Automated Software Reliability Services
Using Reliability Tools Should Be As Easy As Webmail

George Candea, Stefan Bucur, Vitaly Chipounov, Vova Kuznetsov, Cristian Zamfir
School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Software quality assurance (QA) is boring, resource-hungry, labor-intensive, takes a long time, and is prone to human omission and error. While research has offered a wealth of new techniques, few made satisfactory progress in *all* these directions, so industry still relies on relatively primitive techniques for software testing and debugging. Little has changed in QA practice over the last 20 years.

Studies have found bug density in production-ready software to be staying constant over time, while average code volume of software is increasing exponentially, with the net effect that the number of bugs per product is increasing. It is therefore necessary to quickly find a "disruptive technology" to reduce bug density by at least 10×.

We see two key challenges: First, few of today's tools can *thoroughly* handle real-sized software (i.e., 1 million lines of code or more), mainly due to high CPU and memory requirements. Second, there is little *incentive* to produce bug-free software, given the exorbitant cost of doing so—in the words of an software architect at a major software firm, "as long as we don't end up on the front page of the New York Times, our software is reliable enough." Software providers compete on functionality and performance, but don't care much about reliability.

To reduce bug density by an order of magnitude, we must (a) ***lower the cost*** of good QA, and (b) provide the market with mechanisms that ***encourage and reward*** reliable software. Our research agenda therefore aims to build (a) techniques that leverage massive clusters of commodity hardware to make automated testing and debugging fast and cheap for real-sized software, and (b) service models that enable consumers to assess and demand software quality, as well as enable developers to deliver it.

Our vision is AutoSRS—reliability-enhancing tools as simple cloud services. AutoSRS is to the status quo in software QA what web-based email is to installing, configuring, and managing one's own sendmail server. We want reliability services to make developing dependable software as easy as using Webmail to communicate. AutoSRS *automatically* tests/debugs/validates/certifies software, without human involvement from the service user's or provider's side.[1] AutoSRS combines the concept of reliability tools as a competitive, simple web service, with doing automated testing and debugging in the cloud, thus harnessing vast, elastic resources toward making such automation practical for real software. AutoSRS consists of:

1. *Programmer's sidekick* that continuously and thoroughly tests code as it is written, with no upfront investment from the developer;
2. *Debugging service* that turns bug reports into 100% deterministically replayable bug scenarios;
3. *Home edition* on-demand testing service for consumers to verify the software they are about to install on their PC, phone, TiVo, camera, etc.
4. *Certification service*, akin to Underwriters Labs, that independently (and automatically) assesses the reliability, safety, and security of software.

Automated software testing/debugging/validation/certification, available to anyone and everyone at low cost, could transform the current software business model in which users must have blind faith in developers, and developers must invest more than half their time in QA.

Programmer's sidekick: Software testing essentially consists of exercising as many paths through a program as possible, and checking that certain properties hold along those paths (no crashes, no buffer overflows, etc.). In its simplest form, the programmer's sidekick service operates "in a loop" that pulls the latest code from the developers' repository, exercises the various paths through the code, and checks them against a collection of test predicates. Continuous testing integrated into the development environment allows developers to provide higher level specifications of what should be tested: instead of imperative test suites, they write test predicates, which takes considerably less human time. This reduces the developer's burden and allows checking deeper properties faster, by using the resources of the cloud.

[1] Note that this is substantially different from today's "testing as a service" businesses, which employ low paid humans to write tests.

Predicates over program state or control flow characterize undesired behaviors, potentially using abstract, symbolic program state to specify computation properties; e.g., "if ever $factorial(\lambda) \neq \lambda * factorial(\lambda - 1)$, that is a bug." A testing service smartly exercises as many execution paths through a program as possible and checks whether any paths trigger these test predicates. Test predicates fall into two categories: First, universal predicates are broadly accepted as describing bugs (null pointer dereferencing, deadlocks, race conditions, memory safety errors, etc.). Second, application-specific predicates capture semantics that are particular to the tested program (e.g., $numConnections > maxPoolSize + delta$). The goal is to integrate predicate upload and testing into developers' IDEs, and make such continuous testing be as indispensable as automatic spell-checking in word processors.

We are building a system, called Cloud9, which aims to scale symbolic execution—a popular test automation technique—to large clusters. Doing automated testing in a cloud instead of on individual developers' machines increases the available compute power by orders of magnitude, thus alleviating the memory and CPU bottlenecks.

Debugging service: Debugging real systems is hard, requires deep knowledge of the code, and is time-consuming. Developers turn into detectives searching for an explanation of how the program could have arrived at the reported failure point. We developed a technique, called execution synthesis, for automating this detective work: given a program and a bug report, it automatically produces an execution of the program that leads to the reported bug symptoms; it can be played back deterministically in a regular debugger, like gdb. The challenge now is to use large clusters to debug real bugs in seconds.

Testing @ home: We also wish to empower consumers to be in control of the quality of the software they use. Consider the following scenario: Mrs. X, a grandmother who lives by herself, relies on her mobile phone to notify her children whenever she experiences the symptoms that precede her seizures. Her mobile phone operating system recently notified her that it needs to be upgraded, to improve the speech recognition component. Mrs. X uses a testing website for end users to check the software upgrade; within minutes, the service produces a webpage with the results of the test, indicating whether it found any serious bugs. Mrs. X allows the phone to update itself only if the test report says no serious bugs were found.

This scenario is not far-fetched: we have built a standalone tool, called DDT, for testing closed-source binary device drivers. In preliminary experiments, DDT tested six mature Windows-certified closed-source binary drivers for less than 5 minutes each and found 14 different serious bugs. We now need to parallelize DDT on clusters.

Certification service: A public certification service provides an objective assessment of a software product's quality. It analyzes software (either in binary or source code form) and, for each defect found, publishes irrefutable evidence of the defect. Based on the defect density, the service rates each product. Consumers use the ratings to make their purchasing decisions, thus motivating software vendors to compete (also) on reliability.

Feasibility of AutoSRS: AutoSRS providers benefit from economies of scale, as users are likely to test common bodies of code (e.g, many programs use the same libraries). The AutoSRS provider can exploit this redundancy by not re-testing already-tested code, thus amortizing the cost of the first test run. More users means more exploitable redundancy.

On the path to AutoSRS, there are both technical and non-technical challenges. We must find ways to scale automatic testing to thousands of loosely-coupled machines. We need incremental testing techniques that reuse existing test results and compose them with tests focused on new or modified code. We need metrics for quantifying the level of confidence we get from a test suite, which go beyond mere line coverage. We need to aggressively parallelize constraint solvers, a key component of automated test engines. We hope the vision presented here will motivate researchers to design their techniques such that they can be plugged into such cluster-based reliability services.

An open AutoSRS framework can provide a platform for research on reliability techniques, enabling rapid transfer of research results into practice. Community efforts can produce databases of smart test predicates, in the spirit of Wikipedia and Knol. AutoSRS can run on public clouds, like Amazon EC2, on private clouds inside organizations, or even on "cooperative" clouds. A cooperative test cloud is a federation of user machines, akin to SETI@home, in which end users dedicate spare cycles to testing the software they are most interested in. Unlike SETI@home searching for extraterrestrial intelligence, cooperative cloud testing can yield results sooner and be more gratifying to those who provide the cycles. The key to enabling a cooperative test cloud is devising symbolic execution algorithms that scale with potentially an order of magnitude more users, higher churn and network latency, as well as more limited network bandwidth.

In summary, a combination of technical and non-technical forces could achieve order-of-magnitude reduction in bug density. Cloud-based automated testing techniques, together with simple web-service interfaces, can make high-end testing and debugging accessible to all developers and consumers at low cost, or even for free.