A Comprehensive Analysis of Software Testing for Intrinsically Challenging Systems

(Technical Paper)

An Investigation on the Societal Impact of Untested Software

(STS Paper)

A Thesis Prospectus Submitted to the

Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements of the Degree
Bachelor of Science, School of Engineering

Jassiel Mendoza

Fall 2023

Introduction

Any product released to the public undergoes testing to ensure it meets technical standards and works within given constraints. In this sense, software is no different, except that, unlike most physical products, which undergo testing before reaching customers and typically offer warranties, software products are in a perpetual state of testing, known as maintenance, and can never be entirely rid of defects. This phase is incredibly vital; a substantial portion of the budget for a software product or service is allocated to maintaining it post-release, and even more so if critical defects are detected in this stage. This is further complicated by the rise of agile and lean development methods in companies, which emphasize creating a minimal viable product, launching it, and reiterating based on feedback. Although this methodology makes sense for software development as it minimizes risk, it inherently glosses over testing. And considering the resource-intensive nature of testing and maintaining software, companies may be prone to cut corners or overlook details in this process, whether deliberately or due to various constraints, opening the door for adverse consequences. However, this is offset by the fact that there are established methods to test and verify traditional software's functionality. Although there is no such thing as "enough testing," these testing techniques ensure a piece of software works as intended and can uncover unwanted behavior or potential vulnerabilities. So much so that the entire testing process can be automated if an exhaustive testing oracle is developed alongside the software itself. This is not the case for newer, complex systems like machine learning algorithms and mobile applications that we are developing and using today since we do not understand how to analyze and assess these systems properly. As this type of software becomes increasingly integrated into security and safety-critical processes, there is a growing need for a thorough understanding of these systems and the challenges they present. This understanding will help us develop more effective testing methods, reducing the inherent risks and biases associated with these systems.

Technical Discussion

Software testing is a critical step in the software development process that is meant to identify and address defects and issues before reaching end-users, intending to create a high-quality product ready for release. As with any testing, this stage acts as quality assurance that the software meets the client's standards and requirements. It is also pivotal for identifying bugs as the cost of rectifying them escalates significantly the further into development they are found. Along the same lines, assessing and mitigating the potential risks involved with software is crucial in industries that must adhere to strict regulatory guidelines, such as safety-critical applications like aviation or security-critical applications like finance. All exhibit the importance of software testing to verify the functionality and integrity, but also to improve the overall user experience and satisfaction by delivering a more robust and reliable product.

Testing is and should be a systematic process based on the software requirements and specifications agreed upon by the client and the developers. Typically, this is achieved by developing a proper test oracle, which helps objectively determine if a software application is functioning correctly by defining the correct and expected behavior. However, this means that adequate test cases are outlined with expected outcomes and set up in such a manner that simulates real-world conditions in which the software will operate, including the hardware and network configurations. This is done to facilitate the replication of tests that uncover any defects or bugs, which is vital when analyzing and understanding the root cause of the issue to resolve it. These properties of traditional software testing are problematic for more intricate software like machine learning models or mobile applications, however. These have intrinsic properties that directly oppose the above guidelines for traditional testing, presenting challenges for the comprehensive testing of these systems. For instance, the non-deterministic nature of these systems is a major problem as it makes it difficult to specify the expected behavior and output when testing, making it hard to create deterministic test cases to evaluate the model's behavior correctly. This problem is exacerbated among mobile applications due to the device fragmentation problem. The wide range of mobile devices with varying screen sizes, resolutions, operating systems, and hardware capabilities makes it extremely difficult to test and account for all instances of a mobile application. We can go even deeper with this issue as the nature of mobile devices means that the same mobile applications will be used in changing network conditions, whether due to poor network connectivity or a device is moving through various cell tower networks for example. On top of the unpredictable nature of user interruptions, such as phone calls, notifications, or other background processes, all of which consume resources, are all managed differently by each device. How can we account for all these variables systematically and affordably? These large input spaces make it impractical and nearly impossible to model and account for every possibility, reducing the ability to identify failure-revealing inputs.

The objective of the technical section is to assess the current testing practices in place today for more intricate software, analyzing both their weaknesses and exploring proposed solutions to determine if they address the limitations of the current state-of-the-art techniques. The goal is to provide a comprehensive outlook on the improvements required for developing effective testing techniques by consolidating the data and findings gathered in this process. Overall, this will culminate into a meta-study on the current challenges of evaluating these systems while pointing toward new, promising techniques that address the shortcomings of current testing techniques.

STS Discussion

As mentioned, software testing is a critical process ensuring the user's safety. Validating that software behaves as they would expect and does not contain vulnerabilities that might endanger their personal information or, in some cases, even their own lives is essential. This is the developer's primary responsibility. Still, they also hold the ethical responsibility of ensuring their software does not marginalize groups of people or have any unforeseen consequences. In other words, safety for a developer should mean minimizing risk and biases in their software. Although this might be seen as common sense, this is much harder to implement than it might seem since a combination of inputs or actions might trigger events in an unintuitive, illogical manner, which is why testing is such a pivotal and continuous stage in software development. As history teaches us, even with strict regulations and guidelines to ensure a program is safe and dependable, mistakes still occur, with the more sinister cases linked to profit-driven decisions. Such was the case not too long ago with Boeing's 737 MAX incident, where a new flight control system pitched the nose down into an uncontrollable dive after receiving data from a faulty sensor. This may be an extreme example, but it highlights the importance of testing software properly and extensively to minimize risk in situations where there can be loss of life or to protect private and sensitive information such as medical or financial records.

The reality is that the world's digitization is well underway, and complex software is more prevalent in our everyday lives and will only continue to spread its influence as time goes on. The recent rise of artificial intelligence only proves this, as we have seen dramatic growth in the computational ability of AI and the already extensive list of potential applications will only continue to grow. Undoubtedly, AI and similar systems will prove to be tremendously helpful tools that will revolutionize how things are done. Still, as a relatively new technology in the public's eye, we must acknowledge that we currently do not understand or can even begin to imagine the potential negative implications such systems could have on society. Much like the internet, which was initially created as a resource for scientists to easily share information about their research and findings with other scientists and institutions, it quickly became a resource for bad actors to utilize to further their interests. Thus, cybercrimes emerged as a result. The difference, however, is that not only are the people using the software unpredictable, but the software itself can behave in an unstable manner. As a result, it is imperative that we at least attempt to gain a deeper understanding of the potential ethical ramifications of such technology in the public's hand before opening Pandora's box.

The trolley problem is a classic ethical thought experiment that paints a scenario where a choice has a trade-off between what is good and an acceptable sacrifice. The question has no solution by design and is meant to examine our justification for our moral judgments and our understanding of good and bad. So then, how could a computer with no sense of ethics and morals answer the trolley problem; how would the algorithm of a self-driving vehicle respond, for instance? If faced with this impossible situation, the algorithm would have to decide, but what factors influenced it one way or the other? The problem with this question is that it does not have a solution either, as biases in the algorithm can stem from the developers who wrote it or even the data used to train the algorithm. This is the real issue with complex software like this, as its inherent nature and irremovable biases will never allow us to evaluate that software thoroughly and objectively. Programs we create will always be ethically flawed as long as we remain morally flawed. This is why testing software to the best of our ability becomes exponentially important as the software in our everyday lives continues to become more complex yet feeble. This culminates into the main question of my STS topic: what are the societal implications and potential ramifications of prematurely releasing and depending on software that goes beyond our comprehension? As well as considering prospective strategies a bad actor could use or how they could leverage said systems to endanger the well-being of others. This will be somewhat tightly coupled with my technical discussion as I will also examine if the effort and money needed to develop more efficient testing methods to secure these types of software is worth it in the end or if we are better off employing reactive, bolted-on solutions as is typical in software development.

Research Questions and Methods

As alluded to earlier, the technical portion will be conducted via a meta-study which involves completing a systematic and rigorous process of analyzing multiple studies and academic papers. I will focus on assessing the effectiveness of current software testing practices employed today and comparing this to the proposed effectiveness of the envisioned techniques and practices outlined in the most relevant sources I find. This will be done while simultaneously considering the amount of effort needed to integrate new or improved techniques among other factors such as time and money. To remain as objective as possible, I will also assess and identify any discrepancies and potential publication bias in the papers to not compromise the reliability of the analysis. After all of this, the results will be interpreted and discussed along with any implications associated. Lastly, the findings will be outlined thoroughly in my final report. My STS portion follows a similar approach, but due to its more theoretical and conceptual nature, it will be less restrictive and focus more on outlining the most important steps we need to take to prevent undesirable outcomes based on the technology available to us today.

Conclusion

Testing for traditional software is complex already, evident by the continuous patches and updates released as new bugs and exploits are discovered. But even more so with systems, we do not fully understand or at least know how to evaluate correctly, as these have intrinsic properties that present challenges for comprehensive testing. This is problematic as we are integrating and consequently interacting more with systems relying on these types of software in our daily lives before we fully understand these systems' complex intricacies and inner workings. Before we can give these systems more prevalence in our daily lives, it is paramount that we acknowledge our limited understanding and make progress toward addressing these shortcomings to minimize the possibility of a catastrophic failure in a security or safety-critical application.

References

Afzal, A., Goues, C. L., Hilton, M., & Timperley, C. S. (2020). A Study on Challenges of Testing Robotic Systems. *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 96–107. doi:10.1109/ICST46399.2020.00020

Armerding, Taylor. (2018, November 20). *Hard Questions Raised When a Software 'Glitch' Takes Down an Airliner*. Forbes. Retrieved from https://www.forbes.com/sites/taylorarmerding/2018/11/20/hard-questions-raised-when-a-software-glitch-takes-down-an-airliner/?sh=53a2c4877b1d

Berend, D. (2021). Distribution Awareness for AI System Testing. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 96–98. doi:10.1109/ICSE-Companion52605.2021.00045

Berglund, L., Grube, T., Gay, G., de Oliveira Neto, F. G., & Platis, D. (2023, April). Test Maintenance for Machine Learning Systems: A Case Study in the Automotive Industry. *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. Presented at the 2023 IEEE Conference on Software Testing, Verification and Validation (ICST), Dublin, Ireland. doi:10.1109/icst57152.2023.00045

CERN. (2019, September 23). *A Short History of the Web.* CERN. Retrieved from https://home.cern/science/computing/birth-web/short-history-web

Chakraborty, J., Majumder, S., & Menzies, T. (2021). Bias in Machine Learning Software: Why? How? What to Do? *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 429–440. Presented at the Athens, Greece. doi:10.1145/3468264.3468537

GeeksforGeeks. (2018, October 11). *Software Engineering - Software Maintenance*. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/software-engineering-software-maintenance/

Kim, H. E., Son, H. S., Kim, B. G., Cho, J., Shin, S. M., & Kang, H. G. (2018). Input-Domain Software Testing for Failure Probability Estimation of Safety-Critical Applications in Consideration of Past Input Sequence. *IEEE Access, 6,* 8440-8451. doi:10.1109/ACCESS.2017.2765698

Marijan, D., Gotlieb, A., & Kumar Ahuja, M. (2019). Challenges of Testing Machine Learning Based Systems. *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 101–102. doi:10.1109/AITest.2019.00010

Merriam-Webster. (n.d.). *Next Stop: "Trolley Problem"*. Merriam-Webster. Retrieved from https://www.merriam-webster.com/wordplay/trolley-problem-moral-philosophy-ethics

Polonski, Slava. (2017, December 19). *Can We Teach Morality to Machines? Three Perspectives on Ethics for Artificial Intelligence*. Medium. Retrieved from https://medium.com/@slavaxyz/can-we-teach-morality-to-machines-three-perspectives-on-ethics-for-artificial-intelligence-64fe479e25d3

Sun, Y., Huang, X., Kroening, D., Sharp, J., Hill, M., & Ashmore, R. (2019). Structural Test Coverage Criteria for Deep Neural Networks. *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 320–321. doi:10.1109/ICSE-Companion.2019.00134