informática

ESHistory: Ferramenta de Extração de Métricas Históricas para Projetos JavaScript

ESHistory: Historical Metrics Extraction Tool for JavaScript Projects

Eduardo Pereira de Sousa
edupsousa@ifsp.edu.br

Eduardo Martins Guerra
eduardo.guerra@inpe.br

Resumo

Métricas de software são ferramentas importantes na análise de aspectos do desenvolvimento de software, apesar disso os trabalhos com foco nessas métricas são ainda limitados a um grupo restrito de linguagens de programação. Neste trabalho é apresentada a ferramenta ESHistory, que tem por objetivo a extração de métricas em diversas versões de projetos JavaScript. Esse artigo apresenta a estrutura da ferramenta e os seus componentes utilizados para percorrer o repositório, extrair as métricas e gerar exibições dos dados. A avaliação da ferramenta é feita com a extração de métricas das versões de diversos projetos Javascript do GitHub, seu processamento e a exibição dos resultados em diferentes tipos de visualização.

Palavras-Chave: *Engenharia de Software; JavaScript; Análise Histórica; Ferramenta*

Introdução

A análise das métricas de software permite verificar características do design do software que podem embasar decisões futuras sobre seu desenvolvimento e manutenção (MACCORMACK et al., 2006). Diversos trabalhos dedicam-se a extração e análise dessas métricas, e para os mais diversos fins. Como exemplo, os trabalhos de Gill e Kemerer (1991) e Meirelles et al. (2010) dedicam-se, respectivamente, a explorar a relação entre essas métricas e a produtividade nas atividades de manutenção do software, e a relação entre essas métricas e a atratividade de softwares de código aberto.

A análise das métricas torna-se ainda mais rica quando seus resultados

históricos são analisados, tornando-se possível não apenas a visualização do estado atual do software, mas também de sua evolução até este ponto e a predição de possíveis tendências em seu desenvolvimento. Apesar de algumas linguagens, como Java, possuírem diversas ferramentas para extração e análise de métricas, outras linguagens como Javascript, que apesar de ser amplamente utilizada, ainda é pouco explorada por estudos que compreendam a análise estática do código-fonte.

Dentro desse contexto, este artigo apresenta a ferramenta ESHistory, cuja principal funcionalidade é a extração e armazenamento de métricas históricas de projetos desenvolvidos na linguagem JavaScript, estabelecendo uma base de informações que permite que sejam criados componentes para o processamento e sua visualização destes dados, como será mostrado. Seu objetivo é auxiliar desenvolvedores e pesquisadores no estudo e entendimento da evolução de softwares desenvolvidos nessa linguagem.

Como notado por Richards et al. (2010), os softwares desenvolvidos na linguagem JavaScript tem comportamento bastante dinâmico, e a linguagem apresenta características pouco encontradas em linguagens mais tradicionais, como orientação a objetos baseada em protótipos, tipos dinâmicos, funções de primeira classe e uso extensivo de métodos assíncronos. Tais características apresentam novos desafios para análise estática e histórica, bem como um vasto campo de estudo.

Neste trabalho realizou-se um experimento para validação da ferramenta, com a extração de métricas de 182 projetos hospedados na plataforma GitHub. Foram selecionados os projetos de maior popularidade por meio da ferramenta de busca do próprio GitHub e da plataforma NPM.js. Foram obtidos projetos com as mais diversas características, contemplando desde pequenas extensões para a biblioteca jQuery, até mesmo frameworks para criação de *front-ends* completos como Angular.JS e React, e para desenvolvimento de *back-ends* como Express.

Análise Histórica de Métricas

Segundo Oman e Hagemeister (1992), os diversos fatores que contribuem para a capacidade de manutenção de um software podem ser organizados em uma série de atributos mensuráveis, ou métricas. As métricas de software são características do software que podem ser quantificadas, indo desde aspectos simples como número de linhas de código ou quantidade de funções, até aspectos cuja mensuração exige a análise estática do código-fon-

te, como a complexidade ciclomática (MCCABE, 1976).

A análise dessas métricas é objeto de variados estudos que visam mensurar desde a qualidade do software desenvolvido (BOEHM et al., 1976) até sua capacidade de atrair usuários e desenvolvedores no caso das comunidades de software aberto (MEIRELLES et al., 2010). Lanza et al. (2005) utiliza essas métricas para a detecção de desarmonias de design.

A análise histórica das métricas de software visa não somente avaliar o estado atual do software, mas permite uma visão mais aprofundada de todo o processo de desenvolvimento que culminou no estado atual. Como verificado por Mens e Demeyer (2001), a análise histórica e evolutiva permite não somente uma análise retrospectiva, mas também uma análise preditiva de futuras alterações no software. Em estudos anteriores, como em Nagappan (2006), foi possível correlacionar alguns conjuntos de métricas de complexidade a maior propensão para falhas em componentes de software.

Ferramenta

Descrição e Licença

A ferramenta ESHistory[1] foi desenvolvida na linguagem JavaScript, sob a licença MIT, com uso dos recursos e bibliotecas da plataforma Node.JS. Seu uso permite a extração de métricas e metadados de projetos armazenados com uso do sistema de controle de versão Git.

Utilização

Seu uso se dá por meio de um *script* de linha de comando chamado *eshistory*, para execução da ferramenta o usuário deve informar o diretório que armazena o repositório do qual as métricas e metadados serão extraídos e o arquivo no qual estes dados serão armazenados. Os comandos abaixo exemplificam o processo de obtenção de uma cópia do repositório do projeto Node.JS por meio do utilitário *git*, e de extração das métricas históricas deste repositório armazenando-as no arquivo *node-metrics.sql*.

Figura 1
Execução da ferramenta ESHistory no repositório do projeto Node.JS

```
git clone https://github.com/joyent/node.git
eshistory node node-metrics.sql
```

1 http://github.com/edupsousa/eshistory

Componentes de Extração de Dados

A ferramenta ESHistory é executada por meio da plataforma Node.JS, que provê recursos para a execução de programas desenvolvidos em JavaScript de forma autônoma, sem a necessidade de um navegador. A ferramenta foi desenvolvida com uso da técnica de desenvolvimento guiado por testes e possui aproximadamente 91% de seu código-fonte coberto por testes.

A estrutura interna da ferramenta é dividida em diversos módulos ilustrados pelo diagrama mostrado na Figura 2. O módulo *ProjectMetricsCommand* é responsável por receber os parâmetros passados pelo usuário, e utiliza os serviços do módulo *MetricsExtractor* para obter as métricas e metadados do repositório, assim como os serviços do módulo *MySQLScriptFile* para a geração do arquivo de saída. O módulo *MetricsExtractor* é responsável por obter os metadados do repositório e o código-fonte dos arquivos por meio do módulo *GitExplorer*, e por extrair as métricas do código-fonte utilizando o módulo *JSMetrics*. Já o módulo *GitExplorer* extrai os metadados do repositório utilizando a dependência externa *nodegit* e o código-fonte por meio do módulo *GitExplorerWorker*, que é executado em múltiplas instâncias paralelas. Já o módulo *JSMetrics* faz uso de múltiplas instâncias do módulo *JSMetricsWorker* para obter as métricas de código-fonte por meio da dependência externa *escomplex*.

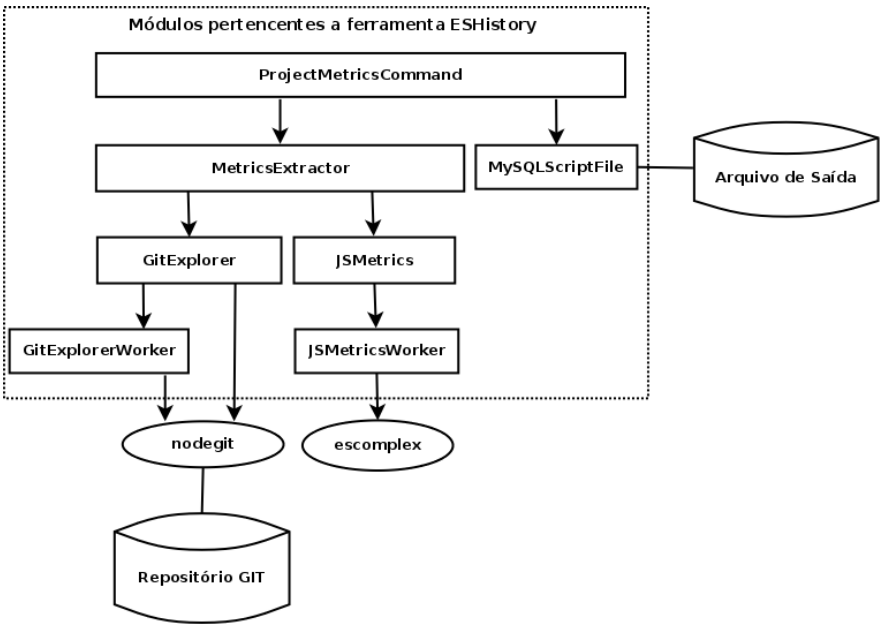


Figura 2 Principais Módulos e Dependências da Ferramenta ESHistory

Além dos módulos descritos, o processo de extração das métricas faz uso das bibliotecas externas NodeGit e ESComplex. NodeGit é uma biblioteca JavaScript utilizada para obtenção de informações de repositórios Git. Já ESComplex é uma biblioteca que tem por objetivo a extração de um amplo conjunto de métricas do código-fonte JavaScript, desde a extração de métricas simples como o número de linhas de código, até métricas dependentes da análise estática do código-fonte, como a complexidade ciclomática e as métricas de Halstead (1977), como volume e vocabulário do código-fonte.

Dados Extraídos

As informações extraídas podem ser divididas em duas categorias: metadados do repositório e métricas do código-fonte. Essas informações são então armazenadas em um arquivo que pode ser importado por um servidor de banco de dados MySQL para posterior análise.

Entre os metadados extraídos estão: (a) nome do projeto; (b) identificação SHA de commits e arquivos; (c) data, autor e mensagem de commits; (d) caminho dos arquivos; e (e) referências aos commits por meio de tags. Dentre as métricas extraídas do código-fonte tem-se: (a) número de linhas lógicas de código para arquivos e funções; (b) número de funções por arquivo; (c) número de dependências por arquivo; (d) complexidade Ciclomática por arquivo e função (MCCABE, 1976); e (e) métricas de complexidade de Halstead (Volume, Vocabulário, Esforço, Tempo) (HALSTEAD, 1977).

Banco de Dados

O arquivo de saída gerado por ESHistory é um *script* no formato SQL, que pode ser utilizado para a importação das métricas e metadados extraídos em um servidor MySQL. O servidor MySQL foi escolhido como formato padrão de saída por seu reconhecido desempenho na execução de consultas sobre grandes volumes de dados, bem como pelo grande número de ferramentas disponíveis para análise e exportação de dados armazenados neste servidor. Cabe ressaltar que o módulo de exportação de dados da ferramenta ESHistory foi implementado de forma que possa ser facilmente modificado ou substituído, permitindo a extensão da ferramenta para exportação em outros formatos.

A Figura 3 exibe o modelo Entidade-Relacionamento do banco de dados. Optou-se a principio pela criação de colunas para cada uma das métricas extraídas do código-fonte dos arquivos JavaScript, presentes na tabela *file_metrics*, bem como para cada uma das funções existentes nestes arquivos, na tabela *function_metrics*. Tal opção tem por objetivo simplificar a consulta a esses dados.

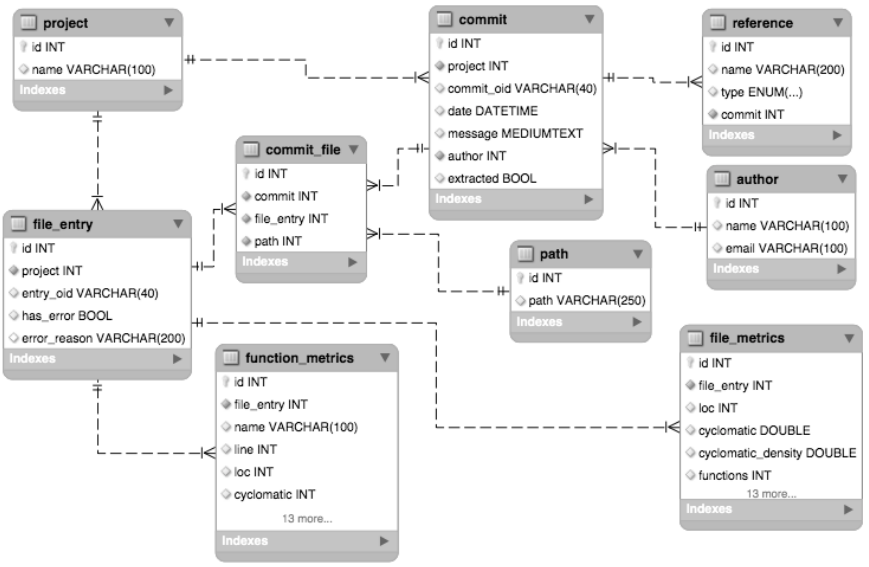


Figura 3
Modelo Entidade-Relacionamento do Banco de Dados

Componentes de Análise e Visualização de Dados

Com base nos dados obtidos pela ferramenta, foram criados alguns componentes externos voltados para análise e visualização destes dados. São consultas na linguagem SQL, *scripts* na linguagem R e componentes de visualização em JavaScript cujo objetivo é analisar a estrutura destes projetos, validando o propósito da ferramenta ESHistory. Uma breve demonstração desses componentes é feita na seção "Mineração e Visualização dos Dados Extraídos".

Aplicação da Ferramenta em Projetos de Código Aberto

Para validação da ferramenta ESHistory foi realizado um experimento consistindo na obtenção e processamento de métricas de softwares de código-aberto na linguagem JavaScript. Optou-se pela obtenção dos 200 repositórios mais populares dos sites GitHub e NPM.JS. O uso dos dois sites teve por finalidade a criação de uma base heterogênea de projetos, uma vez que o site NPM.JS é utilizado para publicação de bibliotecas voltadas para a plataforma Node.JS, enquanto muitos dos repositórios JavaScript mais populares do GitHub são voltados para execução em navegadores Web. Após a obtenção dos dados foi verificada a duplicidade de 18 projetos, e assim a base final ficou reduzida a 182 projetos.

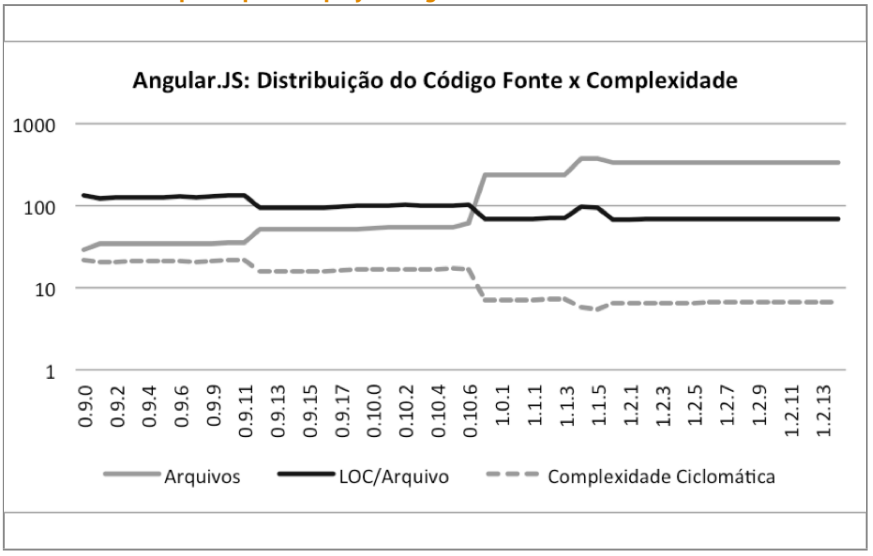
Durante este experimento foram processadas 328.260 diferentes versões (*commits*) destes projetos, uma média aproximada de 1.803 versões

por projeto. O projeto com menor número de versões foi *gulp-autoprefixer* com 26 versões, e o maior número de versões foi encontrado no projeto *brackets* com 16.438 versões. O projeto mais antigo analisado foi a biblioteca *jQuery*, cuja primeira versão do repositório data de 22 de março de 2006.

As versões dos projetos analisados fazem referência a 2.477.155 arquivos, os quais possuem 177.421 versões individuais, destes, 15.538 (8,76%) arquivos apresentaram algum tipo de erro sintático e não puderam ser processados. Estes erros se deram em sua maioria em arquivos que fazem uso de novos recursos da linguagem JavaScript, ainda não suportados pelo *parser* utilizado na ferramenta ESHistory.

O Gráfico 1 exibe a distribuição do código fonte no projeto Angular.JS e a complexidade ciclomática média dos arquivos que compõe cada uma das versões do projeto. Esse gráfico permite supor uma correlação entre o número de arquivos do projeto e a complexidade média destes, evidenciando a refatoração do código-fonte do projeto durante seu ciclo de evolução. Fica clara a possibilidade de uso da ferramenta ESHistory na análise e visualização da correlação temporal entre as diversas métricas de código-fonte, permitindo o uso dessas informações para embasamento de possíveis decisões arquiteturais nos projetos de software.

GRÁFICO 1
Correlação entre a distribuição do código-fonte e a complexidade ciclomática média por arquivo no projeto Angular.JS



A partir desses dados, obtidos diretamente da base de dados gerado pelo ESHistory pode-se verificar a importância deste tipo de ferramenta na análise histórica de projetos de software. O banco de dados do experimento pode ser obtido por meio do endereço *https://goo.gl/pQHhbk* em um arquivo contendo os dados e a estrutura do banco de dados no formato MySQL.

Mineração e Visualização dos Dados Extraídos

A partir dos dados obtidos no experimento de validação da ferramenta foram criados alguns experimentos complementares, voltados para a visualização destes dados. Um desafio particularmente interessante foi demonstrar a forma de organização do código-fonte destes projetos ao longo de seu ciclo de vida. Para isso criou-se por meio do algoritmo Fuzzy C-Means (BEZDEK et al., 1984) 4 categorias distintas para os projetos segundo o número de arquivos e linhas de código, pelas características do algoritmo de agrupamento os seus limites não podem ser expressos com exatidão, mas podem ser descritos como:

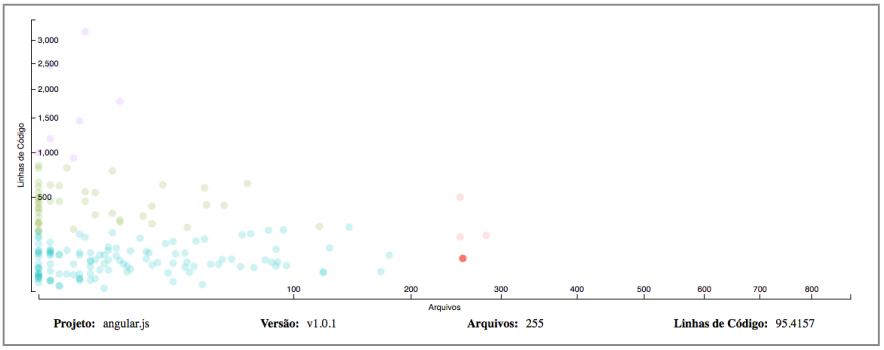
- Roxo: Não mais que uma dezena de arquivos e milhares de linhas de código.
- Verde Escuro: Até uma centena arquivos, com algumas centenas de linhas de código.
- Verde Claro: Até duas centenas de arquivos, com menos linhas de código do que a categoria anterior.
- Vermelho: Acima de duas centenas de arquivos, com número de linhas de código não superior ao da categoria anterior.

A partir dessa divisão foram criadas duas visualizações interativas por meio da biblioteca D3.js, sendo uma delas um *scatter plot* animado, com a representação do número de linhas versus o número de arquivos e suas variações no tempo para cada projeto, apresentado no Gráfico 2. Além disso, foi criada uma segunda visualização, um gráfico do tipo *swin lane*, representando a movimentação dos projetos entre as 4 categorias descritas acima durante seu ciclo de vida, onde cada categoria é representada por uma faixa horizontal e cada um dos projetos por uma linha que se alterna sobre essas faixas durante o ciclo de vida. Demonstrações do *scatter plot* animado[2] e do gráfico *swin lanes*[3] podem ser visualizadas nos endereços indicados no rodapé da página.

2 Scatter plot: https://www.youtube.com/watch?v=hV0btHg5yUI

3 Swin lanes: https://www.youtube.com/watch?v=XocoRiXx2f4

GRÁFICO 2
Arquivos x LOC por projeto. Em destaque o projeto Angular.JS



Trabalhos relacionados

O trabalho de Sokol et al. (2013) apresenta a ferramenta MetricMiner, voltada para a mineração de métricas em repositórios de projetos na linguagem Java, tal trabalho propõe uma ferramenta de mineração e um repositório de métricas que simplificaria o acesso de pesquisadores a esses dados.

Com foco na linguagem JavaScript o trabalho de Ramos e Valente (2014) apresenta uma breve análise de 50 projetos na linguagem JavaScript obtidos a partir do site GitHub com base em sua popularidade, esse trabalho analisa a estrutura dos projetos, seu tamanho, organização e complexidade, permitindo uma visão atual sobre a forma como são desenvolvidos os projetos JavaScript de maior popularidade.

Não foram encontrados pelos autores deste trabalho outros trabalhos dedicados a análise de métricas e da evolução de softwares desenvolvidos na linguagem JavaScript, ressaltando a importância da ferramenta aqui proposta como meio auxiliar para a investigação de projetos desenvolvidos nesta linguagem.

Conclusão

A ferramenta ESHistory apresentada neste trabalho permite a extração de métricas de forma automatizada a partir de projetos JavaScript armazenados em repositórios Git, otimizando este processo e gerando um conjunto de dados em formato adequado para análise. Espera-se que com essa ferramenta seja possível obter uma visão clara do processo de evolução do código-fonte JavaScript em projetos de software, auxiliando na realização estudos focados nestes projetos. Isso foi avaliado através do uso da

ferramenta para extrair dados de diversos projetos Javascript do GitHub, com a geração de uma posterior visualização dinâmica desses dados.

A importância deste estudo é ressaltada pela recente popularização no uso da linguagem JavaScript em projetos de software, tal expansão é demonstrada por meio do reconhecido índice de popularidade das linguagens de programação mantido pela empresa TIOBE[4], no qual a linguagem JavaScript foi aclamada "linguagem de programação do ano", no ano de 2014, e tem se mantido desde então entre as 10 linguagens de programação mais utilizadas. Além disso, como mencionado na seção "Trabalhos relacionados", não foram encontrados outros estudos além daqueles já mencionados neste trabalho que se dediquem a análise de projetos JavaScript, fato que realça o vasto campo de estudos que pode ser auxiliado pela ferramenta proposta.

Possíveis trabalhos futuros, incluem a evolução da ferramenta ESHistory para a adição de um sistema de filtragem, que permita ao usuário selecionar um subconjunto de objetos contidos pelo repositório para exportação. Outra contribuição importante seria a criação de um repositório permanente de métricas, que permitisse ao usuário a análise de métricas previamente extraídas e armazenadas. Além disso, deve-se levar em consideração a ampla gama de trabalhos possíveis com o uso da ferramenta, visto seu objetivo em auxiliar estudos desse tipo. Ela pode ser aplicada para desde a geração de outras visualizações históricas de métricas, até para a execução de algoritmos mais complexos de mineração de dados.

Referências

BEZDEK, J.C.; EHRLICH, R.; FULL, W. FCM: The Fuzzy C-Means Clustering Algorithm. *Computers & Geosciences*, v. 10, n. 2, p. 191–203, 1984.

BOEHM, B.W.; BROWN, J.R.; LIPOW, M. Quantitative Evaluation of Software Quality. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2., 1976, San Francisco, USA. *Proceedings*... San Francisco, USA: IEEE Computer Society, 1976. p. 592–605.

HALSTEAD, M.H. *Elements of Software Science: Operating and Programming Systems Series*. New York: Elsevier Science Inc., 1977.

LANZA, M.; MARINESCU, R.; DUCASSE, S. *Object-Oriented Metrics in Practice*: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Secaucus: Springer Publishing Company Inc., 2005.

MACCORMACK, A.; RUSNAK, J.; BALDWIN, C.Y. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and

4 http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Proprietary Code. *Management Science*, v. 52, n. 7, p. 1015–1030, 2006.

McCabe, T.J. A Complexity Measure: Software Engineering. *ieee Transactions on Software Engineering*, v. 2, n. 4, p. 308–320, 1976.

Meirelles, P.; Santos Junior, C.; Miranda, J.; Kon Terceiro, F.; Chavez, C. A Study of the Relationships Between Source Code Metrics and Attractiveness in Free Software Pro jects. In: 2010 brazilian symposium on software engineering, 24., 2010, Salvador. *Proceedings*... Salvador: ieee Computer Society, 2010. p. 11–20.

Mens, T.; Demeyer, S. Future Trends in Software Evolution Metrics. In: international workshop on principles of software evolution, 4., 2001, Vienna, Austria. *Proceedings*... Vienna, Austria: iwpse, 2001. p. 83–86.

Nagappan, N.; Ball, T.; Zeller, A. Mining Metrics to Predict Component Failures. In: international conference on software engineering, 28., 2006, Shanghai, China. *Anais*... Shanghai, China: icse, 2006. p. 452–461.

Oman, P.; Hagemeister, J. Metrics for Assessing a Software System's Maintainability. In: conference on software maintenance, 1., 1992, Orlando, usa. *Proceedings*... Orlando, usa: icsm, 1992. p. 337–344.

Ramos, M.E.; Valente, M.T. Análise de Métricas Estáticas para Sistemas JavaScript. In: workshop on software visualization, evolution and maintenance, 2., 2014, Maceió. *Proceedings*... Maceió: vem, 2014. p. 30–37.

Richards, G.; Lebresne, S.; Burg, B.; Vitek, J. An Analysis of the Dynamic Behavior of JavaScript Programs. In: conference on programming language design and implementation, 31., 2010, Toronto, Canada. *Proceedings*... Toronto, Canada: pldi, 2010. p. 1–12.

Sokol, F.Z.; Aniche, M.F.; Gerosa, M. Metricminer: Supporting Researchers in Mining Software Repositories. In: international working conference on source code analysis and manipulation, 13., 2013, Eindhoven, The Netherlands. *Proceedings*... Eindhoven, The Netherlands: scam, 2013. p. 142–146.

Abstract

Software metrics are important tools in the analysis of software development aspects, yet the work focusing on these metrics still limited to a small group of programming languages. This paper presents the ESHistory tool, which aims at extraction of historical metrics from JavaScript projects. We discuss the structure of the tool and its main components, which are used to walk through the repository, extract metrics and present the data. The tool is evaluated by extracting metrics of various JavaScript projects obtained from GitHub platform, the processing of obtained metrics and its display.

Keywords: *Software Engineering; JavaScript; Historical Analysis; Tool*