Mutode: Generic JavaScript and Node.js Mutation Testing Tool

Diego Rodríguez-Baquero
Universidad de los Andes
Bogotá, Colombia
d.rodriguez13@uniandes.edu.co

Mario Linares-Vásquez
Universidad de los Andes
Bogotá, Colombia
m.linaresv@uniandes.edu.co

ABSTRACT

Mutation testing is a technique in which faults (mutants) are injected into a program or application to assess its test suite effectiveness. It works by inserting mutants and running the application's test suite to identify if the mutants are detected (killed) or not (survived) by the tests. Although computationally expensive, it has proven to be an effective method to assess application test suites. Several mutation testing frameworks and tools have been built for the various programing languages, however, very few tools have been built for the JavaScript language, more specifically, there is a lack of mutation testing tools for the Node.js runtime and npm based applications. The npm Registry is a public collection of modules of open-source code for Node.js, front-end web applications, mobile applications, robots, routers, and countless other needs of the JavaScript community. The over 700,000 packages hosted in npm are downloaded more than 5 billion times per week. More and more software is published in npm every day, representing a huge opportunity to share code and solutions, but also to share bugs and faulty software. In this paper, we briefly describe prior work for mutation operators in JavaScript and Node.js, and propose `Mutode`, an open source tool which leverages the npm package ecosystem to perform mutation testing for JavaScript and Node.js applications. We empirically evaluated `Mutode` effectiveness by running it on 12 of the top 20 npm modules that have automated test suites.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

Mutation testing, Operators, JavaScript, Node.js

ACM Reference Format:
Diego Rodríguez-Baquero and Mario Linares-Vásquez. 2018. Mutode: Generic JavaScript and Node.js Mutation Testing Tool. In *Proceedings of 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3213846.3229504

ISSTA'18, July 16–21, 2018, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5699-2/18/07…$15.00
https://doi.org/10.1145/3213846.3229504

1 INTRODUCTION

JavaScript and Node.js applications, and their ecosystem, have been growing steadily in the past few years thanks to the possibility of reusing code that runs both in front-end (web browser, mobile applications) and in back-end (server), and the easiness to use published CommonJS-defined[33] and ECMAScript modules[10]. This growth can be seen in comparison to other programming languages and module registries [6], and in the statistics [41] published by the Chief Operating Officer of npm — the package manager for Node.js —, which was built as an open source project in 2009 and later registered as a company in 2014. It is also an ever growing registry of open source modules built for the JavaScript, Node.js and Web ecosystems [26].

The over 700,000 modules hosted in npm are downloaded more than 5 billion times per week [41]. More and more software is published in npm every day, representing an opportunity to share code and solutions, but also to share bugs and faulty software. A recent survey of over sixteen thousand developers shows that 77% of respondents are concerned about the security in open source modules, and 52% of respondents are not satisfied with the tools they have got to evaluate open source code security and quality[28]. As a reaction from the practitioners, current state of the practice for testing, used in the most popular modules, includes unit testing, and an increasing number of developers are opting in using test code coverage tools such as Istanbul[4]. However, using unit testing frameworks and measuring coverage is not enough by itself, because the quality of test suites needs also to be measured. Thus, here is an opportunity for going beyond test code coverage and take advantage of mutation testing, which goes a step above and measures test suites' effectiveness.

Mutation testing is a technique in which faults (mutants) are inserted into a application to assess its test suite effectiveness [18]. It works by inserting mutants and running the application's test suite to identify if the mutants are detected (killed) or not (survived). Although computationally expensive, it has proven to be an effective method to assess applications' test suites [18]. Recent papers have shown that statement and branch coverage are not enough for measuring test suite effectiveness [1, 20] and mutation testing is more effective than data flow testing for detecting faults in applications [31].

While for other types of applications there are well known and maintained mutation testing frameworks, in the case of JavaScript and Node.js applications, practitioners lack an easy to use and well-maintained applications that allows mutation testing. Consequently, there is a big opportunity for creating a *generic* mutation testing tool that targets the latest practices for software development in JavaScript and Node.js, for both back-end and front-end code; one that works with any type of testing framework; and one which requires zero or minor configuration.

We analyzed prior work for mutation operations in JavaScript and Node.js and then propose a catalog of mutation operations grouped by mutators, and a new tool which leverages npm package ecosystem to build a generic tool that allows easy, zero-configuration mutation testing on current generation applications using any testing framework. By generic we mean that it will support any type of test suite that runs with the `npm test` command, without plug-ins or configuration, and does not generate source code from the modified the Abstract Syntax Tree (AST), which could affect the linter rules set by the developer. In this paper, we describe `Mutode`, a mutation testing tool for JavaScript and Node.js applications and modules, that aims to support developers in writing and complementing tests. `Mutode` includes 16 mutators with 43 mutation operations that target both JavaScript and Node.js code, as well as general programming code. We empirically evaluated `Mutode` effectiveness by running it on the top 20 npm modules that have automated test suites.

2 RELATED WORK

Some of the mutation testing frameworks reported or described in previous works are PIT[5], Major[19], μJava [24], MDroid+[22], Milu[17], Mull[7], Mutate++[23], VisualMutator[8], Cosmic Ray[3], MutPy[11], Mutant[38], NinjaTurtlesMutation[37], μDroid [15], and Infection[32]. In the case of JavaScript and Node.js apps, practitioners lack an easy to use and well-maintained tool that allows mutation testing. However, there is significant research about JavaScript testing, bugs and their patterns [9, 12, 29, 30].

For mutation testing of JavaScript and Node.js applications, we found Mutandis[25], a framework built in Java focused on client-side JavaScript applications. It utilizes Mozilla's Rhino AST parser to mutate the application and intercepts network requests to analyze their content. If a test suite is available for a given application, it's used to profile it. This tool, however, wasn't made for Node.js applications and npm modules. Besides the initial work 4 years ago, no work or maintenance has been done to the library since. There is also a lack of documentation in its GitHub repository and instructions on how to use it.

In the practice, the main mutation testing framework available for JavaScript and Node.js is Stryker[16]. An open source framework available at GitHub. It offers over 30 mutation operations and allows running tests with Karma[21], Mocha[14], among other frameworks through a custom built plug-in. It requires configuration and does not support other test frameworks besides Karma and Mocha without a plug-in. This represents a room for improvement for a new tool that works out of the box with any test suite. Another tool is Mutant[13], a mutation testing tool which has 7 mutation operations, is limited to test frameworks with TAP[39] output, and uses babel[40] to parse and regenerate AST; however it is unmaintained, only five versions were released.

3 MUTODE

With the purpose of creating an useful tool that targets a wide range of JavaScript and Node.js applications, `Mutode` takes into account the following design decisions: (i) `Mutode` does not perform operations at the AST-level in order to keep compliance of the existing code with any linter the developer uses within the application or module; (ii) `Mutode` uses the `npm test` command in order to run the tests against each mutant, thus, no configuration or plug-ins are needed to execute existing tests; (iii) the mutants are executed in parallel by workers that consume tasks from a queue to reduce the time required to test mutants and harness the CPU power.

3.1 Implemented Operations

In order to create an effective mutation testing tool for JavaScript and Node.js, we propose a catalog of 43 mutation operations grouped in 16 mutators. Our operations catalog is built on top of operations previously defined for (i) Java applications [5], (ii) JavaScript applications [25], (iii) new operations proposed by us based on our experience developing JavaScript/Node.js applications, (iv) and open source reported bugs in npm modules. The `Mutode` mutators and operations are listed in Table 1. It is worth noting that we call "mutator" to a group of mutation operations (*e.g.,* Numeric literals), and "operations" to specific mutations applied to the source code (*e.g.,* replace a *true* boolean literal with *false*). A detailed explanation of each mutator and their operations is available on the project website[35]. The project website also has a table that maps our mutators to previously operations defined in [5, 25].

3.2 Mutode Execution

The workflow in `Mutode` is described as follows. When an instance of `Mutode` is created, the `index.js` file and the `lib/` folder of the application or module are analyzed (in the case the user does not provide a path or array of paths) looking for files that could be mutated by `Mutode`; if no files are found, then `Mutode` exits with an error. Before starting the mutation process, `Mutode` deletes previous copies of the application (if any), creates a new copy of the application and times the test suite execution with no mutants (*i.e.,* the test suite is only executed on the original application) to set a timeout value that will discard mutants which execution do not exit after 2.5x the expected timing; if the test suite does not pass (*i.e.,* process exits with a code different than 0), then `Mutode` exits with an error. After timing the test suite, `Mutode` creates extra copies for each execution thread; the copies are used as the base for the mutants generated by each thread.

For each file to be mutated an empty queue is created where mutants will be added by mutators, the file content is read and its AST is parsed by babylon[2] (the AST parser for babel[40]). Afterwards, mutators are executed to generate mutants for a given file and add them to the queue; a mutant in the queue is then processed (*i.e.,* the test suite is executed on the mutant) by a free worker, each having its own copy of the application, and logging the mutation results (*i.e.,* killed, survived or discarded) to the console. Finally once the queue is empty, a mutated file is reverted to the original version so no mutants of the file are left in subsequent mutant executions of other files' mutants. A consolidated report is shown at the end of the process, summarizing mutants created, killed, survived and discarded, as well as the mutants coverage, calculated as:

$$\frac{\text{Killed mutants}}{\text{Generated mutants - discarded mutants}} \times 100$$

Mutants Creation. Given the catalog of operations listed in Table 1, `Mutode` generates a mutant for each line or set of lines

Table 1: Mutators and Operations in Mutode

Mutator	Operation Replacement or Action	
	Original	**Mutant(s)**
Boolean Literals	`true`	`false`
	`false`	`true`
Conditionals Boundary	`<`	`<=`
	`<=`	`<`
	`>`	`>=`
	`>=`	`>`
Increments	`++`	`--`
	`--`	`++`
Invert Negatives	`-a`	`a`
Math	`+`	`-`
	`-`	`+`
	`*`	`/`
	`/`	`*`
	`%`	`*`
	`&`	`|`
	`|`	`&`
	`^`	`|`
	`<<`	`>>`
	`>>`	`<<`
	`**`	`*`
Negate Conditionals	`==`	`!=`
	`!=`	`==`
	`===`	`!==`
	`!==`	`===`
	`>`	`<=`
	`>=`	`<`
	`<`	`>=`
	`<=`	`>`
Numeric Literals	a	`a + 1`
		`a - 1`
		Random value
		`0`
Remove Array Elements	`[a, b]`	`[a]`
		`[b]`
Remove Conditionals	`a < 10`	`true`
		`false`
Remove Function Call Arguments	`func(a, b)`	`func(a)`
		`func(b)`
Remove Function Parameters	`func(a, b) {}`	`func (a) {}`
		`func (b) {}`
Remove Functions	Removes functions by commenting them	
Remove Lines	Removes single line statements by commenting them	
Remove Object Props	`{a:1,b:2}`	`{a:1}`
		`{b:2}`
Remove Switch Cases	`switch(v) { case 1: ... case 2: ... }`	`switch(v) { case 1: ... }`
		`switch(v) { case 2: ... }`
String Literals	`My string`	Empty string
		Random string

where each mutator identifies a potential location for applying an operation. This process is performed by first using the AST and identifying statement types that match those in which each mutator could create a new mutant, and then using text comparisons with each operation that can be applied. With the AST information of the statement, the operation is applied into the source code of the file swapping exact text without affecting the code's format, commenting lines without affecting the original code's lines number or removing code from the an exact line in the source code. A mutant ID is assigned and the information is logged to a mutants log to offer the developer a way of knowing what was changed. A full copy of the file with the mutant is then pushed to the tasks queue.

Mutants Execution. Each mutant in the queue is inserted in one of the available workers' copy of the application's source. A child process running `npm test` is then launched on the copy, which captures the exit code and marks it as either survived, if the exit code was 0, or killed if else. In the case where the child process does not exit, a timeout, set to 2.5x the original test suite timing, is set to discard the mutant and kill the child process. Result is logged to the console and used worker is freed for executing next mutant available in the queue. There are various reasons as to why a mutant could be discarded but the main ones are the following: (i) the callback argument in a function call is removed, or the callback parameter is removed from a function declaration, therefore further execution does not happen; (ii) a callback call or a returned function call is removed, therefore further execution does not happen; (iii) a timeout timer's value is modified by the numeric literals mutator to a larger number than the mutant's execution timeout; (iv) a timer or child process unreferencing is removed and process never dies, even if execution is completed; (v) an state variable is switched or removed, blocking the application's continuity.

3.3 Tool Usage and Extensibility

`Mutode` is published as an npm module, available at [34] and intended to be used as a command-line utility in which the developer can decide the specific set of mutators, to be applied in the generation of mutants and execution, as well as a parameter to set the concurrency of mutants execution and a parameter to set the files to mutate. A README, user guide and video showing `Mutode` execution can be found in the project repository[36].

4 EVALUATION

To evaluate `Mutode`, we conducted a study with the following goals: (i) test the applicability of `Mutode` in top npm modules; (ii) fix any execution bugs that arise during the top npm modules mutation analysis, and (iii) improve the core of `Mutode` and mutators' operations. To accomplish this, we executed `Mutode` on the top 20 most depended npm modules according to the npm registry [27].

Study Results. Mutation testing results are listed in Table 2. Out of the 20 top npm modules, `Mutode` was successfully executed in 12. `lodash@4.17.5` could not be tested because of excessive memory consumption; `react@16.3.2`, `react-dom@16.3.2` and `babel-runtime@7.0.0` could not be tested because did not include `npm test` command nor an easy way to include it; `bluebird@3.5.1` could not be tested because it has a restriction that its tests be must run in a folder called 'bluebird', and `Mutode` copies of the application are named as `.mutode-{instance ID}-{copy number}`; `fs-extra@5.0.0` and `mkdirp@0.5.1` could not be tested because their tests were failing; `prop-types@15.5.1` was not included as all of the mutants survived and there was no evidence its tests target the source.

On average 2952 mutants were generated for each module, and the test suites in the modules achieved an average mutation coverage of 70.59%. The highest mutation coverage was achieved by `express@4.16.3` and the lowest by `debug@3.1.0`. Note that the latter only has four tests cases in its test suite, and the test cases are oriented to the very core functionality assuring it does not throw errors and does not repeat its logging when called.

To illustrate the reasons of surviving and discarded mutants, we analyzed `Mutode` execution on `commander@2.15.1`. Out of the 390 surviving mutants: 114 were string literal mutations; 81 were numeric literal mutations; 57 were removed function call mutations;

Table 2: Study results on npm modules

Module Information		Mutode's Timings		Mutants Execution Results				
Name & Version	Test Cases	Generation	Execution	Generated	Killed	Survived	Discarded	Mutant Coverage
`request@2.85.1`	1519	2 s	7 h 3 m	2073	1757	306	10	85.16%
`chalk@2.4.0`	56	<1 s	41 m 2 s	330	307	23	0	93.03%
`async@2.6.0`	516	6 s	15 h 40 m	3636	2214	47	1375	60.89%
`express@4.16.3`	855	2 s	14 h 34 m	4100	2469	0	1631	100%
`commander@2.15.1`	44	<1 s	2 h 44 m	1404	1006	390	8	72.06%
`moment@2.22.1`	3358	8 s	38 h 6 m	10944	5014	1747	4183	74.16%
`debug@3.1.0`	4	3 s	18 m 56 s	892	126	762	4	14.19%
`underscore@1.9.0`	1570	2 s	59 m 45 s	2848	2225	513	110	81.26%
`colors@1.2.1`	32 (assertions)	2 s	4 m 37 s	2292	474	1818	0	20.68%
`body-parser@1.18.2`	229	1 s	7 m 32 s	1228	1058	167	3	86.37%
`glob@7.1.2`	1706	1 s	6h 29 m	2171	1593	530	48	75.04%
`uuid@3.2.1`	17	1 s	12 m 14 s	3510	2936	550	24	84.22%

53 were removed line mutations; 29 were removed conditional mutations; and the rest were distributed in the following mutators: *math*, *remove array elements*, *remove object property*, *remove functions*, *negate conditionals*, *conditionals boundary*, and *boolean literals*. Out of the eight discarded mutants, four were because of removed arguments of a child process execution started by the module, which then will not exit; three were because of modified `for` statements' step, which caused an infinite loop; and one was because of a modified `for` statements' boundary which blocked execution by not finishing.

5 DEMO REMARKS & FUTURE WORK

In this tool demo paper, we presented `Mutode`, a generic mutation testing tool for JavaScript and Node.js applications that supports 43 mutation operations and leverages the npm ecosystem to run test suites using the `npm test` command, supporting any testing framework. `Mutode` is published as a free and open source npm module, hosted on GitHub, under the MIT License. `Mutode` was evaluated against the top-20 most depended upon modules of the npm registry and was shown to generate various surviving mutants which could provide insights on how to improve those modules' test suites. In the future, we plan to improve current mutation operations, implement more mutation operations and improve execution performance. Also future work should be devoted to detect and avoid redundant and equivalent mutants.

Acknowledgements We would like to thank Camilo Escobar and the Master Students from the Automated Testing course at Universidad de los Andes for their help testing `Mutode`.

REFERENCES

[1] James H Andrews, Lionel C Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using mutation analysis for assessing and comparing testing coverage criteria. *TSE* 32, 8 (2006), 608–624.
[2] Babel. 2018. Babylon. https://www.npmjs.com/package/babylon
[3] Austin Bingham. 2017. Cosmic Ray: mutation testing for Python. https://github.com/sixty-north/cosmic-ray
[4] Ben Coe. 2018. Istanbul. https://istanbul.js.org/.
[5] Henry Coles. 2017. PIT. http://pitest.org/.
[6] Erik DeBill. 2018. Modulecounts. http://www.modulecounts.com/.
[7] Alex Denisov and Stanislav Pankevich. [n. d.]. Mull it over: mutation testing based on LLVM. https://github.com/mull-project/mull. ([n. d.]).
[8] Anna Derezińska and Piotr Trzpil. 2015. Mutation Testing Process Combined with Test-Driven Development in. NET Environment. In *Theory and Engineering of Complex Systems and Dependability*. 131–140.
[9] A. M. Fard and A. Mesbah. 2017. JavaScript: The (Un)Covered Parts. In *ICST'17*. 230–240.
[10] Node Foundation. 2018. ECMAScript Modules. https://nodejs.org/api/esm.html.
[11] Konrad Halas. 2017. MutPy: mutation testing tool for Python 3.x source code. https://github.com/mutpy/mutpy
[12] Quinn Hanam, Fernando S. de M. Brito, and Ali Mesbah. 2016. Discovering Bug Patterns in JavaScript. In *ESEC/FSE'16*. 144–156.
[13] Ben Hartley. 2016. Mutant: A mutation testing framework for JavaScript. https://github.com/benhartley/mutant
[14] TJ Holowaychuk. 2018. Mocha. https://mochajs.org/.
[15] Reyhaneh Jabbarvand and Sam Malek. 2017. muDroid: An Energy-aware Mutation Testing Framework for Android. In *ESEC/FSE'17*. 208–219.
[16] Nico Jansen and Simon de Lang. 2018. Stryker: The JavaScript mutation testing framework. https://github.com/stryker-mutator/stryker
[17] Yue Jia and Mark Harman. 2008. MILU: A customizable, runtime-optimized higher order mutation testing tool for the full C language. In *Practice and Research Techniques, 2008. TAIC PART'08. Testing: Academic & Industrial Conference*. 94–98.
[18] Yue Jia and Mark Harman. 2011. An analysis and survey of the development of mutation testing. *TSE* 37, 5 (2011), 649–678.
[19] René Just. 2014. The Major mutation framework: Efficient and scalable mutation analysis for Java. In *ISSTA'14*. 433–436.
[20] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *ESEC/FSE 14*. 654–665.
[21] Vojta Jína. 2018. Karma. https://karma-runner.github.io/2.0/index.html.
[22] Mario Linares-Vásquez, Gabriele Bavota, Michele Tufano, Kevin Moran, Massimiliano Di Penta, Christopher Vendome, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. 2017. Enabling mutation testing for Android apps. In *ESEC/FSE'17*. 233–244.
[23] Niels Lohmann. 2017. Mutate++: C++ Mutation Test Environment. https://github.com/nlohmann/mutate_cpp
[24] Yu-Seung Ma, Jeff Offutt, and Yong Rae Kwon. 2005. MuJava: An automated class mutation system. *Software Testing, Verification and Reliability* 15, 2 (2005), 97–133.
[25] Shabnam Mirshokraie, Ali Mesbah, and Karthik Pattabiraman. 2013. Efficient JavaScript mutation testing. In *ICST'13*. 74–83.
[26] Inc. npm. 2018. About npm. https://www.npmjs.com/about
[27] Inc. npm. 2018. Most depended upon packages. https://www.npmjs.com/browse/depended
[28] npm, Inc., Node.JS Foundation, and JS Foundation. 2018. Attitudes to security in the JavaScript community – npm, Inc. – Medium. https://medium.com/npm-inc/security-in-the-js-community-4bac032e553b
[29] F. Ocariza, K. Bajaj, K. Pattabiraman, and A. Mesbah. 2013. An Empirical Study of Client-Side JavaScript Bugs. In *ESEM'13*. 55–64.
[30] Frolin S Ocariza Jr, Karthik Pattabiraman, and Benjamin Zorn. 2011. JavaScript errors in the wild: An empirical study. In *ISSRE'11*. 100–109.
[31] A Jefferson Offutt, Jie Pan, Kanupriya Tewary, and Tong Zhang. 1996. An experimental evaluation of data flow and mutation testing. *Softw., Pract. Exper.* 26, 2 (1996), 165–176.
[32] Maks Rafalko. 2018. Infection: PHP Mutation Testing Framework. https://github.com/infection/infection
[33] RequireJS. 2018. CommonJS. http://requirejs.org/docs/commonjs.html.
[34] Diego Rodríguez-Baquero. 2018. mutode. https://www.npmjs.com/package/mutode
[35] Diego Rodríguez-Baquero. 2018. Mutode - Mutators Documentation. https://thesoftwaredesignlab.github.io/mutode/module-Mutators.html
[36] Diego Rodríguez-Baquero. 2018. Mutode: Mutation testing for JavaScript and Node.js. https://github.com/TheSoftwareDesignLab/mutode
[37] Tony Roussel. 2016. NinjaTurtlesMutation. https://github.com/criteo/NinjaTurtlesMutation
[38] Markus Schirp. 2018. Mutant: Mutation testing for Ruby. https://github.com/mbj/mutant
[39] Michael G Schwern and Andy Lester. 2018. TAP specification. https://testanything.org/tap-specification.html.
[40] Daniel Tschinder, Logan Smyth, and Henry Zhu. 2018. Babel. https://babeljs.io/.
[41] Laurie Voss. 2018. Rolling weekly downloads of npm packages. https://twitter.com/seldo/status/988477780441481217