coverity®

WHITE PAPER

Transforming Testing Through Automation

The Story of Coverity's Journey to Automate Testing as Part of Development

by Andreas Kuehlmann
Senior Vice President of Research and Development, Coverity

People, process, and technology are critical for succeeding in the software development business, which is increasingly characterized by fierce competition, rapid product delivery cycles and relentless demand for more differentiating features that are secure and high quality. The key is to stay in the "driver's seat" and constantly balance the short-term value of feature delivery with the long-term ability to efficiently expand and maintain the product. In our experience, automating testing as much as possible and interleaving test and feature development are critical elements to succeed in this venture. We are in the middle of transforming our own organization – from one that relied heavily on an outsourced, manual Quality Assurance (QA) operation, to one that automatically tests the code as it is developed. This transition involved changing the skill set of the QA team and adopting new processes, as well as innovating with new technologies and solutions that help us to focus our test development.

Software development is a relatively young engineering discipline and its procedures are still evolving to satisfy the increasing business demands for a manufacturing process that delivers high-quality and secure software products in a sustainable and predictable manner. Much has been written in painstaking procedural particularities about the evolution from "Waterfall" to "Agile" development methodologies. One may argue whether it is important to call the team meetings "standups" or span the "sprints" over two weeks or four weeks. However, one uncontroversial tenet of the "Agile" philosophy is that "you test your code as you write it."

The very presence of a large Quality Assurance (QA) organization may appear to relieve developers of responsibility for testing. Given that the wisdom that developers should be accountable for the quality of their code is not new, one may question the emergence of separate QA organizations with the sole responsibility for quality. Clearly, the QA team's independence from the development process is critical for an unbiased and holistic view of quality from an end-user's perspective. Such an independent end-to-end checkpoint is an important stage-gate in shipping high-quality products. However, too often QA teams get bogged down in finding low-level software bugs that should have been found much earlier in the development process. The resulting frequent iterations between QA and development are the root of two key issues: first, it is difficult to predict how fast the iterations will converge, thus making the timeline for delivering a desired level of quality unpredictable. Second, the long latency and high frequency of the iterations dramatically increase the overall cost of software development.

The value of "testing your code as you write it" is mostly understood and often aspired to by development organizations. However, the reason why it usually remains merely a lofty goal is the lack of metrics that provide a crisp criterion for "adequately tested by development." In the absence of such a criterion, QA teams don't know what quality level they receive from development and similarly developers – under intense pressure to deliver new features – get away with passing on code that was not tested enough. Many organizations closely monitor bug rates and compare them with historical data. Yet such metrics offer only an indirect view of development diligence, thus limiting their use to reactive firefighting without fundamentally attacking the issue. In an Agile development process, the passing of an "acceptance test" for a feature, which is required before a developer can move on, provides a finer-grain stage-gate to ensure interlock between development and testing. However, in the absence of a measurable criterion for the adequacy of such a test, its thoroughness remains subjective, often leading to a practice where a successful execution of a "happy path" is declared as sufficient for a feature to be accepted.

Many software development organizations today have adopted Agile development practices to one extent or another in order to accelerate product delivery and time- to-market. Yet many of them are finding that they still have trouble getting software built as quickly and reliably as needed. In this paper, we describe our own journey to accelerate our product development, to make it more predictable and to deliver higher quality software through increased test automation. We will concentrate our discussion on two testing areas which needed significant rework: the functional testing of Coverity® Connect, our issue management console, and the end-to-end testing process as a whole. Moreover, we will focus on the changes we've made in the last two years and outline some of the work that is still ahead of us.

One Organization's Journey

Our release development cycle for product delivery is six months, adopting elements from a Waterfall process as well as Agile principles such as Scrum and two-week Sprints for building features. We utilize the last six weeks of the release cycle for "hardening" the quality in three two-week stages: "Feature Freeze", "Integration Freeze," and "Release Freeze." This process is a fairly common practice in many enterprise software development shops.

Two years ago, the state of our test automation was highly heterogeneous. The "front-end" component of our product (the part which compiles the customer code), as well as the "analysis" component (the part which identifies potential problem areas in the code), had a fairly good practice of developers building automated tests and delivering them as part of the feature delivery. On the other hand, Coverity Connect was developed utilizing a mostly manual QA process. Similarly, the end-to-end (E2E) QA process was done almost entirely manually. Figure 1 gives a high-level overview of the product components and the functional testing approach as it was applied two years ago.

Both manual QA tasks, the functional testing of Coverity Connect and the E2E testing, were outsourced to Eastern Europe

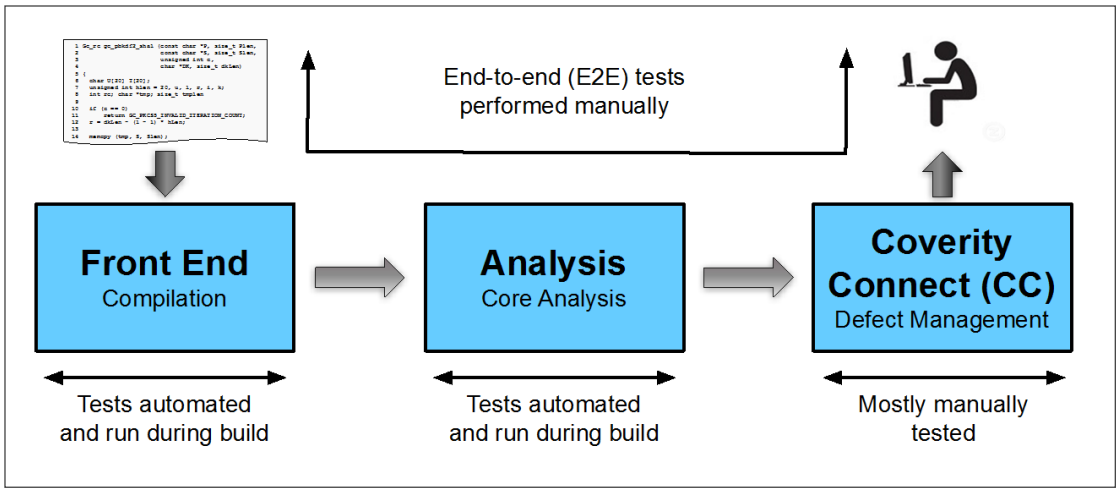


Figure 1: High-level overview of the product components and functional testing approaches as of Summer 2011.

to a team that had limited access to the product source code and operated with a 10-hour time difference. The QA process was not started until the features were "declared" complete. Because there was no crisp criterion for completeness, the quality of the features was unknown at the time of Feature Freeze and the number of outstanding bugs and the effort required to address them was highly unpredictable. This uncertainty regularly caused "crunch time" during the release hardening cycle, with many overtime hours spent by developers and the QA teams while anxiety spread throughout the entire organization to deliver the release on time. Moreover, this process failed to assure the overall quality of the product to which we aspired.

The distance between our development team and our testing team—culturally and geographically–aggravated the problem. Long communication cycles, language barriers and varying levels of product familiarity added to the unpredictability of the release hardening process.

It was clear that the existing process was not working and would certainly not scale for building more features, faster. We were running too many tests manually and lacked meaningful metrics to tell us if we were on track for quality at each release milestone. So while we claimed to have Agile processes, our lack of test automation and interleaving of feature development and testing resulted in problems once we reached the end of the development cycle.

Big Changes

Things had to change. We assessed our existing processes and concluded that the problem lay not in the Agile development philosophy per se, but in the disconnects between the culture and methodology of the development organization and the QA team for Coverity Connect as well as for E2E testing. The way to eliminate the separation was to automate more of the testing and interleave it with the development process.

Our first step was an R&D re-organization that shifted QA resources into development and consolidated responsibility for testing in the development team. In Spring 2011, we moved two thirds of the QA resources residing in Eastern Europe into the Coverity Connect organization and left one third to focus on E2E testing. This move quickly proved to be insufficient: the remote location of the QA staff and inconsistent practices between the two locations still slowed down the work. Moreover, the former QA engineers, now part of the Coverity Connect team, had limited development experience and had difficulties automating tests in an effective manner. We were going to have to do more than tweak the organization.

"Test Automation" was going to be our new catch phrase. We phased out the Eastern European operation altogether and moved the resources to Calgary, Canada, where we already had a small office only one time zone away from our San Francisco office, and where lines of communication would be shorter and

language differences could be overcome. This restructuring, which took place over the period of one year (from August 2011 until September 2012), was the general framework within which we would deal with the specific issues of people, process and technology.

People

The first consideration was what new skills would be needed to achieve automated testing during development. Instead of hiring individuals with traditional QA experience, we hired people with programming skills. Fluency in programming was necessary for writing automated tests, but it also provided a future opportunity to move individuals between the roles of test engineer and development engineer as part of their career growth. Contradicting conventional wisdom, the transition from Eastern Europe to Calgary was executed in a mostly budget-neutral manner. With only a slight increase in cost, we replaced 12 QA engineers from Eastern Europe with 11 software engineers in Calgary (Figure 2).

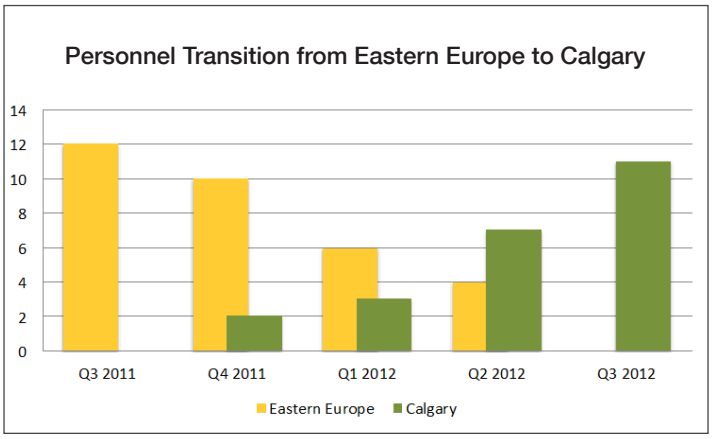


Figure 2: Transition of testing resources from Eastern Europe to Calgary, Canada.

The most critical human factor was getting the development team to take responsibility for the testing side of the equation. This wasn't as difficult as it might seem, since the new structure better enforced accountability for quality and thus made the development organizations their own customers, reaping the direct benefits of investing in testing. The appreciation of effective testing was a strong incentive to automate the process to the highest degree possible. Similarly, the E2E QA team was now composed of software engineers with strong programming skills. They were highly motivated to replace the mundane tasks of manually applying hundreds of QA tests under high time pressure with automation.

Part of the transition from Eastern Europe to Calgary was to ensure long-term sustainability. We developed a relationship with the Computer Science Department of the University of Calgary, and now participate in job fairs, sponsor programming contests and provide technical lectures to build a constant supply of highly qualified interns and future hires.

Process

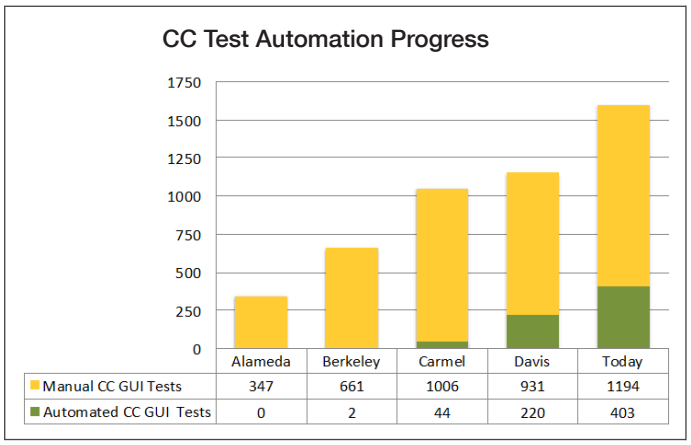
Our big goal was to "test the code as it is developed." The manual testing approach, which was prevalent in Summer 2011 for functional Coverity Connect testing and E2E testing, was fundamentally preventing this.

The first stage to achieve the bigger goal was to increase test automation. Automating tests has two key benefits: first, the tests encode the expected behavior of a feature; executing the tests validates that the code does in fact implement that behavior. Once passed, the feature can be declared complete. Second, after the feature development is complete, the tests prevent regressions by identifying when a feature is unintentionally affected by other work. Test automation can catch such regressions quickly when executing the tests at regular intervals. As a result, the tests serve as "guardrails" for future feature development or architectural rework. A solid test automation-based development process becomes increasingly critical as the complexity of the software increases, teams grow and code ownerships change. It should be noted that, depending on the way features interact, not all tests can be developed in lockstep with the features. Some tests have to be built after sets of interacting features are complete.

Coverity Connect is a web application with a traditional split between a back-end for the server-side operations and a front-end for the web-client. For the back-end, we have a process in place where developers write automated unit tests as part of the code development. Over the past two years, we continued to improve the unit test coverage by increasing the number of tests by 34%, from 2,346 to 3,148. On the other hand, in Fall 2011, we had almost no automated tests for the front-end and all functional tests were performed manually from the browser GUI. Changing that was one of our key initiatives. Our first investment was to build a comprehensive infrastructure that combined

[1] *http://www.seleniumhq.org*

[2] *http://www.testng.org*



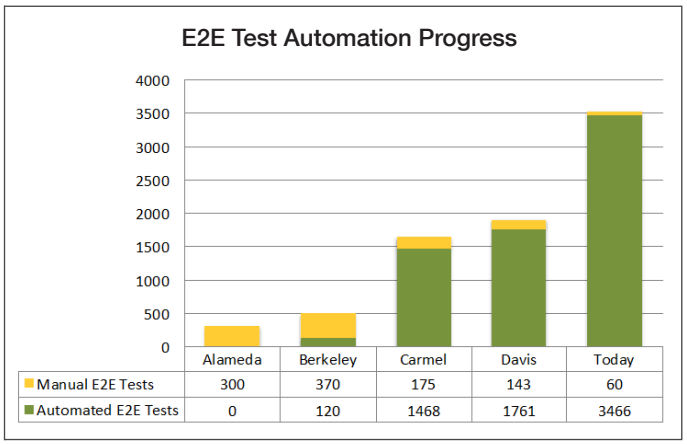
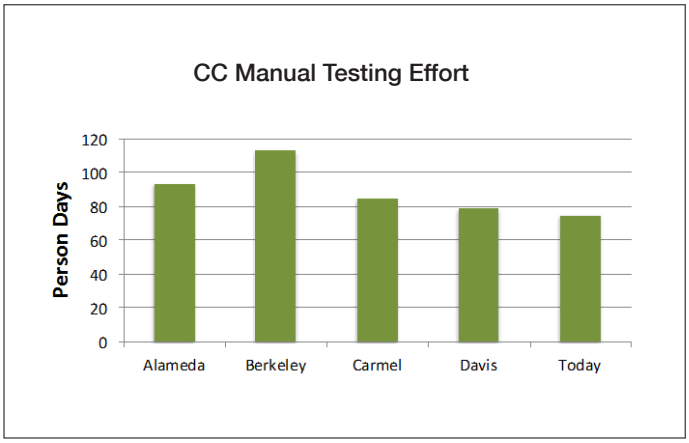


Figure 3: Progress of developing functional Coverity Connect (CC) tests and end-to-end (E2E) tests and automating them from the Alameda release (Fall 2011) until today (Spring 2013). The terms "Alameda," "Berkeley," "Carmel," and "Davis" refer to internal code names of major product releases.

Selenium WebDriver[1], TestNG[2] and internally developed test automation technologies. The key was to create an architecture that was simple to use and allowed test development to scale across the entire application, as well as the organization. Once the infrastructure was in place, we started developing the automated tests on top of it. While catching up with automation, we increased the overall number of GUI-driven functional tests for Coverity Connect to ensure more rigorous testing.

Figure 3 depicts the increase of the total number of functional tests for Coverity Connect and E2E from Fall 2011 until Spring 2013. For Coverity Connect, the number of GUI-based functional tests grew by a factor of 4.5. Once the infrastructure was in place, we quickly raised the level of automation. Today 25% of the Coverity Connect functional tests are fully automated; we expect to finish most of the automation over the next 9-12 months. It should be noted that our test automation infrastructure not only reduces the testing time, it also allows for significantly more diverse test coverage. For example, the Selenium-based GUI tests, once automated, are used for all supported platforms and web browsers. This breadth would be prohibitively expensive if attempted manually. For the E2E testing, we made rapid progress toward automation – today 97% of these tests are automated, including the tests for license validation and web-services. The remaining E2E tests are the ones that are hard to automate and require some form of manual inspection, e.g. visualizing the correct layout of a browser page.

Figure 4 shows the resulting decrease in the manual effort to execute the tests. Today, we can complete a full round of E2E tests in approximately two days, involving three test engineers. This allowed us to broaden the scope of testing and helped reduce the time to harden the latest maintenance release from two weeks down to three days.



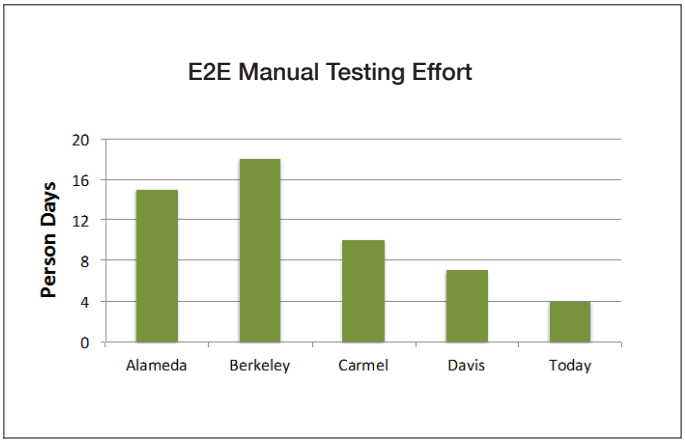


Figure 4: Reduction of labor effort to execute one round of manual Coverity Connect and E2E tests.

The second stage toward the bigger goal is to interleave test creation with feature development. Ultimately, a set of passing tests should be in place before a feature is declared accepted in the Agile development process. This ensures the most efficient development of the feature as it forces the developer to correct issues while the code is still in his or her mind. Moreover, it also provides detailed insight into the progress of the release and allows management to take corrective actions if features are late, e.g., because the scoping underestimated the effort to implement them.

While developing unit tests has always been contemporaneous with code development, we have just started working on introducing Coverity Connect functional tests and E2E testing earlier in the release cycle, as each feature is completed. Key to making this transition successful is a close collaboration between feature developers and test developers, and a product architecture and development schedule that detangles the features to be developed during the release, so that they can be finished and tested in isolation. Due to a major re-architecting effort of Coverity Connect over the past few releases, the second issue has been challenging and prevented a lock-step process thus far. As this rework subsides and test automation catches up, we will increasingly tighten the interleaving of feature and test development. To drive this process, we are utilizing our new Coverity Test Advisor technology, which is described in the next section.

Technology

Clearly, investing in test automation requires significant tools and technology for developing and executing the tests. For the build process, we utilize Continuous Integration[3] using the Jenkins[4] platform. The majority of the unit tests are executed as part of the build. Besides the 3,148 unit tests for Coverity Connect, the front-end and analysis components have 10,215 tests that are executed as part of the build. For executing more complex tests, we built an in-house infrastructure called Scenario Test System (STS) which schedules the test execution on a test farm in a tiered manner. STS executes the automated functional GUI tests for Coverity Connect and the automated E2E tests, as well as 1,978 more complex tests for the front-end and analysis components. Over the last two years, we significantly extended the resources for test execution such that we can now run a complete set of tests in a half-day turn-around for the supported Tier 1 platforms. Figure 5 depicts the increase of the testing hardware resources over the last two years. Note that a large part of our test execution platforms are virtualized.

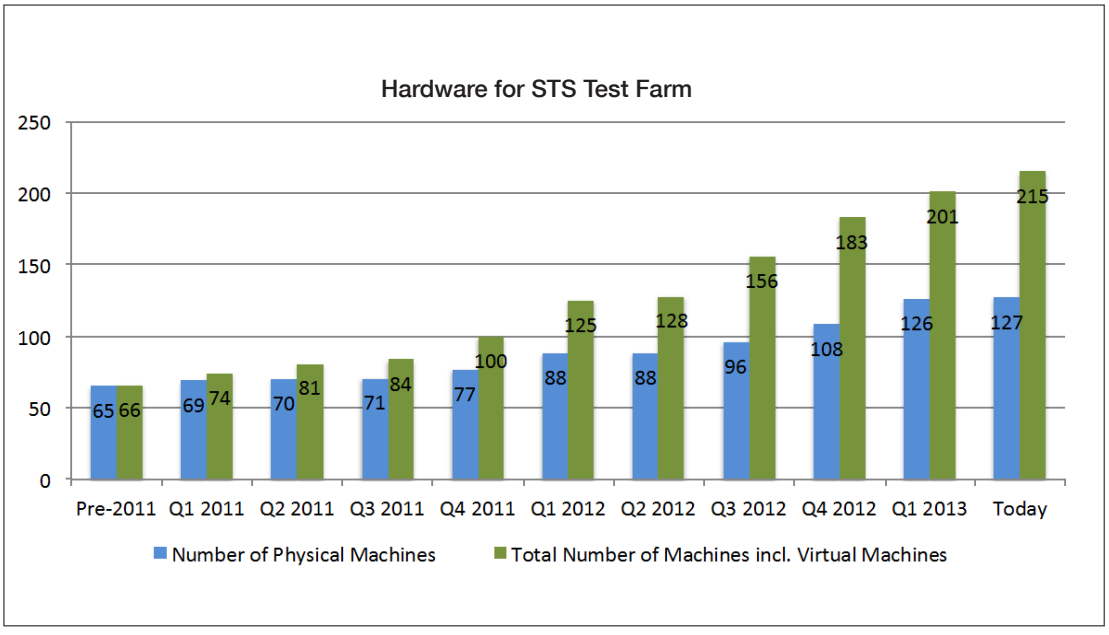


Figure 5: Growth of the STS test farm over the last two years in terms of number of physical and virtual machines.

[3] *http://en.wikipedia.org/wiki/Continuous_integration*

[4] *http://jenkins-ci.org*

As part of our regular code development process, we use our own product as "Product on Product" (PoP). All defects found by Coverity Quality Advisor and Coverity Security Advisor are triaged and fixed by developers on an on-going basis, and the removal of all critical defects is a key element of our release criteria. The fully automated defect detection capability of our static analysis makes this a highly cost-effective way to eliminate bugs early in the process. The types of bugs uncovered by Coverity Quality Advisor and Coverity Security Advisor are highly complementary to the ones found with traditional testing. This is because static code analysis performs a much more extensive search of execution paths for general defect patterns that cause crashes, data corruption, memory leaks or other, severe bugs in the product. On the other hand, traditional dynamic tests are focused on finding functional bugs by validating the intended behavior encoded in the tests. This functional intention of the code cannot be inferred by static analysis. Figure 6 gives an overview of the number of defects we found and eliminated as part of our PoP process.

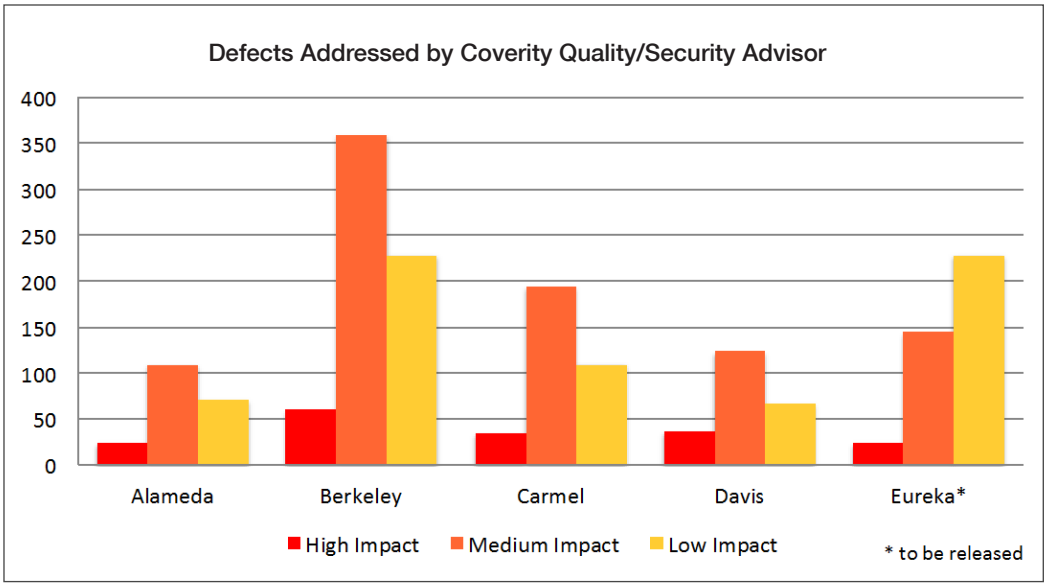


Figure 6: Defects found by Coverity Quality & Coverity Security Advisor and fixed during development.

As our effort to increase test automation as part of the feature development progressed, it became quickly clear that we needed a way to direct our test development efforts to the most-needed areas. Code test coverage is a common proxy to assess test efficiency. In 2011, our code coverage for the front-end and analysis components was approximately 83%, a number that is generally considered as very good. However, we had no understanding of what was hidden in the uncovered 17% and what additional tests we should develop. Moreover, when new code was added or existing code was changed, code coverage did not provide any guidance as to what tests should be added. This was particularly troublesome as these changes are the most risky part of a new release and should be the focus of limited testing resources. This insight led us to explore whether we could utilize our deep understanding of the source code semantics to determine the impact of code changes for guiding effective test development.

In Spring 2011, we performed a small experiment on the C/C++ code of our front-end and analysis components to validate the envisioned principle. Our coverage indicated that of the 1,372,376 lines of code, 83.7% were covered by tests, leaving 223,340 lines untested. Developing tests for covering all this code would be a significant investment and require many person-years of effort, resulting in a temporary stop of feature development.

Our experiment demonstrated that when overlaying the untested code with the updates made during the recent releases, the amount of code requiring more tests was relatively small and developing tests for them became feasible. Figure 7 shows the results of our experiments. When focusing on the most recent changes introduced since the last release, only 851 of the 223,340 lines of uncovered code would require additional tests.

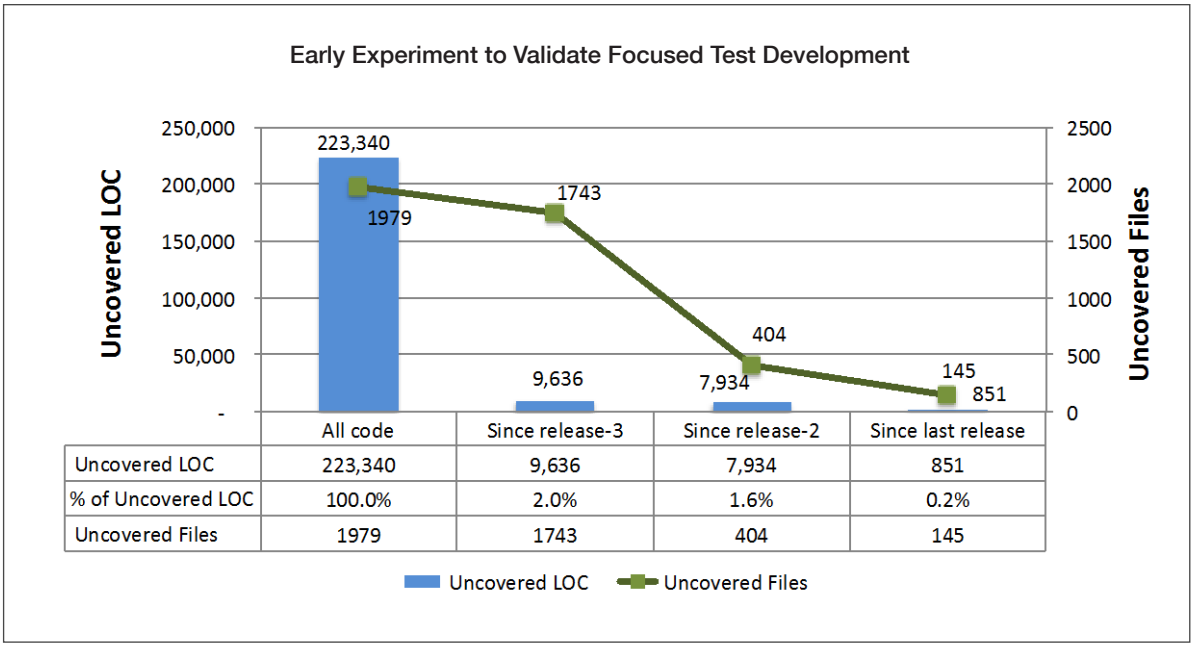


Figure 7: Results of an early experiment to demonstrate value of focusing test development on risky code changes. The chart shows the reduction of code requiring tests when zeroing in on latest changes.

The insight from this experiment led to the development of a new solution for our in-house usage, which also turned into a new product: Coverity Test Advisor. With Coverity Test Advisor, development teams can specify a test policy, which defines on a fine level of granularity what code is most risky and needs to be tested thoroughly. Key to this is our static analysis capabilities which can identify functional impact of code changes, dead code segments and untestable code, as well as specific code fragments that should be excluded from testing, for example debug code or certain exception handling. Insufficient testing is translated into actionable work items for developers or test engineers and management can track progress of closing them out. Coverity Test Advisor provides the above-discussed missing link for an Agile process by establishing a feature acceptance criteria of "adequately tested."

We started full development of Coverity Test Advisor in Summer 2011. In Spring 2012, a first prototype was utilized in an internal experiment for the front-end product component. We asked a test engineer to spend exactly four hours per week triaging Coverity Test Advisor violations and develop tests for the high-risk areas. Figure 8 provides the results of this experiment. After 12 weeks, the test engineer had added 29 tests and uncovered 19 bugs – some of them severely impacting particular compiler functionalities. It should be noted that approximately half of the 19 bugs were found by simply inspecting the high-risk code areas during test development and finding discrepancies or inconsistencies.

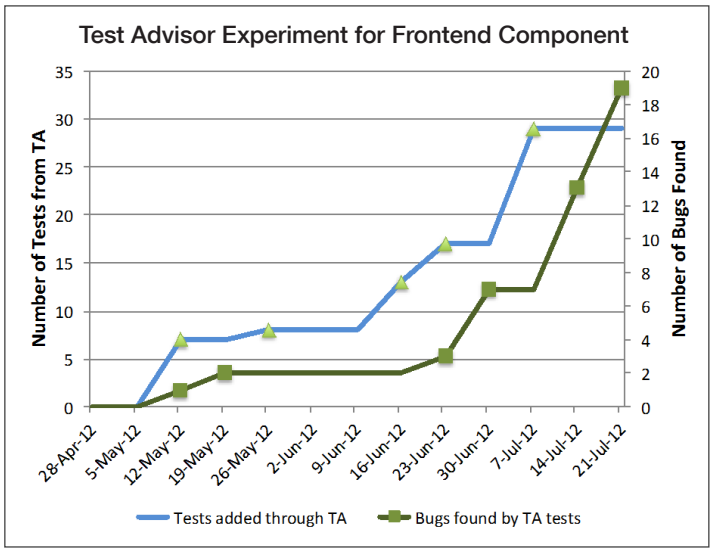


Figure 8: Results of controlled experiment of using Test Advisor. 19 bugs were found with 29 focused tests developed in 48 person hours.

We launched Coverity Test Advisor as a new product in Fall 2012. Since then we have been adopting it for full production for all our product components to identify testing weaknesses and to focus our increased test development.

Results and Future Outlook

Our initiative to increase test automation and move testing into development has shown significant progress and benefits in terms of quality and reliability of product delivery. Figure 9 shows the decrease of the normalized customer-found defects for the last releases. Given that over this period, the number of customers as well as the breadth of our product functionality has significantly increased, this trend indicates a substantial improvement of our test effectiveness. Moreover, we were able to shorten the hardening of a maintenance release from two weeks to three days – making the release of hotfixes and small product updates significantly more efficient. Compared to the past, our major releases are not only delivered more predictably, but they have more features and include entirely new product components that are validated with the same or fewer resources.

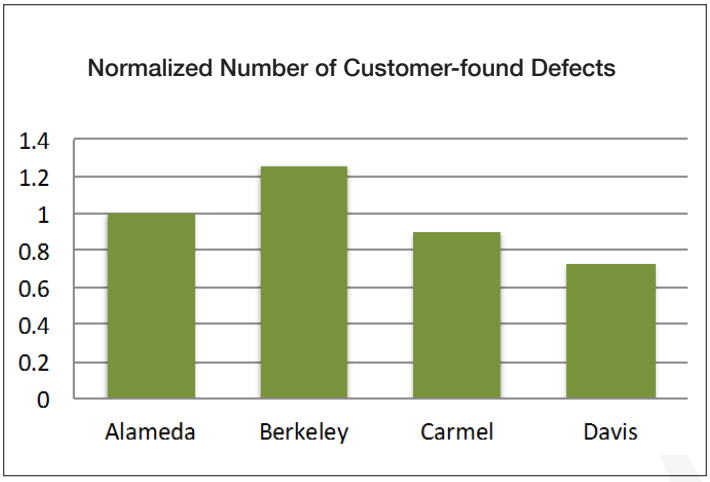


Figure 9: Number of defects found per week by all customers normalized to the "Alameda" release. The number of customers has significantly increased over this period.

Our transformation from a QA-based development process to one that fully interleaves feature development and automated testing is not done yet. We have mostly finished our E2E test automation and built an effective infrastructure for test automation for Coverity Connect. We are on track to fully close the automation gap for Coverity Connect in the next 9-12 months, and make the development of tests an integral part of the code development. Coverity Test Advisor is evolving into an essential solution to focus our testing effort and to gauge the progress of the release cycle by ensuring that features are only accepted once they comply with all Coverity Test Advisor policies. Our vision is to shorten the time required to harden major release cycles to less than a week, and be able to demonstrate product functionality on the fly from any build.

As we evolve our own development process, we will continue to convert lessons learned into innovations and compile these into new products and solutions that not only help us, but also our customers, to build better software. Our vision is to utilize our deep code intelligence derived from static analysis to automate all mundane development and testing steps, and transform software manufacturing into a predictable and cost-efficient engineering process that delivers high-quality products.

Stay tuned for more innovations to come from Coverity!

Acknowledgements

The entire Coverity R&D team has been instrumental in making the transformation described in this paper possible. Special thanks go to Marat Boshernitsan, Johnny Chan, Dino Delyani, Menwin Gatus, Zohar Hirshfeld, Mel Llaguno, Scott McPeak, and Kit Transue, as well as the Coverity Connect and E2E test automation teams for their tireless passion for quality and their effort to make testing part of our daily development work.

coverity®

For More Information
www.coverity.com
Email: info@coverity.com

Coverity Inc. Headquarters
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: (800) 873-8193
International Sales: +1 (415) 321-5237
Email: sales@coverity.com