Reducing Costs with Good Requirements and Code Reviews

Teams constrained by resources are always looking for a competitive advantage in reducing costs and improving software quality. This advantage can be achieved by implementing two best practices:

1. **Writing good requirements**
2. **Providing peer code reviews**

Studies have proven that the later a defect is discovered, the more it costs a team to fix. A study conducted by Dunn - *"Software Defect Removal"* - found that a defect that costs $1000 to fix when discovered during requirements analysis costs $25,000 to fix when left undiscovered until the quality assurance testing phase. Another study conducted by Jason Cohen, highlighted in his book - *"Best Kept Secrets of Peer Code Review"* showed that defects cost half as much to fix when found during code review vs. when discovered during quality assurance testing.

Defining Good Requirements

A good requirement contains these elements:

- A narrative that explains what the requirement is going to accomplish
- A prototype that depicts exactly what is expected (screen shots, report layouts, etc.)
- A list of validation rules (e.g. allowable field sizes, allowed values, data validation, etc.)
- A list of security rules (e.g. only administrators can access specific features)
- A list of performance criteria (e.g. must be able to add a record in 3 seconds or less)
- A list of audit rules (e.g. when adding or changing a record, store the history of the change)
- Success criteria (e.g. objective criteria that can be evaluated to ensure this requirement meets the need)

Let's explore how taking this approach saves costs. Imagine we needed to develop a new screen to track our customer contacts and the screen entry needed to collect company name, contact name, email address and phone number. A poor requirement would simply describe the process (e.g. "create a data entry screen for collecting contacts") but would not include any of the other descriptive elements.

Without a prototype to guide him, the programmer might take liberties and design a screen that had a lot of ancillary fields (like birth date and marriage status) that may not be needed and may not include important fields (like email address and phone number). If no validation rules are specified, the programmers may not test a scenario where a new contact is added whose email address is the same as an existing contact -- causing a duplicate contact record. If no security rules are identified, the screen may allow anyone to delete a contact and this may cause your team to lose an important contact record. If no performance criteria are set, the team may not test a scenario where you have 100,000 contacts in your contact list and your new software may slow to a crawl. If no audit rules are in place and a person deletes a contact, you may never know who deleted an important contact and why.

With a good QA team, many of the issues above would shake out during the quality assurance phase. However, the cost of discovering these issues and re-working the code is dramatically

higher when discovered at this time. If the requirements had been written using the detailed elements above, the programmer would not have taken liberties to add additional fields and leave out important ones, they would have stress-tested the solution with large number of contacts, they would have implemented validation logic to ensure all fields were validated, they would have added logic to prevent duplicate records and would have audited records when deleted.

Defining good requirements is equally crucial for Waterfall and Agile projects -- the difference is that Agile projects will break the requirements into smaller units of work that are described by User Stories (what a user wants to do in each case). In our example above, a Waterfall approach would define all of the elements of each requirement at the outset of the project. An Agile approach would break each of those elements into separate User Stories (a User Story for the prototype, a User Story for the Validation Rules, etc.) and would incrementally implement each item in an iterative fashion.

It's also imperative to review requirements as a team (internal teams and client) once they are developed but before they are signed off. The review process lets extra eyes ensure that the elements above are addressed and that the requirement will meet the actual need.

Reducing Defects with Peer Code Reviews

As code is developed, it is important to review the code with your team. Below are a few items to look for in code reviews:

- Ensure the code functionality meets the specifications in the requirement
- Ensure each subroutine contains proper error trapping and handling
- Inspect logic errors that can cause looping, memory leaks and performance issues
- Inspect variable declarations to ensure that variables are cast properly to reduce boundary issues and invalid type casting
- Review coding standard to ensure the code meets your internal standards, is well documented, and easy to maintain

The most efficient way to perform code reviews is to have team members inspect the code (called Peer Code Reviews) on a regular basis and openly discuss possible issues discovered during the code review. This activity can be done before checking the code in (called *"pre commit"*) or after checking the code in (called *"post commit"*). The decision for pre or post commit is entirely up to your team and the way your team works best together.

By inspecting the code regularly, you can discover defects earlier in the development cycle which, like writing clear and detailed requirements, will reduce costs. Imagine a scenario where you reviewed the code and noticed that no logic had been added for preventing duplicate records from being added. By discovering the issue prior to testing, you eliminate costly QA time to discover and report the issue. And if QA doesn't find the issue, you will spend exponentially more fixing it once the software has shipped to customers.

From a technical perspective, imagine a scenario where you had some logic that reads data from a database using a record set, then executes a loop over and over again until all the data is processed. A common programming mistake is not checking for a "no data found" or "end of file" condition. A peer code review could quickly discover that the loop may be entered even if no data was loaded into the record set, which will only cause a failure under that condition. If this defect is not found until QA testing, your QA testers may see erratic behavior where it works sometimes (when data is found) but not other times (when no data is found). It may take your QA testers 2 to 3 days to discover the actual issue, so you have just added several days of effort to a problem that could easily have been discovered during a peer code review.

Reduce Project Costs with These Best Practices

Obviously, the sooner in your development cycle that you identify discrepancies in requirements and find defects, the less expensive the corrective measures will be. By writing good, detailed, and clear requirements at the outset of a project, and by having both team members and customers review them, you ensure that developers spend their time building the right thing the first time. By doing peer code reviews on your code before it goes to QA, you ensure that bugs are found immediately, while they are still easiest – and cheapest – to fix.

Can Tools Help?

Tools can help you with both of these issues. *Software Planner* (http://www.SoftwarePlanner.com) allows you to keep all your requirements in a single place and allows your team to setup workflow to ensure that requirements are reviewed for best practices. Software Planner works equally well for Waterfall and Agile environments and allows teams to spot issues with requirements before it is too late and more costly to fix. *Code Collaborator* (http://www.CodeCollaborator.com) allows your team to conduct peer code reviews online, a task that is painfully tedious and error prone without a tool.

Helpful Resources

Below are some helpful resources and templates to aid you in developing software solutions:

- **Software Planner** - http://www.SoftwarePlanner.com
- **AutomatedQA TestComplete (Automated Testing Tool)** - http://www.TestComplete.com
- **Code Collaborator (Peer Code Review Tool)** - http://www.SmartBear.com
- **STAR QA (Automated Testing Resources)** - http://www.star-qa.com
- **Software Development /QA Templates** - http://www.softwareplanner.com/Templates.asp
- **Test Case Training** - http://www.SoftwarePlanner.com/Services.asp
- **Pragmatic Agile Development** - http://www.softwareplanner.com/PADOverview.pdf

About the Author

Steve Miller is the Vice President of ALM Solutions for AutomatedQA (http://www.automatedqa.com). With over 24 years of experience, Steve has extensive knowledge in project management, software architecture and test design. Steve publishes a monthly newsletter for companies that design and develop software. You can read other newsletters at http://www.SoftwarePlanner.com/Newsletters.asp.

AutomatedQA Corporation
7935 E. Prentice Ave, Suite 105
Greenwood Village, CO 80111 USA
Tel:+1 303.768.7480