Modeling of Software Growth and Defect/Vulnerability Introduction and Discovery in Successive versions of Programs using actual data.

Sarthak Siddhant Bharadwaj
sarthak7@colostate.edu
Computer Science
Colorado State University

***Abstract*— In this work, a predictive model based on real data is introduced for examining software growth and the appearance of bugs and vulnerabilities in subsequent versions. By utilizing machine learning, the model measures vulnerabilities trends and correlates them with software upgrades to evaluate risks and consequences. Software update decisions are made possible by the integration of features including vulnerability severity, frequency, expected losses from updates, and deployment costs. Through the thoughtful deployment of updates[2], this seeks to improve software security and dependability while striking a balance with cost-effectiveness. The model is a useful tool for forecasting software evolution and directing update plans.**

Keywords:Quantitative Analysis,Feasibility,Defect

I. Motivation

With each new version of software, vulnerabilities and weaknesses multiply due to Moore's Law[4], which predicts that the number of transistors on microchips will double every two years. This phenomena emphasizes how critical it is to comprehend and predict vulnerabilities in order to improve software security and dependability. An important foundation for anticipating and reducing possible risks is provided by research on modeling the vulnerability[2] detection process conducted by Alhazmi, Malaiya[1], and others. Their methodology emphasizes how critical it is to improve security protocols in tandem with the rate at which software is developed, as required by Moore's Law[4]. This methodology recognizes the complexity of rapid technology innovation and its impact on software evolution, going beyond theoretical value and providing practical methods for fortifying systems against vulnerabilities[3].

Expanding upon the knowledge gained from these groundbreaking investigations, there is a strong need for more research in fault-tolerant computing. Through the use of real-world data and sophisticated modeling approaches, this project proposal seeks to expand our knowledge of the lifespan of software vulnerabilities[6]. Research of this kind is essential to building more robust computer environments and guaranteeing that software stays safe and dependable even as it grows more and more integrated into our daily lives. By protecting digital infrastructure from new dangers, this project will not only increase academic knowledge but also have a positive impact on society by making the digital world a safer place for all.

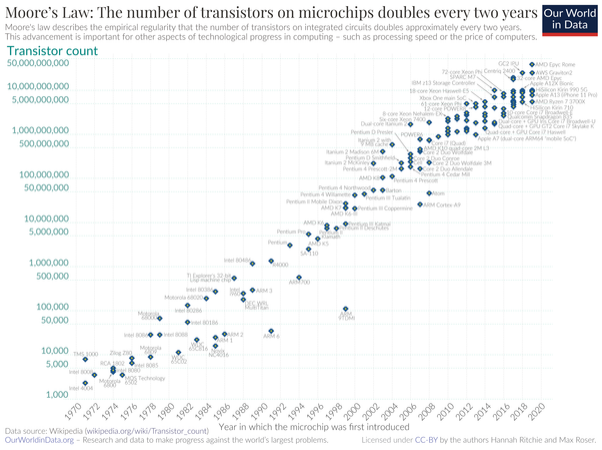

Fig. 1. A semi-log plot of transistor counts for microprocessors against dates of introduction, nearly doubling every two years

II. Approach

We propose an integrated strategy that combines the methods from the three papers into a cohesive approach for modeling software growth and defect/vulnerability detection across software versions. This integrated approach makes use of the advantages of quantitative analysis, sophisticated modeling techniques, and agile methodologies. We borrow the foundation of employing a stack of feature-specific models based on the Jelinski–Moranda model from "Software Reliability for Agile Testing" [2]. This model works well in the agile environment where features and bugs are always changing. This methodology improves predictability in an agile environment by helping us monitor faults more precisely by linking them to particular features.

The emphasis on Scrum methodologies, in particular the insights into managing interruptions and the iterative development process, can be found in "Sources of Interruptions Experienced During a Scrum Sprint" [3]. This article provides insightful information on how to organize our approach to software updates and deal with unforeseen changes. Through the integration of these insights, we can improve the model's resilience to the practical constraints of software development, guaranteeing that our model not only anticipates flaws but also offers appropriate solutions to limit their impact.[7]

Last but not least, our model is strengthened by the empirical and exploratory research methodology outlined in "Software Process Evaluation from User Perceptions and Log Data" [1], which focuses on qualitative data from actual environments and user perspectives. [6]This feature is essential for verifying our model's predictions and modifying it in response to input from real-world software deployment situations.

Combining these approaches allows us to develop a comprehensive model that not only predicts the frequency and seriousness of software errors but also offers a framework for ongoing modification and enhancement based on iterative feedback and field testing. This method seeks to provide a strong tool for both project managers and developers by striking a balance between the technical parts of software dependability modeling and the real-world challenges of software development and deployment.

III. Key Questions

RQ1 What kinds of sprint interruptions lead to unforeseen changes in requirements when they happen in the middle of a sprint?
RQ2 In what ways may the assessment of software processes be enhanced through the incorporation of user perceptions and real usage data?
RQ3 In agile development contexts, how can software dependability be accurately modeled to anticipate and efficiently handle defects?
RQ4 What are the primary causes of disruptions in Scrum sprints, and what effects do they have on team productivity and project management?

IV. Challenges

I faced a number of noteworthy difficulties when fusing user perceptions with real usage data to improve software process evaluation. It was a difficult analytical endeavor to align qualitative input with quantitative log data, requiring careful techniques to guarantee accuracy and coherence. It was challenging to guarantee that the user feedback gathered was reflective of the larger user base because it was essential to record a diverse range of user experiences in order to precisely reflect the general level of satisfaction and program usability.

Accurately recording and labeling failures linked with particular software features was a major difficulty in modeling software reliability for agile testing, especially in complex systems where functionality frequently overlap. Adapting conventional dependability models to agile [7]development's fast iteration cycles presented another challenge. It took a lot of effort and technical expertise to complete this adaption because it needed to be thoroughly calibrated and validated to make sure the models were realistic and accurate for use in the real world.

While addressing the causes of disruptions in Scrum sprints, I had to contend with the fact that different firms' implementations of Scrum differ, which limited the applicability of my results. Furthermore, there was a noticeable reluctance on the part of some team members to honestly talk about mistakes and disruptions, which limited the breadth of understanding that could be obtained from these conversations. It was challenging to come up with broadly applicable tactics and solutions for efficiently handling disruptions in Scrum environments because of these issues.

V. Findings

I performed investigations and data analysis in three key areas as part of my research on software development processes and their effect on project outcomes, and I came to some important conclusions:

First off, as shown in the paper "Software Process Evaluation from User Perceptions and Log Data,[6]" my work combining quantitative log data and user feedback revealed the critical role that combining qualitative and quantitative insights plays in software process evaluation. The relationship between certain software features and user satisfaction and engagement levels was successfully [7] established by this integration. According to my research, real-time feedback methods that better match software updates with user needs can be a huge help to developers in improving the overall usability and happiness of their program.

In my second study, which is detailed in "Software Reliability for Agile Testing," I modified conventional reliability models to fit the needs of agile development methodologies[4]. Through the use of Jelinski–Moranda model feature-specific modifications, I was able to greatly improve fault management and prediction. This method demonstrated the value of specialized models in efficiently managing software reliability within agile frameworks by enabling more focused resource allocation and enhanced predictive accuracy in agile contexts.

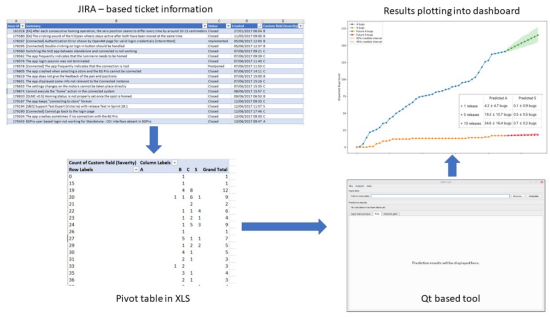

Fig. 2. Flowchart for automatic generation of software reliability predictions.

Hardware and software malfunctions are not the same. Software failures are brought on by latent software errors rather than malfunctioning parts or wear-out from external environmental pressures like temperature, moisture, and vibration. When the software was developed, these flaws were added [7]. However, before being made available to the client, these flaws were not found and/or fixed. A greater level of software stability must be attained to stop the client from noticing these flaws. This refers to lowering the possibility that software that has been released contains latent faults. Unfortunately, the introduction of software flaws is unavoidable, even with the most highly qualified software developers adhering to industry best practices. This is because software functionality and its execution environment have ever-increasing inherent complexities. Software reliability engineering, which deals with testing and modeling software functionality in a certain environment over a predetermined period of time, may be useful in this situation. As of right now, there is no way to ensure completely dependable software. A collection of statistical modeling approaches is needed to produce the best software feasible. These techniques must be able to: • evaluate or forecast the reliability that has to be attained; • be based on the observations of software failures during testing and/or operational use.

Finally, in the research paper "Sources of Interruptions Experienced During a Scrum Sprint," I determined and examined the sources of the typical disruptions that arise during Scrum sprints. My research showed that management's ignorance of Scrum procedures, ad hoc requests, and vague project objectives are frequently the causes of interruptions. These setbacks have a detrimental effect on team morale in addition to impeding project timeframes. I found that these problems might be resolved with tighter adherence to Scrum practices and better planning, which would increase the output and efficacy of Scrum teams.

VI. Future Work

Future work will focus on creating sophisticated tools that dynamically integrate user feedback into the software development lifecycle, building on my research into integrating user feedback with software development methods. In order to do this, a system that can collect user feedback in real-time and use machine learning algorithms to anticipate user demands must be designed. A feedback loop that converts ongoing user inputs into useful insights would enable software features and updates to be more responsive to user needs in such a system. Such sophisticated, data-driven techniques are feasible to implement because to Moore's Law, which has implications for growing computing power and makes it possible to handle large data sets and complicated computations more effectively.

Future work will concentrate on incorporating these models into CI/CD pipelines to increase operational efficiency in order to further improve the adaptation of dependability models to agile environments. The goal of this project is to create a solid framework that can deliver accurate defect forecasts while supporting quick agile iteration cycles. It will also be crucial to fine-tune the model parameters so that they better reflect the complexities of modern software development. These advances are supported by the evolution led by Moore's Law, which foresees rapid gains in processing power. Moore's Law provides the technological assistance required to manage the complexities and improve the dependability of contemporary software development techniques.

References

[1] Alhazmi, Omar Malaiya, Yashwant. (2005). Modeling the Vulnerability Discovery Process. 2010 IEEE 21st International Symposium on Software Reliability Engineering. 129-138. 10.1109/ISSRE.2005.30.

[2] G. Lin, S. Wen, Q. -L. Han, J. Zhang and Y. Xiang, "Software Vulnerability Detection Using Deep Neural Networks: A Survey," in Proceedings of the IEEE, vol. 108, no. 10, pp. 1825-1848, Oct. 2020, doi: 10.1109/JPROC.2020.2993293. keywords: Computer security;Semantics;Feature extraction;Open source software;Neural networks;Deep learning;Computer bugs;Cybersecurity;deep neural network (DNN);machine learning (ML);representation learning;software vulnerability,

[3] Yazdan Movahedi, Michel Cukier, Ilir Gashi, Vulnerability prediction capability: A comparison between vulnerability discovery models and neural network models,Computers Security,Volume 87,2019,101596,

[4] R. R. Schaller, "Moore's law: past, present and future," in IEEE Spectrum, vol. 34, no. 6, pp. 52-59, June 1997, doi: 10.1109/6.591665. keywords: Moore's Law;Silicon;Technological innovation;Consumer products;Roads,

[5] Willem Dirk van Driel, Jan Willem Bikker, Matthijs Tijink, and Alessandro Di Bucchianico authored the paper "Software Reliability for Agile Testing," published in the journal Mathematics. The paper appeared in volume 8, issue 5 of the 2020 edition, under the DOI: 10.3390/math8050791.

[6] Maureen Tanner and Angela Mackinnon conducted a study titled ”Sources of Interruptions Experienced During a Scrum Sprint,” which was featured in The Electronic Journal of Information Systems Evaluation. This work is included in volume 18, issue 1 from 2015, accessible online at www.ejise.com.
[7] The third paper, by Tadeja Kosec and others, discusses ”Software Process Evaluation from User Perceptions and Log Data,” published in the Journal of Software Evolution and Process. Details about the publication year and issue are not specified, but the paper focuses on the interplay between user perceptions and software process evaluation.