificar quais partes do software são mais susceptíveis a mudanças e quais impactos uma mudança naquele setor irá impactar no software em geral [20]. Isso ajuda na tomada de decisão de realizar ou não as mudanças.

Dada toda a importância relacionada à evolução do software, bem como a dificuldade em gerenciá-la, é importante

ter abordagens que facilitam essa tarefa. Uma delas é a Visualização de Software, discutida na próxima seção.

2.2 Visualização de Software

Visualização de software é uma subárea da Visualização da Informação e utiliza recursos e tecnologias visuais para representar de forma organizada os dados coletados de determinada aplicação [18]. A visualização de software pode ajudar profissionais da engenharia de software a lidar com a complexidade que evoluir e manter um software pode requerer [1], uma vez que técnicas visuais de fato têm relevância para observar dados coletados.

No processo de visualização de software é necessário compreender os conceitos de atributos reais e atributos visuais e como são mapeados. Atributos reais são as propriedades do software, tais como valores atribuídos ao software. Por exemplo, um atributo real que existe em todo software é a quantidade de linhas de código que existe em um sistema. Outro exemplo, considerando a orientação a objetos, pode ser a quantidade de classes que um pacote de um determinado sistema possui. Tudo isso são propriedades existentes no software. Esses atributos são medidos através de métricas de software apropriadas, tais como Número de linhas de Código (LOC) e Complexidade Ciclomática (CC).

Por outro lado, os atributos visuais buscam por meio de elementos visuais (e.g., comprimento, largura e cor) representarem a evolução e valores de métricas referentes aos softwares analisados [22]. Ou seja, é possível representar em uma única visualização diversas métricas desde que sejam utilizados atributos visuais diferentes para que cada métrica possa ser identificada. Por exemplo, podemos ter em uma visualização a métrica CC com a cor vermelha para crescimento dessa métrica em diferentes versões do mesmo software. Já a cor verde usada para representar o decrescimento dessa métrica em diferentes versões do software. No mesmo contexto de visualização, podemos ter a métrica LOC mapeada no atributo visual área, em que quanto maior a área de um quadrado por exemplo, maior é o valor da métrica LOC.

Saindo do escopo dos conceitos de atributo visual e real, é importante ressaltar que a quantidade de abstrações de maneiras de produzir visualizações de softwares são imensas [17]. Isso porque as visualizações de software podem ser feitas utilizando diversos tipos de representação. Dentre elas, existem algumas que são bastante utilizadas, tais como o Treemap [12], bubble chart [27], line chart [27], entre outras diversas. O Treemap consiste em uma estrutura hierárquica constituída por diversos retângulos. Esses retângulos são aninhados de forma que elementos que pertencem a outro fiquem sempre no interior do retângulo maior. Já o bubble chart é um tipo de visualização constituída através de várias circunferências que possuem tamanhos diferentes e que são influenciados pelo valor da métrica que está sendo representada. Por fim, o line chart é um tipo de visualização que mostra um conjunto de informações utilizando marcadores que são conectados por linhas e seus valores são atrelados a valores definidos em um plano cartesiano. Essas representações são importantes para entender a visualização de software porque diversos casos de visualização de software utilizam esses tipos de visualização ou algum tipo de customização para representar suas visualizações.

Portanto, a visualização de Software busca por meio de gráficos, tabelas e outros recursos achar uma maneira que torne mais fácil para seres-humanos, por meio de recursos visuais entender, compreender e explorar as informações e dados a cerca de um projeto, com a finalidade de melhorá-lo [18].

2.3 Estratégias de Análise Visual

Quando softwares evoluem, tudo que está envolvido com ele também evolui [22]. Então, tanto suas entidades, artefatos e suas propriedades também podem evoluir. Estas evoluções podem se referir a métodos que foram removidos, adicionados ou alterados, pode se referir também a um módulo novo ou a um módulo removido, entre outros aspectos. Visualizar a evolução de software é a tarefa de se visualizar os softwares sendo evoluídos. Para isso, existem diferentes estratégias de análise visual para representar e compreender visualmente a evolução. O conceito de Estratégia de Análise Visual define como uma parte independente do software pode ser apresentada para análise [22]. As estratégias de análise visual definem a evolução de determinado módulo de software e como pode ser analisado. Portanto, todos os elementos de software que podem ter sido alterados no processo de evolução de software podem ser visualizados por meio de uma estratégia de análise visual. Porém, essa evolução independente de módulos pode ser representada de diversas formas e dependendo da maneira que seja feita, pode dificultar ou facilitar a compreensão dos dados [22]. Portanto, a maneira que é escolhida para representar as mudanças que estão ocorrendo no contexto de um software são de extrema importância para que o desenvolvimento de uma visualização de software seja bem feito.

As estratégias de análise visual são classificadas em duas grandes categorias: temporal e diferencial [22]. As estratégias temporais representam a evolução do software levando em consideração as diferentes versões disponíveis daquele software. Softwares possuem histórias, que são os registros de tudo que ocorreu no desenvolvimento do software. Muitas vezes, a medida que o software cresce, versões do software são definidas. Essas versões do software são os estados do sistema em um determinado espaço de tempo. Em outras palavras, quando um software está sob análise ele possui um histórico de versões que pode ser bastante extenso. Nas estratégias temporais todo esse histórico de versões é levado em consideração nas análises. Então, em um software com n versões e que tenha sido decidido que seriam analisadas as versões (n5) até a versão n, no contexto da estratégia temporal, todas as versões entre as escolhidas também serão levadas em consideração para o desenvolvimento da visualização. Ou seja as versões n4, n3 e assim por diante estarão presentes na representação da visualização. Existem diferentes tipos de estratégias que pertencem a estratégia temporal. Dentre elas existem a Temporal Overview (TO), Temporal Snapshot (TS) e a Temporal Acumulativa (TA).

- **Temporal Overview:** Esta estratégia gera uma visão geral dos dados analisados. Ou seja, gera uma visualização que engloba desde a primeira versão do software até um determinado ponto;
- **Temporal Snapshot:** Esta estratégia preocupa-se mais em mostrar o estado do software em determinada versão específica, assim ela desconsidera módulos que não estejam presentes na versão selecionada. Porém, algum atributo visual é utilizado para representar algum atributo real temporal;

- **Temporal Acumulativa:** Esta estratégia leva em conta os valores absolutos de alterações que ocorreram entre as versões com o intuito de analisar a evolução do software. Levando em consideração todas as versões de um determinado intervalo.

As estratégias diferencias, diferentemente das temporais, levam apenas em consideração duas versões escolhidas para serem analisadas [22]. Ou seja, ao contrário das estratégias temporais se forem escolhidas as versões n e a versão n2, somente essas duas versões serão analisadas e representadas nas visualizações. Os atributos visuais que são representados na visualização são utilizados para representar as diferenças entre as versões selecionadas. As estratégias diferenciais são subdivididas em Diferencial Relativa (DR) e Diferencial Absoluta (DA).

- **Diferencial Relativa:** A estratégia da diferencial relativa foca principalmente em alterações que ocorreram nos módulos do software entre as versões. Geralmente focando na representação de crescimento e decrescimento das métricas relativas as versões do mesmo software. Ou seja, em uma versão x do software um módulo possui a métrica CC com o valor de 50 já na versão y este mesmo módulo possui o valor da mesma métrica em 80, a estratégia diferencial relativa irá representar este acréscimo na visualização considerando essa variação em relação as demais variações do próprio software;
- **Diferencial Absoluta:** A estratégia Diferencial Absoluta não se preocupa com crescimento ou decrescimento dos elementos do software e sim em identificar o aparecimento e/ou desaparecimento de propriedades de elementos entre as versões. Ou seja, ela mostra propriedades de elementos que existiam em determinada versão e que depois deixaram de existir e propriedades de elementos que não existiam em determinada versão e passaram a existir em outra.

A visualização de evolução de software enfrenta diversos problemas e desafios para serem implementados. Cada uma dessas estratégias citadas podem ajudar a abordar de uma melhor maneira as visualizações de software [22]. Portanto, já que cada tipo de estratégia engloba uma maneira de visualizar e analisar o software, a combinação delas pode trazer uma visualização mais completa [24].

3. TRABALHOS RELACIONADOS

Nesta seção, serão detalhados os trabalhos relacionados a este projeto. Neste detalhamento é feito uma descrição geral do trabalho e é descrito como o trabalho se relaciona com este projeto. É considerado também os pontos fortes e fracos identificados.

Combinando estratégias de análise visual de evolução de software em diferentes níveis de detalhe – No trabalho de Novais et al. [22] é apresentado uma proposta de visualização de software em detalhes. Para isso, são utilizados diferentes tipos de estratégias de análise visual. A partir da combinação dessas estratégias, é definido uma infraestrutura para visualização de evolução de software. Essa infraestrutura foi a base para o desenvolvimento de uma abordagem de visualização de evolução de software que combina versões, módulos e atributos de maneira organizada.

O Visminer DEV foi projetado a partir desse trabalho. Do trabalho de Novais et al. [22] são retirados a maioria dos conceitos de análise visual, além de prover também conceitos de visualização de software. O grande diferencial é que Visminer DEV foi desenvolvido para Web enquanto que o trabalho de Novais et al. [22] foi desenvolvido na forma de plugin do Eclipse [3]. Ambos os trabalhos atuam da mesma maneira ao gerar visualizações partindo de métricas que foram calculadas previamente.

Voronoi treemaps for the visualization of software metrics – No artigo de Balzer et al. [2] é apresentado um novo tipo de visualização. Esta visualização trata-se de um Treemap, que ao contrário dos Treemaps tradicionais não é baseado em retângulos. Em vez disso são utilizados polígonos arbitrários.

Este artigo apresenta similaridades ao Visminer DEV por utilizar primordialmente um Treemap, além de possuir zoom interativo para navegação entre os níveis da visualização.

Um dos pontos fortes deste trabalho é a inovação da visualização do Treemap que traz consigo informações adicionais sobre as métricas. Portanto, traz mais informações para análise do software. A Figura 1 mostra a representação abordada no artigo de Balzer et al. [2].

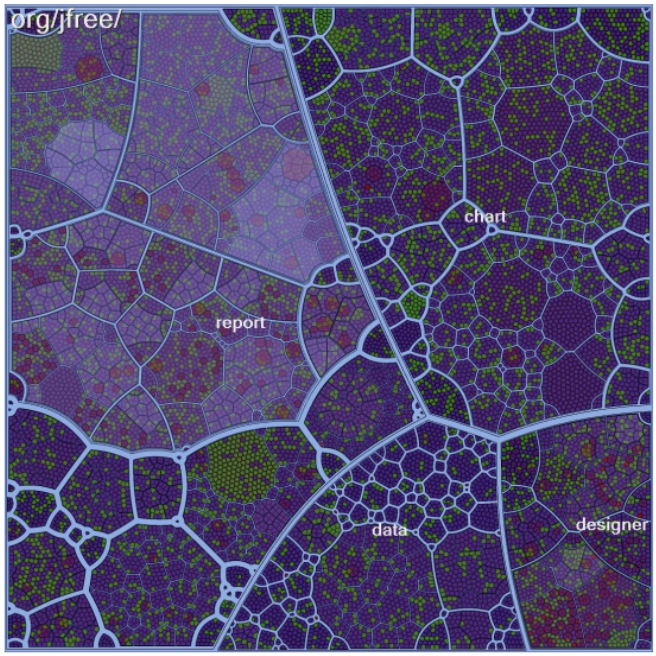

Figura 1: Voronoi TreeMap (Balzer et al., 2005)

Pargnostics: Screen-Space Metrics for parallel coordinates – No artigo de Dasgupta et al. [7] são propostas algumas métricas para gerar visualizações utilizando a técnica do *parallel coordinates*. Essas métricas dão origem a um diagnóstico do parallel coordinates. Isso originou um trabalho de filtragem e delineou o que realmente era necessário ser mostrado no parallel coordinates para o usuário final.

Esse trabalho de Dasgupta et al. [7] se assemelha ao parallel coordinates presente no Visminer DEV, por tentar construir um parallel coordinates que somente mostre informações relevantes sobre os dados analisados. Porém, apesar de se preocupar bastante com as informações que serão mostra-

das, o parallel coordinates do trabalho de Dasgupta et al. [7] apresenta talvez alguns problemas no quesito atributos visuais. Pois, por exemplo as linhas presentes na visualização são todas da mesma cor, confundindo por vezes os dados visualizados, como pode ser visto nas Figuras 2 e 3.

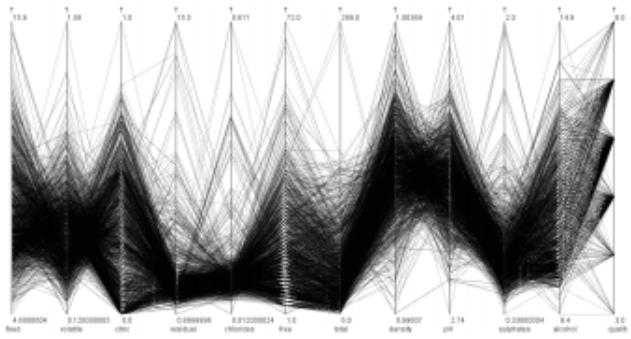

Figura 2: Layout do *parallel coordinates* (Dasgupta et al., 2010)

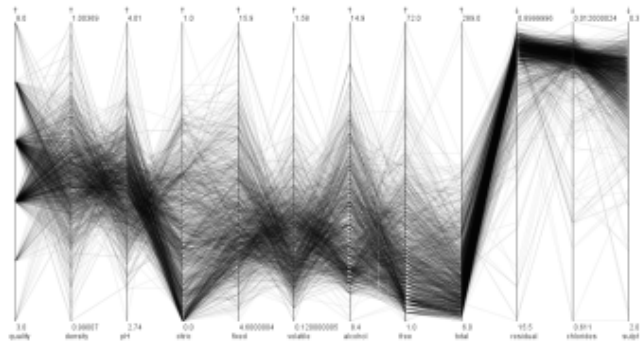

Figura 3: Layout do *parallel coordinates* Variação (Dasgupta et al., 2010)

No Visminer DEV, já houve uma preocupação maior com relação aos atributos visuais. Definindo cores diferentes para cada linha visualizada, além de se possível visualizar dados isolados ocultando algumas linhas horizontais.

Illustrative parallel coordinates – Nesse trabalho de McDonnel et al. [19], são utilizadas técnicas de renderização gráfica para melhorar a experiência da visualização do parallel coordinates. O ponto forte desse trabalho é a melhora no sentido de comparar os dados analisados, pois a definição de cada linha fica mais visível e mais fácil de analisar. Como na Figura 4.

Esse trabalho ajudou na construção do parallel coordinates do Visminer DEV, por utilizar técnicas para melhorar os atributos visuais, de forma que fosse o mais fácil possível visualizar os dados. No parallel coordinates do Visminer DEV, os atributos visuais utilizados para melhorar a experiência do usuário são as cores bem definidas por exemplo.

Visualization-based Analysis of Quality for Large-scale Software Systems – O trabalho de Langelier et al. [14] chega ao consenso de que para melhorar a análise da evolução de software, a combinação das técnicas de visualização de software mais a observação humana tem a capacidade de

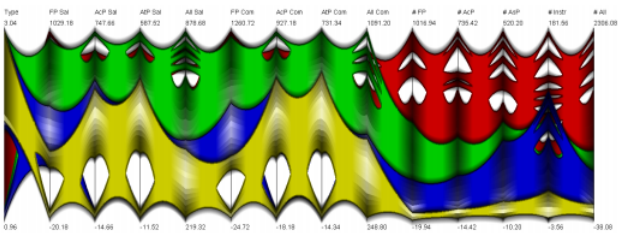

Figura 4: Layout do *parallel coordinates* (McDonnell et al., 2008)

identificar os problemas referentes a evolução. Então, é proposto um framework para ajudar nesse desafio. Como um dos layouts de visualização foi escolhido o Treemap. Nisso, o trabalho de Langelier et al. [14] se assemelha ao Visminer DEV, pois no seu Treemap assim como no do Visminer DEV o uso de cores e linhas para delimitar os pacotes, classes e métodos foi bastante utilizado. Porém, esse Treemap ainda não possui uma navegação adequada para visualizar com maior detalhes os elementos que estão aninhados dentro de outros, como os métodos por exemplo. Já, no Visminer DEV a ferramenta de Zoom facilita bastante essa análise mais detalhada, pois proporciona uma visão mais detalhada dos elementos inferiores na hierarquia do Treemap.

4. VISMINER DEV

Nesta seção será explicado como foi desenvolvido o Visminer DEV – um ambiente de visualização web com suporte a múltiplas Estratégias de Análise Visual. É apresentado também as tecnologias que foram utilizadas para o seu desenvolvimento.

A Figura 5 apresenta uma visão geral de como o Visminer DEV está organizado, dando destaque para sua relação com os principais componentes que fazem parte da solução.

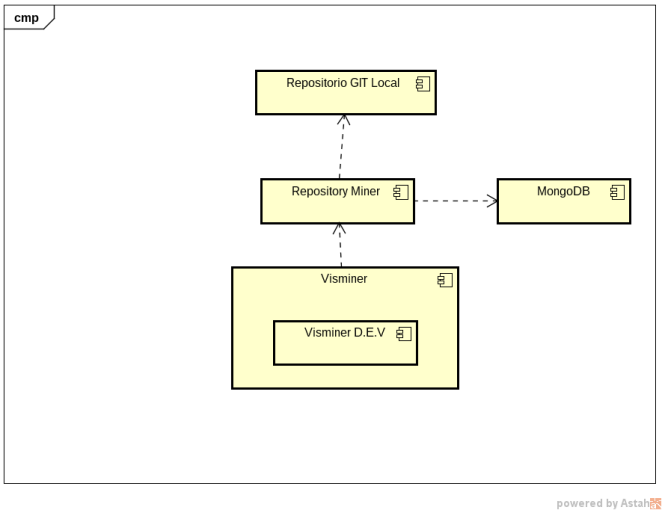

Figura 5: Visão Estrutural de Alto Nível

O Visminer DEV foi construído utilizando como base uma solução de mineração de repositórios de código-fonte: o Repository Miner (RM). O Repository Miner minera os dados dos repositórios de código-fonte coletando informações de ar-

quivos fonte, *commits* e *committers*, calculando os valores de diversas métricas, e identificando *code smells*. O Repository Miner atualmente minera projetos de software escritos na linguagem de programação JAVA e que tenham sido armazenados no sistema de controle de versão baseado no GIT. Após ter minerado os dados do repositório, o Repository Miner armazena os dados em um banco de dados não relacional: o MongoDB. Após ter todos os dados armazenados nas coleções no MongoDB, o Vismíner, pode acessá-las através de consultas presentes no Repository Miner. Para realizar essas consultas é utilizado uma API *RESTful*, que é desenvolvida para que o Front-End (desenvolvido em Angular.js e JavaScript) possa realizar requisições de dados para gerar visualizações. O Visminer DEV é um módulo do Vismíner que tem como objetivo construir visualizações relacionadas à evolução de software utilizando múltiplas estratégias de análise visual.

4.1 Método e Materiais

O método e os materiais utilizados neste trabalho são descritos nas subseções seguintes. Na subseção 4.1.1, são descritas as tecnologias utilizadas na construção do software em questão, bem como suas características. Na subseção 4.1.2, é explicado as metáforas de visualização utilizadas. Na subseção 4.1.3, é demonstrado as etapas que foram feitas para o desenvolvimento da aplicação.

4.1.1 Tecnologias e Características

Para o desenvolvimento da aplicação Visminer DEV se fez necessário identificar qual é a atual conjuntura estrutural que o Visminer possui. A partir dela, foram definidas quais eram as principais tecnologias para manter a compatibilidade entre o Visminer DEV e o Visminer. Portanto, após essa análise, as tecnologias que foram primordialmente utilizadas foram:

JAVA: esta tecnologia é a que está presente em todo o desenvolvimento do Repository Miner e que também é a linguagem de programação utilizada para o desenvolvimento da API *RESTful*. Ou seja, toda a parte do back-end foi desenvolvido em Java. Para o desenvolvimento da API *RESTful* foi utilizado o *Framework* JERSEY [26], que é um *open source Framework* para o desenvolvimento de *RESTful Web Services* em Java.

AngularJS: esta tecnologia já está sendo usada em um dos módulos Web que o Visminer já possui. Então, como modo de manter um padrão de código e um padrão visual esta tecnologia foi utilizada no desenvolvimento do Visminer DEV. O AngularJS é uma plataforma baseada em JavaScript para ser utilizada no *front-end* de aplicações Web. Neste trabalho, ela foi utilizada no desenvolvimento de grande parte da aplicação devido ao Visminer DEV ser basicamente uma aplicação *front-end* que consome dados gerados pelo Repository Miner. Além do AngularJS, como complemento, foram utilizados o Javascript nativo para manipular dados em JSON necessários para gerar as visualizações, bem como o JQUERY que foi amplamente utilizado para manipular alguns dados da aplicação.

MongoDB: é um banco de dados multiplataforma que evita a utilização do modo tradicional de banco de dados relacionais, utilizando estruturas em JSON. Nesta aplicação, o MongoDB foi utilizado como local de armazenamento de dados que o Repository Miner utiliza e por consequência o Visminer DEV consome esses dados para gerar suas visualizações.

HighCharts: é uma biblioteca feita em JavaScript para gerar gráficos e diversos tipos de visualizações. Nesta aplicação, ele foi utilizado para gerar algumas das visualizações definidas pelas estratégias de análise visual.

D3.js: é uma biblioteca JavaScript para manipulação de documentos baseados em JSON com o intuito de gerar visualizações customizadas. Esta biblioteca foi utilizada para gerar algumas das visualizações presentes no Visminer DEV.

Como metáforas de visualização para o Visminer DEV foram escolhidas o TreeMap e o parallel coordinates que serão descritas na subseção abaixo.

4.1.2 Metáforas de Visualização

Treemap: Segundo Shneiderman et al. [12], o Treemap é um modo de preenchimento de espaço para visualização de uma grande quantidade de dados hierárquicos como pode ser visto na Figura 6. Os Treemaps passaram a ser bastante utilizados na visualização de software, uma vez que ele permite mostrar os dados hierárquicos de projetos de software orientados a objetos (pacote, classe método). No caso do Vismner DEV, o Treemap é utilizado para representar métodos, classes e pacotes. A visualização permite visualizar as relações de pertencimento entre os níveis, tornando por exemplo a compreensão de qual é o elemento em relação a determinada métrica que possui a maior significância. Treemaps necessitam que um peso seja dado para cada elemento representado [12]. No Visminer DEV foram atribuídos dois tipos de pesos para os elementos que o Treemap representa. O primeiro é a área. No Visminer DEV a área de cada retângulo presente no Treemap representa o valor de uma métrica previamente selecionada. Quanto maior for o valor dessa métrica, maior será o tamanho do retângulo de um elemento no Treemap que representa um componente relacionado, como um método. Neste caso, ambas visualizações que utilizam o Treemap no Visminer DEV representam os valores das métricas de uma versão na área dos retângulos do Treemap. O segundo atributo visual associado a cada elemento presente no Treemap no Visminer DEV é a sua coloração. Este atributo é abordado da seguinte forma: uma métrica é atribuída para ser representada por cores no Treemap, então é analisado os valores da métrica selecionada e de acordo com o tipo de estratégia de análise visual as cores são atribuídas. No caso da estratégia *single version*, é definida uma escala da cor marrom em que a medida que a tonalidade do marrom fica mais escuro significa que aquele elemento possui um valor associado a métrica maior. No caso do marrom ser mais claro, significa que aquele elemento possui um valor associado a métrica menor. No caso da diferencial relativa, são definidas duas escalas de cores. Uma relacionada a crescimento do valor da métrica, representada pela cor vermelha e outra para definir decrescimento do valor da métrica, representada pela cor verde. Nestes dois casos, são definidas escalas de cores assim como no single version. Em que quando a tonalidade do vermelho varia, significa que o crescimento do valor da métrica para aquele elemento teve um maior aumento (tom mais escuro do vermelho) ou um menor aumento (tom mais claro do vermelho). A tonalidade do verde segue o mesmo fluxo, porém com significado diferente. No caso do verde, as variações das tonalidades determinam se o valor da métrica do elemento colorido diminuiu em maior quantidade (verde mais escuro) ou diminuiu em menor quantidade (verde mais

claro).

Por fim, o Treemap foi escolhido como uma metáfora de visualização do Visminer DEV por conseguir separar os elementos do software de um modo explícito, em outras palavras, de uma maneira que não permita confusões acerca de qual elemento está sendo representado na visualização.

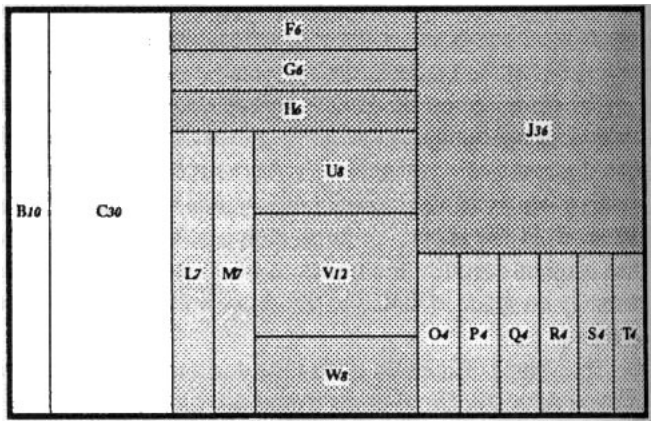

Figura 6: Shneiderman et al. Treemap.

Parallel Coordinates: De acordo com Johansson et al. [11], o tipo de visualização do *parallel coordinates* tem sido por alguns anos uma técnica bem popular para visualizar dados multivariados. Ela é a representação de múltiplas linhas que descreve diferentes itens, essas linhas intersectam outras linhas verticais que representam variáveis [9] como pode ser visto na Figura 7. No Visminer DEV, o parallel coordinates foi utilizado para representar os valores das métricas para cada versão do software, representando somente um elemento. No Visminer DEV, as métricas referentes ao elemento selecionado fica na horizontal à esquerda. Cada métrica é representada por uma linha de cor diferente. Essas cores são definidas por uma sequência pré-definida, em que a cada métrica que será representada na visualização é designada uma cor que a representará. Essas linhas fazem intersecções em linhas paralelas na vertical, em que cada uma delas representa uma versão diferente do repositório analisado. Isso pode tornar mais fácil a identificação e visualização das mudanças que cada elemento sofre com relação as suas métricas quando as versões vão sendo alteradas. Isso porque o *parallel* coordinates consegue proporcionar uma visão geral dos valores das métricas para todas versões do repositório.

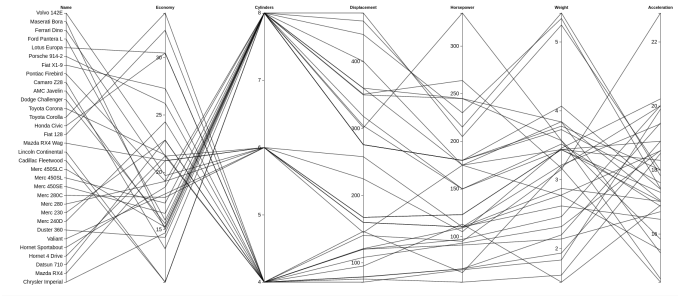

Figura 7: Heinrich Parallel Coordinates.

4.1.3 Etapas de Desenvolvimento

Para o desenvolvimento deste projeto foi necessário realizar algumas etapas descritas abaixo.

Primeiramente, foi feito um estudo sobre como funciona o Repository Miner. Para isso foi feito o download do software em questão e analisado suas classes e métodos, para que fosse possível entender como o Repository Miner analisa um repositório local GIT e calcula métricas baseadas em seu código. Após esse estudo, foi realizado um estudo de como esses resultados eram armazenados no MongoDB. Foi feita uma análise das estruturas de coleção que o Repository Miner gera no âmbito do MongoDB para que se tornasse mais fácil identificar em futuras consultas em que coleção os dados que precisavam ser consumidos estavam armazenados. Depois disso, foi iniciado um estudo de como as consultas do Repository Miner funcionavam e quais dados eram passíveis de consulta.

Feito isso, foi necessário investigar como o Visminer estava estruturado no contexto do seu HTML e JavaScript para que dessa forma a inclusão da perspectiva do Visminer DEV pudesse ser incluída sem maiores problemas de compatibilidade.

Após ter conhecimento do funcionamento do Visminer, fez-se necessário o estudo de como as tecnologias escolhidas podiam ser aplicadas às necessidades da aplicação. Dessa forma, foram iniciados estudos mais profundos acerca de todas as tecnologias citadas na subseção 4.1.1.

Com todos esses conceitos estudados e devidamente entendidos, iniciou-se o processo de definição das visualizações que estariam presentes no Visminer DEV. Para isso foi realizado uma seleção de possíveis visualizações e foi decidido inicialmente que a principal visualização seria um TreeMap. Posteriormente foi escolhido como fonte desse Treemap o Large TreeMap [10] provido pela biblioteca HighCharts. Antes de descrever como o Treemap do Visminer DEV foi definido, é importante salientar que para atingir a visualização desejada foi necessário realizar algumas alterações no large treemap original. Uma das alterações, foi a necessidade de alterar tanto as bordas que separam os elementos, quanto a maneira que o Highcharts coloria cada elemento na visualização. Foi necessário também, adicionar alguns eventos de interação com os elementos do Treemap, como por exemplo, a adição do evento do botão direito sobre o nome do elemento, necessário para gerar a visualização parallel coordinates. Foi definido que o Treemap disporia de três níveis navegáveis para um melhor aproveitamento da visualização. O primeiro nível foi destinado para a representação dos pacotes do repositório analisado. Neste nível, cada pacote seria representado por um retângulo que seriam divididos por linhas na cor preta para definir seus limites. No segundo nível, são representadas as classes presentes em cada pacote, ou seja, todas as classes pertencentes a determinado pacote devem estar aninhadas no retângulo referente ao seu pacote. Essas classes, são representadas por retângulos com bordas na cor amarela. Por fim, no nível 3 podem ser vistos os métodos de cada classe. Seguindo o mesmo raciocínio presente na relação classe-pacote, os métodos são aninhados dentro do retângulo da classe à qual pertencem e representados por outros retângulos. Porém, suas bordas são na cor vermelha.

Primeiramente, o Treemap foi utilizado para representar uma única versão (single version) no Visminer DEV. Neste caso, é permitido ao usuário escolher duas métricas que serão representadas no Treemap, uma que seus valores refleti-

rão na área de cada retângulo do Treemap e outra que seus valores refletirão na tonalidade da cor de cada retângulo do Treemap. A seguir, as Figuras 8 9 10 representam um pouco das visualizações produzidas.

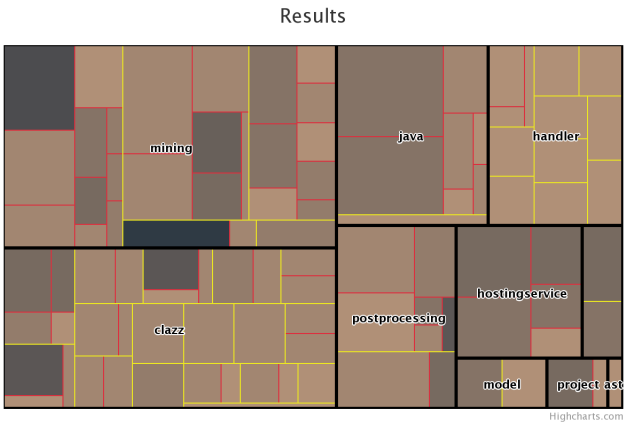

Figura 8: Nível 1 do Treemap-single version

Na Figura 8 é mostrado o nível 1 do Treemap (single version). O tamanho e as tonalidades das cores estão variando de acordo com a descrição que foi explicada anteriormente. Já na Figura 9 pode ser visto o segundo nível da visualização. Por fim, na Figura 10 é mostrado o último nível presente no Treemap single version.

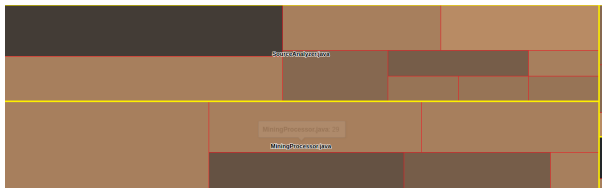

Figura 9: Nível 2 do Treemap-single version

Figura 10: Nível 3 do Treemap-single version

Após definir o Treemap com a perspectiva do single version, foi estudado a possibilidade de inserção de uma visualização mais detalhada dos dados que estão contidos no Treemap. Então, foi decidido a implementação da perspectiva do parallel coordinates. No Visminer DEV, essa visualização foi utilizada para demonstrar a variação de valores das métricas a medida que o software vai evoluindo suas versões. Na Figura 11 é possível observar as versões de um software e os valores de suas métricas para cada versão. Para visualizar esses resultados é necessário clicar com o botão direito do mouse sobre o nome do elemento que se deseja visualizar separadamente nesta visualização do parallel coordinates, podendo realizar este evento em qualquer elemento do Treemap, seja pacote, classe ou método. Então, são coletados os valores das métricas para cada versão do software do elemento clicado e a visualização pode ser vista logo abaixo do Treemap. Quando o parallel coordinates já está na tela, se outro elemento for clicado da maneira que foi descrita anteriormente, o parallel coordinates é refeito com os dados do novo elemento clicado.

Para desenvolver essa aplicação foi necessário a utilização da biblioteca JavaScript D3.js. As Figuras 12, 13 e 14 mostram as colunas presentes na perspectiva parallel coordinates. Nelas é possível perceber as variações das métricas de acordo com versões do repositório.

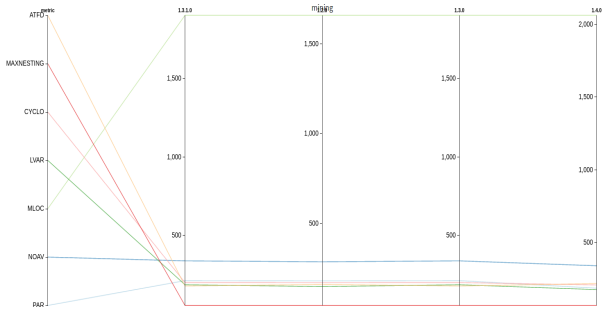

Figura 11: parallel coordinates Visão Geral

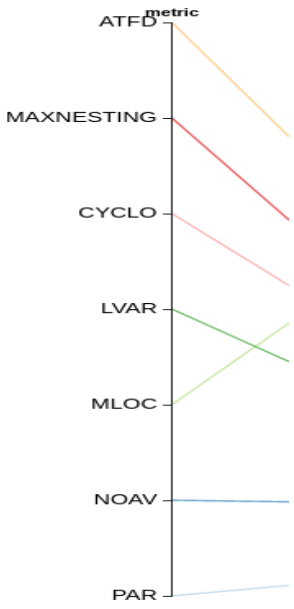

Figura 12: Coluna Métricas

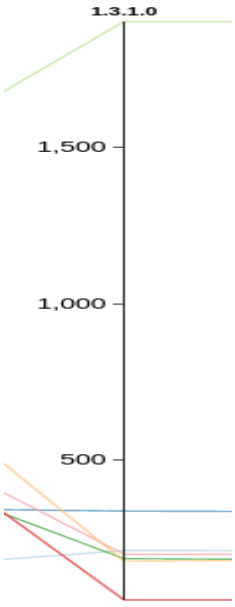

Figura 13: Coluna Versão 1

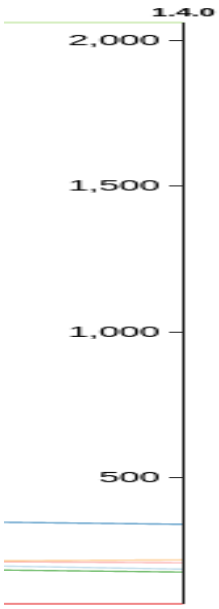

Figura 14: Coluna Versão 2

O Treemap também pode ser utilizado para a visualização de evolução de software. Para isso foi implementada uma estratégia de análise visual no Visminer DEV, denominada diferencial relativa. Essa estratégia que é semelhante ao que é proposto em Novais et al. [21], utiliza cores para distinguir as diferenças de valores entre as métricas pertencentes aos elementos comparados de duas versões do repositório analisado. Para implementar essa visualização, também foi utilizado o Treemap fornecido pelo Highcharts. Neste Treemap, são representados todos os elementos que as duas versões analisadas possuem em comum e as diferenças dos valores das métricas selecionadas. Os valores das métricas e os elementos a qual elas pertencem são representados da seguinte maneira: para gerar essa visualização é necessário a seleção de duas métricas, uma para ser representada pela área (A) da Figura 15 dos componentes do Treemap e a outra para ser representada através das cores (B) da Figura 15. Também é necessário escolher duas versões do software analisado para que sejam comparadas (E) e (F) da Figura 15. As letras (C) e (D) representam o tipo de visualização que será utilizada, caso seja escolhida o single version a letra (E) é ocultada. A letra (G) é o local onde pode ser escolhido a estratégia que será utilizada, porém atualmente só está implementada a diferencial relativa. E a letra (H) é o botão para a visualização ser gerada. O sistema de representação da área é bem simples, ela simplesmente exibe os valores das métricas referentes aos elementos da versão número 2. Já o sistema de cores segue as seguintes regras: As subtrações que obtiverem o resultado igual a zero são coloridas com a cor branca. As subtrações que apresentarem crescimento (resultados positivos) são representados em tonalidades da cor vermelha. É definida uma escala de cores com base no maior valor encontrado de crescimento e partir dele são estabelecidos em que grau de tonalidade da escala da cor vermelha o elemento se encontra. Ou seja, quanto maior for o crescimento do valor da métrica, mais escuro será o vermelho que o representará, como pode-se observar na Figura 16 letra (A). Para os decrescimentos (valores negativos) é usada a mesma lógica que é utilizado para crescimento de valores das métricas, só que neste caso a escala é feita com tonalidades da cor verde. Como na Figura 16 letra (B). Nas Figuras 16 17 é possível observar dois níveis da representação da estratégia diferencial relativa no Visminer DEV.

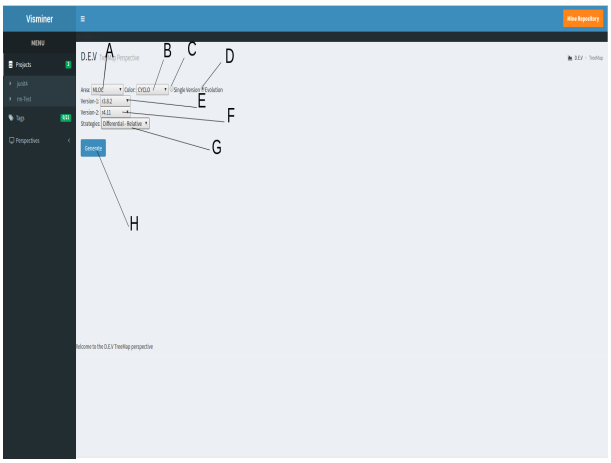

Figura 15: Configuração Visminer DEV

Após a definição do Treemap, para que fosse possível obter mais detalhes sobres os resultados do Treemap, tanto do single version quanto da diferencial relativa, foi construído a visualização do parallel coordinates, possibilitando a observação individual de cada elemento com relação aos valores de suas métricas por versão.

5. ESTUDO EXPLORATÓRIO

Esta seção apresenta o estudo exploratório conduzido para avaliar o potencial das visualizações produzidas pelo Visminer DEV. Outros resultados desta avaliação podem ser encontrados em http://rafabispo93.github.io/JunitAnalysis.

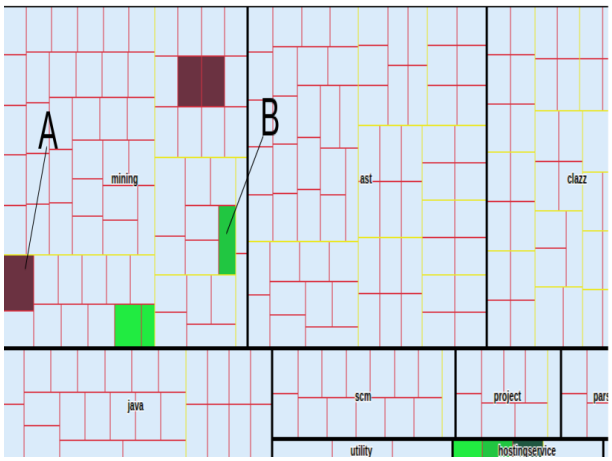

Figura 16: diferencial relativa Nível 1

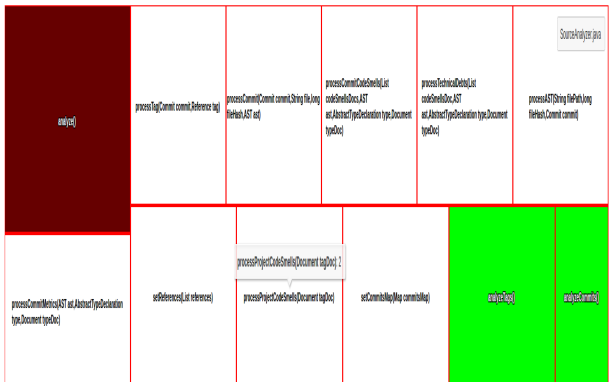

Figura 17: diferencial relativa Nível 3

Para conduzir esse estudo foi utilizado como software objeto o JUnit4: um *Framework Open Source* para testes automatizados criado por Erich Gamma e Kent Beck [8]. O JUnit4 é escrito em Java e permite a criação de testes automatizados para a mesma linguagem. Com o JUnit é possível verificar se os métodos de cada classe funcionam da maneira que se espera, exibindo possíveis erros, falhas ou inconsistências que possam haver nos métodos.

Nas seções seguintes, detalhamos o *design*, execução e resultados do estudo.

5.1 Planejamento

Objetivo. O principal objetivo do estudo foi testar os recursos visuais utilizados na solução desenvolvida, utilizando um software real, com o intuito de verificar possíveis comportamentos e problemas presentes no JUnit4 que poderiam não ser identificados com facilidade sem os recursos de visualizações de software. Esta avaliação tem como objetivo também analisar e descrever como as mudanças que ocorreram durante a evolução do JUnit4 foram refletidas nas visualizações do Visminer DEV.

Questões a serem respondidas. Esta avaliação é uma análise exploratória dos recursos visuais do Visminer DEV. Para atingir essa meta, foram realizadas análises de 5 versões do repositório do JUnit4 utilizando o Visminer DEV. Para validar o experimento algumas questões foram definidas. Segue abaixo as questões:

1. O que foi possível identificar acerca do JUnit4 após essa avaliação ? Existem problemas ou comportamentos em algum elemento que possuem grande influência no funcionamento do JUnit4 ?
2. Existe alguma relação entre os valores das métricas deste software ?
3. O Visminer DEV reflete as mudanças que ocorreram no JUnit4, de acordo com sua documentação ?

Essas perguntas são importantes, pois ao responder a questão 1 é possível destacar se a visualização de software no Visminer DEV torna capaz a identificação de métodos que tem grande influência dentro do software, ou seja métodos que são cruciais para o funcionamento do software. Ao responder a questão 2, é possível checar quais métricas podem talvez influenciar outras, e assim por exemplo determinar que ao aumentar a complexidade ciclomática de métodos outra métrica também pode crescer. Desta forma pode ser possível realizar intervenções no software com maior conhecimento de em que isso afetará. Já a questão 3, serve para identificar se as mudanças ocasionadas pela evolução do software ficam mais fáceis de serem identificadas quando são utilizadas as visualizações que o Visminer DEV implementa. Isso comparando o uso dessas visualizações com o ato de ler a documentação do software. Algumas estratégias foram planejadas com o intuito de responder essas perguntas. Primeiro, analisar principalmente os métodos que na maioria das versões sofreram variações tanto de sua área tanto de sua cor. Isso pode ajudar na identificação de métodos importantes. Outra estratégia, foi comparar as versões mais próximas entre si, para que desta forma o maior número de variações das métricas pudessem ser identificadas.

5.2 Execução

As avaliações foram executadas entre os dias 20 e 31 de Março de 2017. Durante este tempo, foram analisadas 5 versões do JUnit4. Esta avaliação foi conduzida da seguinte forma: Foram utilizados para análise as versões do JUnit4 que possuíam *Release Notes*. Esses *Release Notes* podem ser encontrados nesta url http://junit.org/junit4/. Esta estratégia foi seguida pelo fato de através dos *Release Notes* e de sua documentação ser possível identificar as mudanças que ocorreram entre as versões do software. Dessa forma, verificar se os resultados obtidos nas visualizações do Visminer DEV estavam coerentes e adequados.

Foram geradas 15 visualizações para que fosse possível realizar uma análise sobre elas. Primeiro, foi gerada as visualizações single version do repositório, para analisar como se comportava cada versão individualmente. Após a análise de cada versão individual, foi analisado como essas versões mudavam de uma para outra, utilizando a estratégia de análise visual diferencial relativa, observando quais métodos que se destacavam pelos seus valores de métricas tanto na visualização do *single version* quanto na diferencial relativa. Concomitantemente, quando um método era observado e fosse necessário obter mais informações sobre ele, a visualização *parallel coordinates* era acionada, para que fosse possível obter uma visão individual do método.

Foram utilizadas 2 métricas, a *Method Lines of Code* (MLOC) e a *Cyclomatic Complexity* (CC), pois ambas facilitam a

identificação de alterações nos métodos do software analisado.

5.3 Resultados

Inicialmente foram feitas algumas análises quanto a visualização single version para a versão r4.9 e r4.10 em que a métrica para a área foi a *Method Lines of Code* (MLOC) e a cor foi a *Cyclomatic Complexity* (CC). Os resultados foram os seguintes:

Nesta versão comparando com uma versão anterior, existem vários elementos que apareceram, talvez por conta da quantidade de pacotes que aumentou. Porém, apesar de ser mais fácil encontrar os novos elementos usando o *single version* do que não utilizando uma visualização, encontrar todos esses elementos requer atenção e observação. Ou seja, o single version consegue mostrar o que mudou no sentido de quantidade de pacotes, classes e métodos, entretanto é preciso observar cada elemento presente na visualização e checar se na versão anterior ele já existia. Nesta versão existem 29 pacotes, ou seja um aumento de 21 pacotes comparado com a versão anterior. O projeto nesta versão está muito maior, isso afeta diretamente os valores gerais das métricas. Uma das grandes mudanças é perceptível no MLOC, por estar medindo exatamente quantidade de linhas dos métodos. Para chegar a essa conclusão, foi necessário observar o tooltip de método por método, pois nele contém os valores da métrica que é representada pela área. Os métodos *start*/(método) que pertence a classe e ao pacote respectivamente *TestRunner.java*/(classe), *textui*/(pacote) letra (A) da Figura 18 e *processArguments*/(método) que pertence a classe e ao pacote respectivamente, *BaseTestRunner*/(Classe), *runner*/(pacote) letra (B) da Figura 18 continuam sendo os maiores com relação a métrica MLOC, porém apareceram diversos métodos que estão bastante próximos deles como por exemplo: o método *validateNTypePaameterONtype* da classe e pacote respectivos, *NoGenericTypeParametersValidator*/(Classe), *model*(pacote) letra (C) da Figura 18. Quanto à complexidade ciclomática, o elemento que apresentou o maior valor foi o método *getTest* que pertence a classe e ao pacote respectivamente, *BaseTestRunner*/(classe), *runner*/(pacote). Comparando com a versão anterior alguns elementos tiveram suas complexidades ciclomáticas reduzidas. Porém o número de elementos com complexidade ciclomática próxima a do *getTest* cresceu consideravelmente. Na Figura 18 é possível observar a visão geral dessa visualização.

Depois ainda analisando repositório a partir da visualização do single version, a versão r4.10 foi observada. Nesta versão em termos de elementos novos não tem praticamente nenhuma grande diferença comparada com a versão r4.9, ainda existe o mesmo número de pacotes e um número semelhante de classes e métodos. Os maiores valores de MLOC continuam sendo dos mesmos elementos da versão r4.9. Na verdade houveram alguns cortes de linhas em métodos talvez devido alguns *bugs* que foram corrigidos nessa versão(visto no *changeLog* do JUnit4). Com relação a métrica CC também não houveram grandes alterações com relação a versão r4.9. Mantendo padrão de tamanho e cor de praticamente todos os elementos.

Após essas análises, um achado interessante é que a maioria dos casos relatados estão de alguma forma relacionados com as mudanças da documentação do JUnit4. Porém, um achado interessante que não foi sinalizado na documentação é que o método *start* da *TestRunner* geralmente cresce quando outros métodos aparecem no mesmo pacote a qual ele pertence.

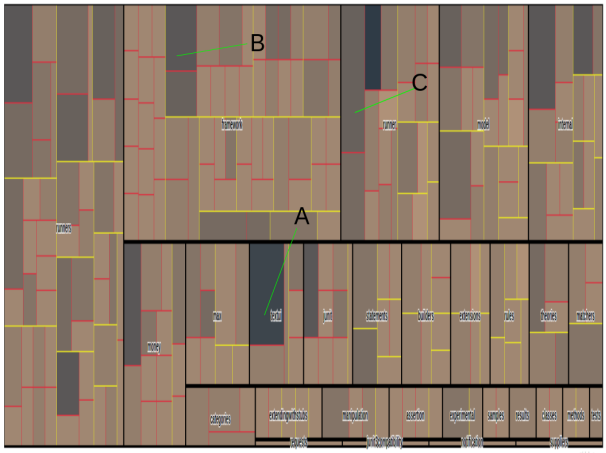

Figura 18: Visão Geral do resultado do single version r4.9. Área : MLOC - Cor: CC

Em seguida foram feitas algumas análises com a estratégia de análise visual diferencial relativa. Aqui é mostrado uma comparação das versões r4.10 para a r4.11 com a área: MLOC e a COR: CC.

Na comparação relativa dessas duas versões houve diversas variações. Houve crescimentos em dois métodos. 1: *assertEquals*/(método) que pertence a classe e ao pacote respectivamente, *Assert*/(classe), *junit*/(pacote) letra (A) da Figura 19 e 2: *assertEquals*/(método) com outra assinatura e que pertence a classe e ao pacote respectivamente, *Assert*/(classe), *junit*/(pacote) letra (B) da Figura 19. Já o método *assertNull*/(método) que pertence a classe e ao pacote respectivamente, *Assert*/(classe), *junit*/(pacote) letra (C) da Figura 19 apresentou decrescimento da métrica CC. Isso provavelmente devido aos *"Improvements to Assert and Assume"* sinalizados no *Changelog*. Na Figura 19 é possível observar esse comportamento. Todos esses comportamentos sinalizados, são facilmente identificados na documentação do Jnit4. Entretanto, um comportamento interessante identificado que não está descrito na documentação do software, é que os métodos de *assert* sofrem bastante alterações com a evolução do software, e isso tem grande influência, pois em diversos lugares métodos de *assert* são utilizados, e partindo da observação que esses métodos sofrem bastante alterações, pode-se concluir que talvez eles sejam um ponto de aparecimento de *bugs*.

Já o método *assertNull*/(método) da classe e pacote respectivos, *Assert*/(classe), *framework*/(pacote) letra (A) da Figura 20 apresentou decrescimento da métrica CC. Também provavelmente devido as mudanças que vieram nos *"Asserts"* nesta versão. Como na Figura 20 pode-se observar.

No pacote *rules* percebe-se também decrescimentos na classe *TemporaryFolder* nos métodos *newFolder*, *newFile* e delete. Isso pode ser devido as melhoras em "Rules"sinalizadas no *Summary of changes* desta versão. Observa-se esse comportamento na Figura 21. Outros métodos apresentaram decrescimento da métrica CC. Estes métodos foram: *getTestClass* da classe e pacote respectivos, *Description*/(classe), *runner*/(pacote). *validateDataPointFields*/(método) da classe

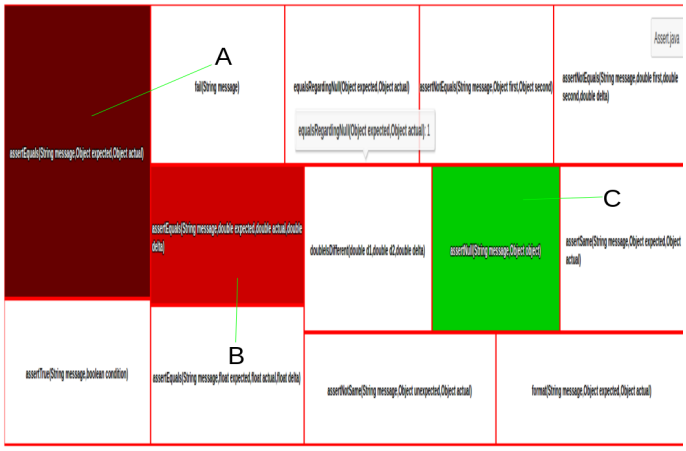

Figura 19: Resultados diferencial relativa.

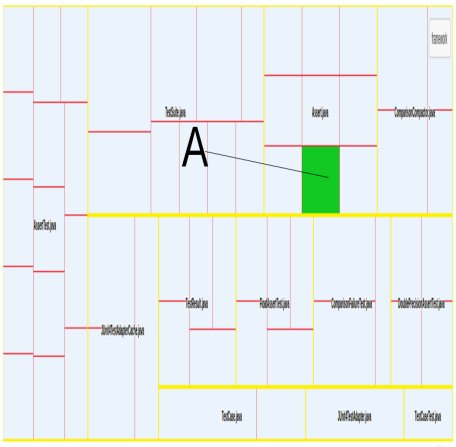

Figura 20: Resultados diferencial relativa.

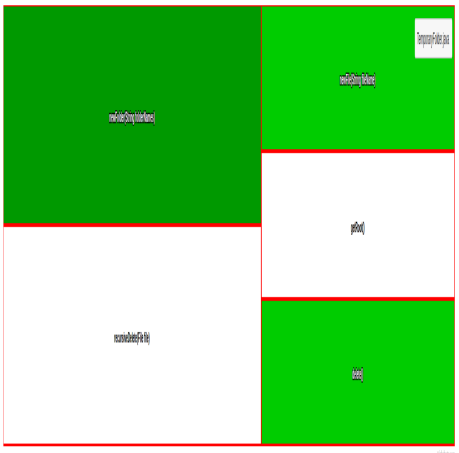

Figura 21: Resultados diferencial relativa.

e pacote respectivos, *Theories*/(classe), *theories*/(pacote). *getRunner*/(método) que pertence a classe e pacote respectivos, *ClassRequest*/(classe), *requests*/(pacote) e *evaluateStatement*/(método) da classe *FailOnTimeout*/(classe) e pacote *statement*/(pacote).

Estes decrescimentos devem ter ocorrido devido as melhoras nos testes e em *"Rules"* nesta versão. E mais um método obteve crescimento da métrica CC nesta versão que foi o método *parallelize*/(método) da classe *ParallelComputer*/(classe) e pacote *experimental*/(pacote). A Figura 22 mostra uma visão geral desta visualização.

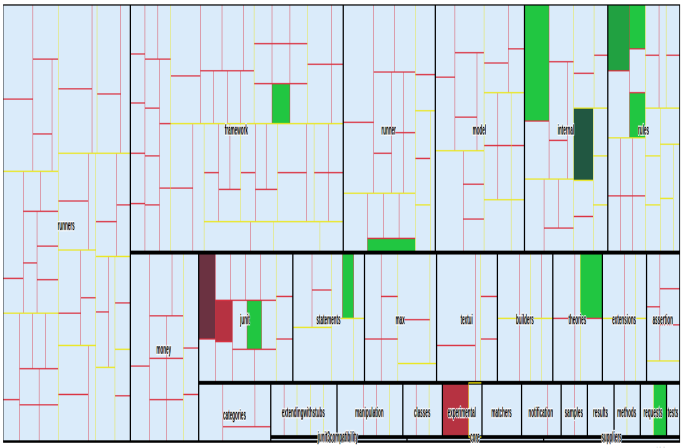

Figura 22: Resultados diferencial relativa.

Após isso foram realizados alguns testes com relação a visualização parallel coordinates. Os elementos escolhidos para serem analisados foram os elementos que apresentaram variações significantes nas visualizações da diferencial relativa. Esses elementos foram: o pacote *junit* representado na Figura 23, o método *getTest* que pode ser visto na Figura 25 e a classe *TestRunner.java* apresentado na Figura 24

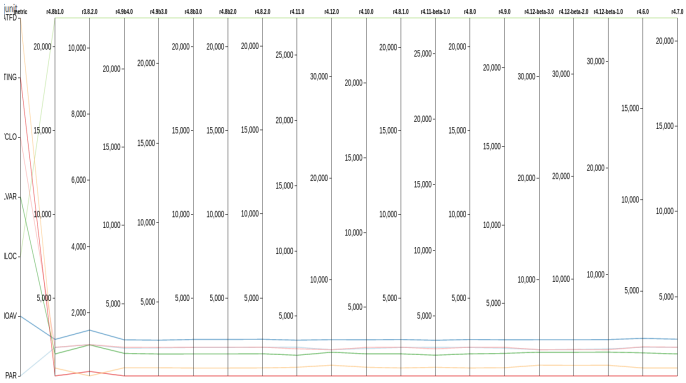

Figura 23: Resultados Parallel Cordinates.

Percebe-se que a métrica MLOC é a que sofre maiores alterações durante as mudanças de versões. As outras se mantém proporcionalmente estáveis. O método *getTest* obteve também grandes variações de MLOC. Isso foi um padrão encontrado em todos os elementos que foram analisados na visualização parallel coordinates.

Na Figura 25 é apresentado mais uma visão do JUnit4 através da visualização parallel coordinates.

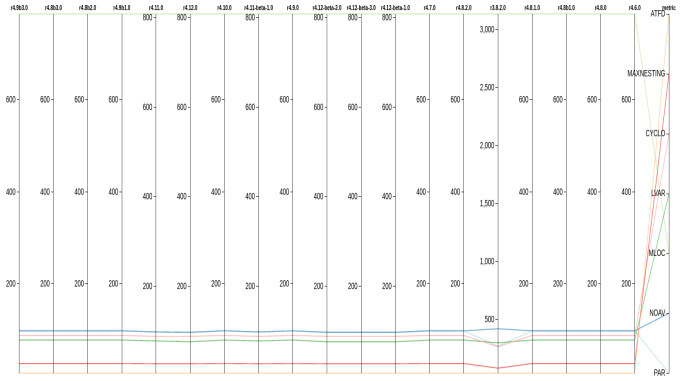

Figura 24: Resultados Parallel Cordinates.

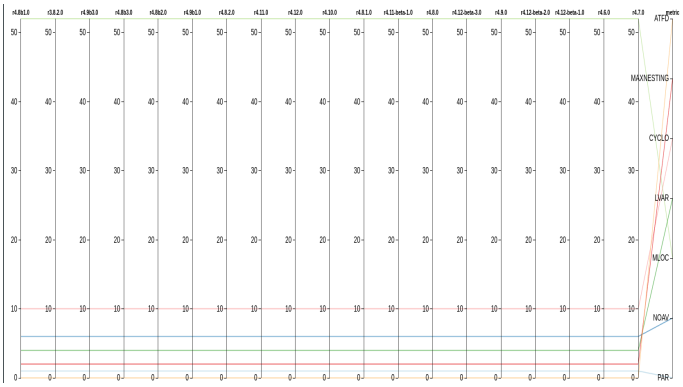

Figura 25: Resultados Parallel Cordinates.

5.4 Discussão

Respondendo as questões levantadas nesta avaliação:

1. O que foi possível identificar acerca do JUnit4 após essa avaliação ? Existem problemas ou comportamentos em algum elemento que possuem grande influência no funcionamento do JUnit4 ?

   É possível perceber que quando as versões são analisadas de maneira isolada como na visualização *single version*, são identificados elementos que possuem valores elevados com relação as métricas MLOC e CC. Porém, somente com a estratégia diferencial relativa é possível identificar quais métodos sofrem a maior quantidade de alterações dos valores de suas métricas. Isso pode levar a identificação dos métodos principais, que geralmente com qualquer alteração podem melhorar ou piorar. Este ponto é algo positivo das visualizações do Visminer DEV, pois somente observando mudanças de código não seria trivial identificar esses métodos citados. Além disso, foi possível perceber que ao longo as versões determinados métodos, como o *assertEqual*, sofreram constantes alterações ao longo das versões. Isso demonstra que ou esse método teve que evoluir por causa de novas funcionalidades que foram surgindo ou

2. Existe alguma relação entre os valores das métricas deste software ?

   É visível que os elementos analisados se comportam de maneira semelhante quanto a proporção das variações dos valores das métricas. Ou seja, analisando os métodos que sofrem mais alterações (identificados na diferencial relativa) com a visualização do parallel coordinates, os valores das métricas podem aumentar ou diminuir, entretanto sempre aumentam de forma proporcional uma com as outras. Então, caso a métrica MLOC tenha um aumento de um certo valor a métrica CC subirá proporcionalmente. Comparando também o *Number of Parameters* (PAR) e *Number of Acessed Variables* (NOAV), também possuem leves variações proporcionais. Entretanto, na maioria das versões elas se mantém estáticas.

3. O Visminer DEV reflete as mudanças que ocorreram no JUnit4, de acordo com sua documentação ?

   Na maioria dos casos que mudanças foram sinalizadas nos *Release Notes* presente na documentação do JUnit4, o Visminer DEV conseguiu representar de alguma maneira essas mudanças. Tanto refatorações que por consequência diminuíram o MLOC de algum método, quanto mudança de assinaturas de métodos que podem diminuir ou aumentar o PAR.

Os mecanismos de interação que o Visminer DEV proporciona ajudaram bastante na identificação dessas informações discutidas. Um deles foi a possibilidade de rapidamente alterar as métricas e versões do repositório analisado. Isso ajudou, pois foi possível comparar de uma forma mais dinâmica como determinada versão se comporta em relação a outras. A navegação entre os níveis como pacotes, classes e métodos ajudaram a visualizar de uma melhor maneira principalmente os métodos de uma mesma classe, sendo possível identificar qual método poderia ser considerado o mais importante de determinada classe. Os diversos atributos visuais ajudaram de diversas formas a visualizar os dados de uma maneira mais organizada. As escalas de cores para representar valores de métricas, tanto na visualização single version quanto na diferencial relativa, ajudaram a identificar rapidamente quais métodos possuíam, por exemplo, a maior complexidade ciclomática. Outro mecanismo de interação proporcionado pelo Visminer DEV, que enriqueceu a experiência da análise do repositório, foi a possibilidade de com o clique do botão direito do mouse em determinado elemento, fosse possível visualizar os valores de todas suas métricas em diversas versões do software. Com isso, foi possível identificar o quanto um método foi afetado por certas mudanças que ocorreram de uma versão para outra.

Por fim, grande parte das mudanças que apareceram no *Release Notes* do JUnit4 refletiram de alguma forma nas visualizações do Visminer DEV. Ou seja, por exemplo se no *Release Notes* aparece que determinado método foi melhorado ou alterado, em vários casos esse método obteve um diminuição ou aumento de sua complexidade ciclomática. Estes casos são de extrema importância, porque isso mostra que o Visminer DEV reflete as alterações que ocorrem entre as versões.

6. CONCLUSÃO

A visualização de software proporciona técnicas que ajudam a engenharia de software evoluir e manter projetos [1]. Por isso, um ambiente de visualização com suporte a estratégias de análise visual, pode contribuir para a visualização

de software durante sua evolução.

O Visminer DEV foi implementado com o intuito de utilizar estratégias de análise visual detalhadas em uma aplicação web, para que pudesse contribuir na análise de softwares que estão evoluindo. O Visminer DEV, também possui o intuito de tornar as informações oriundas das evoluções dos softwares mais fáceis de analisar e entender, utilizando para esse fim as estratégias de análise visual diferencial relativa e single version.

Como resultados, foi produzido um sistema que está integrado com o Repository Miner e o Visminer, sendo possível analisar e utilizar os valores de métricas que eles produzem.

Uma contribuição que o Visminer DEV produziu foi a construção de duas visualizações integradas com o Visminer. A primeira foi o Treemap, que foi utilizado na implementação da estratégia diferencial relativa e na implementação da estratégia do single version. Outra visualização construída foi a parallel coordinates, que foi utilizada para visualizar elementos do software de maneira isolada dos demais.

Os resultados do estudo realizado com o JUnit4 utilizando as ferramentas proporcionadas pelo Visminer DEV foram bastante positivos. Isso porque, foi possível identificar que uma boa quantidade das mudanças que ocorreram durante a evolução do JUnit4 refletiram em certos atributos visuais nas visualizações do Visminer DEV. Outro ponto positivo, foi a identificação de padrões entre as métricas, como o crescimento e decrescimento de MLOC e CC.

Implementar uma nova visualização dentro do contexto do Repository Miner e do Visminer, possui uma curva de aprendizado difícil no início do desenvolvimento. Pois, aprender e entender como todos os mecanismos que os sistemas possuem, não é uma tarefa trivial. Porém, a partir do momento que o desenvolvedor entende como funciona esses sistemas e principalmente como o Repository Miner armazena os dados gerados nas minerações, torna-se mais fácil desenvolver novas visualizações.

Por isso tudo, a construção de uma visualização de software pode trazer diversos benefícios para a engenharia de software. Porém, para desenvolver uma nova visualização é preciso escolher bem os atributos visuais que essa visualização terá. Pois, os atributos visuais adequados ajudam bastante nas análises de software que são feitas através de tipos de visualização. Na construção do Visminer DEV, isso tudo foi estudado através das referências, principalmente a de Novais et al. [22], para que as visualizações presentes no Visminer DEV pudessem ser bem elaboradas.

6.1 Limitações deste Trabalho

6.1.1 Limitações Científicas

Entende-se por uma limitações deste trabalho a não possibilidade de utilizar um maior número de Estratégias de Análise Visual nas visualizações, como por exemplo, a estratégia diferencial absoluta. Essa estratégia serviria para representar propriedades que apareceram ou desapareceram de uma versão para outra. Isso traria uma ganho científico para o Visminer DEV e consequentemente o Visminer.

6.1.2 Limitações Tecnológicas

Uma limitação identificada neste trabalho, é a necessidade de que para gerar a visualização do parallel coordinates a partir do Treemap é necessário clicar exatamente no nome referente ao pacote, classe ou método. Isso talvez torne mais difícil a utilização do Visminer DEV por possíveis usuários em termos de usabilidade.

No mais, é necessário uma uniformização e organização maior do código do JavaScript, tentando seguir padrões e boas práticas de implementações em JavaScript, pois isso, pode tornar o Visminer DEV mais fácil de ser compreendido e por consequência, mais fácil de evoluir e ser mantido. Outro ponto limitante, é a dependência de minerar os dados separadamente com o Repository Miner de forma não totalmente integrada com o Visminer. Isso pode dificultar a utilização da ferramenta, pois existem muitos passos necessários para que o Visminer consiga identificar os dados referentes ao repositório minerado.

Por fim, existem alguns problemas de performance, tornando algumas vezes a produção da visualização um pouco lenta. Isso ocorre devido a navegação dos dados provenientes do Repository Miner, que por estarem aninhados de maneira hierárquica a todo momento é necessário navegar por essa hierarquia e realizar comparações e operações entre os dados de diferentes versões.

6.2 Trabalhos Futuros

Para trabalhos futuros, pode-se produzir outras visualizações que implementem outras estratégias de análise visual, pois atualmente o Visminer DEV não aborda estratégias de análise visual temporal por exemplo. Ou seja, ainda existem diversas estratégias que podem ser implementadas. O layout da aplicação deverá ser melhorado, principalmente o menu de configuração para gerar as visualizações.

7. REFERÊNCIAS

[1] T. Ball and S. Eick. Software visualization in the large. *Computer*, 29(4):33–42, 1996.

[2] M. Balzer, O. Deussen, and C. Lewerent. Voronoi treemaps for the visualization of software metrics. *Proceedings of the 2005 ACM symposium on Software visualization*, 05:165–172, 2005.

[3] W. Beaton and J. d. Rivieres. Eclipse platform technical overview. Technical report, The Eclipse Foundation, 2006.

[4] K. H. Bennett and V. Rajlich. Software maintenance and evolution: a roadmap. *ICSE - Future of SE Track*, pages 73–87, 2000.

[5] G. Carneiro, M. Mendonça, and R. Magnavita. Visminertd - an open source tool to support the monitoring of the technical debt evolution using software visualization.

[6] L. Carvalho, R. Novais, and M. Mendonça. Visminerservice: a rest web service for source mining. *Workshop on Software Visualization, Evolution and Maintenance*, 11, 2015.

[7] A. Dasgupta and R. Kosara. Pargnostics: Screen-space metrics for parallel coordinates. *IEEE Transactions on Visualization Computer Graphics*, 16(6):1017–1026, 2010.

[8] E. Gamma and K. Beck. Junit a cook's tour. *Java Report*, 4(5), 1999.

[9] J. Heinrich, J. Stasko, and D. Weiskopf. *The Parallel Coordinates Matrix*. The Eurographics Association, 2012.

[10] Highcarts. *Large Treemap*. Available at http: //www.highcharts.com/demo/treemap-large-dataset.

[11] J. Johansson and C. Forsell. Evaluation of parallel coordinates: Overview, categorization and guidelines for future research. *IEEE Transactions on Visualization and Computer Graphics*, 22(1)(579-88), 2016.

[12] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. *Proceedings of the 2nd conference on Visualization*, page 284–291, 1991.

[13] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering:a research survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 15:87–109, 2003.

[14] G. Langelier, H. Sahraoui, and P. Poulin. Visualization-based analysis of quality for large-scale software systems. *IEEE/ACM International Conference on Automated software engineering*, pages 214–223, 2005.

[15] M. Lanza and S. Ducasse. Understanding software evolution using a combination of software visualization and software metrics. *Proceedings of Languages et Modèles à Objets*, pages 135–149, 2002.

[16] M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[17] M. Lungu. The small project observatory: Visualizing software ecosystems. *Sci. Comput. Program.*, 75(4):264–275, 2010.

[18] R. Mazza. Introduction to information visualization. *IEEE Computer Society*, Icorporated(1 edition), 2009.

[19] K. McDonnell and K. Mueller. Illustrative parallel coordinates. *European Symposium on Visualization*, 27(3), 2008.

[20] T. Mens, Y. Guéhéneuc, J. Fernández, and M. D'Hondt. Software evolution. *IEEE Computer Society*, 2010.

[21] R. Novais, G. Carneiro, P. Júnior, and M. Mendonca. On the use of software visualization to analyze sofware evolution: An interactive differential approach. *Lecture Notes in Business Information Processing*, 3:15–24, 2011.

[22] R. Novais and M. Mendonça. Combinando estratégias de análise visual de evolução de software em diferentes níveis de detalhe. *Programa Multi-institucional de Pós-Graduação em Ciência da Computação- PMCC*, 2013.

[23] R. Novais, C. Nunes, C. Lima, E. Cirilo, F. Dantas, A. Garcia, and M. Mendonça. On the proactive and interactive visualization for feature evolution comprehension: An industrial investigation. *International Conference on Software Engineering*, pages 1044–1053, 2012.

[24] R. Novais, J. Santos, and M. Mendonça. Experimentally assessing the combination of multiple visualization strategies for software evolution analysis. *The Journal of Systems and Software*, 127:56–71, 2017.

[25] R. L. Novais, G. de Figueiredo Carneiro, P. R. M. S. Júnior, and M. G. Mendonça. On the use of software visualization to analyze software evolution: An interactive differential approach. In *ICEIS*, volume 102 of *Lecture Notes in Business Information Processing*, pages 241–255. Springer, 2011.

[26] Oracle. *JERSEY Framework*. Available at https://jersey.java.net/, version 2.25.1.

[27] E. Tufte. *The Visual Display of Quantitative Information*. 2nd edition. Graphics Press, 2001.