Association for Computing Machinery
Advancing Computing as a Science & Profession

Ubiquity, an ACM publication
December, 2012

Ubiquity Symposium

Evolutionary Computation and the Processes of Life

Darwinian Software Engineering: The Short Term, the Middle Ground, and the Long Haul

by Moshe Sipper

Editor's Introduction

In this article, Moshe Sipper discusses a foreseeable future in which an entirely new paradigm of producing software will emerge. Sipper calls this software engineering revolution, "Darwinian Software Engineering"—a time when it will be possible to program computers by means of evolution.

Editor

 ©2012 Association for Computing Machinery

Ubiquity Symposium

Evolutionary Computation and the Processes of Life

Darwinian Software Engineering: The Short Term, the Middle Ground, and the Long Haul

by Moshe Sipper

"Programming is an unnatural act," wrote renowned computer scientist Alan J. Perlis in a popular medley of humorous, pithy epigrams on programming published in 1982. Quite likely he was referring to the daunting task facing programmers. Laborious to write, arduous to maintain, buggy, constantly out of date, or, in short: software.

If only we could just say what we want! A wish to which Perlis provided a ready-made reply among his epigrams: "When someone says 'I want a programming language in which I need only say what I wish done', give him a lollipop." Life in the fast (software) lane is hard—and getting harder by the day.

A report published in 2006 notes that, "in software we continue to accept failure rates, quality problems, and costs that would be unacceptable in any other field of engineering [1]." The report further explains that our current practices, already costly and problematic, will simply not scale to future systems comprising billions of lines of code.

In the foreseeable future standard software engineering may simply hit a complexity wall. We will need more than a linear extension of our current practices—we will need an entirely new paradigm of producing software. I believe this new paradigm is what I call "Darwinian Software Engineering." Because the one process that has proven successful time and again at engendering complex objects is evolution by natural selection.

 ©2012 Association for Computing Machinery

The report by Northrop et al. avers that, "Judiciously used, digital evolution can substantially augment the cognitive limits of human designers and can find novel (possibly counterintuitive) solutions to complex [ultra-large- scale] system design problems."

Software engineering will require a revolution—and evolution may well be the answer. It is already providing ample support to software engineers in the short term, and I foresee benefits in the long run as well.

The Short Term

Rising to the challenge of revolutionizing software engineering, the field of search-based software engineering (SBSE) has come into being over the past few years [2]. SBSE is a field wherein metaheuristic search techniques, chiefly (though not solely) evolutionary algorithms, are applied to software engineering problems. The field is based on the observation that many activities in software engineering can be formulated as optimization problems, which—due to their computational complexity—can only be solved through heuristic means. The field's importance and scope has been growing steadily, with a major conference now held annually as well as several tracks in other top conferences. To date, researchers have addressed hard, contemporary problems of import, including: cost estimation, project planning, the next-release problem, design decisions, source-code optimization, test-data generation, and software maintenance. These efforts are aiding present-day software engineers, and will continue to develop in the near future.

The Middle Ground

Over the past few years my graduate student Michael Orlov and I have been developing the FINCH system, intended to evolve and improve existing programs in the highly popular Java programming language [3, 4, 5]. FINCH evolutionarily improves actual, extant software, which was not intentionally written for the purpose of serving as an evolutionary algorithm representation in particular, or for evolution in general. The only requirement is the software source code be either written in Java or can be compiled to Java bytecode. We have demonstrated several examples of evolved Java programs that solve hard problems: symbolic regression, trail navigation, image classification, array sum, and tic-tac-toe. Beginning from "bad seeds" (poor Java programs), or even from good seeds, evolution provides an ability to take human-written, functioning software—and improve it.

 ©2012 Association for Computing Machinery

We chose to automatically improve extant Java programs by evolving the respective compiled bytecode versions. This allows us to leverage the power of a well-defined, cross-platform, intermediate machine language at just the right level of abstraction: We do not need to define a special evolutionary language, thus necessitating an elaborate two-way transformation between Java and our language; nor do we evolve at the Java level, with its encumbering syntactic constraints, which render the genetic operators of crossover and mutation arduous to implement. FINCH thus advances both the state-of-the-art in automated software engineering as well as in evolutionary computation, wherein we have shown how a complex, existing programming language can be turned into an evolutionary parlance.

This latter point is important in the current context: We did not wish to invent a language to improve upon some aspect or other of evolutionary computation (efficiency, terseness, readability, etc.), which has been amply done. Nor did we wish to extend standard genetic programming to become Turing complete, an issue that has also been addressed [6]. Rather, conversely, our point of departure was an extant, highly popular, general-purpose language, with our aim being to render it evolvable. This opens the door to a whole new direction in evolutionary computation, one wherein an extant computational language can be "converted" to Darwinism.

Orlov and I recently predicted that in about 50 years' time it will be possible to program computers by means of evolution; not merely possible but indeed prevalent [7].

The Long Haul

Evolutionary computation is now routinely used to design complex objects that stretch—or overstretch—our classical engineering techniques. But these results are still quite limited in scope. Can something truly astounding—something entirely new—emerge out of an artificially set stage? This is a question I posed a few years ago in my book *Machine Nature* [8]. In my mind this is one of our grandest challenges, and it may still be many years in the coming. Think, for one, what an astounding feat it would be to evolve a cricket-level intelligence from first principles.

Natural evolution is an open-ended process and is thus distinguished from artificial evolution, which is guided, admitting a "hand of god": The (human) user who defines the problem to be solved. When we apply evolution with a well-defined goal in mind—such as designing a bridge, constructing a robotic brain, or developing a computer program—what we are doing is akin to animal husbandry. Farmers have been using the power of evolution for hundreds of years, in

 ©2012 Association for Computing Machinery

effect doing evolutionary computation on domestic animals. Perhaps one would need to set up a process of open-ended evolution, which, to date, only nature has been able to do so.

Whether open-ended evolution is the answer to the long-haul challenge is uncertain, but there are most definitely significant—and exciting—challenges that lie ahead.

References

[1] Northrop, L. et al. *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Carnegie Mellon University, Pittsburgh, PA, July 2006. URL http://www.sei.cmu.edu/uls.

[2] Harman, M. The current state and future of search based software engineering. In Proceedings of FOSE '07, *2007 Future of Software Engineering* (May 20 - 26, Minneapolis, MN) IEEE Computer Society, Washington, DC, 2007, 342–357,

[3] Orlov, M. and Sipper, M. Genetic Programming in the Wild: Evolving unrestricted bytecode. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation* (July 8-12, Montreal). ACM Press, New York, 2009, 1043–1050,

[4] Orlov, M. and Sipper, M. FINCH: A system for evolving Java (bytecode). In R. Riolo, T. McConaghy, and E. Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*. Springer, Berlin, 2010, 1–16.

[5] Orlov, M. and Sipper, M. Flight of the FINCH through the Java wilderness. *IEEE Transactions on Evolutionary Computation* 15, 2 (2011), 166–182.

[6] Woodward, J. Evolving Turing complete representations. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003* (Dec. 8-12, Canberra, Australia). IEEE Press, Washington D.C., 2003, 830–837.

[7] Sipper, M. 2011. URL http://www.moshesipper.com/finch/.

[8] Sipper, M. *Machine Nature: The Coming Age of Bio-Inspired Computing*. McGraw-Hill, New York, 2002.

Acknowledgement

The author is with the Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel. Email: sipper@cs.bgu.ac.il. This research was supported by the Israel Science Foundation (Grant No. 123/11).

 ©2012 Association for Computing Machinery

About the Author

Moshe Sipper is a professor of computer science at Ben-Gurion University of the Negev, Israel. He received his B.A. degree from the Technion- Israel Institute of Technology, and his M.Sc. and Ph.D. degrees from Tel Aviv University, all in computer science. During 1995-2001 he was a Senior Researcher at the Swiss Federal Institute of Technology in Lausanne. Dr. Sipper's current research focuses on evolutionary computation, mainly as applied to software development and games. At some point or other he also did research in the following areas: bio-inspired computing, cellular automata, cellular computing, artificial self-replication, evolvable hardware, artificial life, artificial neural networks, fuzzy logic, and robotics. He has published more than 140 scientific papers, and is the author of three books: *Evolved to Win*, *Machine Nature: The Coming Age of Bio-Inspired Computing*, and *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. He is an Associate Editor of the *IEEE Transactions on Computational Intelligence and AI in Games* and *Genetic Programming and Evolvable Machines*, an editorial board member of *Memetic Computing*, and a past associate editor of the *IEEE Transactions on Evolutionary Computation*.

DOI: 10.1145/2406356.2406358

 ©2012 Association for Computing Machinery