

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA TRIÂNGULO MINEIRO PRÓ-REITORIA DE PESQUISA E INOVAÇÃO PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA – PIBIC

RELATÓRIO FINAL

PARTE I

Título do plano de trabalho do bolsista	Uma Ferramenta para Conversão de Código JavaScript Orientado a Objetos em ECMA 5 para ECMA 6
Título do projeto do orientador	Uma Ferramenta para Conversão de Código JavaScript Orientado a Objetos em ECMA 5 para ECMA 6
Nome do bolsista	Matheus Russignoli do Nascimento
Nome do orientador	César Francisco de Moura Couto
Grupo de pesquisa	Linguagens e Ambientes de Programação - UFMG
Período de vigência	01/03/2017 a 28/02/2018

RESUMO (Máximo 220 palavras)

ECMAScript, em sua nova versão, ECMAScript 6, introduz uma nova sintaxe para programação orientada a objetos. A proposta é manter o funcionamento baseado em protótipos, mas com uma forma de implementação mais conveniente e sintaticamente próxima dos conceitos tradicionais de orientação a objetos. Para que os desenvolvedores possam usufruir dos benefícios desta nova versão, este trabalho de pesquisa apresenta uma ferramenta, intitulada *ecms5to6* para migração de classes de código em ECMAScript 5 para nova sintaxe proposta em ECMAScript 6. A ferramenta proposta é responsável por converter estruturas de código orientado a objetos usando *prototypes* implementados em ECMAScript 5 para estruturas de código orientada a objetos em ECMAScript 6, tais como class, constructor e extends. Adicionalmente, este trabalho de pesquisa reporta um estudo de caso com 10 softwares reais e populares implementados em JavaScript. Este estudo de caso foi usado para avaliar a eficácia e eficiência da ferramenta *ecms5to6* em realizar a conversão de estruturas de código para ECMAScript 6. Foi concluído que em todos 10 softwares analisados, a ferramenta proposta foi capaz de realizar a conversão de código para ECMAScript 6.

1. INTRODUÇÃO

JavaScript é uma linguagem de programação inicialmente projetada em meados da década de 90 para estender páginas web com código executável. Desde então, sua popularidade e relevância tem crescido consideravelmente [1, 2]. Por exemplo, JavaScript é a linguagem mais popular no GitHub (um repositório de software público), considerando novos repositórios criados por linguagem. É também reportado que a linguagem é usada por 97 de todos os 100 mais populares sites da web [3]. Concomitantemente com o seu aumento de popularidade, o tamanho e a complexidade de softwares JavaScript está em constante crescimento. A linguagem é usada para implementar diversos tipos de aplicações, tais como clientes de email, aplicações de escritórios, IDEs, etc, as quais podem alcançar centenas de milhares de linhas de código¹.

Desde sua criação, programas em JavaScript tem sido implementados seguindo diversos paradigmas, como por exemplo os paradigmas de programação imperativa e funcional. Alem destes, o paradigma da programação orientada a objetos é bastante utilizado [4], usando princípios de prototipagem [5]. Este conceito de prototipagem define uma abordagem diferente da orientação a objetos tradicionalmente utilizada em linguagens como Java ou C#. Adicionalmente, existe um problema sintático, dado que não existem em JavaScript palavras reservadas comumente utilizadas, como por exemplo, class. Essas diferenças podem acabar causando dificuldade na implementação do código orientado a objetos.

.

http://sohommajumder.wordpress.com/2013/06/05/gmail-has-biggest-collection-of-javascript-code-lines-in-the-world, verificado em 11/02/2015

Essa divergência sintática foi alvo das atualizações da ECMAScript, que em sua nova versão, ES 6 (ECMAScript 6) [6] introduz uma nova sintaxe para classes. A proposta é manter o funcionamento baseado em protótipos [7], mas com uma forma de implementação mais conveniente e sintaticamente próxima dos conceitos tradicionais de orientação a objetos. Para que os desenvolvedores possam usufruir dos benefícios desta nova versão, este trabalho de pesquisa apresenta uma ferramenta, intitulada *ecms5to6* para migração de classes de código JavaScript ES 5 (ECMAScript 5) [8] para nova sintaxe proposta de ES 6. Com isso, espera-se ajudar os milhares de desenvolvedores que possuem código que emula classes de acordo com a sintaxe antiga a se beneficiarem de forma automática da sintaxe nativa de classes proposta pela ES 6.

A ferramenta *ecms5to6* é *open-source* e foi implementada utilizando a linguagem de programação Java. Sua principal função é permitir desenvolvedores de software Javascript converter código JavaScript orientado a objeto em ES 5 para ES 6. Mais especificamente, a ferramenta proposta é responsável por converter estruturas de código orientado a objetos usando *prototypes* implementados em JavaScript ES 5 para estruturas de código orientada a objetos em JavaScript ES 6, tais como class, constructor e extends.

Adicionalmente, este trabalho de pesquisa reporta um estudo de caso com 10 softwares reais e populares implementados em JavaScript. Este estudo de caso foi usado para avaliar a eficácia e eficiência da ferramenta *ecms5to6* em realizar a conversão de estruturas de código JavaScript ES 5 para estruturas de código JavaScript ES 6. Foi concluído que em todos 10 softwares analisados, a ferramenta proposta foi capaz de realizar a conversão de código ES 5 para ES 6.

2. MATERIAIS E MÉTODOS

Conforme ilustrado na Figura 1, a ferramenta *ecms5to6* é composta de dois módulos: detecção de classes já implementado em~[9] e conversão para ES 6. O módulo de detecção de classes é responsável por ler arquivos JavaScript (JS), gerar arquivos JSON contendo a árvore de sintaxe abstrata e detectar classes, métodos, atributos e relacionamentos de herança entre classes a partir da árvore de sintaxe abstrata. O módulo de conversão para ES 6 é responsável por converter estruturas de código orientado a objetos usando *prototypes* implementados em JavaScript ES 5 para estruturas de código orientada a objetos específicas para JavaScript ES 6, tais como class, constructor e extends.

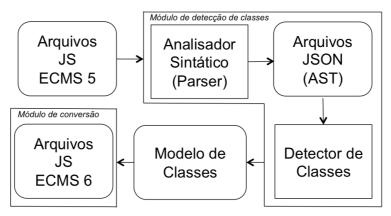


Figura 1 - Arquitetura da ferramenta ecms 5to 6

Módulo de detecção de classes: este módulo recebe como entrada arquivos JavaScript (JS) representando o código fonte da aplicação de origem a ser analisada. A partir dos arquivos JS, este módulo gera a árvore de sintaxe abstrata (AST) no formato JSON utilizando a biblioteca Esprima--um parser JavaScript. Em seguida, este módulo persiste as ASTs geradas em arquivos JSON. A partir dos arquivos JSON, este módulo aplica a estratégia descrita em [9] para detectar classes e gera objetos Java que representam classes contendo atributos, métodos e relacionamentos de herança.

Módulo de conversão em ES 6: este módulo receberá como entrada uma lista de objetos representando classes provenientes do módulo de detecção de classes. Esta lista de objetos compõe o modelo de classes. Em seguida, este módulo implementará um algoritmo de conversão de código que mapea o modelo de classes para estruturas de código específicas de programação orientada a objetos em JavaScript ES 6.

3. RESULTADOS

Um estudo de caso com sistemas de software reais implementados em JavaScript foi realizado para avaliar se a ferramenta *ecms5to6* é efetivamente capaz de converter estruturas de código JavaScript ES 5 para estruturas de código JavaScript ES 6. O contexto deste estudo consiste de 10 softwares JavaScript populares e reais disponíveis no GitHub.

Basicamente, este estudo objetivou responder a seguinte questão de pesquisa (QP):

QP: A ferramenta *ecms5to6* é capaz de converter estruturas de código JavaScript ES 5 para estruturas de código JavaScript ES 6?

A Figura 2 apresenta as diferenças na codificação do arquivo /algorithms.js/data_structures/bst.js pertencente ao software *algorithms.js* antes e após a execução da ferramenta *ecms5to6*. Como pode ser observado, a função Node foi classificada como uma classe pela ferramenta. Consequentemente, a ferramenta converteu a função Node para a nova sintaxe de classe especificada em JavaScript ES 6 ao adicionar as palavras reservadas class e constructor nas linhas 46 e 47 do arquivo bst_6.js.

```
    JavaScript Source Compare ▼

ecms5to6/temp/bst .js
                                                               ecms5to6/temp/bst_6.js
                                                                43 /
 44 /**
                                                                44 * Tree node
 45 * Tree node
                                                                45 */
                                                                                                                         46 */
                                                                46 class Node {
 47 function Node(value, parent) {
                                                                      constructor(value, parent) {
                                                                47
 48 this.value = value;
                                                                48
                                                                          this.value = value;
 49 this.parent = parent;
                                                                49
                                                                           this.parent = parent;
 50 this.left = null;
                                                                          this.left = null;
                                                                50
 51 this.right = null;
                                                                51
                                                                          this.right = null;
 52 }
                                                                52
 53
                                                                53}
 54 /**
                                                                54/
 55 * Insert elements to the tree respecting the BST restr
                                                                55 * Insert elements to the tree respecting the BST re
```

Figura 2 - Diferenças na codificação do arquivo bst. js antes e após a execução da ferramenta.

A Figura 3 apresenta as modificações que foram realizadas no arquivo /algorithms.js/data_structures/heap.js. Como pode ser observado, a função MinHeap foi identificada e classificada como classe. A ferramenta converteu a função para a sintaxe em JavaScript ES 6 assim como na anterior Figura 2. Mais especificamente, foram adicionadas as palavras class e constructor. Entretanto nessa conversão pode-se notar o aparecimento da palavra reserva super. Super é uma palavra reservada para invocar o método construtor da classe pai.

```
    JavaScript Source Compare ▼

                                                                                                  ecms5to6/temp/heap .js
24
                                                              24 /**
                                                               25 * Basic Heap structure
26 * Basic Heap structure
                                                              26- */
27 */
                                                               27 class MinHeap {
28 function MinHeap(compareFn) {
                                                                     constructor(compareFn) {
                                                               28
29 this._elements = [null];
                                                               29
                                                                         this._elements = [null];
30 this._comparator = new Comparator(compareFn);
                                                                         this._comparator = new Comparator(compareFn)
                                                               30
31
                                                               31
                                                                         super(this, 'n', { get: super(this) });
 32
    Object.defineProperty(this, 'n', {
                                                               32
33
      get: function () {
                                                               33-3
34
        return this._elements.length - 1;
                                                               34 MinHeap.prototype._swap = function (a, b) {
 35
      }.bind(this)
                                                               35
                                                                     var tmp = this._elements[a];
 36 });
                                                                     this. elements[a] = this. elements[b]:
```

Figura 3 - Diferenças na codificação do arquivo heap.js antes e após a execução da ferramenta.

A Figura 4 apresenta a refatoração do arquivo /mustache.js/mustache.js/Scanner.js pertencente ao software *mustache.js*. Como pode ser observado, a ferramenta converteu para nova sintaxe em JavaScript ES 6. É possível observar também que o arquivo resultante teve uma redução em sua quantidade de linhas. Além da conversão, a ferramenta é capaz de remover linhas desnecessárias, como pode ser visto nas linhas 271 e 281 do aquivo ecms5to6/temp/mustache.js.

```
    JavaScript Source Compare ▼

                                                                                                     ecms5to6/temp/mustache.js
                                                                ecms5to6/temp/mustache_6.js
      * A simple string scanner that is used by the templa
                                                                      * A simple string scanner that is used by the tem
                                                                242
      * tokens in template strings.
                                                                243
                                                                      * tokens in template strings.
275
                                                                244
     function Scanner(string) {
276
                                                                245
                                                                       class Scanner {
277
       this.string = string;
                                                                246
                                                                           constructor(string) {
278
       this.tail = string;
                                                                247
                                                                               this.string = string;
279
      this.pos = 0;
                                                                248
                                                                               this.tail = string;
280 }
                                                                249
                                                                               this.pos = 0:
281
                                                                250
282 /**
                                                                251
                                                                       3
283
      * Returns `true` if the tail is empty (end of string
                                                                252
                                                                      * Returns `true` if the tail is empty (end of str
284
                                                                253
285 Scanner.prototype.eos = function () {
                                                               254
```

Figura 4 - Diferenças na codificação do arquivo mustache.js antes e após a execução da ferramenta.

A Figura 5 apresenta o arquivo /pdf.js/src/core/Pattern.js pertencente ao software *pdf.js*. Nesse exemplo, a ferramenta foi capaz de identificar e refatorar a função MeshStreamReader que é classificada como classe em JavaScript ES 6 aninhada a função MeshClosure.

```
M 🗗 🗗 🗫 😂 🔥 🏡 🕰 🐍
ecms5to6/temp/pattern.js
                                                                          ecms5to6/temp/pattern 6.js
222 };
224
      return RadialAxial;
225 })();
                                                                                  return RadialAxial;
                                                                         225}();
225}() All mesh shading. For now, they will be presented
226
220
227// All mesh shading. For now, they will be presented as
228// to be drawn on the canvas and rgb color for each ver
228// to be drawn on the canvas and rgb color for each ver
229<mark>Shadings.Mesh = (function MeshClosure() {</mark>
                                                                         230 function MeshStreamReam
231 this.stream = stream
                            mReader(stream, context)
                                                                                      constructor(stream, conte
    this.stream = stream;
                                                                         230
                                                                                                              context) {
         this.context = context;
                                                                                           this.context = context;
this.buffer = 0;
233
         this.buffer = 0;
                                                                         232
         this.bufferLength = 0;
                                                                         233
234
235
                                                                         234
                                                                                            this.bufferLength = 0;
      MeshStreamReader.prototype = {
236
237
        get hasData() {
  if (this.stream.end) {
                                                                         236
                                                                                  MeshStreamReader.prototype = {
238
                                                                                       get hasData() {
239
             return this.stream.pos < this.stream.end;
                                                                         238
240
                                                                         239
           if (this.bufferLength > 0) {
                                                                                                return this.stream.pos < this.stream
```

Figura 5 - Diferenças na codificação do arquivo pattern. js antes e após a execução da ferramenta.

A Figura 6 apresenta a refatoração do arquivo /backbone/backbone.js/surrogate.js pertencente ao software *backbone.js*. A Ferramenta foi capaz de converter a estrutura de código de JavaScript ES 5 para JavaScript ES 6. Nessa figura, pode-se observar na linha 1653 do arquivo ecms5to6/temp/backbone.js que desenvolvedor declarou uma *Function Expression* ao utilizar a variável Surrogate e atribuir uma função a ela. Mesmo neste caso, a ferramenta *ecms5to6* foi capaz de realizar a conversão. O trecho de código entre as linhas 1630 e 1634 do arquivo ecms5to6/temp/backbone_6.js apresenta o código convertido e indentado.

```
M 🗗 🗗 🐎 👙 🔥 🏡 🙅 ፟ 🛦
ecms5to6/temp/backbone.js
                                                                                        ecms5to6/temp/backbone_6.js
        // by us to simply call the parent's constructor.
if (protoProps && _.has(protoProps, 'constructor')) {
                                                                                                          child = protoProps.constructor;
                                                                                        1620
1642
                                                                                        1621
           child = protoProps.constructor;
1643
                                                                                                          child = function () {
                                                                                        1622
1644
                                                                                                                return parent.apply(this, argumen
                                                                                        1623
          child = function(){ return parent.apply(this, argument
1646
                                                                                        1625
1647
                                                                                        1626
                                                                                                     // Add static properties to the construct
1648
        // Add static properties to the constructor function, if
                                                                                                       .extend(child, parent, staticProps);
/ Set the prototype chain to inherit fro
/ `parent`'s constructor function.
                                                                                        1627
        _.extend(child, parent, staticProps);
                                                                                        1628
1650
                                                                                        1629
        // Set the prototype chain to inherit from `parent`, wit // `parent`'s constructor function.
1651
                                                                                                          Surrogate = class Surrogate {
constructor() {
                                                                                       1630
1652
                                                                                        1631
1653 var Surrogate = function(){ this.constructor = child; };

1654 Surrogate.prototype = parent.prototype;

1655 child.prototype = new Surrogate;
                                                                                        1632
                                                                                                               this.constructor = child;
                                                                                       1634
                                                                                                     };
Surrogate.prototype = parent.prototype;
child.prototype = new Surrogate();
// Add prototype properties (instance p
// if guestied
            Add prototype properties (instance properties) to the
1657
                                                                                        1636
                                                                                        1637
1659 if (protoProps) _.extend(child.prototype, protoProps);
                                                                                                     if (protoProps)
        // Set a convenience property in case the parent's proto
1661
```

Figura 6 - Diferenças na codificação do arquivo backbone.js antes e após a execução da ferramenta.

A p	artir das ilustraç	ões apresentada	s nas figuras	anteriores,	pode-se	concluir qu	ie a resposta	a para a (QΡ é
que a ferran	enta é completa	amente capaz de	converter e	estruturas de	e código	JavaScript	ES 5 para	estrutura	s de
código JavaS	Script ES 6 em se	oftware reais. A	ferramenta s	se encontra	publicam	ente dispor	nível em:		

https://github.com/cesarfmc/ecms5to6

4. DISCUSSÃO

A discussão dos resultados foi descrita detalhadamente na Seção 3.

5. REFERÊNCIA BIBLIOGRÁFICA

- [1] H. Kienle. It's about time to take javascript (more) seriously. *IEEE Software*, 27(3):60–92, 2010.
- [2] A. Nederlof, A. Mesbah, and A. van Deursen. Software engineering for the web: the state of the practice. In *36th International Conference on Software Engineering (ICSE)*, pages 4–13, 2014.
- [3] G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dy-namic behavior of JavaScript programs. In *Conference on Programming Language Design and Implementation (PLDI)*, pages 1–12, 2010.
- [4] Leonardo Silva, Miguel Ramos, Marco Tulio Valente, Nicolas Anquetil, Alexandre Bergel. Does Javascript Software Embrace Classes?, In *22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1-10, 2015.
- [5] Marcus Arnstrom, Mikael Christiansen, and Daniel Sehl- berg. Prototype-based programming, 2003 (acessado em 21 de novembro de 2016). http://www.idt.mdh.se/kurser/-cd5130/msl/2003lp4/reports/prototypebased.pdf.
- [6] European association for standardizing information and communication systems (ECMA). ECMA-262: ECMAScript language specification. 6.0 edition, 2015.
- [7] Gunther Blaschek. Object-Oriented Programming: with Prototypes. 1th edition edition, 2012.
- [8] European association for standardizing information and communication systems (ECMA). ECMA-262: ECMAScript language specification. 5.1 edition, 2011.
- [9] Mauro Mendonça and César Couto. Uma ferramenta para detectar e inspecionar classes em JavaScript. In V Seminário de Iniciação Científica do Instituto Federal do Triângulo Mineiro (SIN), pages 1–1, 2015.

PARTE II

1. ATIVIDADES REALIZADAS NO PERÍODO

- Testes no módulo de detecção de classes em sistemas reais.
- Desenvolvimento de uma aplicação teste em Java para exercitar a programação orientada a objetos.
- Estudo da sintaxe POO de JavaScript ES 6.
- Estudo da ferramenta Esprima, um parser JavaScript responsável por gerar a árvore de sintaxe abstrata (AST) no formato JSON.
- Estudo da AST no formato JSON.
- Estudo de algoritmos e técnicas de conversão de código.
- Implementação do módulo de conversão.
- Realização de um estudo de caso com 10 softwares populares em JavaScript para avaliar a eficácia e eficiência da ferramenta *ecms5to6*.

2. PARTICIPAÇÃO EM EVENTOS CIENTÍFICOS E PUBLICAÇÕES

[1] Giullian Barbosa, César Couto. "Uma Ferramenta para Conversão de Código JavaScript Orientado a Objetos em ECMA 5 para ECMA 6". In VII Seminário de Iniciação Científica do Instituto Federal do Triângulo Mineiro (SIN), 2017.





VII Seminário de Iniciação Científica e Inovação Tecnológica do IFTM – VII SIN-IFTM Uberaba, MG, 8 de junho de 2017

UMA FERRAMENTA PARA CONVERSÁO DE CÓDIGO JAVASCRIPT ORIENTADO A OBJETOS EM ECMA 5 PARA ECMA 6

Giullian Faria Barbosa¹; César Francisco de Moura Couto²

JavaScript é uma linguagem de programação inicialmente projetada em meados da década de 90 para estender páginas web com código executável. Desde então, sua popularidade e relevância tem crescido consideravelmente. Por exemplo, JavaScript é a linguagem mais popular no GitHub (um repositório de software público), considerando novos repositórios criados por linguagem. É também reportado que a linguagem é usada por 97 de todos os 100 mais populares sites da web. Concomitantemente com o seu aumento de popularidade, o tamanho e a complexidade de softwares JavaScript está em constante crescimento. A linguagem é usada para implementar diversos tipos de aplicações, tais como clientes de email, aplicações de escritórios, IDEs, etc, as quais podem alcançar centenas de milhares de linhas de código . Desde sua criação, programas em JavaScript tem sido implementados seguindo diversos paradigmas, como por exemplo os paradigmas de programação imperativa e funcional. Alem destes, o paradigma da programação orientada a objetos é bastante utilizado, usando princípios de prototipagem. Este conceito de prototipagem define uma abordagem diferente da orientação a objetos tradicionalmente utilizada em linguagens como Java ou C#. Adicionalmente, existe um problema sintático, dado que não existem em JavaScript palavras reservadas comumente utilizadas, como por exemplo, class. Essas diferencas podem acabar causando dificuldade na implementação do código orientado a objetos. Essa divergência sintática foi alvo das atualizações da ECMAScript, que em sua nova versão, ES 6 (ECMAScript 6) introduz uma nova sintaxe para classes. A proposta e manter o funcionamento baseado em protótipos, mas com uma forma de implementação mais conveniente e sintaticamente próxima dos conceitos tradicionais de orientação a objetos. Para que os desenvolvedores possam usufruir dos benefícios desta nova versão, este trabalho de pesquisa apresenta uma ferramenta, intitulada ecms5to6 para migração de classes de código JavaScript ES 5 para nova sintaxe proposta de ES 6. Com isso, espera-se ajudar os milhares de desenvolvedores que possuem código que emula classes de acordo com a sintaxe antiga a se beneficiarem de forma automática da sintaxe nativa de classes proposta pela

Palavras-chave: Classes em JavaScript, Parser, ECMAScript 6.

Apoio: Fapemig.

1Estudante, IFTM Campus Paracatu, MG, bolsista FAPEMIG. giullianfb@yahoo.com.br 2Professor, IFTM Campus Paracatu, MG, Dr. Ciência da Computação. cesarcouto@iftm.edu.br

Local: Paracatu-MG Data: 12/06/2018.

Assinatura do Coordenador