Full Spectrum Software

Stop the Endless Alpha Blues

How to rescue a hopelessly stalled software project

You are in charge of a software project that is already hopelessly stalled and which is many months, or even years behind schedule. You're in a seemingly endless cycle of test and de-bug mode and the soft ware is still not stable enough to even consider releasing it. The team is stressed out, management is putting pressure on you and sales people are explaining to customers why the product is delayed.

Everyone in the business of building commercial software has either been there or seen this happen.

Here are some practical, hands-on techniques that you can use right now to turn things around and start making forward progress. Full Spectrum Software has performed many software rescue missions over the last 12 years. In this paper, we'll share the techniques, tools and processes that have led to our 100% success rate in rescuing hopelessly stalled software projects and getting those products to market.

A large part of any software rescue mission is called product hardening and the code we receive from a client is in a state that we call "fragile." Generally, the software being worked on is either an older product that the team is trying to upgrade and add new features and capabilities to or the software is more complex than the software team is prepared to manage.

For older products, at one time the software may have had a great architecture, but over the years many different engineers have worked on the product and have not adhered to that architecture. In some cases the software is based on technology from the 1990's, perhaps MS-DOS which has been "wrappered" to give it a Windows look and feel but they are not truly Windows applications.

For newer products, the product may have expanded in scope to require complex threading, may require expertise that the development team simply doesn't posses or, in the worst case, the product may have been designed and implemented by engineers who did not understand the technology.

More Pressure Doesn't Help

Whatever the case may be, there are commercial software tools available including both static analysis and run time analysis that can help immediately. However, the most important change that needs to be implemented immediately is a fundamental shift in mindset. The goal at this stage is to stop the test and de-bug cycle and begin a controlled process called code hardening.

The goals, objectives, processes and methodologies for product hardening are different from simply trying to de-bug the software and release it. It is a major shift in strategy and almost always requires executive buy-in. Needless to say, executing this internally and selling this internally is a very, very tough sell.

It is particularly challenging because executives don't want to hear that you are essentially stopping all development and shifting to a completely different effort to analyze, stabilize and ultimately prioritize how development will proceed. However, the truth is, you really are stopping development to redirect your resources.

The first step is to pull everyone off the project except for the software lead and the SQA lead. You will need their domain expertise and intimate knowledge of the code. Next you need to acquire the commercial grade static analysis tools and one or more run time analysis tools. Both of these toolsets are very challenging to configure correctly so a reasonably large amount of time should be allocated for configuration. This is especially true if your team is not already using the tools.

Once the tools are configured correctly and applied to your product, you will see an immediate picture of the state of the code. The static analysis tools will give you insight into the quality of craftsmanship of the code. For example, they will show you exactly where engineers have failed to initialize variables properly or failed to implement memory-freeing techniques and calls properly or have incomplete states that can result in unpredictable behavior.

The run time analysis tools compliment the static analysis tools and will track and report problems such as unhandled exceptions or failures in the code, out of bounds parameters that are being passed to functions within the code and report memory errors such as freeing the same block of memory twice.

These types of defects are exceptionally difficult to find without the use of these automated tools and are the source of many types of very serious problem conditions that are often hard to reproduce.

Concurrently, you need to bring in your most experienced software systems architect and a team of your most experienced software quality assurance engineers. The systems architect should have experience in doing formal code reviews and should be tasked with conducting a thorough analysis of the code.

Their focus should be on evaluating the architecture, craftsmanship of the code and areas of the code where there are known defects and problems. Their report should identify where the major problems exist within the code base. This allows the software team leader to help prioritize where resources should be allocated when they begin fixing defects.

Recently the FDA forensics lab announced that they were using static analysis tools to test software that had been re-called due to adverse events. Brian Fitzgerald of the FDA forensics lab said "We're hoping that by medical device manufacturers staying on top of their technology, that we can introduce this up-to-date vision that we have." The implication is clear. Automated test tools must become part of the software development and SQA process.

The new SQA team should be tasked with developing formal test plans based on the result of the code review and output from the automated tools. It's important to emphasize that the new SQA team must be composed of the most highly experienced people in the company. Highly experienced software quality assurance engineers have a mindset and the professional training to test and design test protocols so that those defects can then be entered into a commercial grade defect tracking system. (It goes without saying that if you're not already doing so, you must implement a commercial source code management system).

The SQA team's ability to test other people's code and uncover difficult to find and clearly document difficult to reproduce defects is crucial in this product hardening process. This is an absolute necessity and is not the same as software engineers testing their own code. Do not under any circumstances allow the software developers to test and approve their own code. Experienced SQA engineers have an entirely different mindset about testing other people's code. While software developers look for elegant ways to make thing work, SQA engineers have a profound fascination with looking for ways to make things break. Only they should be allowed to approve code.

Once the systems architect has finished his or her report, the static and runtime analysis tools have been correctly configured and the tests have been conducted, the software quality assurance team has completed their detailed test plans and test protocols and completed at least one round of testing, you're ready to begin developing your plan for project completion. You should expect that if you execute this process internally that this phase will take approximately 8 to 12 weeks to complete.

There are two issues you should be prepared for and set the correct expectations internally. One is that when you complete this process, the bug count will skyrocket. If the bug count does not jump substantially, it is a sign that something has been done incorrectly. Either the test tools have not been configured correctly or the new teams don't have the maturity or experience to execute the process described here.

The key point is that you need to set the expectation that the results of doing this analysis will reveal many more bugs than were originally thought to exist. If you don't set this expectation before you begin the product hardening process you may have some very upset executives.

The second issue is that you will have a fairly good sense of the "life expectancy" of the product. You may make the informed judgment that no further upgrades will be possible for the product and that management should expect that a complete re-design and re-write will be required within a certain time frame.

At this point, you are ready to bring the original development team back onto the project. However, the new SQA team should largely be kept in place until product release. At minimum, you should keep your most experienced SQA lead and your software architect on the project until completion.

You now have a detailed view of the code base and you can prioritize the order in which each defect should be addressed. Start with the low hanging fruit. Fix the easy bugs first so that additional defects are not introduced during the fix process. Address each bug methodically and release the fix to the SQA team for formal testing.

Do not begin bug fixing at a frenzied pace, even though you will be under pressure to do so. Wait until the SQA team completes its testing and review the results of each fix. From this point to final release it is imperative that no additional bugs are generated as a result of a fix. Follow the now documented test procedures and protocols. Review your secure defect tracking system daily. Continue to run both the static and runtime analysis tools periodically. Talk to each team member at least once per day regarding their progress and any problems. Plan to hold formal weekly review meetings to be absolutely certain you are making forward progress.

If you follow this highly structured process consistently, you now have a very good chance at successfully releasing the product on the revised schedule.

Techniques that don't work

Having just described a process that will work, we'll take a moment to discuss techniques that do not work. There's always a temptation to throw more resources at a project in the futile hope that more cooks won't spoil the soup. Many companies try to accelerate product completion by adding more software engineers. The practical problem with this approach is that a software team leader has certain bandwidth in terms of how many people he or she can manage and still make valuable contributions to the project.

The software team leader is usually intimately familiar with the code. The leader is usually the most productive member of the team. Give them too many new engineers to manage and bring up to speed and their personal productivity drops precipitously, slowing progress as a result.

Simply adding more software engineers will achieve the opposite of the desired outcome. It just slows everything down. This is especially true when adding more junior engineers who have not been through multiple product releases.

The second technique that also doesn't work is to add many more SQA engineers and software testers. This sometimes creates the illusion that the bug count is going down and forward progress is suddenly being achieved. In fact, the opposite is happening.

The new SQA engineers and the often very inexperienced software testers are simply not familiar with the code. The bug count may actually be increasing, but in fact, duplicate defects are reported or defects are reported without proper steps to reproduce. In the worst case, they don't have the experience or familiarity with the code to find new or meaningful bugs. This can be a very dangerous situation if management makes a decision to release the product based on a low bug count.

The last technique is the worst one of all. Put even more pressure on the team to finish. Anyone who has been in this situation knows that there is a physical limit to the number of hours an engineer can work over a sustained period of time before the quality of their work begins to deteriorate. Highly experienced engineers start to make rookie mistakes because they are mentally and physically exhausted. Even though management has been kind enough to install cots so people can sleep an hour or two a night, this technique also achieves the opposite of the desired outcome.

About Full Spectrum Software

Full Spectrum Software has performed a wide variety of software rescue missions over the past 12 years with a 100% success rate. The company employs hundreds of pre-configured templates and test procedures that are used in conjunction with dozens of our commercial grade static and runtime analysis tools. This allows us to quickly customize the correct configuration of our static and runtime analysis tools to meet the needs of virtually any type of code base. Our SQA teams have an extensive library of processes and protocols that can be quickly customized to our client's needs. Our software architects are experts in product hardening and incorporate a full arsenal of automated systems in our development and testing services. We leverage the expertise of your internal teams to rapidly achieve results for our clients. If you have a hopelessly stalled software project and need an expert partner to help you get to completion, give us a call. We'll introduce you to our clients who worked with us to get a high quality product out the door.

Full Spectrum Software
225 Turnpike Road
Southborough, MA 01772
508-620-6400
www.FullSpectrumSoftware.com