Thesis Project Portfolio

Program Analysis of Educational Hardware-Description Language

(Technical Report)

On the Normalization of Software Bugs

(STS Research Paper)

An Undergraduate Thesis

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

James Leo Yuan Huang

Spring, 2023

Department of Computer Science

Contents of Portfolio

Executive Summary

Even though digital computers are a recent invention, their current capability and pervasiveness in modern society is impressive. However, it is unclear whether the software they run is equally as impressive. Software engineering is unique in that what is engineered (software) is primarily abstract and does not struggle against the "real world" as much as, say, a bridge that struggles against the forces of its pedestrians. Yet even though our computing power has grown exponentially in the last few decades, the same software engineering woes of yesterday continue to ring true, and we still produce software with defects. My STS research analyzes our current perceptions of software and its engineering, investigating whether we have normalized software bugs. My technical work explores the boundaries of traditional software engineering, experimenting with how nontraditional testing methods can be used to assess the correctness of educational programming assignments.

Software has a concerning amount of bugs. Software engineers continue to "debug" their software, but it is unclear why the bugs are there in the first place. For my STS research project I analyzed this problem to determine whether bugs have been normalized in software engineering. I identified two contrasting viewpoints of software bugs: that they are a manifestation of the essential nature of software engineering, or that they are humans' failure to write correct programs. Further analyzing the latter viewpoint, I discovered that mathematical and formal methods to software development have seen success where applied but have ultimately not been applied very widely, likely because they clash with the informal nature of most software development. I thus concluded that bugs have been normalized in software engineering due to the normality of the fluid, informal software development process that treats bugs as part of the process.

Traditional program testing works by feeding a program some inputs and comparing the resulting outputs to predetermined expected outputs. This can easily reveal the program's incorrectness (if the resulting outputs do not match the expected outputs), but for all but the most simple programs, it is infeasible to verify that a program is completely correct by testing all possible inputs and outputs as it would take too long. For my technical project I developed an autograding tool that analyzes using nontraditional methods the correctness of student programs written in HCLRS, an educational hardware-description language. My final approach was to use "symbolic execution" to evaluate symbolic (as opposed to concrete) inputs into symbolic outputs that are compared with predetermined symbolic outputs in such a way that avoids incorrectly concluding that a program is false. The final tool offered advantages over traditional testing approaches (e.g. it avoided double-jeopardy and produced more detailed feedback), but ultimately it could not be applied to all of the available HCLRS assignments due to time constraints and design issues stemming from its experimental nature.

My work took an interesting look at current software-engineering practices, analyzing them in my STS research and exploring their boundaries in my technical work. While my STS research offers a preliminary argument that bugs have been normalized in software engineering, it does not offer any discussion of whether this normalization is ethical and thus should be denormalized, or whether denormalization would be feasible in the first place. And while the tool I produced in my technical work was a good proof-of-concept for the relevant ideas, it was developed experimentally and thus lacks formal correctness, and also has not been applied to all available assignments. Despite these major problems that would ideally be addressed in future work, I enjoyed and am satisfied with what I have accomplished. Though it feels like software

engineering is proceeding in the exact opposite direction that I poked at with my work, I am content having done what I wanted to do.