



Smartphone Software Quality: Detecting and Eliminating Bugs Critical to User Satisfaction

Technology research firm, Gartner, Inc., reports that smartphones now account for over half of mobile phone purchases in North America and Western Europe. As consumers increasingly upgrade to smartphones from “feature” phones, device performance has become the leading determinant of how brands are perceived. Performance problems lead to churn away from the manufacturer or network operator. The importance of smartphones in users’ lives, aided by social media, turns issues like antenna performance or software glitches from annoyances into headlines. And ultimately into a significant impact on the bottomline.

The purpose of this paper is to suggest strategies to avoid what industry watchers have been warning: Major problems, unseen till now, are likely to emerge in the smartphone market due to increased dependence on software, coupled with a lack of adequate software quality solutions and testing in the smartphone industry.

The intense competition, both at the top and bottom of the smartphone market, acts as a compounding factor for device manufactures. Vendors continue to drive technology enhancements, on thinner and thinner profit margins, while attempting to outdo each other in time-to-market. Unless new methods are adopted to ensure software integrity, we believe this set of circumstances will result in high profile quality failures, with potential market share decline.

Mobile Device Makers Hold the Key to Consumer Perceptions of Quality

Recent studies suggest over half of users are disappointed with the overall performance of their smartphone. Nearly a third experience “continual” problems with newly acquired applications, and a majority of users have required some form of software patching to fix issues on their smartphone. A majority of users blames the handset manufacturer and is eager to make their dissatisfaction public via social media.

End users say they would prefer to change handsets than to change networks. But it is clear all major players in the mobile sector must have a hand in implementing solutions to address the rapid growth of software defects.

Previously Fixed Bugs Pose the Largest Risk to Handset Quality

The accidental release of previously fixed bugs back into production is a serious problem overlooked by many in engineering management in the mobile industry. But the facts are clear:

- > Up to 45% of software bugs are previously fixed bugs¹
- > Nearly 50% of software releases in the field contain known security vulnerabilities²
- > On average, 10% of the bugs fixes made in development do not make it into the final release. (This is based on field work—both at the University of Illinois, and with customers at Pattern Insight™.)

To compound the problem, bugs that have been previously identified and prioritized to fix, are usually higher in severity. One of the biggest mistakes an engineering organization can make is to re-release a Priority-1 bug that has been previously reported and fixed.

¹Nam H. Pham, Tung Thanh Nguyen, Hoan Anh Nguyen, Tien N. Nguyen, Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, Tien N. Nguyen: Recurring bug fixes in object-oriented programs. ICSE 2010, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering.

²Nam H. Pham, Tung Thanh Nguyen, Hoan Anh Nguyen, Tien N. Nguyen, Detection of recurring software vulnerabilities, ASE '10, Proceedings of the IEEE/ACM international conference on Automated software engineering.

How Previously Fixed Bugs are Released: The Android Case

Since its launch in November 2007, the Android mobile market has grown with unprecedented speed. Not simply measured in the number of users or applications, the boost in the number of companies developing and building smartphone handsets has outstripped most analysts' predictions. Google claims that over 150 million Android devices have been activated worldwide, with "over 550,000 devices now lit up every day" per Google CEO, Larry Page this August, 2011.

A network of about 39 manufacturers and 231 carriers in 123 countries develops the devices that include smartphones, tablets and eReaders. The Android ecosystem puts tremendous pressure on mobile device makers to rapidly develop new models in order to stay competitive and satisfy growing global needs.

**Android Sales Worldwide
2008 – 2011 (Projected)**

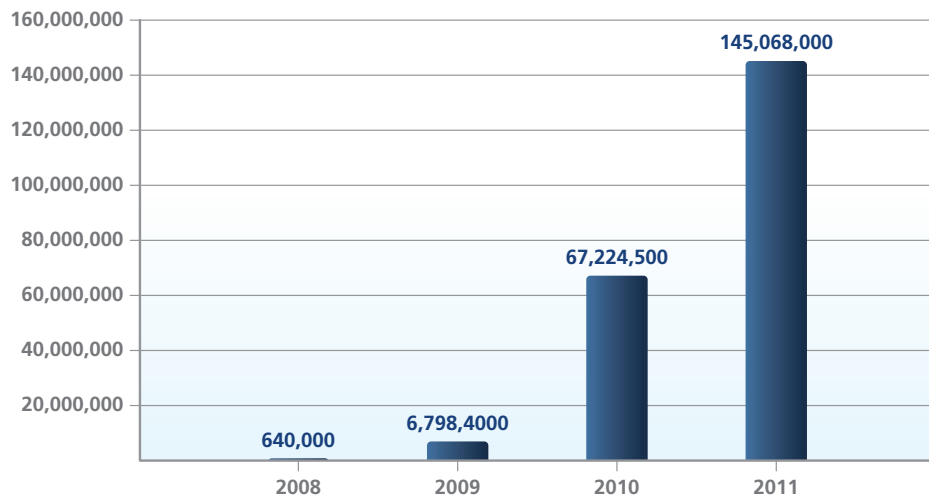


Table 1: WW sales in Android handsets, 2008 – 2011, based on Gartner Research statistics.

Time-to-market has come to mean extraordinarily short turnaround times with no margin of error in quality within this extremely competitive industry. When it comes to software development activities that enable this explosive growth, large-scale component and software reuse has become the chief means to deliver the numerous models of handsets to customers in this expedited environment.

Let's examine one example. Samsung is now considered a high-growth player in the Android space. It has achieved this by releasing many models in a short period of time: Nearly 40 Android smartphone models have been launched in the U.S. market alone since 2008. These models are developed to satisfy different carriers' marketplace demands, appeal to different user needs, hit predefined price points, etc.

Two models, in particular, offer an insight into the diversity of the product range necessary to compete in this space. Both are based on Google Android OS 2.2 (Froyo) and were released in May 2011. The Samsung Droid Charge targets high-end users with a \$659 price tag and runs on Verizon's 4G LTE network, while the Samsung Infuse 4G sells for \$199, and is supported on AT&T's HSPA+ network. And like the rest of their 40 models, these two are U.S. specific.

Clearly, to deliver smartphone handsets in today's market means mobile device vendors must work on hundreds of models simultaneously. Mobile development teams use a technique called **parallel development streams**, releasing new handsets in a matter of days.

This development method requires large-scale reuse of code bases or software components across development streams. While this brings down development costs and expedites the development process drastically, it has a downside. Defects are rapidly propagated due to code replication. Further, components are often modified after reuse, making it extremely difficult to track the replicated code.

The combination of staggering market growth and ultra-fast software development shines a spotlight on the handset industry's reliance on manual processes to assure software quality. The result is predictable: Handsets are often significantly delayed or, even more troubling; models are released that don't meet the vendors' quality standards.

Additionally, these three phenomena described below amplify the problem of large-scale code reuse in the mobile industry:

- > **Agile development.** To further accelerate the time-to-market, many mobile device vendors adopt the agile development model that emphasizes quick iterations with incremental development. Therefore, during a short period of time, many replicated artifacts (releases, builds, feature branches, etc.) are created and need to be tracked properly, often for an arbitrary amount of time.
- > **Software supply chain component reuse.** In the mobile world, components are reused not only within an organization, but also across organizations. For example, many Android components are created by Google, but reused and modified by other companies that produce Android devices. As a result, software supply chains have become commonplace, in which code is frequently replicated and reused. It is critically important for a downstream company to manage their third-party code effectively, and for an upstream company not to release buggy and insecure code to their partners.
- > **New development tools.** Most device vendors, in particular Android-based vendors, use Git/Repo as their SCM system to support its component-based development model. Git, is the free, fast and distributed version control system. In Git, branching is the "default" procedure, both because of its distributed nature and also because merging is so fast. As a result, these companies that use Git tend to create and manage many of both ephemeral and persistent branches.

The increasing number of branches creates significant problems including *missing branch patches*, *snippet-level conflicts* and *reuse of code containing bugs* resulting in a far too common scenario—known bugs are leaked into a final release.

Current Tools and Manual Processes Fail in the Time-to-Market Race

Software development teams use engineers to manually verify that the final release contains all necessary changes (including bug fixes). This is an expensive, time-consuming process. Unfortunately, it is error-prone as well. To improve efficiency and reduce defects, companies build in-house tools. These generally fall into two categories: SCM-based or keyword/regex-based.

Here is how they work and why they fail.

SCM-based tools. Modern SCM systems have built-in mechanisms to integrate one change set from one branch to another. If an integration action is taken, it leaves an entry in the SCM log. Based on the history in the SCM log, companies build in-house tools to track where a bug fix goes. Although these help to some extent, they are far from solving the problem for the following reasons:

First, these tools are not automated—requiring developers to do the right thing or key in the right information. It is common for developers not to use the integration command to port bug fixes. Rather, the bug is manually fixed in other branches. In such cases, there is no trace in the SCM log.

Second, SCM systems lack necessary information. SCM systems cannot track duplicated content—if a file is removed and later added back, SCM systems cannot track the changes between these two files. Further, SCM systems are not aware of the syntax/semantics of source code; they treat it as text and cannot understand if two pieces are semantically equivalent.

Third, there are many legacy SCM systems regularly used in development shops. These do not support the notion of change sets, which makes tracking where a bug fix goes nearly impossible, again leading back to manual workarounds.

Keyword/regex-based tools. Another set of commonly used tools includes keyword search or regular expression based tools. They have many significant drawbacks.

First, it's difficult to determine the right query to get the results you want. A bug tends to involve many snippets in different files. How would a simple keyword search or regex-based search capture this complexity?

Second, the results are usually very noisy, since the search lacks context. Search results contain extraordinary volumes of irrelevant results. The overhead of processing them and narrowing down the candidate code snippets is prohibitive.

Finally, it is impossible to be certain that the results are complete. Cases are commonly missed. These include examples where buggy code has been copy-pasted into another area, and then modified.

The ROI Factor: Closing the Case

This paper has mentioned various driving factors behind the mobile handset industry's need to reinvent its approach to software quality. But here is one more significant reason: *It pays big dividends to make the changes urged here.*

Calculations of return on investment in software quality can be difficult to determine and even more difficult to believe. However, the savings are straightforward for smartphone makers who eliminate all instances of previously fixed bugs. The simple calculation model below is based on the number of vendor releases, the number of bugs detected and fixed before each release is given the OK, and the cost to fix bugs before and after release.

Assumptions:

- > 300 releases for a smartphone maker
- > \$500 cost to find and fix bug before release
- > \$5,000 cost to find and fix bug after release
- > 10 bugs detected and fixed before each release

Results:

- > \$50,000 saved per release
- > \$18,900,000 Total Savings Per Year for ~400 versions of phones (across different locales, carriers, etc.)

There are important intangible benefits from adopting the automated approach, as well. These include:

- > Reduced cycle time and faster time-to-market
- > Significant improvement in the quality of rapid releases from reduced time to find and fix bugs
- > Quality improvement leads to customer loyalty and market share gains
- > Engineering resources freed to work on additional features and functionality

But the net ROI that we have seen instance after instance with vendors in the mobile space is a significant bottom line enhancing ratio of nearly 1:20.

The Solution: Automated Insight into Code

Ultimately, the only true evidence of a bug's existence in a source branch resides in the code itself. No matter how much you strengthen manual processes, they break down due to reliance on human behavior. The source code, on the other hand, is absolutely reliable and independent of any engineering process. Therefore, any solution promising to solve the problems described above, must gain "direct insight" into the source code itself.

Here are the key criteria an effective automated technology solution must have:

Capability to identify similar versus identical matches. Code quality research tells us that 2/3rds of the code is modified after the reuse³ – therefore, duplicate code is not exact. A valid solution to this challenge must find every instance of a bug across all versions and branches by identifying similar matches and not just identical ones.

Fast, easy and accurate. The ideal solution must be fast, easy to use, and accurate. Fast in this case means the solution must scale to millions or even billions of lines of code, while still providing response time in seconds across the entire codebase instead of hours or days like many defect detection technologies. Ease of use means the solution must be intuitive for the developer and integrate with the development process and other development tools, notably the SCM system. Accuracy is defined as the degree of precision of the solution necessary to provide engineers confidence that a bug is completely eliminated from the codebase. False alarms are unacceptable as they can compromise the confidence the developers have in the tool.

Adapts to multiple workflow scenarios. Developing software in today's environments, with diverse models, such as Agile, Scrum, RUP, etc., demands code quality tools that can be used at various "check points". The gatekeeper role common in many organizations relies on checking the build at release time. Additionally, individual developers want to know exactly where to look for previously fixed bugs before code check in. Further, many organizations want a solution that can be incrementally rolled into their software development processes.

Built to handle multiple snippets and files. Because bug fixes involve multiple snippet changes in numerous files, the right solution must be designed to intelligently determine the proper context and level of fuzziness for each snippet and consolidate the results from multiple snippets to draw an accurate conclusion.

Pattern Insight's Code Assurance

Code Assurance from Pattern Insight is the only software solution that ensures every instance of a bug is found and never released again. It is based on our patent-pending fuzzy matching technology, which can tolerate any variable name change or statement insertion and deletion.

Pattern Insight's Code Assurance has the characteristics required to provide automated defect elimination:

- > **Fast:** Returns results in seconds, even for billions of lines of code.
- > **Accurate:** Extremely low false positives.
- > **Easy-to-use:** Fully integrates with all SCM systems and is capable of being used in any development, build, and release process.

Plus, it can be used in numerous workflow scenarios:

Ensure releases are clean. For Build/Release Owners, Pattern Insight's Code Assurance easily integrates into the release process or continuous integration to identify previously fixed bugs in releases going out the door. For example, one of our customers in the mobile device industry built a catalog of thousands of security and other Priority-1 bugs and runs nightly reports to determine if any of these have "leaked" into its 500 builds. If a match is found, the build is blocked and the developer is notified automatically.

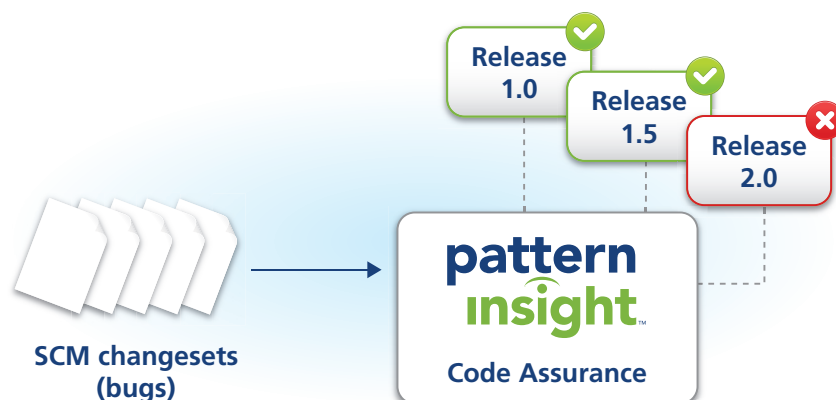


Figure 1: Ensure releases are clean.

Eliminate all instances of bugs in development. Developers want to ensure that a bug has been fixed across all locations branches and components in the development process. In this “catch point” use, the developer uses Pattern Insight’s Code Assurance to quickly run a report to find where every instance of a bug is and fixes them immediately. More automatically, the developer can be informed in code review or code check-in.

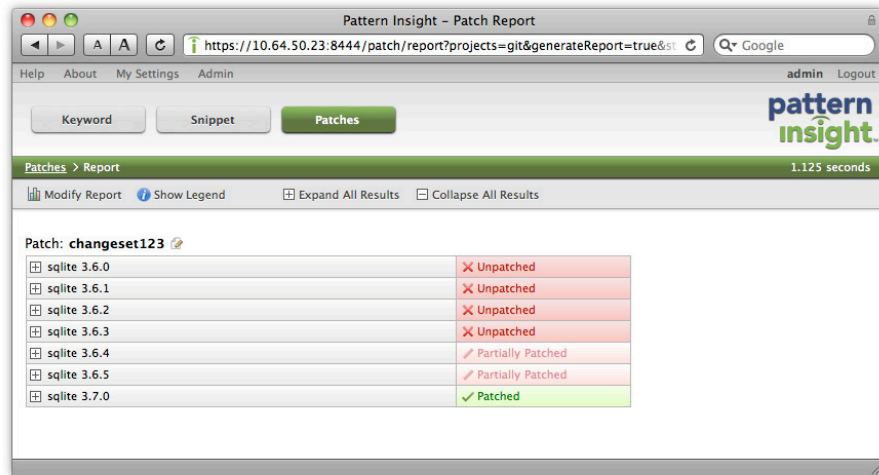


Figure 2: Eliminate all instances of bugs in development.

Ensure that all changes are ported into the final release. Release engineers deploy Pattern Insight for use beyond bug fixes. Code Assurance can be used to automatically ensure that changes made in separate branches make it into the final release. The number of the changes checked can be hundreds or thousands in one single run. This process takes a matter of minutes, whereas manual verification may take months.

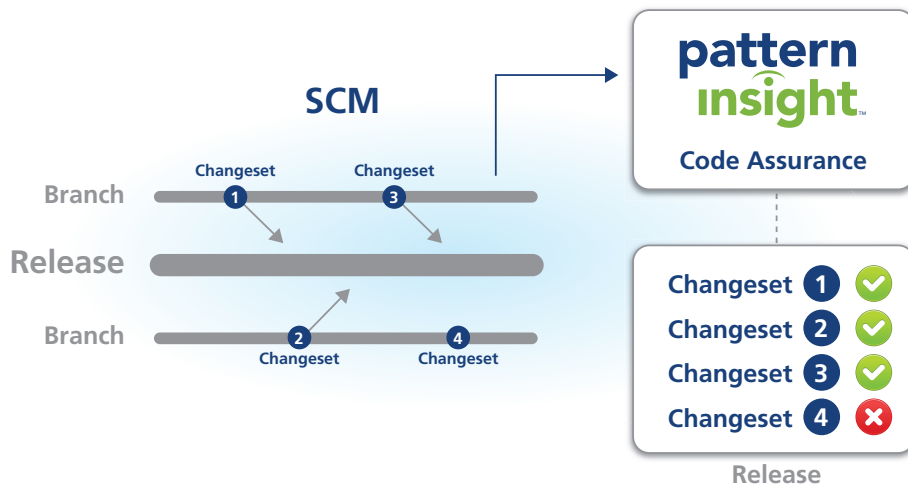


Figure 3: Ensure that all changes are ported into final release.

Pattern Insight matches other workflow scenarios as well:

- > Ensuring that all changes are migrated properly when the underlying platform is upgraded (for example, from Android 2.0 to Android 3.0).
- > Ensuring that patches released by open source software components or third-party commercial software components are incorporated correctly.
- > Ensuring that a code style deemed to be detrimental to software quality is not used.

Conclusion: Focus on Previously Fixed Bugs to Increase Smartphone Code Quality and Retain Customers

Previously fixed defects account for nearly 45% of all bugs in production software. Often they are the most devastating because they escape rigorous quality assurance processes, are missed by traditional static analysis tools, and can be detrimental to customer relationships. Major mobile companies, including Qualcomm and Motorola, have been using Pattern Insight's Code Assurance product to make sure they never release software with bugs they've already fixed. Pattern Insight is the only solution that provides direct insight into the code itself. Most important, Pattern Insight is the only means to effectively automate the manual processes engineering departments rely on today to ensure error-free releases.



Pattern Insight Headquarters

465 Fairchild Drive, Suite 209
Mountain View, CA 94043
P: 866 582 2655
F: 408 573 7855
E: info@patterninsight.com

PatternInsight.com