DOI:10.1145/1735223.1735248

Technical Perspective
Automated Patching Techniques: The Fix Is In

By Mark Harman

OVER THE PAST 40 years, much effort has been devoted to testing software to find bugs. Testing is difficult because the underlying problem involves undecidable questions such as statement reachability. However, testing cannot be ignored. The National Institute of Standards and Techniques estimated the cost of software failure to the U.S. economy at $60 billion, indicating that improved software testing could reduce this by at least one-third.[6]

If finding bugs is technically demanding and yet economically vital, how much more difficult yet valuable would it be to automatically fix bugs? This question is answered precisely by the work reported in the following paper by Weimer, Forrest, Le Goues, and Nguyen. The authors use evolutionary computation to evolve patches that fix bugs. Their work is the first to show how Genetic Programming can evolve patches that fix real bugs (an idea first proposed by Arcuri and Yao, who experimented on toy programs[3]).

There is an increasing realization within the software engineering research and development community that evolutionary computation and related search-based optimization can be used to search for solutions to software problems. Software engineers often face problems typified by enormous spaces of possible requirements, designs, test cases, and code. Search-based optimization allows them to deploy optimization algorithms that automatically search these spaces, guided by fitness functions that encode solution desirability.

The search-based optimization approach has come to be known as Search Based Software Engineering (SBSE).[4] This approach is widely appealing because it is very generic, applying equally well to many varied software engineering problems. SBSE also comfortably handles the multiple, conflicting, and noisy optimization objectives often found in software engineering scenarios. Using SBSE, it has proved possible to automate the search for requirements that balance cost and benefit, designs that maximize cohesion and minimize coupling and to find test cases that balance fault finding against execution time.[1,2,5,7]

Work on SBSE has gathered pace over the past 10 years and there are now over 650 papers[a] on various applications of search-based optimization to software engineering problems. The following paper is sure to be looked back upon as a significant and tangible breakthrough in this history of SBSE research. The authors' work demonstrates that an optimization algorithm is capable of repairing broken programs. Since it is automated, it can fix bugs much faster than any human. For example, the Zune bug was patched in a little over three minutes on standard equipment.

Currently, both developers and researchers remain cautious about the idea of deploying evolved program code. However, with the benefit of hindsight, we may look back on this with the same historical perspective that we now apply to compiled code. That is, despite its current widespread use, there was, within living memory, equal skepticism about whether compiled code could be trusted. If a similar change of attitude to evolved code occurs over time, then automated patching will surely be seen to have played a crucial part is this paradigm shift.

The successful application of search-based optimization techniques requires careful configuration and tailoring of the algorithms employed. The more information that can be exploited to guide the search, the better the results. The authors use several innovative techniques to guide their search for patches. Their approach automatically fixes software faults in a manner analogous to the way in which a surgeon might go about healing a wound. In order to localize the wound, they use positive and negative test cases, finding code segments most likely to contain the bug. One important insight employed in the patching process is the use of what might be called "plastic surgery;" rather than evolving a patch from scratch, the authors start their evolution from portions of existing code with similar functionality. The evolved patch is subsequently tidied up using techniques for dead code removal, thereby ensuring neat and effective wound healing.

All of these stages are described in detail in the following paper. The authors illustrate the application of their automated patching techniques and give compelling results to show how it finds and fixes real faults. Given the enormous costs of software faults mentioned earlier, it is surely impossible to overstate the potential impact that this work may have. C

a Source: SBSE repository at http://www.sebase.org/sbse/publications/.

References

1. Afzal, W., Torkar, R. and Feldt, R. A systematic review of search-based testing for non-functional system properties. *Info.Softw.Tech. 51*, (2009), 957–976.
2. Ali, S., Briand, L.C., Hemmati, H. and Panesar-Walawege, R.K. A systematic review of the application and empirical investigation of search-based test-case generation. *IEEE Trans. Softw. Eng.* (2010, to appear).
3. Arcuri, A. and Yao, X. A novel co-evolutionary approach to automatic software bug fixing. In *Proceedings of the IEEE Congress on Evolutionary Computation* (Hong Kong, June 1–6, 2008), 162–168.
4. Harman, M. The current state and future of search based software engineering. The Future of Software Engineering. L. Briand and A. Wolf, Eds. *IEEE CS Press*, Alamitos, CA, 342–357.
5. McMinn, P. Search-based software test data generation: A survey. *Softw. Testing, Verification and Reliability 14*, 2 (June 2004), 105–156.
6. National Institute of Standards and Technology. *The Economic Impacts of Inadequate Infrastructure for Software Testing. Planning Report 02–3*, May 2002.
7. Räihä, O. A survey on search based software design. Tech. Report Technical Report D-2009-1, Dept. of Computer Sciences, University of Tampere, 2009.

Mark Harman is a professor of software engineering and leads the Software Engineering Group at King's College, London. He is also director of the Centre for Research on Evolution Search and Testing (CREST).