Research Statement

Eric Schulte

My work treats software as an evolved system and yields novel solutions to difficult software engineering (SE) problems. In my dissertation, I have developed systems that automatically fix bugs in off-the-shelf software [3], patch exploits in closed source binaries [7], generate diverse implementations of a software specification [6], and reduce the energy consumption of benchmark programs by 20% as compared to the best available compiler optimizations [4].

I view the current software ecosystem as the product of an evolutionary process in which developers, through re-using and modifying effective tools and techniques, serve as the agents of reproduction and variation. I have uncovered empirical evidence of evolution by studying the *mutational robustness* of software [6]; work which helps to motivate the use of randomized mutation operations to modify extant software.

The field of genetic programming (GP) has long used biologically inspired program mutations to evolve programs written in custom-built, simplified domain specific languages. My work adapts GP techniques to real-world computer software and architectures. The ultimate goal of this work is the realization of the original goal of GP—the automated construction of wholly new software artifacts and functionality—inside of the existing software development ecosystem.

Previous Contributions

I research the natural aspects of computation emphasizing the production of usable tools. I have made the following research contributions.

Genetic Program Repair As part of the Genprog team, I developed automated software repair techniques that are applicable to real-world software. Given a software project with a regression test suite and at least one failing test, Genprog applies random mutations to the source code of the original program yielding alternative program implementations. In a process mirroring natural selection, an evolutionary algorithm searches for alternate program implementations until a version of the program is found which passes the originally failing test while still passing the entire regression test suite. In a systematic study of bugs culled from 8 popular open-source software projects Genprog was able to repair 55 of 105 bugs in less than 2 hours of runtime per repair. This work is largely responsible for the recent interest in automatic program repair in the SE community.

Genetic Program Optimization I developed the Genetic Optimization Algorithm (GOA) [4], a technique of post-compiler software optimization capable of reducing the runtime and energy consumption of popular benchmark programs by 20% compared to the best optimizations provided by GCC. Using generic mutation operations, an evolutionary algorithm is used to evolve x86 assembler compiled from an original program to optimize any aspect of the program's runtime behavior which may be efficiently measured or modeled. Recent advances in profiling techniques and the prevalence of hardware performance counters allow even sophisticated runtime properties like energy consumption to be targets of optimization. GOA is able to find optimizations specific to both the target hardware and the workload used for optimization.

Software Representation I have explored methods of representing and modifying extant software. I have extended Genprog from the modification of C source code, to the modification of LLVM IR and multiple assembler languages including x86 and ARM [5], and I have extended these techniques to directly modify binary executables on multiple architectures including x86 and MIPS [3, 7]. This work allows for wider application of genetic techniques to multiple programming languages, in resource constrained devices, and to software for which no source code is available (e.g., closed source binary executables). I have demonstrated this increased applicability by repairing exploits in a popular NETGEAR router firmware before the company itself addressed the exploit. I hope to continue this work by investigating the power and efficiency of genetic techniques across multiple representations of extant software.

Software Mutational Robustness I have demonstrated that extant software (including both large open source projects and a number of extremely well-tested benchmark programs taken from the software testing community) has the *mutational robustness*, meaning the software's functionality is robust to random mutations at the level of source code and compiled assembly code, one would expect from an evolved system [6]. Mutational robustness has been studied extensively in biological systems where it is associated with development through evolution. Software mutations often yield algorithmically distinct implementations of a project's implicit specification (demonstrated by passing the software's test suite). This work motivates a new view of software as fundamentally robust to random modification, helps to explain the success of previous mutation-based SE techniques, and opens the door to a variety of new methods of software development.

Physical Evolutionary Computation I developed a physical evolutionary computation (PEC) algorithm, designed to run on a unique hardware platform composed of homogeneous live-plugable distributed computational tiles [1]. By eliminating global state and points of control the PEC algorithm is able to run over an unbounded number of tiles, make use of new tiles attached mid-computation, gracefully degrade when tiles are removed mid-computation, and function as tile groups are split and later rejoined. PEC introduced a number of unique features including real-time mutation rates, and the ability to map aspects of the problem to the physical dimensions of the hardware. Although this work explored novel and naturally robust models of computation, the impact was limited by its departure from standard hardware and tooling.

Reproducible Research I work to ensure that my published empirical results are reproducible. To this end, I am the author of a widely used open source reproducible research tool kit [2] with support for 52 programming languages and contributions from 63 software developers. This project has been used to in publications in fields as diverse as Anthropology, Genetics and of course Computer Science.

Next Steps

I hope to continue the integration of GP techniques into the daily work-flow of software developers. This will require new areas of focus in GP research and will result in the development of new SE tools.

(GP) Representation Genetic techniques are dependent upon software representations which integrate into existing software development toolchains, and are modifiable by *both* software developers and genetic mutations. Currently ASM performs very well as a target of genetic modifications. ASM has many properties in common with DNA including a linear representation, sequential execution in contiguous segments, the use of reading frames and local jumps. Unfortunately ASM modifications are not easily communicated back to software developers. More work is required to find new program representations

facilitating communication between software developers and genetic techniques.

(GP) Near Optimal Search Software requirements, like ecosystems, change gradually. Thus the evolution of extant software is similar to the evolution of biological systems in that the starting point of the search, the original program or the wild type organism respectively, is near optimal. This deviates sharply from traditional work in GP, in which the starting population is often a collection of random candidates with little or no fitness. New GP techniques for the evolution of extant software should be informed by the ways in which biological adaptation is tailored to this process of repeated gradual improvement.

(SE) Software Husbandry In addition to work on the GP front, I plan to integrate recently developed genetic optimization more tightly into the software development process. This work will combine the current processes of software development followed by post-compiler genetic optimization into a collaborative model in which a population of software objects is maintained and is contributed to in parallel by both software developers and evolutionary processes. This new practice should have the twin benefits of providing GP techniques increased time to improve software and by granting software developers a chance to influence, understand, and work with the evolutionary process.

Purpose

I hope to make GP techniques part of the standard software development toolbox. I see this as a multi-stage project which is already underway. Program repair and optimization currently use GP to modify real-world programs. Further work refining software representations and biologically-inspired mutations can increase the power of such techniques. Interactive evolutionary tools will apply GP techniques to the development of novel software in a setting where human developers may compensate for GP's shortcomings. Through gradually reducing the role of software developers, I hope this work will culminate in the full GP of useful real-world software. I believe that this goal, the original vision of GP, is now becoming practical, and although there are certainly unforeseen hurdles and research questions along the way, this future is starting to take shape.

References

[1] Eric Schulte and David Ackley. Physical evolutionary computation. Technical report, University of New Mexico, Albuquerque, NM, USA, 2011.

[2] Eric Schulte, Dan Davison, Thomas Dye, and Carsten Dominik. A multi-language computing environment for literate programming and reproducible research. *Journal of Statistical Software*, 46(3):1–24, 1 2012.

[3] Eric Schulte, Jonathan DiLorenzo, Westley Weimer, and Stephanie Forrest. Automated repair of binary and assembly programs for cooperating embedded devices. In *Proceedings of the eighteenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13. ACM, 2013.

[4] Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, and Westley Weimer. Post-compiler software optimization for reducing energy. In *Proceedings of the eighteenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14. ACM, 2014. To appear.

[5] Eric Schulte, Stephanie Forrest, and Westley Weimer. Automated program repair through the evolution of assembly code. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 313–316, New York, NY, USA, 2010. ACM.

[6] Eric Schulte, Zachary. Fry, Ethan Fast, Westley Weimer, and Stephanie Forrest. Software mutational robustness. *Genetic Programming and Evolvable Machines*, pages 1–32, 2013.

[7] Eric Schulte, Westley Weimer, and Stephanie Forrest. Repairing security vulnerabilities in the netgear router binary. Submitted, December 2013.