

The CodeScrubbers™ Process

By Chuck Clifton
Maple Canyon Co.

February 16, 2012

Copyright 2012 by Maple Canyon Co. All rights reserved

The phrase “CodeScrubbers” is a trademark of Maple Canyon Co.

Email: CustomerService@MapleCanyon.com

Website: www.CodeScrubbers.com

Table of Contents

Introduction to the CodeScrubbers Process	4
Background	4
Software Defects (Bugs)	4
How Much Does it Cost to Repair a Bug in Software?	5
Why Are There Bugs in Software?	5
What is a Code Review?	6
Why Don't the Software Authors Perform Quality Code Reviews?	7
What is the CodeScrubbers Process?	7
How the CodeScrubbers Process Works.....	8
What the CodeScrubbers Process is Not.....	8
Advantages of Using the CodeScrubbers Process	9
The Pros and Cons of the CodeScrubbers Process	10
Pros of Using the CodeScrubbers Process:	10
Cons of Using the CodeScrubbers Process:.....	10
Markets that Could Benefit from Using the CodeScrubbers Process	10
Most Likely Customers of the CodeScrubbers Process	11
Conclusion	12

Introduction to the CodeScrubbers Process

CodeScrubbers is a process by which computer software can be thoroughly reviewed to uncover dangerous defects (aka bugs) that may exist in the software. There are many reasons that bugs get inserted in the code in the first place, but regardless of why they are inserted, there is no reason that they should remain. It should be the desire of the software manufacturer and its customers that bugs are discovered and removed as soon as reasonably possible.

Background

Software Defects (Bugs)

Software bugs are mistakes in the instructions or data inside a computer program.

Bugs come in different levels of criticality and those levels may differ based on the nature of the product into which the software is installed. For example, if there is a bug that forces a program to terminate, there can be different levels of criticality between software installed in an MP3 music player and that installed inside a jet engine. In the former case, if the program hits a bug and terminates, the poor user just might have to go 30 seconds without hearing their music. ☹ On the other hand, in the latter case if the program hits a bug and terminates, all on board the aircraft could be at risk of never hearing music again! ☹☹

Table 1 shows how a given type of bug can have different Criticality Levels based on the usage of the software.

Table 1: Criticality Level of Bugs Based on Usage

Bug Causes	Criticality Level for MP3 Music Player	Criticality Level for Jet Engine
Extra slow initialization time	Low – The user will get used to the startup time needed for the player	High – Because if an engine restart is needed in the air, time is critical
Intermittent Communication Loss	Moderate – The user may not even notice this kind of bug	High – Because an important message (e.g. Report Fire Message) may be lost
Program Shuts Down	High - It may be annoying to the user, but it may not ruin their day	High – Engine shutting down in flight is highly undesirable

One might draw the conclusion that software that runs in less important environments may not need to be reviewed as much as software that runs in more critical environments. In some respects, that may be true, especially when human safety is at stake. However, even bugs in less important environments can drive customers away which can cause revenue to fall which can have a huge impact on the business. After all, companies that are in the software business want to make money for their efforts and investments.

So, for many reasons, software bugs are undesirable.

How Much Does it Cost to Repair a Bug in Software?

To repair a bug in software requires the following steps:

- **Find the Bug** – Often software development companies spend a very large part of their budget on using the testing process to find bugs.
- **Fix the Bug** – This is the process of making actual changes to the program (instructions and/or data) to fix the bug. Sometimes, a bug can be fixed by merely changing the requirements. However, that type of bug is not the kind that will be addressed in this discussion.
- **Review the Fix** – This is often a quick, informal review by whoever happens to be available to make sure the fix actually fixes the problem and did not break something else.
- **Test the Fix** – This usually requires that a new build of the software be run and tested in a lab environment that best simulates the actual, final configuration of the hardware and software. Sometimes, tests can be run on the actual hardware itself in the same form in which the final customers will use it.
- **Document the Fix** – If the fix requires a change to any user documentation, that will need to be done. For sure, the changes made to the code will need to be documented to maintain the proper configuration control of the software.
- **Deliver the Fix** – This can be rather simple or rather complicated, depending on the nature of the final product involved. For example, a fix to the software in an online software service might be rather easy and quick because there might only be one place where the software is located that needs to be fixed. On the other hand, if the fix needs to be installed on millions of cars and the owners of those cars need to return them to the dealers to receive the software fix, this can have huge costs and take a lot of time, not to mention what it can do to the reputation of the product manufacturers involved.

The cost to repair a bug in software can be anywhere from minimal to astronomical. It seems logical that the closer the bug is to the customer, the more expensive it can be to repair. Sometimes the actual cost of a bug cannot be given a monetary value, such as when human life is involved. However, studies have shown that if it costs \$100 to repair a bug at the developer's desk, it could likely cost \$10,000 to repair the bug once it has been delivered to the customer.

Why Are There Bugs in Software?

Some of the reasons that bugs are introduced into software may be a combination of one or more of the following:

- Nobody is perfect
- Insufficient requirements
- Inadequate design
- Not following good coding practices
- Inadequate software development lifecycle process
- Insufficient hardware documentation
- Undocumented software
- Rushed schedules
- Inexperienced software personnel
- Lack of training for software personnel
- Incorrect management choices
- Inadequate Configuration Management

- Incorrect assumptions
- Undo pressures from “those above” (e.g. customers, upper management, marketing, shareholders)

Some of the reasons that bugs remain in software that reaches the customer may be a combination of one or more of the following:

- Insufficient testing
- Inadequate software development lifecycle process
- Ineffective (or nonexistent) code reviews
- Rushed schedules
- Inexperienced software personnel
- Incorrect management choices

What is a Code Review?

A code review consists of one or more persons, other than the author, reading the code and searching for and documenting any defects found. Depending on the experience level of the reviewers and the time allotted for the review, there are obviously different levels of quality of code reviews.

Code reviews should be distributed for review with sufficient time to allow all the reviewers to perform a quality review. This rarely happens. The review itself can either be a “desk” review or a “meeting” review.

The former allows the reviewer to perform the review at their desk and then submit their findings to the author. No meeting is used in the case of a “desk” review. One problem is that without a “meeting” to act as a deadline, the reviewers may not complete the review in a timely manner. As often happens on software projects, they may be called away to supposedly more pressing issues.

The latter requests the reviewers to perform their reviews at their desks, submit their findings to the author and then attend a meeting to discuss the findings. One benefit of this type of review is that as defects are discussed, all present at the review can learn about defects that others have found and thus they may improve their own software development skills. The downside of this type of review is that often the reviewers will not get the code soon enough to have time to properly review it before the meeting. Or, even if they do have ample time to perform the review, they still may not review the material ahead of time but instead they will rely on the discussions during the meeting to either look at the code for the first time or to report their defects. This can cause the time spent in code review meetings to grow astronomically and that has an impact on the schedule that usually causes a truncation of the code review process. The code is the victim here.

Typically, the reviewer will use some kind of abbreviated checklist when looking through the code. There are sometimes Software Coding Standards to which the code needs to be written. Some items in the Coding Standards may appear on the checklist and there is a slight chance that they will be reviewed and accounted for correctly. However, there are many topics in Computer Science that should be followed to produce the safest code. These may not always be in the coding standards and they almost never are on the Code Review Checklist. Therefore, these items may only be caught if an experienced reviewer aggressively checks for them. Some of these are very complicated and, even with software tools to help, can take a lot of time to review.

Why Don't the Software Authors Perform Quality Code Reviews?

In most software development environments, the responsibilities for code reviews fall on software authors to review each other's code. This does not happen often enough. And, perhaps, when the code reviews do happen they may lack quality for a variety of reasons.

Some of the reasons that the software authors don't perform quality code reviews may be a combination of the following:

- Only a very small percent of their time is scheduled to perform code reviews. Even that can be cut short due to "higher priority" needs
- Often, code reviews are skipped or minimized as schedule crunches occur
- Insufficient Training – Schedules are often tight enough to train only the software developer in other areas and training for code reviews is often minimized or non-existent
- Insufficient Expertise - Often, even if a developer puts some effort into a code review, they may not know how to perform a quality code review. Since it is not their primary duty, software authors do not obtain much needed experience in performing code reviews
- Insufficient Checklists – There may be checklists (even provided by strict standards like DO178B) but those probably do not come close to catching the kinds of bugs that can be caught by the CodeScrubbers process
- Lack of desire - Most software authors do not enjoy performing code reviews
- Lack of enforcement – Often management will not force code reviews to be performed properly, but only well enough to "check the box".
- Management should focus the training and growth of their software engineers in areas particular to the projects upon which they are working and outsource the code reviews to those who are experts in performing good quality code reviews.

What is the CodeScrubbers Process?

The CodeScrubbers process is a process of using software tools and highly trained software engineers to review code and find and report bugs. It is based on many decades of experience in the software industry where certain types of bugs have been prevalent. Many bugs are caused by the misuse of one or more programming languages. The members of the CodeScrubbers team have been trained to find a variety of bugs by searching for their root causes in the source code. Some of this searching can be done by software tools and some can only be done by manual inspections by those who know what to look for and have the time to do so.

The CodeScrubbers Process looks at the uses of the various programming languages to ensure that they are being used correctly. This means that in order to scrub a piece of software, the members of the CodeScrubbers team only need to have access to a copy of the code and its build information. They do not need access to requirements, design documents and/or test plans. Members of the CodeScrubbers team will NOT examine the code or its design to ensure that it correctly implements the requirements. That is left to others. However, the expertise of those on the CodeScrubbers team lies in the uses of the various programming languages. **Our research has shown that there are countless bugs introduced into software just because those who wrote it did not make proper use of the programming languages in which it was written.**

The members of the CodeScrubbers team collaborate on different programming practices to keep the team up to date on the kinds of bugs detected in software. The CodeScrubbers team maintains a proprietary list of dangerous bugs that can sometimes be found in code, even after it has been inspected. Through time, the

level of expertise of this team in finding bugs grows well above that of the average authors in the software development companies.

How the CodeScrubbers Process Works

The CodeScrubbers process focuses primarily on compensating for the lack of effective code reviews. Realizing that most code is never properly reviewed, one can use the CodeScrubbers process to help fill the void and perform these reviews in parallel with the full development effort of the software with minimal impact on the schedule of the software. Usually, the only impact on the software development project will be that required to fix the bugs exposed by the CodeScrubbers process. But that is a small price to pay for removing bugs and drastically improving the quality of the software. Besides that, in most cases, the bugs exposed and reported by the CodeScrubbers process are reported in such a way that the fixes can usually be made quite easily.

The CodeScrubbers process only requires that a snapshot read-only copy of the software and its build information be provided. In no cases will anyone performing the CodeScrubbers process ever need to access the master source of the software to make changes to the code. That is left to the software developers at their site. The CodeScrubbers process is instead performed offsite from the development site.

In the following discussion, to make it easy to convey the points, the company that owns the software to be reviewed will be referred to as “SW” and the company that performs the CodeScrubbers process will be referred to as “CScr”.

The main steps involved in using the CodeScrubbers process are as follows:

- SW enters into a contractual agreement with CScr to have the CodeScrubbers process performed on some of its software
- CScr signs a non-disclosure agreement as provided by SW
- SW provides a read-only copy of the code to be reviewed to CScr
- CScr performs a review of the code using the CodeScrubbers process
- CScr provides a Bug Report to SW that details the bugs found along with recommendations on how to fix the bugs
- CScr invoices SW for the services per the terms of the contractual agreement
- SW pays CScr per the terms of the contractual agreement
- SW personnel make any desired changes to their code
- SW is pleased with the CodeScrubbers process and tells their counterparts around the world of the benefits of using this service

What the CodeScrubbers Process is Not

Although the CodeScrubbers process can be used to expose many hard to find bugs in code, it is **not** to take the place of other formal code reviewing efforts. The CodeScrubbers process does not concern itself with any company’s coding standards and other items that are on the checklists used in code reviews today. Instead, the members of the CodeScrubbers team will use proprietary methods and software tools to examine the code in a way in which it has probably not been examined before. The things that they check for are not usually found on the lists of things most code reviewers of the world might use. So, with that said, the CodeScrubbers process is not a normal, standard and certainly not a complete code review. It should be thought of as an auxiliary review that can expose bugs that dwell deep in the system.

Advantages of Using the CodeScrubbers Process

Since the efforts involved in performing the CodeScrubbers process are made by those not working on the development efforts, there are many advantages as follows:

- Minimal lost time by those responsible to develop the software
- “A second set of eyes” can recognize bugs not noticed by the code authors
- Personnel who perform the CodeScrubbers process are highly trained in using that process and this is what they do all day long. Consequently, over time, they have reached a higher level of proficiency in finding bugs than the average software developer. In a nutshell, they are experts at finding bugs
- If the CodeScrubbers process does actually find bugs in code, when would they have been found if this process were not used?
- Independence – Because the members of the CodeScrubbers team are not the authors and they do not make any connections to the master source code, there is a level of independence that satisfies any standard.
- Better quality software would emerge as bugs are identified and removed. The CodeScrubbers team would identify them and the authors would fix them.
- Finding bugs *before* the code reaches the lab may reduce the following:
 - Number of Bugs in the Lab – For every bug that the CodeScrubbers team finds and the Authors fix, there will be one less bug that makes it to the lab. Actual experience will show that the CodeScrubbers process may expose hundreds or thousands of bugs.
 - Crossover Bugs – These are bugs that are not in the area expected but that are caused by some other bug in a totally different area. This can result in wasted efforts debugging the wrong code. Bad pointers can easily cause this kind of bug. With the removal of crossover bugs, other bugs may “disappear”.
 - Number of Software Problem Reports (Software Change Requests) – If there are fewer bugs in the lab, then there will be less effort to report those bugs after lengthy testing sessions.
 - Lab Time - Any bugs that do not make it to the lab will reduce the amount of lab time that is needed by that army of testers/developers who spend countless hours each day pursuing bugs. This will also reduce the contention for resources in the labs.
 - Lab Equipment – If the testers spend fewer hours in the lab each week, perhaps some of the lab equipment will not be needed.
 - Overtime – If the testers are spending less time in the lab, they just might not need to work all that overtime that strangles the budgets of the project managers and hampers the lives of the people who work it.
 - Schedule – With fewer bugs in the system, perhaps the scheduled deliveries will happen sooner rather than later.
- Confidence in the eyes of the customer will improve as they see better quality code and much shorter bug lists and problem reports.
- Improved bottom line – In many projects, it has been the lack of quality in the software that has affected the bottom line in a negative way. Finding numerous bugs early and fixing them will have the opposite effect and affect the bottom line in a positive way. Any cost to use the CodeScrubbers process can be well compensated for by the overall savings that will be seen in many areas of the software development lifecycle.

The Pros and Cons of the CodeScrubbers Process

Pros of Using the CodeScrubbers Process:

- The CodeScrubbers process was specially developed to find bugs in code. It is not an afterthought or a simple checklist that minimizes finding bugs by reviewing code.
- The CodeScrubbers process is accomplished in parallel with the code development effort. It does not take time away from those who need to develop the code. This is a win-win situation where all involved get to do what they do best: code authors write the code and the CodeScrubbers process team members get to review it.
- The CodeScrubbers process is performed by experts in the code review process who have been specially trained. This is their full time job. In addition to following a strict manual process, they use commercial and in-house developed software tools to assist in using the CodeScrubbers process.
- The members of the CodeScrubbers team do not have any kind of relationship (i.e. they are not fellow employees) with the code authors so they can provide unbiased reviews of the code.
- The CodeScrubbers team members will collaborate among themselves to help team members become aware of new types of bugs to look for as time goes by. Their expertise in finding bugs will increase.
- The companies that use the CodeScrubbers process will only pay for bugs that are found in their code.

Cons of Using the CodeScrubbers Process:

- This space is available ☺

Markets that Could Benefit from Using the CodeScrubbers Process

Markets that could benefit by using the CodeScrubbers process:

- Aviation industry – If a bug is not caught until an aircraft is flying, its effects may put human life at risk. The number of people affected by a severe bug on an aircraft would consist of up to many hundreds onboard the plane and potentially many on the ground. The cost to fix a bug that makes its way to the aircraft can be rather expensive because each aircraft that has the bug will need to have the fix installed. This could be anywhere from dozens to thousands of aircraft.
- Auto industry – If a bug is not caught until the software has been installed in an auto, its effects may put human life at risk. The number of people at risk of being injured may range from one to several per incident. If a bug needs to be fixed after it has been installed on millions of autos, typically, a recall is announced and the auto owners will need to visit an auto dealer to have the bug fixed. With each car today having dozens of computers onboard and there being hundreds of millions of cars on the road worldwide, fixing bugs that get through to the final customers can be astronomical.
- Medical industry – If a medical device fails to work because of a software bug, it is likely to only affect one person per incident, but there may be up to millions of people using that type of device for their medical needs. This makes fixing a bug in a medical device very costly. Imagine the cost of fixing bugs in software that has been installed in medical devices which have been implanted inside the bodies of human beings.
- Mobile devices industry – If a bug is not caught until it reaches the mobile devices already in the field, perhaps there is not so much threat to human life. The cost to fix such a bug are related to the notification of the users of the need to upgrade their software and then keeping track of which ones get the upgrades and which ones still need it. However, this industry does have the benefit of having a simple upgrade method whereby a user can merely dial a phone number and press a key or use the

Internet to have the new software installed. However, there may be times when the user would need to either take the mobile device back to a dealer or mail it in to the manufacturer in order to get the bug fixed.

- Home entertainment industry – If a TV or DVD player has a bug in its software, either a technician will need to visit the home or the device will need to be taken into the dealer or manufacturer to get the bug fixed. There are hundreds of millions of these devices worldwide and even a simple bug that needs to be fixed could overwhelm a company.
- Remember the Y2K bug? – Some people have said that it was just a rip-off and that it was a fake because there were no real effects from the bug when the clock struck the New Year of 2000. However, those people just might be wrong. The Y2K bug did not turn out to be a problem because the companies and governments of the world literally spent many billions of dollars fixing the bug everywhere it existed before the time that it would have caused problems. Had they not done so, the bug could have caused catastrophic results worldwide.
- Financial Institutions – People want their investments to be safe and secure
- Educational Institutions – Every aspect of a college campus demands well written software.
- Governments
- Militaries
- Space Programs

Most Likely Customers of the CodeScrubbers Process

Suppose that Company A (the vendor) has a contract to develop custom software and sell it to Company B (the customer). The best practice would be for Company A to make full use of the CodeScrubbers process before delivering the code to Company B. By so doing, the bugs that will be caught will be caught earlier in the lifecycle and this would provide the following benefits:

- Bugs are cheaper to fix, the earlier they are caught in the development cycle.
- By catching the bugs earlier in the development cycle, there is a better chance that those who are responsible for inserting the bugs in the code are more likely to become educated to not create those kinds of bugs anymore. In essence, they would be more likely to be held accountable for their work and they would learn **from** their mistakes while they are still around to learn **of** their mistakes.

However, there are many factors that might prevent Company A from using the CodeScrubbers process such as:

- Company A does not see a need for the CodeScrubbers process.
- Company A does not want to spend the money for the CodeScrubbers process.
- Company A thinks that by their own in-house code reviews and extensive (sounds like: expensive) testing efforts they can find the dangerous bugs in the code.
- Company A has pride and does not want to expose their deficiencies to outsiders.
- There is no requirement for Company A to use the CodeScrubbers process.

On the other hand, here are some reasons why Company B might want to use the CodeScrubbers process:

- Company B might want to make sure that the code that they are purchasing is of the best quality possible.
- Company B might want someone to provide an independent second opinion on the quality of the code for which they are spending their money and basing their reputation to their own customers.

- Company B might want to see if there are really bugs that can be found in the software after it is delivered to them by Company A.
- Chances are that if Company B is buying custom software from Company A, they just might be buying software from companies for dozens, hundreds or even thousands of other projects companywide. Their use of the CodeScrubbers process can become a major part of the standard by which all software is accepted or rejected as it is delivered by the software vendors.

Conclusion

In conclusion, there are a lot of bugs in software delivered today. This does not need to be so. Many of the hardest to find bugs are not found in the labs via monstrous testing efforts but they can likely be found by using the CodeScrubbers Process.

Although there is more to gain in having the software vendors of the world use the CodeScrubbers process, there is a much higher probability of convincing the software customers of the world to use the CodeScrubbers process. So, if those who purchase custom made software would have it scrubbed before they accept it from their software vendors, they can request that the bugs uncovered by the CodeScrubbers Process be fixed.

However, after a few rounds of the software vendors being embarrassed by the bugs that are found in code they thought was good enough for delivery, they will short circuit the process and utilize the CodeScrubbers Process **prior** to their shipping of the code to their customers. That would be a small price to pay for the improvement in their reputations.

Currently, the CodeScrubbers team only supports scrubbing code that has been written in C/C++. Plans are to add support for more languages in the future.