



Eliminate Previously Fixed Bugs to Improve Your Software Quality

SCM Systems Fail to Help Detect and Eliminate “Known”
Defects, Leading to Production Release Failures

The Problem: The Bugs You've Already Fixed Pose the Largest Risk to Your Next Release

Software companies continuously release previously fixed bugs to their customers, despite their best efforts. This is a serious problem overlooked by many in Engineering management, but the facts are clear:

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

Software quality is not the only factor impacted by this problem. Customer satisfaction is severely diminished when previously fixed bugs keep reappearing in the field. Brand reputation can be put on the line. And the productivity of your team is drastically reduced. Engineering resources are typically redirected to diagnose and fix the same bugs repeatedly. Further, the manual approaches adopted to address this problem invariably fail to solve it. So, cycle times are prolonged with no discernible increase in software quality.

To compound the problem even more, 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.

How Previously Fixed Bugs are Released

The overarching problem is a pervasive reliance on manual processes within development teams and lack of awareness of new technologies to automate these processes. More specifically, the underlying cause of releasing previously fixed bugs to customers lies in the conflict between modern software development with its shorter release cycles, proliferation of components, and numerous product variants; and the attempt to maintain code quality using traditional tools and processes.

This has its root in two practices:

Component and software reuse. This is a good practice in modern software development, often leading to significantly lower development costs. However, it creates a large number of variants. For example, software produced for the telecom, mobile, aerospace, automotive, medical device, and consumer electronics industries, contains significant quantities of product variants that share software components.

Parallel branching. This is the predominant strategy employed to streamline software development and support release management needs. For example, it is very common for software companies to maintain many old releases in the field even while working on a new release. In some organizations, it is also necessary to maintain multiple customized versions, each specialized for a specific customer. Developers continuously make code changes, including bug fixes, in these versions. Keeping these versions in sync is an enormous challenge.

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

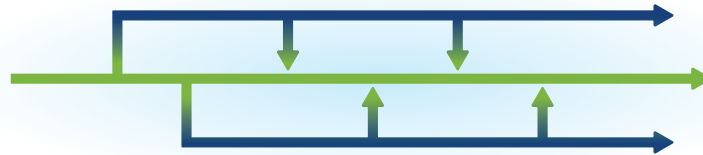


Figure 1: Parallel Branches

These practices lead to common scenarios in which known bugs are leaked into a final release:

Missing branch patches. Bug fixes in one branch must be integrated into all other branches. In most cases, this is done manually. Companies usually impose a policy that if a bug is fixed in BRANCH A, the fix needs to be ported to BRANCH X, Y, and Z. However, the increasing number of branches and bug fixes subvert the manual process. Add the constant pressure to deliver a release on time, and the result is that patching tends to become secondary in priority lists. Commonly, some branches are not patched.

Often it is also very challenging to determine which branches a bug fix should be ported to because developers often don't know when a bug was originally introduced.

Snippet-level conflicts. A bug fix usually consists of several snippets in different files. When bug fixes are integrated into another branch, it is very common for conflicts to arise with other changes made to the same snippets. At that moment, the developer only sees the conflict at the snippet level. Because the purpose of conflict resolution is often to make the build clean, the developer is likely to unintentionally break another developers' bug fix. Many companies use dedicated integration or build engineers to perform this task, which adds to the problem as these engineers tend to have much less knowledge about the specific changes involved in the conflicts.

In many cases, a bug fix consists of several change sets in the SCM system. It's quite common for another developer to mistakenly revert one of the numerous change sets, subsequently breaking the fix.

Code containing bugs is reused. Copy-paste is a widespread programming practice used to save time. Research shows that software typically contains 15-50% duplicated code. In many cases, an entire component is reused and modified, even within the same branch. When a component of code is reused, buggy code within it is also propagated. Currently there is no effective method to keep track of these duplicated bugs.

Current Tools are Insufficient

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. It is not uncommon for the information from developers to be missing, incomplete, or misleading, even when the correct integration means is used.

For example, many companies ask developers to write the bug ID, or ticket ID in the comment section of a change set before code is checked in. Not doing this precisely makes it nearly impossible to perform cross-reference analyses between systems.

Second, SCM systems lack necessary information. For example, SCM systems cannot track duplicated content. Also, 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.

A Rapidly Expanding Problem

New trends in software development amplify the problem of exploding replicated code.

New development tools. Distributed version control systems (DVCS) readily come to mind. Git, for example, the free, fast and distributed version control system, has gained increasing popularity in both the open source community as well as commercial companies. In Git, branching is the “default” procedure, both because of its distributed nature and also because merging is so fast. As a result, companies that use Git tend to create and manage many ephemeral and persistent branches.

Agile development. Many companies nowadays adopt the agile development model that emphasizes quick iterations with incremental development. Therefore, during a short period of time, many artifacts (releases, versions, and product variants) are created and need to be tracked properly, often for an arbitrary amount of time.

Software supply chain component reuse. In modern software development, 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 many other companies that produce Android devices. Most networking device companies have licensed code from their third-party partners who develop the chips they use. 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.

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 vs 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 (not just a single branch) instead of hours or days like many defect detection technologies. Ease of use means the solution must be intuitive for the developer and it must integrate both 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 code base. False alarms are unacceptable as they can compromise the confidence of the developers on 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 & 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 product is the only direct intelligence 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.

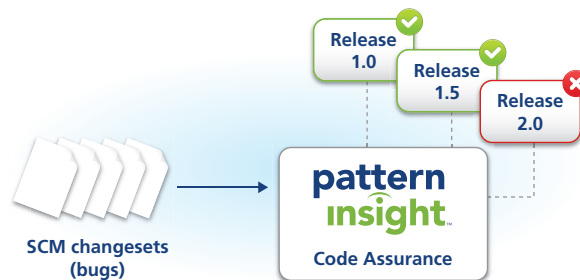


Figure 2: Ensure releases are clean.

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.

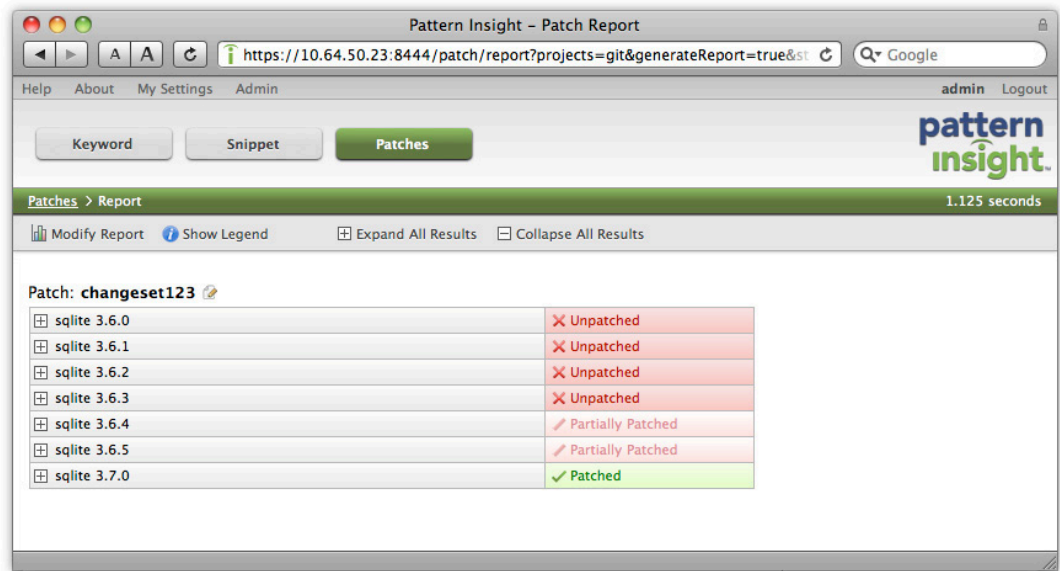


Figure 3: Eliminate all instances of bugs in development.

Ensure that all changes are ported into 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.

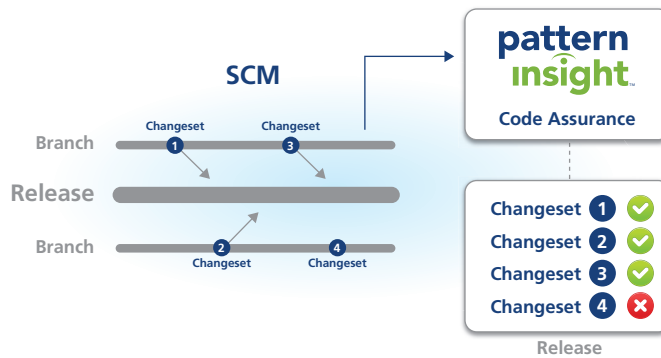


Figure 4: Ensure That All Changes Are Ported Into Final Release.

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 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. Many major technology companies, including Qualcomm, Cisco, and Motorola, have been using Pattern Insight's Code Assurance 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

Europe

SCL
Jubilee House, Jubilee Walk, Crawley, RH10 1IQ, UK
P: 44 1293 403636
F: 44 1293 403641
E: info@scl.com

PatternInsight.com