A spoonful of DevOps helps the GI go down

Benoit Baudry, Nicolas Harrand
KTH,
baudry,harrand@kth.se

Eric Schulte
Grammatech
schulte.eric@gmail.com

Chris Timperley
Carnegie Mellon University
ctimperley@cmu.edu

Shin Hwei Tan
National University of Singapore
shinhwei@comp.nus.edu.sg

Marija Selakovic
TU Darmstadt
m.selakovic89@gmail.com

Emamurho Ugherughe
SAP
emamurho@gmail.com

ABSTRACT

DevOps emphasizes a high degree of automation at all phases of the software development lifecyle. Meanwhile, Genetic Improvement (GI) focuses on the automatic improvement of software artifacts. In this paper, we discuss why we believe that DevOps offers an excellent technical context for easing the adoption of GI techniques by software developers. We also discuss A/B testing as a prominent and clear example of GI taking place in the wild today, albeit one with human-supervised fitness and mutation operators.

KEYWORDS

Genetic Improvement, Continuous Integration, DevOps

ACM Reference Format:
Benoit Baudry, Nicolas Harrand, Eric Schulte, Chris Timperley, Shin Hwei Tan, Marija Selakovic, and Emamurho Ugherughe. 2018. A spoonful of DevOps helps the GI go down. In *Proceedings of Workshop on Genetic Improvement (GI).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Genetic Improvement aims at improving existing software artifacts through search via automatic transformations and evaluation [5]. GI has been used to automatically improve software in a variety of ways, including fixing bugs [10] and reducing energy consumption [2]. In this paper, we outline how GI could have a broader impact on the software-engineering community by leveraging existing DevOps tooling to enable the deployment of GI at scale.

DevOps, a portmanteau of development and operations, aims at continuous end-to-end automation in software development and delivery [1]. DevOps has been adopted by most web-based companies and is gaining momentum in other software-intensive industries (e.g., telecom[1], finance[2], IT[3], etc.). From a software-technology perspective, DevOps emphasizes the construction of software factories that assemble tens of tools, each of them in charge of automating one task of the process from commit to deployment to end users. From a process perspective, DevOps is a way of closing the gap between development teams and the operational teams in charge of deployment and production operations. The ∞-shape in Figure 1 illustrates this continuous flow of software evolution.

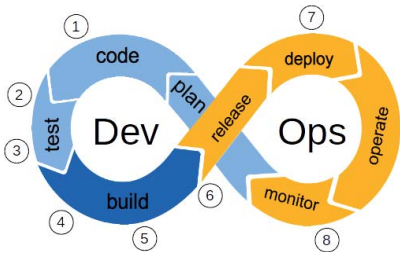

Figure 1: DevOps: end-to-end automation of software development and operation for continuous evolution.

Both the social and technical dimensions of DevOps pave the way for the adoption of GI. From a social perspective, we often hear in the GI community: "developers do not want a tool to mess with their code." From a technical dimension, application of GI requires tooling that is able to automatically build and evaluate software. The DevOps process provides for full automation. DevOps developers are accustomed to tools that manipulate their code in various ways (e.g., code review, deployment, merging). We believe that integrating GI technologies into this tool-supported process and finding appropriate niches for automatic improvement within DevOps is one way to promote and accelerate the adoption of GI among software developers.

In the rest of this paper, we discuss technical aspects of the integration of GI into DevOps: we identify artifacts produced in DevOps pipelines that may be genetically improved, as well as tools that automate DevOps tasks that can be leveraged by GI.

2 OPPORTUNITIES FOR GI IN DEVOPS

The existence of sets of software artifacts, i.e. of populations of artifacts, is a key feature that shall be leveraged by GI. To name a few of these populations in DevOps; there is populations of pull requests in the *code* phase; populations of tests in the *test* phase; populations of dependencies in the *build* phase; populations of instances of server and client side parts of an application in the *deploy* phase. These populations of artifacts open several opportunities for GI.

Although most GI techniques modify source code to automatically fix bugs or improve performance, the scope of GI is much broader: to automatically improve any software artifact produced by human developers. In this section, we identify several non-source

[1] https://www.ericsson.com/en/publications/ericsson-technology-review/archive/2017/devops-fueling-the-evolution-toward-5g-networks
[2] http://tech.finn.no/2017/03/10/unleash-your-features-gradually/
[3] https://atos.net/en/blog/devops-or-die

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GI, 2018, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

code artifacts produced in DevOps, and discuss how they can benefit from genetic improvement. The opportunities discussed below correspond to the numbers annotating the DevOps in Figure 1.

(1) Fixing merge conflicts. In a collaborative development setting merge conflicts, in which multiple developers simultaneously edit a section of code, are frequent. Solving these conflicts is a time consuming manual task, which can benefit from GI automation. In this scenario there are at least three versions of code (master and 2 conflicting versions), each of which comes with a test suite. GI may be used to genetically recombine these versions using the combined test suite as a fitness function. Prior work that fixes software errors using multiple program versions shows the feasibility of this approach [8]. Such automation may result in significant time savings for software engineers.
(2) Genetically improve the test suite. Automatic testing is essential in DevOps. As a result, developers write sets of test cases that can be automatically executed. These are software artifacts that can be genetically improved. Improvement here consists in augmenting the test suites with new cases that execute uncovered branches or increases the mutation score of the suites. DSpot[4] and AFL[5] are examples of tools that address this form of GI.
(3) Genetically fix flaky tests, i.e., tests that randomly pass or fail on the same program version. Such tests are problematic for continuous integration (CI) since the test results vary across runs. There are many causes for flaky behavior, e.g., concurrency, network access, I/O operations, etc. One mitigation strategy is to increase isolation of the code under test. Here, GI can search for a version of the test code that is more stable (test stability could be measured by the changes in test outcomes across runs).
(4) The (CI) engine collects build dependencies, builds the complete application, and runs the tests automatically. If a bug appears in new code it will first manifest in the CI pipeline. That and the fact that the CI can run tests on demand suggests that CI provides an excellent entry point for automatic GI bug repair in DevOps. The very first tool filling this niche is now available [9].
(5) Building large projects requires assembling multiple parts, which independently declare dependencies to third party libraries. This results in new kinds of build bugs, dependency conflicts. For example, if multiple modules depend on different versions of a library, the build will fail. Fixes require finding a combination of edits in dependency declaration to resolve conflicts. This search is a classic GI task.
(6) The *release* phase consists of bundling the result of a build, with a runtime environment, into a single container. While this process is fully automated, it also produces container images that contain much more code than is necessary to run the application code. Similar to super optimizers [3], the container could be genetically minimized to include only the necessary code.
(7) The *deploy* phase consists of deploying multiple clones of the same image to let the application scale. A search strategy could synthesize diverse versions of the image for exploiting the trade-off between quality and resource consumption. Initial results show the feasibility of this type of improvement [6, 7].
(8) While the *monitor* phase appears less amenable to GI, it is important to consider all the data it can feed back to the development teams, enabling further genetic improvement. This data includes statistics on resource consumption, load, crashes, etc. In the next section, we discuss how A/B testing can be revisited as live GI that exploits this feedback to drive an evolutionary process spanning the DevOps ∞-cycle.

[4]https://github.com/STAMP-project/dspot
[5]http://lcamtuf.coredump.cx/afl/

3 A/B TESTING

To answer questions such as "what features of my game will users prefer" or "what design will attract most users," web companies use a technique called *A/B testing*. This technique involves developing multiple versions of an application along with a quantifiable criteria to score these versions. Next, the versions are deployed side by side and the companies collect usage statistics for each version. Finally, the company calculates *fitness* from these usage statistics and *selects* the fittest version for subsequent deployment.

In the context of DevOps, the alternative solutions are designed and developed manually, but the rest happens fully automatically, including; monitoring, assessing acceptance criteria through automatic analytics, and migrating all deployments to the fittest.[6]

A/B testing appears to be a clear example of evolutionary improvement that is currently widely practiced by industry. The initial variants is built manually, but all the other steps are fully automated, and clearly adopt an approach based on the survival of the fittest, using machine learning for real time data analytics and DevOps tool chains to modify the deployed software according to this feedback.

4 CONCLUSION

We invite the GI community to consider the deployment of their techniques in the DevOps environment. The existing automated tools and processes involved in DevOps have removed classic technical and social barriers to the deployment of GI. The DevOps environment is ripe for the insinuation of GI techniques.

REFERENCES

[1] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE Software*, 33(3):94–100, 2016.
[2] William B Langdon and Mark Harman. Optimizing existing software with genetic programming. *TEVC*, 19(1):118–135, 2015.
[3] Henry Massalin. Superoptimizer: a look at the smallest program. In *ACM SIGPLAN Notices*, volume 22, pages 122–126. IEEE Computer Society Press, 1987.
[4] Risto Miikkulainen, Neil Iscoe, Aaron Shagrin, Ron Cordell, Sam Nazari, Cory Schoolland, Myles Brundage, Jonathan Epstein, Randy Dean, and Gurmeet Lamba. Conversion rate optimization through evolutionary computation. In *GECCO '17*, GECCO '17, pages 1193–1199, New York, NY, USA, 2017. ACM.
[5] Justyna Petke, Saemundur Haraldsson, Mark Harman, David White, John Woodward, et al. Genetic improvement of software: a comprehensive survey. *TEVC*, 2017.
[6] Marcelino Rodriguez-Cancio, Jules White, and Benoit Baudry. Images of code: Lossy compression for native instructions. In *ICSE 2018, NIER Track*, Gothenburg, Sweden, 2018.
[7] Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, and Westley Weimer. Post-compiler software optimization for reducing energy. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 639–652. ACM, 2014.
[8] Shin Hwei Tan and Abhik Roychoudhury. relifix: Automated repair of software regressions. In *ICSE 2015*, pages 471–482. IEEE Press, 2015.
[9] Simon Urli, Zhongxing Yu, Lionel Seinturier, and Martin Monperrus. How to Design a Program Repair Bot? Insights from the Repairnator Project. In *ICSE 2018, Track Software Engineering in Practice (SEIP)*, Gothenburg, Sweden, 2018.
[10] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically finding patches using genetic programming. In *ICSE 2009*, pages 364–374. IEEE Computer Society, 2009.

[6]for example at King, an online game company https://blog.crisp.se/2016/11/22/yassalsundman/ab-testing-at-king, also "ugly logos" designed automatically using A/B testing [4]