

Reducing Your Cost of Quality

Executive Summary

How high is your Cost of Quality? The answer might surprise you. Yes, it includes doing reviews, maintaining the QA infrastructure, preparing tests, and so forth—those are called your “Appraisal Costs”. But how high are your “Failure Costs”? ... the costs that you incur every time a defect in your software comes to light?

How badly is your budget busted by the effort your engineers spend diagnosing and fixing defects, both during testing and after release? How badly does your development schedule slip because of the constant parade of defects turned up during testing? How costly is it to provide support for your customers as they deal with the defects that you shipped to them? And what effect do those cost, schedule and support issues have on your product’s reputation in the marketplace?

Clearly, your Failure Costs are your more significant Costs of Quality, but how can you gain control over them? Although your Failure Costs are beyond your direct control, you *can* gain control over them indirectly through your Appraisal activities. By focusing on activities that remove defects most effectively, you can reduce your total Cost of Quality, making your budget better, your schedule stable, your customers cheerful, and your market magnificent.

Cost of Quality: Two Components

We tend to think of Quality Assurance as a necessary evil. It is a cost of doing business, and like all costs, we must focus on keeping these costs down so they don’t eat us alive. What we often overlook is that the cost of quality has two distinct components: Appraisal Costs and Failure Costs.

Appraisal Costs of Quality are the ones we tend to focus on because we have direct control over them. We explicitly plan for certain appraisal activities, and we decide when we have done enough of them. We budget for the cost of the QA staff and their tools, the time they spend and the resources they use doing test planning, testing, and other parts of their jobs. We decide if our developers will do peer reviews, and how extensive those reviews will be. But Appraisal Costs aren’t even the biggest part of our Cost of Quality.

Failure Costs of Quality are the ones that happen *to* us. They are variable costs that are hard to predict, and are the more significant portion of our Cost of Quality. These costs (often called the “Cost of Poor Quality”) are the ones that are caused by defects in our work products. Failure Costs have an insidious way of sneaking up on us in the form of costs that we don’t immediately see as Costs of Quality. Let’s look at a few of those.

Failure Cost: Busted Budget

The single largest Failure Cost of Quality is the effort that our developers spend investigating and diagnosing defects, and then reworking designs and code to correct them. Although we may try to minimize those costs by putting less expensive people on the line, we find that our most experienced (and most expensive) developers end up being sucked into the fray on the most challenging and elusive of the defects. The bottom line is that the people we most need to have focusing on the next product—the future of our company—are stuck processing an endless stream of bug reports.

Reducing Your Cost of Quality

These costs are so significant that we routinely engage in triage of problem reports. We decide that a few of the defects must be fixed immediately. Others can wait till later. (Maybe next week we'll have time to fix them.) We will defer many of the problems until the next release. (This can cause a firestorm of protest from QA or from Marketing, and the ensuing negotiations waste even more of everyone's effort as we sort it all out.) Finally, we will choose to ignore some defects. (After all, there *is* a reasonable workaround, so it won't cause the customers *too* much trouble.)

The vast majority of us no longer expect to fix all known defects before releasing our software. We simply hold it back until we decide that the cost of *not* shipping the software has exceeded the cost of shipping a defective product. That is why so many of us can identify with the statement made by a recent client of mine who said, "We don't release software, it escapes!"

Of course, fixing the problem next week or next release, or letting customer support deal with it does not get rid of the failure cost; it only defers it until later. The company ends up paying the cost one way or the other.

Failure Cost: Slipped Schedule

Developers seem to be lousy estimators. No matter how long they say a project will take, even if we double the schedule, they *still* can't deliver on time! This is a common frustration. But if you could gain insight into *why* schedules slip, you might be surprised at the most common cause. While there are times when technical challenges or gross underestimates are to blame, the most common cause of schedule problems is defects.

Testing is like a big black hole. Software is sucked into test, and it is never seen again. Sometimes we blame the testers; "We delivered the software to QA on schedule, but they are holding up the release!" But QA is only finding defects, and it is the defective software that is holding the schedule up.

More often than not, defects prevent test suites from being completed. Testing is halted until the defects that block execution can be removed. When the fixed software is available, QA first re-tests to be sure the defect was fixed, then they move forward—until the next blocking defect is encountered. This cycle continues for a much longer time than anyone would like. Even when the testing can be completed, the backlog of defects that are being fixed requires continual re-testing. And of course too often, defect fixes introduce new defects into the system, starting the cycle all over again.

While delivering the software on time is critically important, we often find that defects simply prevent it. We must hold the product back until we can reach some reasonable quality level. Of course, the definition of "reasonable quality" is a matter of great debate. No matter when we finally release, some people will think we waited too long, and others will fear the consequences of the remaining defects.

Reducing Your Cost of Quality

Failure Cost: Costly Customers

Once the product has been released, the Failure Costs associated with it are not over. In fact a significant portion of them will continue for as long as the product is in use. Let's face it: The reason we want to force our customers to upgrade is to stop the hemorrhaging of support costs!

Most of the effort in our customer support group is spent helping customers to deal with all of the defects we shipped to them. But the cost of support is not limited to the support group. Every time a customer reports a *new* defect, a developer must investigate it and diagnose the problem. Fixing the defect in the next release does not avoid the cost; it only defers it, while aggravating the customer.

For those defects that are deemed to be significant, the costs can be staggering. The developer must rework the design or code or develop a patch, QA must test the fix to be sure it works and it did not cause unforeseen problems, and finally, it must be installed at the customer site to fix their particular problems. It is easy to see why the research shows that the cost of the average defect found by the customer is 50 to 100 times higher than those found in-house.

Failure Cost: Meager Market

Alienating customers is easy. Deliver the product late. Defer shipping promised features while you are correcting defects. Ship lots of defects for customers to find. Continually tell them "it will be fixed in the next version". Placate them with "ingenious" work-arounds.

The biggest Failure Costs are nearly impossible to quantify; loss of customer good will, tarnished reputation in the market, and loss of product momentum. The customers who dump your product in favor of your competitor's immediately erode your market share. But those who don't go that far are still unlikely to encourage others to buy your products, and your market share will continue to erode.

Reducing the Failure Cost of Quality

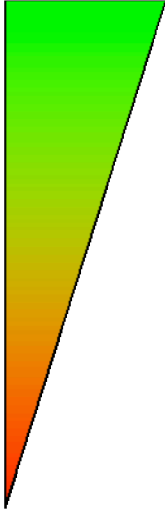
Conventional wisdom would tell us that when you have some costs you can directly control (Appraisal Costs) and others that you can not (Failure Costs), that you should keep a tight reign on the controllable costs and hope for the best with the others. Like most paragraphs that begin with a statement about "conventional wisdom", this one argues against that strategy. The fact is that you can exercise *indirect* control over your organization's Failure Costs by appropriately managing the Appraisal Costs. This is not to say that you should allow your Appraisal Costs to run wild. But it also infers that you may not want to cut them too deeply.

All ... ALL of your Failure costs (every dollar of them) are caused by a finite number of defects in your software. Every defect that you can remove more economically than you currently do represents money on your company's bottom line. Every defect you can remove in a more timely way represents hours or days (or weeks!) of schedule saved. Every defect that you avoid shipping to you customer, and every useful feature that you *do* ship is priceless good will that builds your reputation in the marketplace. The key is to find more efficient methods to detect and remove defects.

Reducing Your Cost of Quality

Defect Removal Activities

This is a list of the various methods that different organizations use to remove defects from their software. They are listed roughly in order from most effective to least effective (in terms of both time and cost per defect removed). Beside each is the word “Appraisal” or “Failure”, indicating how most of the effort involved in that activity would be classified.

<u>Activity*</u>	<u>Cost of Quality</u>	<u>Effectiveness</u>
• Personal Reviews (PSP reviews)	Appraisal	
• Software Inspections (Fagan Inspections)	Appraisal	
• Peer Reviews	Appraisal	
• Compiling	Failure	
• Unit Testing	Failure	
• Integration Testing	Failure	
• Beta Testing	Failure	
• System Testing (and performance & other testing)	Failure	
• Acceptance Testing	Failure	
• Walkthroughs	Appraisal	

Testing is a relatively ineffective way to remove defects, but it is still a necessary part of your development lifecycle. Rather than continuing to make it your main defect removal mechanism, you would do better to use it to gage of the effectiveness of your earlier defect removal activities like reviews, inspections and unit testing.

Many of the above methods will be discussed in future articles on this site. In the mean time, you can begin to reduce your total cost of quality by moving your defect removal activities up toward the top of this list.

* Notes on Defect Removal Activities:

The placement of Peer Reviews in the list is based on best practices. In some cases, their effectiveness is quite low.

The order of Compiling, Unit Testing, Integration Testing, Beta Testing, System Testing and Acceptance Testing in the above list really does indicate their relative efficiency in removing defects (not just their lifecycle order). For example, Compiling can find 50% of the defects in a program, and each of those defects is generally inexpensive to remove. By the time you get to Acceptance Testing, only about 35% of the existing defects are detected, and each one is *much* more expensive to correct.

The placement of Unit Testing in the list is based on *best* practices. Many developers have never been trained in testing, and do a poor job of it.

Walkthroughs are more effective for training purposes than for defect removal.