

Quality Assurance in Software Development: The Impact of Unit Test Automation

Ariful Islam¹, Fazle Elahi Fahim¹, Abu Sakib¹, Fernaz Narin Nur¹, A. H. M. Saiful Islam¹

¹Department of Computer Science and Engineering, Notre Dame University Bangladesh, Dhaka, Bangladesh

Article Info

Article history:

Received month dd, yyyy

Revised month dd, yyyy

Accepted month dd, yyyy

Keywords:

Automating Manual Unit Tests

Unit tests

Developer Freedom in Test

Parameters

Accurate Unit Testing

ABSTRACT

Automatic unit testing plays a pivotal role in modern software development, offering numerous advantages for ensuring code quality, reliability, and maintainability. A key benefit is the early detection of defects in the development process. However, challenges arise, such as the upfront time and effort required to establish comprehensive test suites. Developers often grapple with crafting tests that cover all possible scenarios, leading to potential gaps in test coverage. To address these challenges, we pose two essential questions. First, we explore the extent to which developers benefit when given the freedom to choose function parameter values. Second, we investigate the time savings achievable by automating traditional manual unit tests, where developers manually select parameter values. In this paper, we present a prototype that addresses these questions, enabling users to test their code with chosen inputs. Through the development of this prototype, we conducted a user study, revealing its significant advantages over conventional automatic unit test tools like Randoop, which employs random values to test Java code. Our prototype not only reduces testing time compared to manual tests but also ensures more accurate unit testing. The insights gained from our study suggest that developers can derive substantial benefits from the flexibility our prototype provides, leading to improved efficiency and precision in unit testing.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Fernaz Narin Nur

Department of Computer Science and Engineering, Notre Dame University Bangladesh

Dhaka, Bangladesh

Email: fernaznur@gmail.com

1. INTRODUCTION

Software testing is an essential method used to validate that a system meets both its functional and non-functional requirements [1]. Various testing methods can be broadly classified into manual and automated approaches. In manual testing, testers execute programs with predetermined test data and assess whether the results align with their expectations. In contrast, automated tests involve the creation of code or scripts by testers or developers, which are typically integrated directly into the ongoing development process [2].

The significance of different testing methods extends to both customers and developers. Testing not only demonstrates that a product can deliver its promised features but also verifies its ability to meet performance, security, and availability requirements. Additionally, testing plays a critical role in identifying defects that could lead to suboptimal user experiences, unpredictable system behavior, or even complete system failures [1]. Some defects have the potential to result in severe vulnerabilities, impacting both customers and product owners. A notable example is the Ariane 5 rocket's maiden flight, which ended in a catastrophic failure just 40

seconds after liftoff. The failure resulted from a software error where a 64-bit floating-point number related to the rocket's horizontal velocity was incorrectly converted to a 16-bit signed integer, causing an overflow and system failure. Adequate unit testing and boundary checks could have identified this issue before launch. Another significant incident is the Equifax Data Breach, exposing the sensitive data of millions of Americans and leading to Equifax incurring 45.45 million dollars in settlements [3]. These examples underscore the critical role of testing in preventing costly and potentially disastrous consequences, highlighting the need for effective testing methodologies. In this paper, we contribute to this discourse by embarking on a journey to explore the realm of automatic unit testing in modern software development, with a specific focus on its advantages and potential challenges.

Our investigation led us to formulate two key questions: the impact of developer freedom in choosing function parameter values and the time efficiency gained by automating traditional manual unit tests. Through the development of a Python prototype, we provided developers with a flexible environment to test their code with dynamically chosen inputs. The subsequent user study demonstrated a positive response, highlighting the prototype's adaptability, user-friendliness, and potential as a powerful tool for precise unit testing. In addressing the time efficiency aspect, we conducted a comprehensive analysis of manual versus automated unit testing. The results unequivocally showed that manual testing, while essential, is time-consuming, often discouraging developers from conducting thorough unit tests. Conversely, our automated testing prototype showcased a significant reduction in testing time, making unit testing more feasible and attractive for developers. The visual representation of testing time trends vividly illustrated the superiority of automated testing over its manual counterpart. The negligible time required for automated testing, as depicted in the trending lines, emphasized its potential to revolutionize the landscape of unit testing practices. The development of a fully functioning application capable of supporting test generation across various programming languages poses significant challenges.

The remainder of this paper is structured as follows: Section 2. reviews related work in the field of automated unit testing, providing context for our research. In Section 3., we detail the methodology employed in creating the Python prototype and conducting the user study. Section 4. presents the results and analysis of our investigation, comparing automated testing to manual testing in terms of time efficiency. Finally, Section 5. offers concluding remarks and outlines avenues for future research in the realm of automatic unit testing.

2. RELATED WORK

Efficient localization of faults in software systems is an ongoing challenge, especially when functions become more complex, making the process time-consuming. In order to address this, W. Eric Wong and Vidroha Debroy (2022) carried out an extensive analysis of fault localization approaches, pointing out problems and suggesting fixes [4]. To improve efficiency and effectiveness in complex software environments, they support scalable approaches and tools that help testers maintain unit tests for large systems [4]. In order to address issues with code selection and system evolution, Mårten Björkman and Jonathan Bergqvist (2022) presented a prototype unit testing application designed for JavaScript [5]. Their user study emphasized the ease of use of the application and emphasized the necessity of open-source collaboration and comprehensive testing applications [5].

While there are issues with code coverage and test sensitivity, Sina Shamshi et al. (2015) evaluated the efficacy of tools such as Randoop, EvoSuite, and Agitar in the field of automated unit test generation and found that they are useful for fault detection citeb5. Runeson (2006) conducted a study that examined unit testing practices in the industry. The study found that while unit testing is widely used, there are differences in its implementation and obstacles like time constraints and maintenance issues [6]. Moreover, the H. Krasner (2020) [3] report on the U.S. cost of subpar software quality confirms the practical ramifications of software flaws.

Furthermore, the research conducted by Shamshi et al. (2015) [7] offers a thorough empirical evaluation of automatic unit test generation tools, assessing how well they identify actual defects. In the meantime, Per Runeson's industry study (2006) [6] reveals the difficulties that businesses face while also illuminating the extensive usage of unit testing.

To sum up, while earlier research has established the foundation for automatic test generation, unit testing applications, and fault localization, our paper builds on this work by offering useful suggestions for improving unit testing procedures. Our prototype's blend of efficient automated testing and developer autonomy

opens up a promising path for further improvements in software testing techniques.

3. METHODOLOGY

The methodology employed in this study is designed to comprehensively address the research questions concerning the impact of developer freedom in choosing function parameter values and the time efficiency gained by automating traditional manual unit tests. Our approach encompasses the development of a Python-based prototype and the execution of a user study to evaluate its effectiveness. The methodology unfolds in several stages, starting with the design and implementation of the prototype that allows developers to exercise their code with dynamically chosen inputs. The flowchart in Fig. 1 shows the proposed methodology. First, we

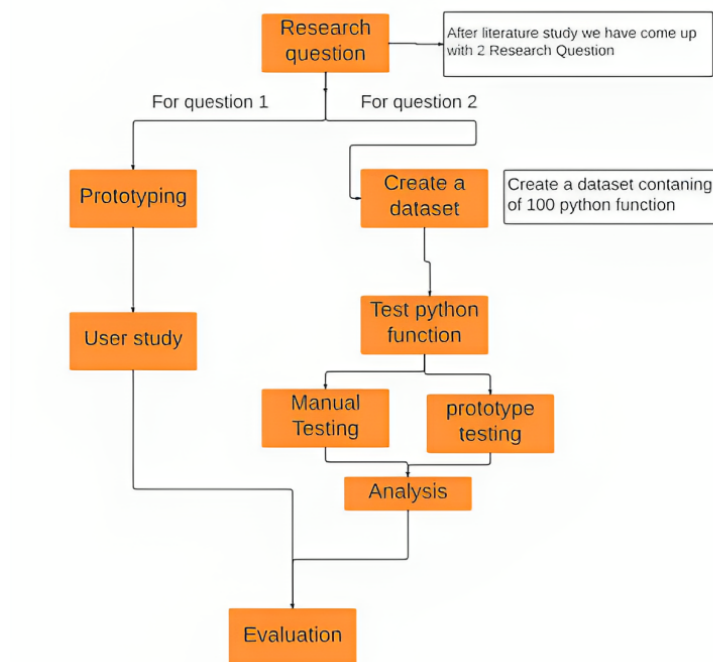


Figure 1. Workflow of RESEARCH PROCESS.

have to conduct our literature study to gain some knowledge about the unit tests after that we come up with 2 questions.

1. If the developer has the freedom to choose a function parameter value how much it would be helpful?
2. In manual testing developer also can choose parameter values. so How much time is reduced if we automate the traditional manual unit test?

3.1. Prototyping

To address the research questions, we developed a Python-based prototype for automatic unit testing. The prototype enables users to input their Python functions and dynamically set parameter values for comprehensive testing.

3.2. User Study

The subsequent phase involves a qualitative user study. Multiple interviews were conducted, allowing participants to interact with the testing prototype and provide insights into their experience. This approach, chosen over surveys, aimed to offer users a hands-on understanding of the prototype's functionality. To recruit participants, we approached three software developers who were invited to test any function of their choice, emphasizing complete freedom in function selection and test creation.

3.3. Dataset creation

To evaluate the prototype's performance, we collected a dataset comprising 100 real-world Python functions sourced from online repositories. This dataset serves as a benchmark for assessing the prototype's efficiency compared to manual testing.

3.4. Evaluation

The final stage involves the evaluation phase, where we seek to answer the research questions. The impact of developer freedom is addressed through prototype development and user study. Additionally, the efficiency of automated testing is assessed using the dataset of 100 Python functions. This holistic approach ensures a comprehensive understanding of the implications and potential improvements for both the research process and the prototype.

3.5. Code Analysis and Test Generation

Testers submit Python code to the prototype, covering a range of functionalities from arithmetic and string manipulation to logical operations and data processing. The Flask app facilitates interactive testing and refinement of user-provided code, accommodating various user-defined functions.

3.6. Analysis of a Python prototype

Our web application, built with Flask, simplifies the testing and automatic generation of unit tests for Python functions. Leveraging Flask's flexibility, the application includes components such as routes, functions, HTML templates, CSS styles, and JavaScript. The core functionality involves parsing user-provided Python functions, extracting essential details, and generating unit test code. Property-based random testing enhances function robustness, and a clean user interface is provided through HTML templates styled with CSS. The application, initiated through a local web server, ensures easy user access.

3.7. Analysis of how the prototype tester code

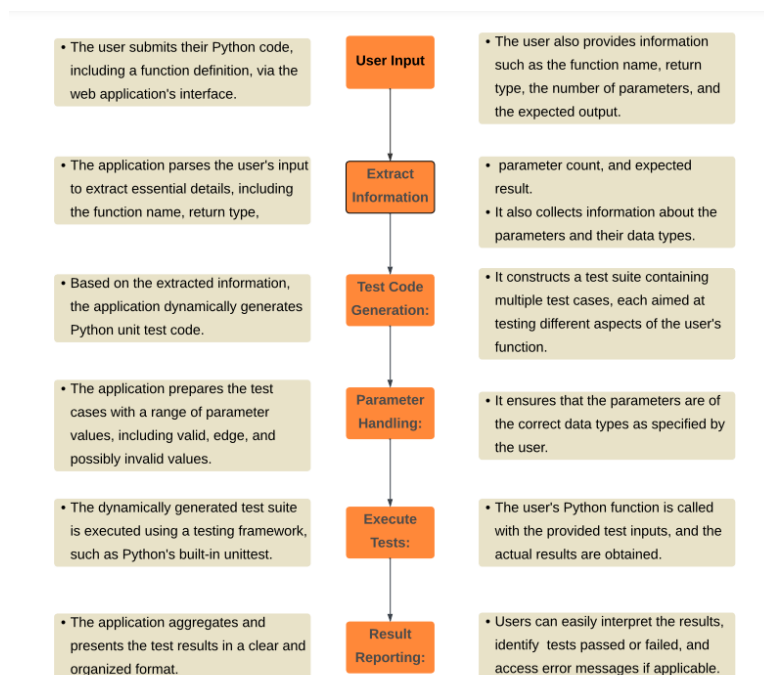


Figure 2. Workflow of prototype code testing

Fig. 2 illustrates the flowchart of our prototype code testing. Users submit Python code, and the application extracts essential information, dynamically generating unit test code covering various scenarios. The generated test code is executed, comparing actual and expected results to determine the user's function behavior. This comprehensive process ensures the intended functionality of the user's code.

4. RESULTS AND DISCUSSION

The results of two main questions of investigation, namely “developer has the freedom to choose function parameter value and how much it would be helpful”, and “In manual testing developer also can choose parameter values. So how much time is reduced if we automate the traditional manual unit test?” are analyzed. These answers will shed light on unit test fields.

4.1. Answer of our First question

Through user study, we have got the answer to our first question. The result was positive, users were satisfied with our prototype function and result.

User feedback summarize: Feedback from Three interview participants indicates that three have provided valuable insights about the prototype for unit testing. Users commend the platform’s flexibility, allowing customization of input parameters and expected results, enhancing precision in testing. The support for various data types and a user-friendly interface simplify the testing process, with clear error reporting aiding debugging efforts. While users appreciate the interactive and user-driven testing approach, there is room for improvement, such as expanding data type support and adding advanced testing features. Overall, with ongoing development and user feedback, the prototype holds the potential to become a powerful tool for unit testing, offering testers effective ways to verify code correctness.

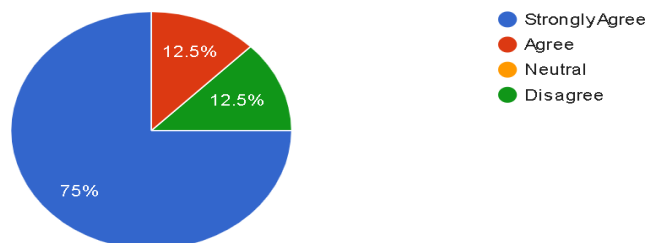


Figure 3. Result of user study

4.2. Answer of our last question

we have collected 100 Python functions to test and compare the time that each test and how much time is needed. In Fig 3 for each function test it almost took 5 minutes and that’s too much time to just test one function. Because of this reason, developers do not want to unit-test.

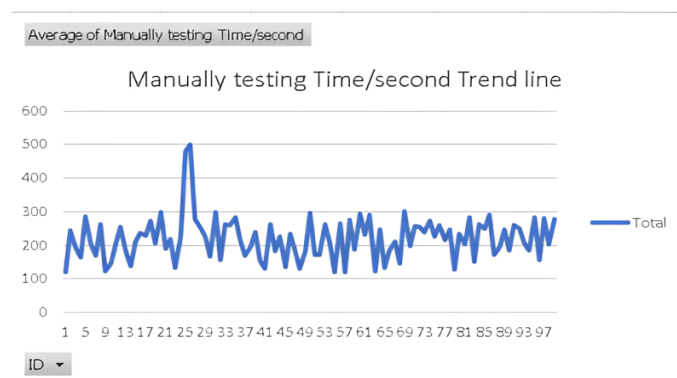


Figure 4. Manual testing time trending line.

To overcome this limitation we have developed a prototype for automating the unit test and it took less time to test one function. In Fig 4 for each function test it almost took 5 minutes and that’s too much time to just test one function. Because of this reason, developers do not want to unit test.

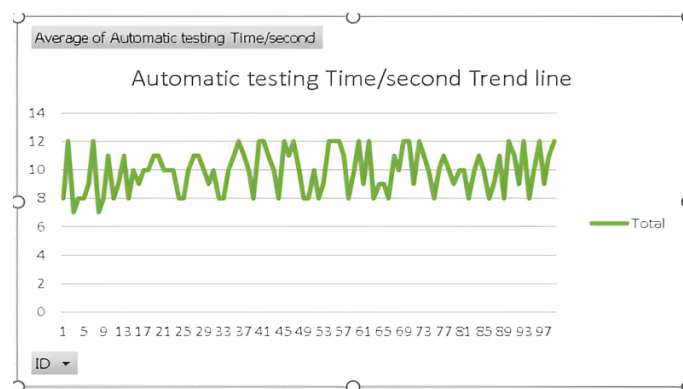


Figure 5. Automating testing time trending line.

And now we have compared both testing trending lines to see which one is better and also to have a better look. In Fig 5 we can see that the the blue trending line is for automatic testing and the yellow one is for manual testing and if we compare both the blue trending line almost took negligible time compared to the yellow line.

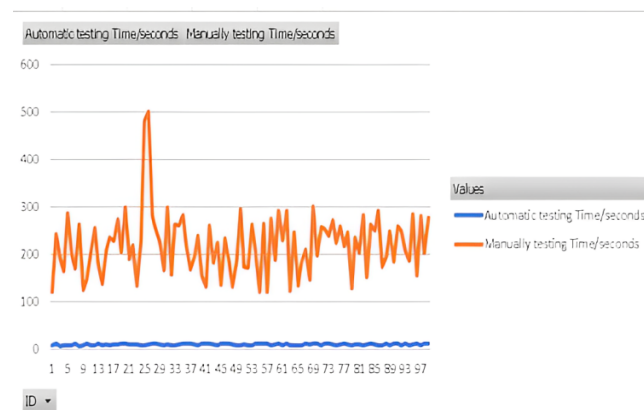


Figure 6. Manual VS Automating testing time trending line.

5. CONCLUSION

In our exploration of automatic unit testing within modern software development, our primary focus was to examine the advantages and challenges inherent in this approach. The formulation of two critical questions led to the development of a Python prototype, offering developers a flexible testing environment with dynamically chosen inputs. A subsequent user study underscored positive responses, highlighting the prototype's adaptability, user-friendliness, and its potential as a potent tool for precise unit testing. The comprehensive analysis of time efficiency between manual and automated unit testing revealed the inherent time-consuming nature of manual testing, often acting as a deterrent to thorough unit tests. Conversely, our automated testing prototype showcased a significant reduction in testing time, rendering unit testing more feasible and appealing for developers. The visual representation of testing time trends emphasized the superiority of automated testing, hinting at its potential to revolutionize unit testing practices.

However, the journey encountered challenges in extending the fully functioning application's capability to support test generation across various programming languages. Presently confined to dynamic input, exception expectations, and dynamic output, there's a need for expansion to support lists of strings. Efforts to extend the prototype to other languages faced obstacles such as API key pricing and unsuccessful Python library usage. Estimating the time and effort required for a comprehensive prototype in this category remains elusive, even for Python alone. Despite these challenges, we maintain that overcoming them is not an insurmountable task. For future studies, enhancing the evaluation process with a more advanced product and comprehensive user studies involving participants with broader development and testing experiences will be crucial. Extensive

testing of the prototype in real-world scenarios could yield more accurate and valuable insights, affirming the promising potential of automated unit testing in shaping the landscape of software development practices.

REFERENCES

- [1] I. Sommerville, "Software engineering 9th edition," *ISBN-10*, vol. 137035152, p. 18, 2011.
- [2] A. Fontes and G. Gay, "The integration of machine learning into automated test generation: A systematic literature review," *arXiv preprint arXiv:2206.10210*, 2022.
- [3] H. Krasner, "The cost of poor software quality in the us: A 2020 report," *Proc. Consortium Inf. Softw. QualityTM (CISQTM)*, pp. 1–46, 2021.
- [4] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.
- [5] M. Björkman and J. Bergqvist, "Creating a unit testing application prototype for javascript," 2022.
- [6] P. Runeson, "A survey of unit testing practices," *IEEE software*, vol. 23, no. 4, pp. 22–29, 2006.
- [7] S. Shamshiri, R. Just, J. M. Rojas, G. Fraser, P. McMinn, and A. Arcuri, "Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 201–211.

BIOGRAPHIES OF AUTHORS



Ariful Islam is a current undergraduate student in the Department of Computer Science and Engineering at Notre Dame University Bangladesh. He harbors a keen interest in research areas such as Artificial Intelligence, Machine Learning, and the Internet of Things. He can be reached via email at ariful81848@gmail.com.






Fazle Elahi Fahim is a dedicated undergraduate student majoring in Computer Science and Engineering at Notre Dame University Bangladesh. His primary research interests include Software Engineering, Artificial Intelligence (AI), and Machine Learning (ML). He devotes his entire time to developing knowledge and skills in Application Development, alongside his academic pursuits. he can be contacted by email: fahim201120030@student.ndub.edu.bd





Abu Sakib is a dedicated undergraduate student majoring in Computer Science and Engineering at Notre Dame University Bangladesh. His primary research interests include Software Engineering, and Machine Learning (ML). he devotes his entire time to developing knowledge and skills in Application Development, alongside his academic pursuits. he can be contacted by email: abusakib201120014@student.ndub.edu.bd



Dr. Fernaz Narin Nur    received her B.Sc. (Hons) from the Department of CSE, Jahangirnagar University, Bangladesh in 2008 and M.S. from Institute of Information Technology, University of Dhaka, Bangladesh in 2010. She obtained her Ph.D. degree from the Department of CSE, University of Dhaka, Bangladesh in 2017. In her professional life, she is an Associate Professor at the Notre Dame University Bangladesh (NDUB) in the Dept. of CSE. She is a passionate researcher in the fields of Wireless Sensor Network, Directional Wireless Sensor Network, Ad Hoc Networks, MAC Protocols, Cloud Computing, Internet of Things, Machine Learning etc. She has published a good number of research papers in international conferences and journals. She is a widely-traveled personality and a novice golfer at Army Golf Club. She is a member of Green Networking Research group, IEEE, Internet Society, Bangladesh Women in IT.



A. H. M. Saiful Islam   A. H. M. Saiful Islam, a Bangladesh native, earned a B.Sc. in Computer Science and Engineering with a focus on Bioinformatics from Khulna University. His M.Sc. in CSE from North South University included work published in ICIEV 2013. In his professional life, currently he is a Professor at the Notre Dame University Bangladesh (NDUB) in the Dept. of CSE. With research interests in Programming, Discrete Mathematics, IT Organization, Software Engineering, and Theory of Computation. He is also interested in Wireless Computing and Mobile Computing.