JSCity – Visualização de Sistemas JavaScript em 3D

Marcos Viana, Estevão Moraes, Guilherme Barbosa,
André Hora, Marco Tulio Valente
{longuinho,estevaoma,barbosa,hora,mtov}@dcc.ufmg.br

[1]Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – Brasil

Abstract. *JavaScript is one of the most used languages on the web. A wide range of frameworks and libraries widely adopted on the market make use of Javascript. In order to support the development and maintenance of such systems, source code visual representations can be used for restructuring, refactoring and understanding JavaScript software. Although there are tools that generate visual representations of code in other languages such as CodeCity for Java, no similar tool is available for JavaScript applications. This paper presents JSCity, a tool for the interactive visualization of JavaScript systems in 3D, using a city metaphor. For this work, we analyzed 40 popular open source systems written in JavaScript hosted in GitHub.*

Resumo. *JavaScript é uma das linguagens mais utilizadas da web. Muitos frameworks e bibliotecas adotados no mercado fazem uso de JavaScript. Para dar suporte ao desenvolvimento e manutenção desses sistemas, representações visuais podem ser utilizadas na reestruturação, refatoração e entendimento de código. Embora existam ferramentas que gerem representações visuais de código em outras linguagens, como proposta pelo sistema CodeCity, nenhuma ferramenta realiza essas representações para sistemas em JavaScript. Esse artigo apresenta JSCity, uma ferramenta para a visualização de sistemas JavaScript em 3D usando a metáfora de uma cidade. Para esse trabalho, foram analisados 40 sistemas populares escritos em JavaScript, que estão hospedados no GitHub.*

1. Introdução

JavaScript é uma linguagem de programação cada vez mais popular, destacando-se cada vez mais no mercado. De acordo com o site `http://githut.info`, JavaScript é a linguagem mais popular do GitHub. Sua principal característica é a dinamicidade, pois executa comandos em um navegador sem recarregamento da página ou interpretação de servidor [ECMA International 2011]. A linguagem foi inicialmente concebida em meados da década de 1990 com o objetivo de estender páginas web com pequenos trechos de código executáveis. Desde então, sua popularidade e relevância só tem crescido [Kienle 2010] [Nederlof et al. 2014]. A linguagem atualmente é utilizada inclusive para implementar clientes de e-mail, aplicações de escritório, IDEs, dentre outros, que podem atingir milhares de linhas de código [Richards et al. 2010]. Junto com a sua crescente popularidade, o tamanho e a complexidade de sistemas JavaScript também está em constante ascensão.

Por outro lado, a compreensão de sistemas, mesmo que com baixa complexidade, é uma tarefa árdua e cara. Estima-se que a manutenção do software represente 90% dos

custos totais de um sistema [Aparecido et al. 2011] e que grande parte do tempo seja dedicado ao entendimento do software [Guha et al. 2010]. Nesse contexto, a visualização de software é uma técnica utilizada para ajudar os desenvolvedores. Com o auxílio de tecnologias de visualização gráfica, ferramentas de apoio podem ser desenvolvidas para representar diversos aspectos de um sistema, principalmente sua estrutura, comportamento e evolução [Stasko et al. 1997].

Nesse artigo, apresenta-se JSCity, uma adaptação da metáfora de cidades para a linguagem JavaScript. Através dessa metáfora, é possível representar funções e aninhamento de funções, além de identificar arquivos que agrupam essas funções. JSCity é uma ferramenta de código aberto que oferece aos desenvolvedores uma forma intuitiva de representar, modelar e visualizar grandes quantidades de dados de desenvolvimento por meio de uma cidade 3D, como originalmente proposto por Wettel e Lanza [Wettel et al. 2011] para sistemas Java. Essa visualização facilita o entendimento da organização do código de uma aplicação e oferece uma analogia visual para que equipes de desenvolvimento possam se comunicar de forma mais eficiente. A ferramenta foi utilizada para criar as representações de 40 sistemas populares hospedados no GitHub.

Desse modo, as principais contribuições desse trabalho são:

1. Adaptação da metáfora de cidade para a linguagem JavaScript, considerando suas principais estruturas e os usos mais comuns da linguagem;
2. Análise de JavaScript, visualização de funções, arquivos e diretórios;
3. Disponibilização de uma solução computacional para visualização das cidades em 3D facilmente acessível através de uma página web, facilitando compartilhamento entre desenvolvedores.

O artigo está organizado como descrito a seguir. Na Seção 2, descreve-se a metáfora da cidade para representação de código. A Seção 3 apresenta a ferramenta JSCity. Em seguida, na Seção 4, são descritos casos de uso da ferramenta. Finalmente, apresentam-se os trabalhos relacionados na Seção 5 e as conclusões na Seção 6.

2. A Metáfora da Cidade em JavaScript

JavaScript é uma linguagem de programação dinâmica, sendo classificada como uma linguagem de script, baseada em protótipos, com tipagem dinâmica e funções de primeira classe [ECMA International 2011]. Assim, JavaScript é multi-paradigma, possibilitando programação orientada por objetos, imperativa e funcional. Apesar da possibilidade de representar classes em JavaScript por meio de protótipos [Silva et al. 2015], a metáfora proposta não representa classes diretamente e sim funções, que são as principais estruturas utilizadas em sistemas JavaScript.

O uso da metáfora de cidade para representação do código foi inspirado na ferramenta CodeCity [Wettel et al. 2011]. Uma cidade é uma forma intuitiva de representação, uma vez que está inserida em nossa vida cotidiana. Assim, propõe-se uma metáfora de cidade adaptada para JavaScript conforme apresentado na Figura 1. Nessa representação, distritos são diretórios, subdistritos são arquivos e prédios são funções.

Distritos e subdistritos são representados pelas cores amarelo e vermelho, respectivamente. Os prédios representam as funções, que podem ser funções anônimas (cor verde) ou funções nomeadas (cor azul). A altura dos prédios é representada pelo *número*

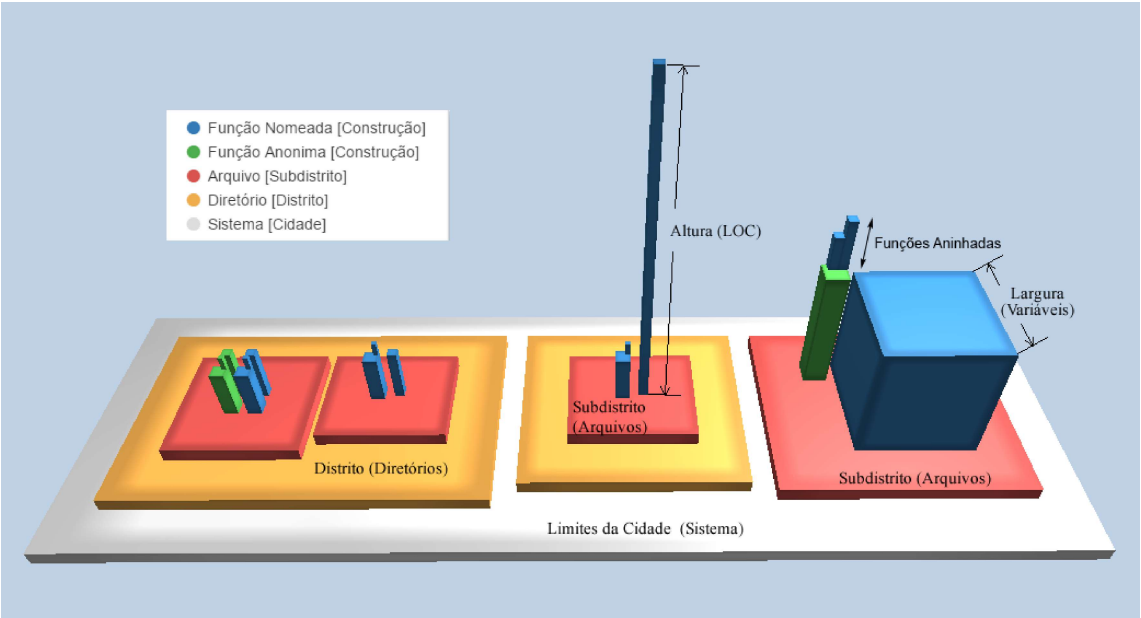


Figura 1: Princípios da metáfora da cidade

de linhas de código da função (LOC) e a largura é representada pelo *número de variáveis da função*.

Uma prática muito comum em JavaScript é declarar uma função dentro de outra, recurso esse denominado de funções aninhadas. Para esses casos, foi adaptado a metáfora da cidade para exibir um prédio sobre o outro. A altura total do prédio é o somatório das linhas de todas as funções aninhadas. A largura da função pai é o somatório das suas variáveis com as variáveis das suas funções filhas, garantindo a construção de um prédio mais largo na base e pequenos prédios acima deste.

A utilização de funções aninhadas é muito comum em JavaScript. Essa propriedade favorece a criação de estilos de programação, como por exemplo, criar uma estrutura que represente uma classe, declarar uma função e dentro dessas outras *n* funções e chamá-las da mesma forma que se chama um método em orientação a objetos. Dessa forma, entende-se que representar esse comportamento não apenas gera uma visualização mais intuitiva, como também possibilita diferenciar alguns padrões no código e possivelmente a distribuição arquitetural do projeto.

3. JSCity

A metáfora proposta foi implementada na ferramenta JSCity[1] para analisar sistemas desenvolvidos em JavaScript através de visualizações interativas de código em 3D. A construção da ferramenta foi realizada utilizando a própria linguagem JavaScript, o framework *Esprima* [Hidayat 2012], cujo papel é gerar uma Árvore Sintática Abstrata (AST) e o framework *ThreeJS* [2], cujo papel é desenhar toda a cidade. Como ilustrado na Figura 2, a ferramenta funciona em cinco passos:

[1] `https://github.com/ASERG-UFMG/JSCity/wiki/JSCITY`
[2] http://threejs.org

1. Execução de um script Node.js para analisar o código com o *Esprima* e gerar uma Árvore de Sintaxe Abstrata (AST) no formato JSON;
2. Interpretação da AST e persistência dos dados relativos às principais estruturas de JavaScript para representação da cidade. No final dessa etapa, os dados necessários para o desenho da cidade já estão gravadas no banco de dados.
3. Usuário escolhe repositório para visualização da cidade em uma página web;
4. Leitura da base de dados para desenho da cidade;
5. Desenho da cidade utilizando o framework *ThreeJS*.

O framework *ThreeJS* oferece recursos gráficos que possibilitam a representação de cenas em 3D. Dentre os recursos oferecidos, podemos enumerar a criação de cenas, geometrias, animações e perspectivas sem o recarregamento da página. Esses recursos foram utilizados pela ferramenta para desenhar a cidade e possibilitar a navegação pelos elementos da cidade, realizar operações de *zoom*, mudar o ângulo da câmera e posicionar o *cursor* no prédio para exibir os dados da função.

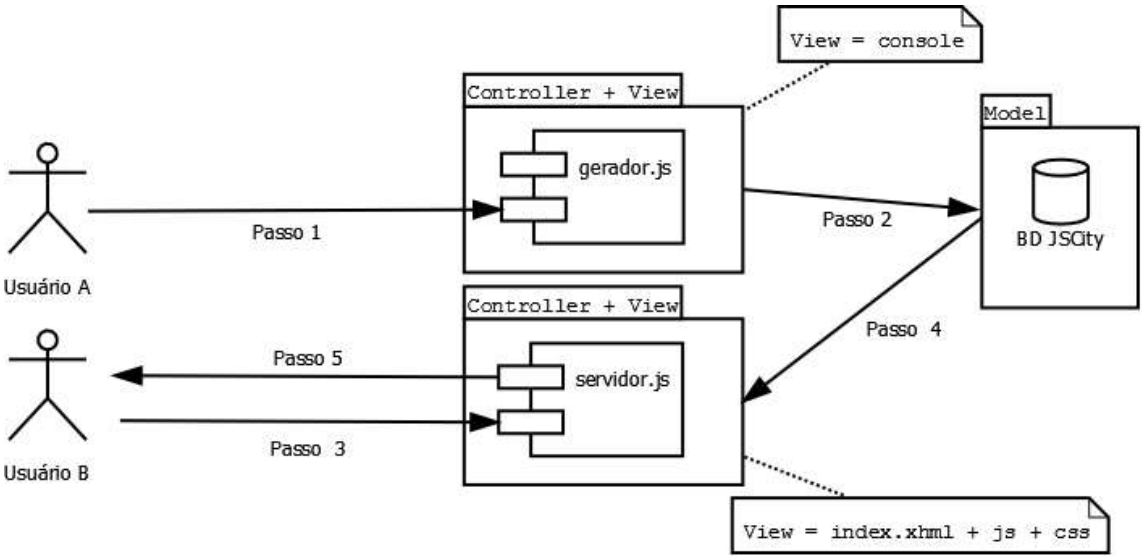
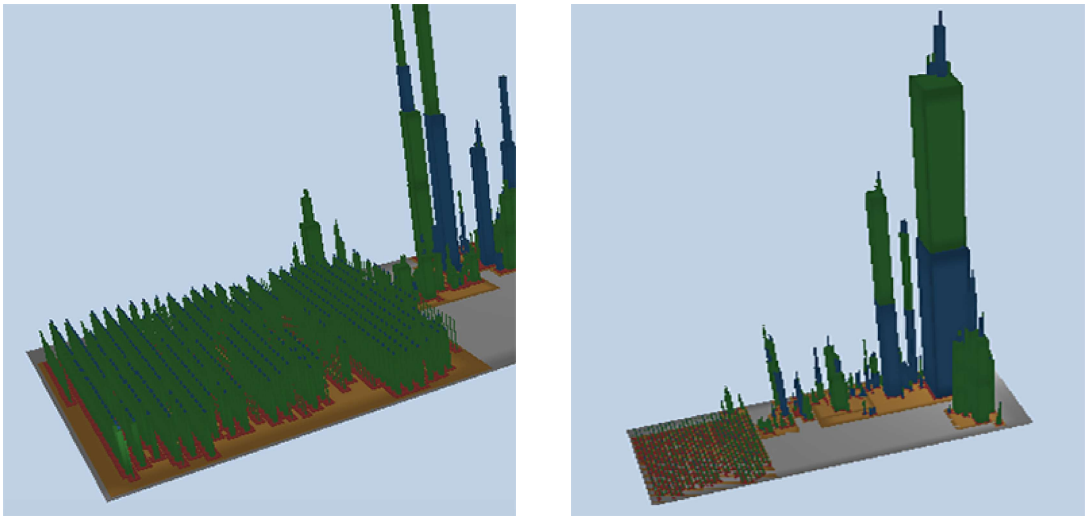


Figura 2: Diagrama de Componentes da Ferramenta JSCity

A ferramenta desconsidera na análise todos os arquivos de teste, exemplos de uso, arquivos minificados, arquivos de *copyright*, documentações e qualquer arquivo que não corresponda ao *core* do sistema em análise. Por ser uma primeira versão, o sistema possui a limitação de realizar análise de código somente em Javascript, não permitindo essa interpretação em outras linguagens de programação. Além disso, a definição das variáveis das metáforas não ocorre em tempo real, o que faz com que seja necessária uma análise prévia do que será ou não relevante para a geração das visualizações.

4. Exemplos de Uso

Seleção dos Repositórios. Para avaliar a aplicação da metáfora e solução propostas para JavaScript, foram selecionados sistemas populares escritos em JavaScript hospedados no GitHub. O critério para seleção de sistemas foi possuir uma quantidade mínima de 250 estrelas. A pesquisa ocorreu em maio de 2015 e dentre os resultados optou-se por 40 sistemas conhecidos, amplamente utilizados na web e que abrangem diferentes aplicações, tais



(a) Funções de Internacionalização (b) Funções do Núcleo

Figura 3: Cidade do sistema AngularJS

como *frameworks*, editores de código, plugins de navegadores, jogos, dentre outros. Para ilustrar o uso da metáfora de forma mais detalhada, foram selecionados três repositórios: *AngularJS, jQuery e Bower*. Os dois primeiros são *frameworks* para desenvolvimento web e o terceiro é um gerenciador de pacotes. A lista completa de repositórios analisados e suas cidades em 3D pode ser encontrada em:
`https://github.com/ASERG-UFMG/JSCity/wiki/JSCITY`.

4.1. Exemplo 1: AngularJS

A Figura 3 apresenta a cidade do sistema JavaScript mais popular no GitHub, o AngularJS. Com 39,032 estrelas, esse conhecido framework para desenvolvimento de aplicações web possui 233,785 linhas de código dividas em 20 diretórios, 863 arquivos, 10,362 funções anônimas e 6,050 funções nomeadas. Através da visualização pode-se observar duas áreas distintas: um distrito com prédios pequenos à esquerda na Figura 3a (sugerindo que contém arquivos estruturalmente similares) em contraste com arranha-céus do lado direito na Figura 3b. De fato, no diretório mostrado à esquerda estão os arquivos para internacionalização, enquanto que nos diretórios à direita estão os arquivos que fazem parte do núcleo desse sistema. Assim, a visualização proposta torna mais simples o entendimento da modularização de um sistema altamente complexo.

4.2. Exemplo 2: jQuery

A Figura 4 apresenta a cidade de um dos *frameworks* mais populares no desenvolvimento web, o jQuery. O código possui na sua maior parte funções anônimas (prédios verdes), e os grandes prédios representam funções do núcleo do *framework*, como as que tratam eventos. Esse comportamento pode ser justificado pelo uso comum de *callbacks* que são funções "passadas como argumento de outra função" e/ou chamadas quando um evento é acionado. Pode-se observar também (através das interações da visualização) que os diretórios estão separados por módulos, como por exemplo, os módulos *core*, *event*, *data*, dentre outros.

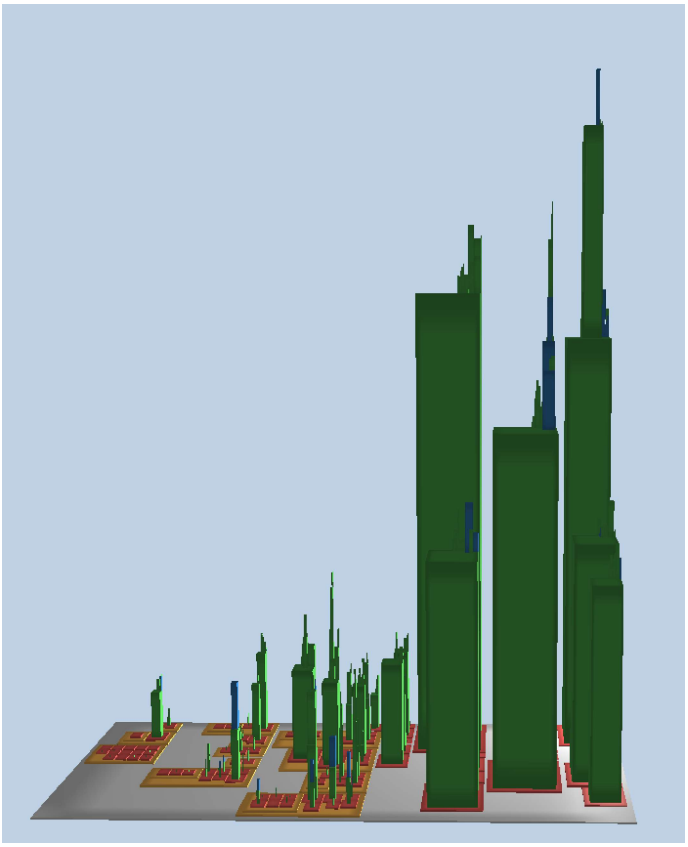


Figura 4: Cidade do sistema jQuery

4.3. Exemplo 3: Bower

A Figura 5 apresenta a cidade de um sistema JavaScript para gerenciamento de pacotes para desenvolvimento web, o Bower. Pode-se observar que não existe um diretório para internacionalização, já que esse sistema não oferece mensagens para outras línguas. Nota-se também que grande parte das funções são anônimas (prédios verdes), mas funções nomeadas também são relativamente comuns (prédios azuis). Além disso, observa-se o frequente uso de funções aninhadas (prédios sobre prédios).

Em suma, a partir da análise detalhada dos repositórios, foi possível encontrar alguns padrões de desenvolvimento, conforme sumarizado a seguir:

1. **Prédios altos e largos**: representam funções do núcleo do sistema;
2. **Distritos grandes com muitos prédios pequenos**: arquivos estruturalmente similares, por exemplo, para implementar internacionalização;
3. **Prédios verdes**: funções anônimas é um recurso da linguagem JavaScript amplamente utilizado;
4. **Prédios sobre outros prédios**: o uso de funções aninhadas é comum nos sistemas analisados, principalmente no núcleo dos sistemas.

5. Trabalhos Relacionados

Nesse trabalho, apresentou-se JSCity, inspirada no CodeCity [Wettel and Lanza 2008, Wettel et al. 2011]. CodeCity tem por objetivo a análise de software, em que sistemas

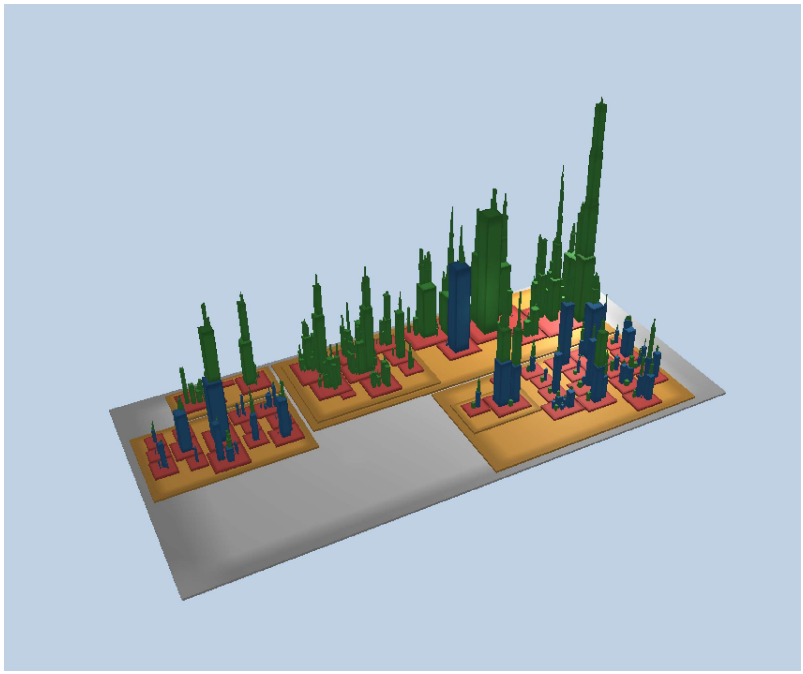


Figura 5: Cidade do sistema Bower

são visualizados como cidades 3D navegáveis e interativas. Classes são representadas como edifícios da cidade, enquanto módulos são retratados como distritos. O número de métodos representa a altura dos prédios, a quantidade de atributos representa a largura e o número de linhas de código é representado por cores - de cinza escuro (menor quantidade) a azul intenso (maior quantidade). CodeCity está disponível para a plataforma de desenvolvimento Eclipse[3] e para a ferramenta Moose[4]. No entanto, nesses casos, estão restritos a análise de sistemas orientados a objetos, nas linguagens, Java e Smalltalk. Por fim, visualizações— não necessariamente na forma de cidade—já foram propostas para outras dimensões de um sistema, como para análise de *bugs* [Hora et al. 2012].

6. Conclusões

JSCity estende CodeCity para JavaScript e oferece para a comunidade uma forma alternativa de analisar sistemas de software desenvolvidos nessa linguagem. A metáfora da cidade foi adaptada para representar sistemas JavaScript, por exemplo, através da visualização de funções anônimas e funções aninhadas. JSCity é facilmente acessível para os desenvolvedores pois roda diretamente em uma página web. As visualizações podem ser utilizadas, por exemplo, em revisões de código e para recomendação de refatorações [Sales et al. 2013, Silva et al. 2014]. Como trabalho futuro, sugere-se que seja verificada a eficácia do uso das metáforas no aumento da produtividade dos desenvolvedores. Além disso, sugere-se que sejam implementadas melhorias na ferramenta que permitam a alteração das métricas das metáforas em tempo real com o objetivo de possibilitar a geração de outros tipos de visualização.

[3]https://marketplace.eclipse.org/content/codecity
[4]http://www.inf.usi.ch/phd/wettel/codecity.html

Agradecimentos

Essa pesquisa foi apoiada pelo CNPq e FAPEMIG.

Referências

Aparecido, G., Nassau, M., Mossri, H., Marques-Neto, H., and Valente, M. T. (2011). On the benefits of planning and grouping software maintenance requests. In *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 55–64.

ECMA International (2011). *Standard ECMA-262 - ECMAScript Language Specification*.

Guha, A., Saftoiu, C., and Krishnamurthi, S. (2010). The essence of JavaScript. In *European Conference on Object-oriented Programming*.

Hidayat, A. (2012). Esprima: Ecmascript parsing infrastructure for multipurpose analysis. http://esprima.org.

Hora, A., Couto, C., Anquetil, N., Ducasse, S., Bhatti, M., Valente, M. T., and Martins, J. (2012). BugMaps: A tool for the visual exploration and analysis of bugs. In *16th European Conference on Software Maintenance and Reengineering (CSMR), Tool Demonstration Track*, pages 523–526.

Kienle, H. M. (2010). It's about time to take JavaScript (more) seriously. *IEEE Software*, 27(3).

Nederlof, A., Mesbah, A., and Deursen, A. v. (2014). Software engineering for the web: The state of the practice. In *Companion Proceedings of the 36th International Conference on Software Engineering*.

Richards, G., Lebresne, S., Burg, B., and Vitek, J. (2010). An analysis of the dynamic behavior of JavaScript programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*.

Sales, V., Terra, R., Miranda, L. F., and Valente, M. T. (2013). Recommending move method refactorings using dependency sets. In *20th Working Conference on Reverse Engineering (WCRE)*, pages 232–241.

Silva, D., Terra, R., and Valente, M. T. (2014). Recommending automated extract method refactorings. In *22nd IEEE International Conference on Program Comprehension (ICPC)*, pages 146–156.

Silva, L., Ramos, M., Valente, M. T., Bergel, A., and Anquetil, N. (2015). Does JavaScript software embrace classes? In *International Conference on Software Analysis, Evolution, and Reengineering*.

Stasko, J. T., Brown, M. H., and Price, B. A., editors (1997). *Software Visualization*. MIT Press.

Wettel, R. and Lanza, M. (2008). CodeCity: 3D Visualization of Large-scale Software. In *International Conference on Software Engineering*.

Wettel, R., Lanza, M., and Robbes, R. (2011). Software systems as cities: A controlled experiment. In *International Conference on Software Engineering*.