SNUTS.js: Sniffing Nasty Unit Test Smells in Javascript

Jhonatan Oliveira
University of the State of Bahia (UNEB)
Bahia, Brazil
natanjesuss20@gmail.com

Luigi Mateus
University of the State of Bahia (UNEB)
Bahia, Brazil
luigimee15@gmail.com

Tássio Virgínio
Federal University of Bahia (UFBA)
Bahia, Brazil
tassio.virginio@ufba.br

Larissa Rocha
University of the State of Bahia (UNEB)
State University of Feira de Santana (PGCC/UEFS)
Bahia, Brazil
larissabastos@uneb.br

ABSTRACT

Test smells indicate potential issues or weaknesses within the test code, which can compromise its effectiveness and maintainability. They highlight areas where improvements can enhance the overall quality of the test suite or testing practices. For instance, an example of a test smell is the Anonymous Test, where the test's name lacks descriptive information about its function or purpose. Addressing these test smells can result in more robust and maintainable test suites, thus improving the reliability of the testing process. Despite significant research on these issues, tools are scarce for automatically detecting them, particularly in certain programming languages such as JavaScript. In the current landscape, existing test smell detection tools for JavaScript lack intuitiveness and graphical interfaces, and require extensive configuration, which may lead to low adoption within the developer community. To address this gap, we propose SNUTS.js, a tool designed to streamline the detection of test smells in JavaScript. Designed as an API, SNUTS.js offers versatility, allowing integration with various tools and environments. This tool goes beyond existing solutions by identifying previously undetected test smells, including the Anonymous Test, Comments Only Test, Overcommented, General Fixture, Transcripting Test, and Sensitive Equality. We also introduce a new test smell termed Test Without Description, which denotes a test case lacking descriptive text. In a preliminary evaluation, we constructed a dataset of tests sourced from real-world projects on GitHub. Through manual analysis, we identified 285 instances of test smells. SNUTS.js demonstrated a detection accuracy of 100% for three specific types of test smells, Anonymous Test, Overcommented, and General Fixture, all tailored to the JavaScript environment. Link to the video: https://youtu.be/89z0jy4Nu0s

KEYWORDS

Test Smells, JavaScript, Test Quality, Tool

1 INTRODUCTION

Software testing is a cornerstone of high-quality software development. It plays a critical role in identifying and preventing bugs before they reach end users, thereby enhancing confidence in software deliveries and preventing regressions throughout the development process [8, 12, 18]. Despite its importance, testing demands substantial effort in both development and maintenance. Effective test coverage often requires multiple methods, and any changes to production code necessitate the creation or modification of tests [10, 22].

High-quality tests ensure that code functions correctly and is maintainable and evolvable. This requires test code to be comprehensible and clear, which facilitates easier maintenance and updates to production code [10, 22]. Automated testing, a prevalent practice in industrial software development, provides several advantages such as repeatability, predictability of results, and speed, all of which contribute to efficient fault detection [13, 18, 22].

However, automated tests are not without their issues. They are susceptible to errors and must adhere to quality guidelines to minimize such occurrences [13, 15, 18, 22]. One key metric for assessing test quality is coverage, which measures how thoroughly the tests exercise the production code [7]. Nevertheless, high coverage does not necessarily equate to high-quality test code in terms of readability and maintainability. Poorly written test code, known as "test smells," can negatively impact the overall quality and effectiveness of tests [4].

Research into test smells has largely focused on statically typed languages such as Java, Scala, Smalltalk, and C++ [5, 14, 20]. In contrast, JavaScript, despite being one of the most widely used programming languages, has received comparatively little attention in this area. This oversight is significant given JavaScript's unique syntactical and paradigmatic characteristics, which could influence the introduction and detection of test smells.

A recent systematic mapping study identified 22 test smell detection tools, yet none for JavaScript [1]. This gap presents a promising research opportunity, especially considering JavaScript's extensive use and the potential for unique test smells due to its distinct language features. Further investigation into the literature post-Aljedaani et al. [1] revealed only two studies focused on JavaScript test smells [11, 17], with only one tool available online for downloading [17]. These tools together detect 23 types of test smells but are not user-friendly or straightforward to use.

This study introduces SNUTS.js, a novel tool designed to detect test smells in JavaScript through a user-friendly interface. SNUTS.js enhances the detection process by providing detailed visualizations, including the number of smells found, the files they are located in, and the specific lines where they occur. Implemented as an API, SNUTS.js offers versatility and seamless integration with various tools and environments. The current version of SNUTS.js detects a set of test smells, including *Anonymous Test*, *Comments Only*

Test, *Overcommented*, *General Fixture*, *Transcripting Test*, and *Sensitive Equality*. Additionally, it introduces a new smell, *Test Without Description*, characterized by test cases lacking descriptive text. Preliminary results indicate that SNUTS.js achieves 100% detection accuracy for certain test smells, demonstrating its effectiveness in the JavaScript environment. We also provide a dataset with more than 500 manually identified test smells.

2 BACKGROUND AND RELATED WORK

2.1 Test Smells

Unit testing refers to testing software units in isolation (e.g., methods), ensuring that the production code functions as intended Aniche [2]. Analogous to this, when there is poor structuring of the test code, the presence of Test Smells is inevitable. Test smells are poor design choices that degrade the overall quality and effectiveness of the test suites. The presence of test smells can lead to less reliable, harder to maintain, and less efficient tests, ultimately compromising the ability to ensure software correctness and robustness [3, 9]. Identifying and addressing these test smells is crucial to maintaining high standards in software testing practices and ensuring the delivery of high-quality software products.

Aljedaani et al. [1] identified and mapped a total of 66 types of test smells from various programming languages, including Java, Scala, Smalltalk, C++, and Python. The Aljedaani et al. [1] catalog does not include JavaScript, which has some specificities. For example, in Java, the name of the test is the method's name itself, whereas in JavaScript, a string defines the test's name. In Java, the test structure is the body of the method, while in JavaScript, it is passed as a function. Due to this different structure, we were able to identify a new test smell, "Test without description (TWD)", a test smell that occurs when a test case lacks a description, which is not possible in Java.

In the following subsections, we provide seven examples of these test smells, along with corresponding code snippets in JavaScript that we created ourselves. To the best of our knowledge, these specific test smells have not been previously demonstrated in JavaScript. To select this set of smells, we started from the catalog provided by Aljedaani et al. [1] and adopted the following criteria: the absence of detection by other solutions; ease of implementation; and the relevance of these smells for JavaScript.

2.1.1 Anonymous Test (AT). : type of test whose name is not descriptive enough to indicate its function or purpose. In the following test example, the designation '*check date*' can lead to different interpretations within the 'it' block, which does not fully explain the functionality or purpose of the test and can result in a lack of clarity and difficulty in maintenance. Descriptive names make it clearer to those reading the test what is being verified.

```
it("check date", () => {
    const date = new Date();
    expect(date).toBeInstanceOf(Date);
});
```

2.1.2 Overcommented Test (OT). : refers to a type of test that has been excessively commented on. In the following example, the comments redundantly describe what the code is doing, such as explaining the purpose of calling the add function or checking if the result is equal to 5. These comments add noise to the code without providing any additional value.

```
// Test to verify if the getUserInfo() function returns user information correctly
test('getUserInfo returns user information', () => {
  // Calls the getUserInfo() function to obtain user information
  const userInfo = getUserInfo();
  // Equality-sensitive test: Compares the object returned by the function with an expected object using the toEqual() method
  // This test verifies if all properties of the returned object are identical to those of the expected object
  expect(userInfo).toEqual({
    id: 1, // Checks if the id is 1
    username: 'john_doe',
    email: 'john@example.com' // Checks if the email is 'john@example.com'
  });
});
```

2.1.3 Sensitive Equality (SE). : type of test smell that occurs when an assertion involves an equality with the *toString* method. In the following example, the test block uses the *toString* method in the assertion, resulting in a test that is sensitive to the specific implementation of the toString() method, which can make it fragile and less reliable.

```
test ('it should check user age to be equal to 13',()=>{
    const user = getUser()
    expect(user.age.toString()).toBeEqual('13')
}
```

2.1.4 Test without description (TWD). : type of test smell that occurs when a test case lacks a description. In the example below, the test block lacks a description, which can result in a lack of understanding and difficulty in maintenance. It is relevant to provide clear and concise descriptions for each test to ensure they are understandable and easy to maintain.

```
const sum = require('./sum');
test('', () => {
  expect(sum(1, 2)).toBe(3);
});
```

2.1.5 Comments Only Test (COT). : type of test smell that occurs when the test or a test block is entirely commented out, as the following example shows.

```
const UserManager = require('./userManager');
// test('should remove a user successfully', () => {
// const user = { id: 1, name: 'John Doe' };
// userManager.addUser(user);
// userManager.removeUser(user.id);
// expect(userManager.getUsers())
// .not.toContainEqual(user);
});
```

2.1.6 Transcripting Test (TT). : refers to a test smell that occurs when print commands are used within the test block. The following example shows that different types of commands are used within the test blocks to display messages in the terminal, which can result in a performance impact, security risk, and visual clutter.

```
test("Test 1", () => {
  console.log("Logging to the console");
  expect(someFunction()).toBe(true);
});
test("Test 2", () => {
  console.warn("Warning message");
  expect(anotherFunction()).toBe(false);
});
test("Test 3", () => {
  console.error("Error message");
  expect(anotherFunction()).toBe(false);
});
```

2.1.7 General Fixture (GF). : It is a type of test smell that occurs when the test setup defines multiple data or objects but only uses a subset of them. The following example shows that the tests are preceded by a *beforeEach* block, which is executed before each test. This ensures that each instance of User, Admin, and Guest is initialized with consistent values before the execution of each test. However, no test uses the *guest* variable.

```
let user;
let admin;
let guest;
beforeEach(() => {
  user = new User("Alice", 30);
  admin = new Admin("Bob", 40);
  guest = new Guest("Charlie", 25);
});
test("user should have a name", () => {
  expect(user.name).toBe("Alice");
});
test("admin should have an age", () => {
  expect(admin.age).toBe(40);
});
```

2.2 Tools for test smell detection

Identifying test smells is a crucial practice for ensuring the quality and maintainability of software. Given the complexity and time-consuming nature of manually identifying test smells, automated tools have become essential. In the study presented by Garousi and Küçük [6], they highlighted the inefficiency and challenges associated with the manual identification and analysis of test smells. The manual process is not only labor-intensive but also prone to oversight, making it ineffective for large codebases where comprehensive results are necessary.

Recognizing these challenges, Aljedaani et al. [1] conducted a systematic mapping study and identified 22 tools specifically designed for test smell detection. These tools were gathered from peer-reviewed scientific publications and encompass a wide range of programming languages. Additionally, their study compiled a dataset featuring 66 distinct types of test smells.

One notable tool is JNose, presented by Virgínio et al. [19], which identifies 21 types of test smells in Java projects. JNose allows developers to analyze code quality and track improvements over time as the project evolves. It also calculates unit test coverage, providing a comprehensive view of test effectiveness within a project.

Similarly, Santana et al. [16] introduced RAIDE (Refactoring Test Design Errors), a tool that uses Abstract Syntax Tree (AST) analysis to identify and refactor *Assertion Roulette* and *Duplicate Assert* test smells in Java with JUnit. Developed as an Eclipse IDE plugin, RAIDE leverages AST to accurately pinpoint test smells in the source code.

For Python, Wang et al. [21] proposed PyNose, a tool designed to detect 17 types of test smells. PyNose also introduced a new test smell called “Suboptimal Assert", highlighting the evolving nature of test smell research and the need for language-specific solutions.

Each programming language presents unique characteristics that influence test smell detection. Test smells that are highly relevant in one language might not be as significant in another due to these inherent differences.

For JavaScript, Jorge et al. [11] developed Steel, a tool capable of identifying 15 types of test smells adapted to the JavaScript language. These include Eager Test, Lazy Test, Assertion Roulette, Conditional Test Logic, Duplicate Assert, Empty Test, Exception Handling, Ignored Test, Sleepy Test, Magic Number Test, Redundant Assertion, Redundant Print, Mystery Guest, Unknown Test, and Resource Optimism.

Building on this, Silva [17] introduced a tool that identifies eight types of test smells in JavaScript, implementing five additional smells beyond those covered by Steel. Silva’s tool adapted test smells identified by Wang et al. [21] in PyNose, tailoring them to the JavaScript context due to certain similarities between the languages. Specifically, Silva [17] implemented detection for test smells such as Suboptimal Assert, Verifying in Setup Method, Non-Functional Statement, Verbose Test, and Unused Imports. Additionally, they identified three new test smells: Identical Test Description, Only Test (OT), and Complex Snapshot (CS).

In conclusion, while significant progress has been made in developing tools for test smell detection across various programming languages, there remains a gap in comprehensive solutions for JavaScript. To the best of our knowledge, Steel and Silva’s tools are currently the only ones available specifically for JavaScript. These tools have made important strides by adapting test smells from other languages and introducing new ones relevant to JavaScript. However, the range of detected test smells is not as extensive as in languages like Java and Python, where tools like JNose and PyNose

provide more comprehensive coverage. As the field of test smell detection continues to evolve, it is crucial to further develop and refine tools for JavaScript to ensure high-quality and maintainable code in this widely used language.

3 SNUTS.JS

Before conducting this study, we performed a comprehensive review of the state of the art to examine detection tools for test smells across various programming languages, such as JNose for Java and PyNose for Python. Our aim was to map existing and common test smells. We also utilized the test smells cataloged in the study by Aljedaani et al. [1], which identified 22 test smell detection tools provided by the research community and established a dataset containing 66 test smells. From this mapping, we analyzed which test smells were already automatically detected by tools for the JavaScript language. After identifying and excluding these, we compiled a list of test smells that were not yet automatically detected. We then assessed the feasibility of the remaining test smells based on the characteristics of the JavaScript language.

Currently, the two existing JavaScript test smell detection tools are not very intuitive, requiring extensive configuration, which might result in low adoption by the developer community. Therefore, we developed SNUTS.JS to offer a versatile and user-friendly solution with an intuitive interface. Licensed under the GNU General Public License, SNUTS.JS is implemented as a JavaScript project and comprises four main packages: (i) detectors, responsible for the logic of detecting smells; (ii) controllers, responsible for repository analysis and generating CSV files from them; (iii) routes, which configure API routes using the aforementioned controllers; and (iv) services, responsible for repository upload and delivering results in JSON and CSV formats, as well as providing methods for working with Abstract Syntax Trees (AST). SNUTS.JS can analyze public repositories and export results to CSV. It includes an API for programmatic analyses that facilitates integration with CI / CD pipelines.

SNUTS.JS improves the adoption of test smell detection by providing an easy-to-integrate and comprehensive tool. Developed using web technologies, the tool features an API created with the Fastify microframework. This API facilitates integration with various web solutions by providing results in the widely used JSON (JavaScript Object Notation) format. Detailed documentation is available through the web interface with instructions and usage examples. Currently, SNUTS.JS identifies the following test smells, which have been adapted to the context of the JavaScript programming language: Anonymous Test, Comments Only Test, Overcommented, General Fixture, Transcripting Test, and Sensitive Equality. In addition, we defined a new smell called Test Without Description, which consists of a test case lacking descriptive text.

SNUTS.JS can identify key JavaScript test file patterns within a repository and analyze the test code using AST. Seven standards are recognized by SNUTS.JS, which are listed in Table 1. Additionally, SNUTS.JS can analyze unit test files that follow the patterns of the most commonly used testing libraries in the JavaScript ecosystem, such as Jest[1] and Jasmine[2]. The API documentation, created using the Swagger library, can be accessed using the /documentation route in its URL.

Test File Standards
`**/*.test.js` | `**/*.tests.js` | `**/*.spec.js`
`**/*.specs.js` | `**/*test_*.js` | `**/*test-*.js`
`**/*Spec*.js`

Table 1: Test File Standards

The tool analysis process consists of several steps, as exemplified in Figure 1. **Data input** is the stage where the initial data required for the analysis is provided. It is necessary to pass the repository URL in the request body. The repository must be public to be downloaded by the tool. Additionally, the API has an optional parameter called *hasTestSmell* which, when sent along with the repository URL, returns only the files where test smells were found. Therefore, a POST request is made by sending this data.

Next, in the **Project Analysis** step, the repository is downloaded and a search is performed for all unit test files present in the project. Furthermore, the source code is transformed into an **AST (Abstract Syntax Tree)** using the Babel library, which performs code transpilation, making it compatible with older versions of browsers or environments that support the latest language features, facilitating the process of checking for the presence of test smells. Finally, detection functions are used to find possible test smells. Therefore, in the **Data Output** step, the API returns a JSON as the response to the request. The following code snippet shows an example of the JSON file. In this example, the API returns a JSON file with the data, which primarily contains the path of the analyzed file, the *type* of the detected smell, and the *start* and *end lines* of the test block where the smell was found. Additionally, the *info* section refers to the number of individual test blocks defined by "it" or "test" (*itCount*) and the number of test description blocks defined by *describeCount*.

```
{
  "file": "chartjs/Chart/test/specs/scale.logarithmic.tests.js",
  "type": "AnonymousTest",
  "smells": [
    {
      "startLine": 8,
      "endLine": 12
    }
  ],
  "info": {
    "itCount": 27,
    "describeCount": 8
  }
}
```

3.1 Usage Example

This section presents concrete examples of the SNUTS.JS usage. As an API, SNUTS.JS offers a broad range of functionalities, and we next explore several usage scenarios to highlight its adaptability and usefulness in various applications.

Usage With SNUTS.js Frontend. To provide a better user experience with SNUTS.JS, a frontend application has been designed and developed utilizing the core of SNUTS.JS, namely, the API, which can be integrated with various other tools, as illustrated in Figure 2. To use the tool, the user needs to enter the public repository to be analyzed in the search field. Then, upon clicking the search button, a POST request is made to the API, specifying this repository.

[1]https://jestjs.io/
[2]https://jasmine.github.io/

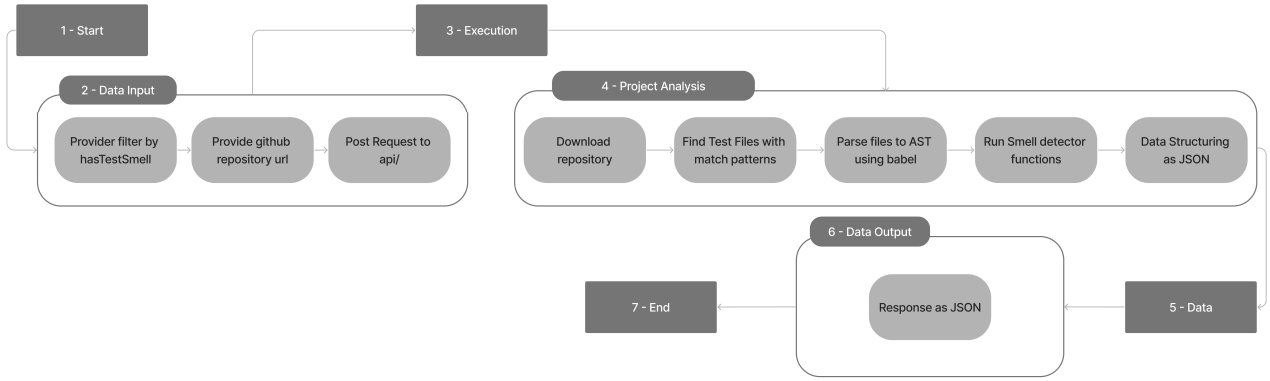

Figure 1: Schematic overview of the SNUTS.js

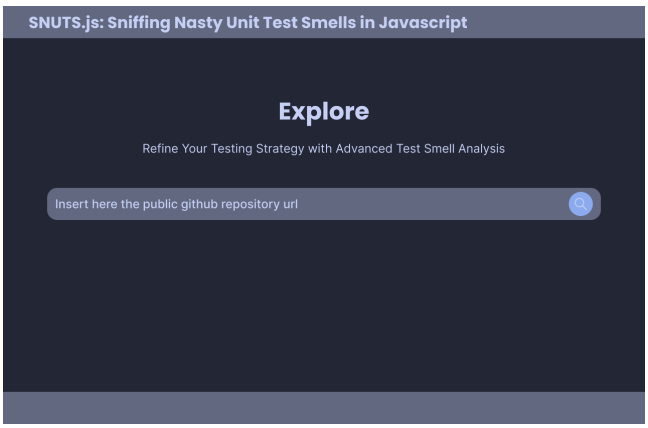

Figure 2: SNUTS.js Frontend

Figure 3: result found example

Following this, the frontend anticipates the arrival of the response, transitioning to a waiting state.

Next, upon successful completion, all files in which test smells have been identified are listed. Additionally, there is the option to generate a CSV table containing the project name, file name, the type of identified smell, and the location of the smell in that file, i.e., their respective start and end lines. Finally, a count of "Describe"

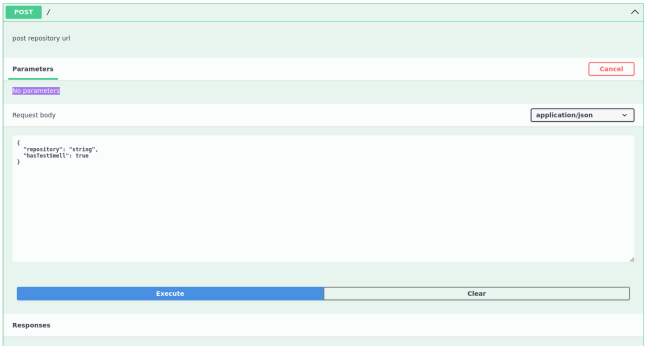

Figure 4: Request example

blocks and "it" or "test" blocks is performed. An example of the analysis result is shown in Figure 3

Usage With SNUTS.js API. To aid the integration of the API with other tools, a documentation of the API was created using the Swagger library. This simplifies the integration process, as the documentation includes response examples and the parameters required to use the API. The API documentation can be accessed by appending "/documentation" to the end of the URL. In this way, developers who need to use the API will find it easy to understand the application routes and the required parameters without accessing the source code of SNUTS.js. Consequently, they can make requests using Swagger, as demonstrated in Figure 4.

4 EVALUATION

This section describes the preliminary evaluation of the tool. To perform the evaluation, we selected three JavaScript projects based on specific criteria, including popularity (one thousand stars or more), use of unit testing libraries such as Jest, Jasmine, and Vitest, and version releases in the last six months on GitHub. The selected projects are presented in Table 2.

Also, Table 3 shows the number of test files in the projects and the occurrence of test smells per file. The dispersion of test smells among the test files of the three selected projects was significant, with the Play Canvas project having 7.69% of test files with at

Name	Repositories	Stars
ChartJS	https://github.com/chartjs/Chart.js	64K
PlayCanvas	https://github.com/playcanvas/engine	9.3K
OpenLayers	https://github.com/openlayers/openlayers	11.1K

Table 2: Repositories used in the study

least one "test smell", the Open Layers project with 30.29%, and the Chart.js project with 14.55%.

Project	Smelly Test Files	Total test files	Dispersal (%)
Play Canvas	6	78	7.69%
Open Layers	93	307	30.29%
Chart.js	8	55	14.55%

Table 3: Dispersion of test smells

Initially, two researchers manually examined the projects listed in Table 2. They performed this analysis in isolation and then conducted a meeting to check for inconsistencies. This comprehensive analysis involved a total of 153 test files, which we make publicly available as a dataset of Test Smells in JavaScript on Zenodo.

The manual analysis identified 285 test smells: 170 Anonymous Tests, 89 General Fixtures, and 26 Overcommented smells. Both researchers agreed on the final list of smells. However, four classes of test smells were not present in the selected projects: Test without Description, Comments Only Test, Sensitivity Equality, and Transcripting Test smells. In detail, for the ChartJs project, we found 14 instances of Anonymous Tests, 8 Overcommented, and only 1 General Fixture. Regarding the Play Canvas, we found 11 Anonymous Tests, 10 Overcommented, and none of General Fixture. Finally, in the Open Layers, we identified 145 Anonymous Tests, 88 General Fixture, and 8 Overcommented smells, as Table 4 shows.

Project	TWD	OT	SE	AT	GF	COT	TT
Play Canvas	0	10	0	11	0	0	0
Open Layers	0	8	0	145	88	0	0
Chart.js	0	8	0	14	1	0	0
TOTAL	0	26	0	170	89	0	0

Table 4: Test Smell Analysis

It is important to highlight that the adaptation of test smells for JavaScript was a process grounded in empirical methods due to the lack of precise definitions in the existing literature. In this case, specific acceptance thresholds were established for the Overcommented and Anonymous Test categories. Hence, an Overcommented test case was delineated by exceeding a set threshold of 5 comments per test, and an Anonymous Test was detected for tests in which its description comprised fewer than two words. Importantly, users retain the flexibility to customize these thresholds to suit their requirements.

After building the dataset, we analyzed the same projects with the SNUTS.js tool. We used the same thresholds defined in the manual analysis. Remarkably, the SNUTS.js results mirrored those obtained manually, exactly all the test smells were detected. This affirms the tool's efficacy in identifying smells within the mentioned repositories, while also offering an intuitive interface for users, thereby simplifying adoption by researchers and devs.

5 THREATS TO VALIDITY

In our study, several validity concerns arose during the construction and validation of our test oracle. These concerns were addressed through careful consideration and methodology adjustments.

Internal Validity. Potential disparities in researchers' analyses during the manual construction of the dataset were acknowledged. Disagreements were collectively resolved to ensure consistency.

External Validity. While our results offer valuable insights, they may not apply universally. Future research could broaden the scope to enhance generalizability.

Conclusion Validity. In our preliminary evaluation, we found three out of seven test smell types detected by SNUTS.js. However, the absence of the other four types of smells limited the validation scope, suggesting the need for broader data inclusion.

Construct Validity. The dataset construction involved two programmers with testing experience, enhancing thoroughness. However, dependency on their expertise poses a potential threat.

6 CONCLUSIONS

SNUTS.JS is a comprehensive tool designed to analyze and improve JavaScript unit test code by detecting test smells. Currently, it identifies six specific types of test smells, adapted from existing literature to fit the JavaScript programming environment. Emphasizing usability and ease of integration, SNUTS.JS offers a robust API documented with Swagger and a user-friendly frontend application for detailed repository analysis. Looking ahead, the core functionality of SNUTS.JS can be extended to create plugins for IDEs and other code quality analysis tools, thus enhancing its utility for the JavaScript development community. We also intend to extend both the tool to identify more smells and the dataset to include more projects. We intend to evaluate the usability of the tool with developers in real production projects and conduct further studies to verify the relevance of test smells in the JavaScript language.

ARTEFACT AVAILABILITY

The supplementary material is available on Zenodo: https://doi.org/10.5281/zenodo.11475024. The SNUTS.js frontend is available on GitHub at https://github.com/Jhonatanmizu/snuts.js_frontend and the SNUTS.js API is available at https://github.com/Jhonatanmizu/SNUTS.js

ACKNOWLEDGMENTS

This work was partially supported by UEFS-AUXPPG 2023 and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001; PROAP 2023 grants; CNPq grant 403361/2023-0 and 140587/2024-1.

REFERENCES

[1] Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mohamed Wiem Mkaouer, Ali Ouni, Christian D. Newman, Abdullatif Ghallab, and Stephanie Ludi. 2021. Test Smell Detection Tools: A Systematic Mapping Study. In *Proceedings of the 25th International Conference on Evaluation*

and Assessment in Software Engineering (Trondheim, Norway) *(EASE '21)*. Association for Computing Machinery, New York, NY, USA, 170–180. https://doi.org/10.1145/3463274.3463335

[2] Maurício Aniche. 2022. *Effective Software Testing: A developer's guide*. Simon and Schuster.

[3] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. Are test smells really harmful? an empirical study. *Empirical Software Engineering* 20 (2015), 1052–1094.

[4] Arie Deursen, Leon M.F. Moonen, A. Bergh, and Gerard Kok. 2001. Refactoring Test Code. In *Refactoring Test Code*. CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands.

[5] Daniel Fernandes, Ivan Machado, and Rita Maciel. 2022. TEMPY: Test Smell Detector for Python. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering* (<conf-loc>, <city>Virtual Event</city>, <country>Brazil</country>, </conf-loc>) *(SBES '22)*. Association for Computing Machinery, New York, NY, USA, 214–219. https://doi.org/10.1145/3555228.3555280

[6] Vahid Garousi and Barış Küçük. 2018. Smells in software test code: A survey of knowledge in industry and academia. *Journal of Systems and Software* 138 (2018), 52–81. https://doi.org/10.1016/j.jss.2017.12.013

[7] Rahul Gopinath, Carlos Jensen, and Alex Groce. 2014. Code Coverage for Suite Evaluation by Developers. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. ACM, New York, NY, USA, 72–82. https://doi.org/10.1145/2568225.2568278

[8] Giovanni Grano, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Harald C Gall. 2019. Scented since the beginning: On the diffuseness of test smells in automatically generated test code. *Journal of Systems and Software* 156 (2019), 312–327.

[9] Giovanni Grano, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Harald C Gall. 2019. Scented since the beginning: On the diffuseness of test smells in automatically generated test code. *Journal of Systems and Software* 156 (2019), 312–327.

[10] Dayne Guerra Calle, Julien Delplanque, and Stéphane Ducasse. 2019. Exposing Test Analysis Results with DrTests. In *International Workshop on Smalltalk Technologies*. Cologne, Germany. https://hal.inria.fr/hal-02404040

[11] Dalton Jorge, Patricia Machado, and Wilkerson Andrade. 2021. Investigating Test Smells in JavaScript Test Code. In *Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing* (Joinville, Brazil) *(SAST '21)*. Association for Computing Machinery, New York, NY, USA, 36–45. https://doi.org/10.1145/3482909.3482915

[12] Gerard Meszaros, Shaun M. Smith, and Jennitta Andrea. 2003. The Test Automation Manifesto. In *Extreme Programming and Agile Methods - XP/Agile Universe 2003*, Frank Maurer and Don Wells (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg.

[13] F. Palomba, D. Di Nucci, A. Panichella, R. Oliveto, and A. De Lucia. 2016. On the Diffusion of Test Smells in Automatically Generated Test Code: An Empirical Study. In *2016 IEEE/ACM 9th International Workshop on Search-Based Software Testing (SBST)*. IEEE, Austin, TX, United States.

[14] Anthony Peruma, Khalid Almalki, Christian D. Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2020. tsDetect: an open source test smells detection tool. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1650–1654. https://doi.org/10.1145/3368089.3417921

[15] Roger Pressman. 2016. *Software Engineering: A Practitioner's Approach* (8 ed.). McGraw-Hill, Inc., USA.

[16] Railana Santana, Luana Martins, Larissa Rocha, Tássio Virgínio, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. RAIDE: a tool for Assertion Roulette and Duplicate Assert identification and refactoring. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (<conf-loc>, <city>Natal</city>, <country>Brazil</country>, </conf-loc>) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 374–379. https://doi.org/10.1145/3422392.3422510

[17] Andrew Costa Silva. 2022. Identificação e Caracterização de Test Smells em JavaScript. *Instituto de Ciencias Exatas e Informática - Pontifícia Universidade* 138 (2022), 52–81. http://bib.pucminas.br:8080/pergamumweb/vinculos/000014/000014ce.pdf

[18] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli. 2018. On the Relation of Test Smells to Software Code Quality. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 1–12. https://doi.org/10.1109/ICSME.2018.00010

[19] Tássio Virgínio, Luana Almeida Martins, Larissa Rocha Soares, Railana Santana, Heitor Costa, and Ivan Machado. 2020. An empirical study of automatically-generated tests from the perspective of test smells. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (<conf-loc>, <city>Natal</city>, <country>Brazil</country>, </conf-loc>) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 92–96. https://doi.org/10.1145/3422392.3422412

[20] Tássio Virgínio, Luana Martins, Railana Santana, Adriana Cruz, Larissa Rocha, Heitor Costa, and Ivan Machado. 2021. On the test smells detection: an empirical study on the JNose Test accuracy. *Journal of Software Engineering Research and Development* 9, 1 (Sep. 2021), 8:1 – 8:14. https://doi.org/10.5753/jserd.2021.1893

[21] Tongjie Wang, Yaroslav Golubev, Oleg Smirnov, Jiawei Li, Timofey Bryksin, and Iftekhar Ahmed. 2022. PyNose: a test smell detector for python. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering* (Melbourne, Australia) *(ASE '21)*. IEEE Press, 593–605. https://doi.org/10.1109/ASE51524.2021.9678615

[22] Vahid Garousi Yusifoğlu, Yasaman Amannejad, and Aysu Betin Can. 2015. Software test-code engineering: A systematic mapping. *Information and Software Technology* 58 (2015), 123 – 147. https://doi.org/10.1016/j.infsof.2014.06.009