Realtime publishers

The Definitive Guide™ To

Quality Application Delivery

sponsored by

Don Jones

Copyright Statement

Chapter 4: What's Your Quality Level?

It's nearly impossible to instantly go from little or no quality all the way to amazing quality; much like building a skyscraper, you have to start with a solid foundation, complete the lower levels, and only after getting everything in place underneath can you finish the top. With quality, you need to get some fundamentals in place, and learn to build an organization capable of implementing and sustaining quality in a repeatable, logical fashion. So before going any further, it'll be useful to assess your organization's *current* quality level so that we can decide what the next steps will be to improve that level.

The Quality Quiz

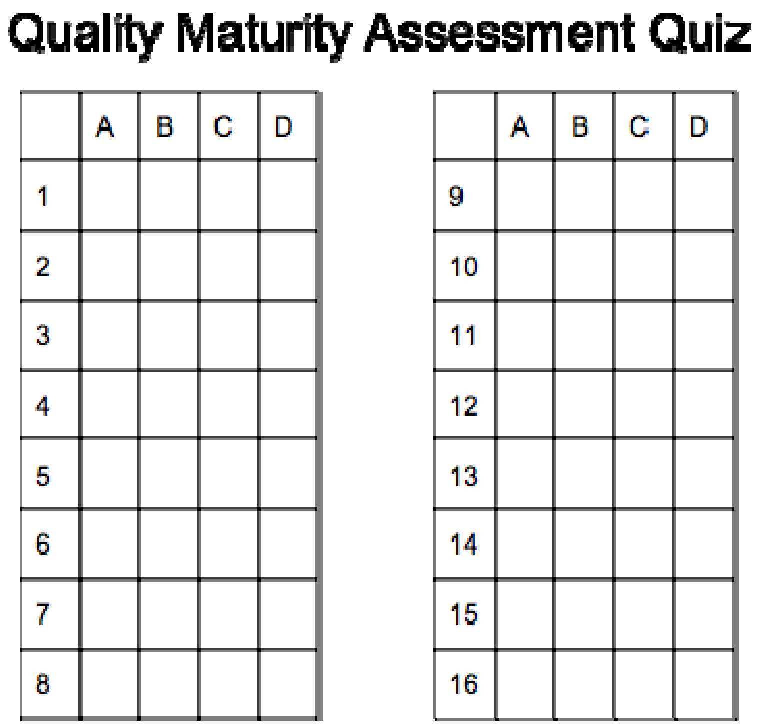
The easiest way to quickly assess your current quality level is to take a short quiz. Simply answer, as honestly as possible, each of these questions. Keep track of your answers on a separate sheet (or print the answer card in Figure 5.1, which follows the quiz), and at the conclusion, you'll create a score that points to your quality level. Choose only the *best* answer for each question—that is, the *one* answer that fits your organization the most.

1. Your organization's quality team…
   a. Doesn't exist
   b. Exists, but struggles to communicate quality
   c. We have several quality teams that don't always deliver consistent quality
   d. Exists and consistently delivers quality
2. Our quality tools…
   a. Don't exist
   b. Consist primarily of testing automation tools
   c. Include automated testing and static code analysis tools
   d. Track specific requirements and their business impact
3. We measure quality…
   a. Primarily through end-user acceptance testing
   b. Through mostly manual testing and some automated testing
   c. Primarily through automated testing
   d. Mainly by seeing how well business requirements are met by the software
4. "Quality" means…
   a. Bug-free
   b. Thoroughly tested and accepted
   c. Different things for different projects
   d. Consistently bug-free software that does what the business requires

5. Our business leaders…
    a. Have little faith in IT's ability to deliver quality
    b. Are making a commitment to quality but don't really know what to do about it
    c. See IT delivering quality sometimes but don't know exactly when it will happen or not happen
    d. Have faith in IT's ability to consistently delivery quality
6. Our company's bottom line…
    a. Has clear, negative impact attributable to poor software quality
    b. Is hindered by buggy software, but we're improving
    c. Is hit less by bugs than by software that isn't as productive as it needs to be
    d. Has a clear, positive impact attributable to good software quality
7. Our executives…
    a. Have no idea where or why quality is being lost
    b. See some automated testing but have no real connection to quality
    c. Know that quality is being achieved sometimes but can't point to exactly why
    d. Have visibility into how we achieve quality and know how to replicate it
8. Our users…
    a. Are frustrated with software that doesn't do what they need, how they need it done
    b. Are patient with bugs but often feel that the software doesn't work the way they need—and feel that *that* outweighs the bugs they see
    c. Generally receive bug-free software but are always asking for minor changes and new features
    d. Are typically pleased with their software and feel it makes them more productive
9. We track software defects…
    a. Primarily through Help desk tickets if at all
    b. In a bug-tracking system that only our developers use
    c. In a combined system that allows management reporting to reveal bug correction rates per build
    d. In a system that allows the existence of known bugs to influence the next phase of requirements

10. "Testing" means…
    a. Beating up on the software to find the bugs
    b. Repeating the same testing process over and over—automation!
    c. Automated testing to run through most user experiences
    d. Automated tests that focus on the original requirements not just specific user paths
11. When we look at our software quality tools…
    a. We don't see many
    b. We see a few
    c. We see several, but different groups use different ones
    d. We see a consistent set being used across the organization
12. Our main software development metrics include…
    a. How long it takes to deliver the next build
    b. How many bugs we found in the last build
    c. How many defects we find in each test cycle, along with how many were corrected
    d. How well each build passes business-focused tests, including trending from each test cycle
13. We explain and communicate quality…
    a. Poorly, if at all
    b. Using simple statistics, often based on defects
    c. Through detailed statistics covering development and testing
    d. With detailed statistics from every step of the software development life cycle
14. Return on Investment (ROI) for software development…
    a. Is something we're still waiting for
    b. Is something we can measure a bit but haven't achieved
    c. Is something we see on only some projects
    d. Is something we consistently look for and achieve
15. "Test data" is…
    a. Whatever our developers make up as they're testing
    b. Ad-hoc data plus a standard test data set the developers created
    c. Drawn from standard test data libraries we developed or purchased
    d. Copied directly from production systems and includes both out-of-bounds data as well as edge-case data

16. Our developers…
    a. Aren't always aware of best practices in coding
    b. Are educated in best practices but don't use them consistently
    c. Consistently use all major best practices
    d. Use all major best practices as well as define and use major internal standards for coding quality



Quality Maturity Assessment Quiz

	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				

	A	B	C	D
9				
10				
11				
12				
13				
14				
15				
16				

Figure 5.1: Print this scorecard to track your answers to the quiz.

All done? Let's see how you did:

- If you mostly answered "A," your organization is mainly treating quality as a hobby. Don't take that as a complete negative—it means that quality is on the radar in your organization but that it's not being done consistently or constantly.
- If you mostly answered "B," quality is an ongoing effort—and a lot of it—in your organization. That's good: It means you're focused on quality and are striving hard to achieve it, although the organization needs some improvement to make quality more consistent.
- If you mostly answered "C," quality is being treated as a professional resource, even through it's still a struggle sometimes. That's great because now you just need to take the last steps to make quality a repeatable, consistent science.
- If you mostly answered "D," you've made quality a science, and you know how to get it every time. You've achieved quality nirvana!

Of course, many organizations will find themselves split between one of these quality levels, which isn't unusual; IT folks are, if nothing else, incredibly clever and often work hard to produce a better product. But if you see facets of your organization in some of these questions, then read on: We're going to look at what makes up each of these quality levels, and more importantly, look at the steps necessary to move from one to another.

Quality as a Hobby

At this level of maturity, organizations typically have little or no formalized Quality Assurance (QA) processes, team, or tools, and quality—if it occurs at all—is usually a random happenstance. This scenario is extremely common for organizations that are just ramping up a software development effort or are trying to produce major software projects on a very minor budget and are simply flying by the seat of their pants. Organizations with a very small number of software developers (say, less than four) often find themselves operating at this quality level.

The Scenario

This type of organization often lacks any kind of dedicated QA team or personnel and usually has few or no tools—such as automated testing tools—designed to help improve quality. Typically, few people in the organization have formal software QA training, and so everyone struggles to make quality happen. In many cases, few team members even realize that tools and processes are available to help improve quality. Software releases are often ad-hoc and are definitely reactive: user finds bug, developer fixes bug and runs a quick test, new software build is deployed.

These organizations tend to focus heavily on bugs as their main problem, and spend considerable effort trying to root out bugs and eliminate them. It's not surprising, then, that automated testing is often considered the ultimate solution to quality because the right automated testing can certainly contribute to a drastic decrease in bug counts.

Developers in this type of organization are often allowed to accumulate bad habits, either through laziness or, more frequently, through a simple lack of education in proper best practices. Code may be undocumented or sparely commented, and standards do not exist for basic practices such as input validation, boundary checking, and so forth. Developers may be aware of some basic best practices but may not always follow those practices consistently.

Testing—when it occurs—is typically unit testing, conducted by developers as they go. "Test data" is usually made up on the fly by developers—look for names like "AAAAA" in the customer table, for example—and rarely pushes data boundaries or includes edge-case data (such as names with punctuation marks or names that exceed the maximum length allowed). Some organizations may rely on peer testing—having developers test each others' code—and may find that peer testing doesn't add a lot of quality to the situation. New releases often bring back bugs that were fixed in a previous version, and it's rare that a new release fixes any bugs *without* introducing any new ones.

Notice that these organizations typically spend very little up-front time planning application releases. In fact, that is one of the key indicators of the "as a hobby" quality level. Software development usually begins with only a cursory design phase, and there is a lot of confidence that the IT group knows exactly how the business does its business and knows exactly how end users do their jobs. Design phases, if they exist, are typically focused on the technologies—database entity relationship diagrams, for example, network diagrams, or software module flow and connectivity diagrams. Some organizations may spend *significant* time on designing (for example) the application's database, and consider that a logical starting point to kick off a new application or an application revision.

The "requirements phase" in these organizations—if such a phase exists—often consists of a senior developer or application designer interviewing a number of managers from throughout the organization, and may result in a fairly comprehensive list of reports and outputs—although these are seldom as detailed as they could or should be. The developer or designer is often given contradictory requirements from different business leaders (a common and unavoidable situation) and is left to him or herself to resolve those conflicts and decide how the application will behave. In many cases, a requirements phase reveals nothing as much as the fact that the business leaders themselves aren't always really firm on how the business operates, and it winds up on the developers' shoulders to decide which way the application will function.

The Problem

Some of the problems in this type of organization are obvious—unstable software that doesn't work the way it needs to may be late in releasing and often doesn't accomplish everything the business needs.

- Software often doesn't do what the business needs, typically because the business didn't provide sufficient input early in the process
- Software often doesn't work the way users want it to, usually because those users had little or no input into the software's design
- Software is often buggy, which is most attributable to poor coding practices as well as a lack of formalized testing

The situation can be depressing for developers, too, who are often *trying* to do a great job and are usually surprised when the software isn't a big hit. They know their testing didn't turn up any additional bugs, so why do so many bugs show up as soon as the software is released? Most likely because the developers made two critical testing mistakes:

- First, they didn't use good test data. Ad-hoc test data often doesn't reflect real-world test data, such as punctuation in people's names, overly long entries, and so forth. The simple test data the developers used probably worked fine; it's when real-world data starts to enter the picture that the application starts to encounter problems.
- Second, they're using poor coding practices. Nothing as obvious as putting comments in their code but rather more serious issues such as failing to include boundary checks for data inputs, failing to normalize data prior to processing it, failing to handle database errors or exceptions properly, and so forth.

Another problem—one that is often only too easy to spot—of this situation is that developers spend a great deal of time firefighting. The Help desk spends a great deal of time logging problem calls. Users spend a great deal of time fighting the software. Managers spend a great deal of time commiserating about the problems and trying to figure out how to fix it. The Human Resources (HR) department spends a great deal of time trying to replace frustrated IT personnel who are leaving the company. *Time* is spent—and you know, there's a good reason that the phrase uses the word *spent*: Time is, of course, money.

The Solution

It would be nice if there was a simple set of steps that could take your quality organization from zero to hero. Unfortunately, there aren't—any more than there are a simple set of steps that could teach trigonometry to a second grader. Instead, there's a *path* that your quality organization needs to take, skills they need to develop, and techniques they need to learn—much as the second grader needs to learn basic arithmetic before moving on to algebra, geometry, and eventually trig. And, like that second grader, these steps are going to take some time—although, in the case of improving your quality, it shouldn't take 10 years!

The first step is to establish a defect-tracking process. Even if you already have a Help desk ticketing system that you're using to track bugs, look at a dedicated system for tracking software bugs. You want to be able to classify bugs—keeping track of those that resulted from simple syntax errors, for example, as well as those resulting from logic errors, poor coding practices, and so forth. You need to begin establishing coding practices, such as naming conventions, standards for input validation, and so forth. You also need to start taking control of testing. Here are some specifics:

- Establish a defect-tracking system that identifies and classifies every defect—not just bugs, mind you, but *everything* that the *end users* regard as a defect. This should include sub-optimal user interfaces, missing report columns, and other "usability" issues as well as actual bugs.
- Begin to create sets of test data that reflect actual production data and scope. If possible, you may export production data from an existing system or you may purchase test data sets. If necessary, be prepared to manually create test data. Tie the test data set back into your defect system: Every time you discover a defect that *could* have been caught through the use of better test data, add the necessary test cases to your test data set. In other words, suppose you log a defect that describes a software crash. Developers determine that the crash results whenever a customer name like "O'Shea," which contains a single quote, is used. You should immediately modify your test data set to include names of this kind so that future software releases can be tested for that scenario.
- Start to build a framework for testing automation. You might not be ready to move into testing automation, which is fine, but start to document your testing processes and procedures. Document what gets tested, how it gets tested, and which data set it gets tested with. Testing might remain a manual process for now, but by documenting those processes, you get better consistency and you get a leg up when it comes time to begin automating.

- Begin tracking testing statistics. Each test suite that you document should have some kind of identification name or number; as you begin a round of tests, keep track of which test suites have been completed and which ones are still waiting to be run. Keep track of pass/fail statistics for each test you run. This might be done in a spreadsheet, a simple database, or even in a piece of software designed specifically for tracking software testing. The idea is to get some visibility into testing: How often are you sending code back to the developers for fixes? What problems are you catching in tests, and how can you modify your development processes to proactively eliminate those problems?
- Tie your test suites into your defect tracking, as well. Each time a new defect is found, examine the testing documents and determine where that problem *should* have been caught prior to release. Modify test suites to include additional tests, test data, or other measures.

The point here is that mistakes aren't bad if you can learn from them and avoid repeating them in the future. Figure 5.2 shows the relationship between your test data, the test suites, the application itself, and the defect data collected from the application in production.

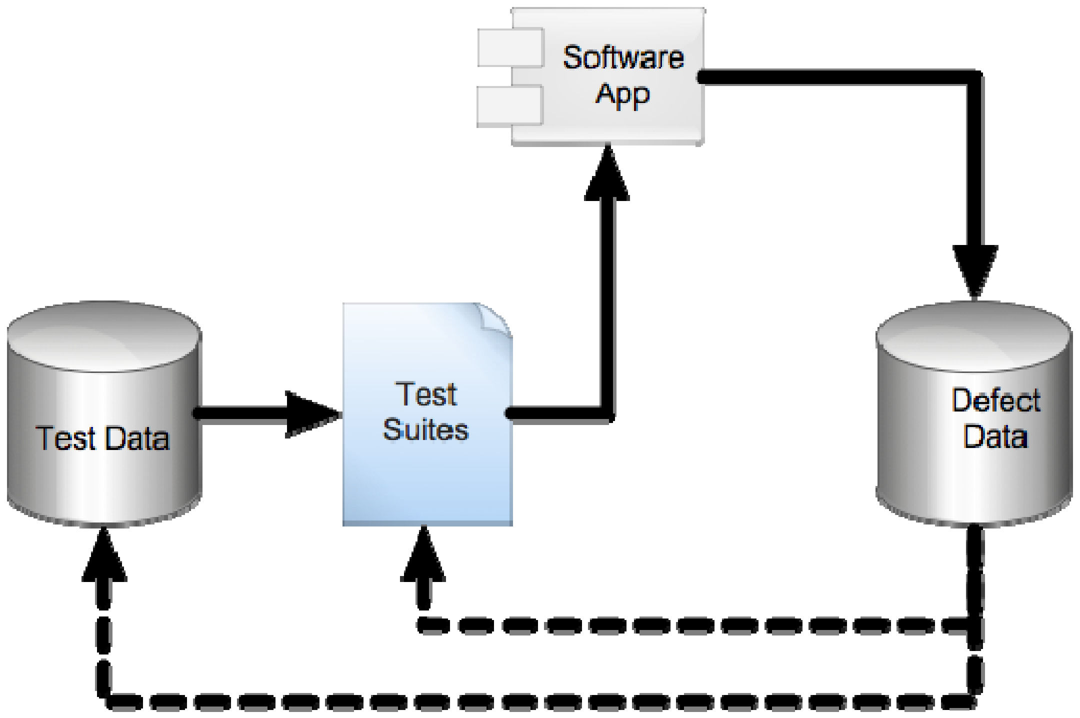


Figure 5.2: Test data feeds test suites, which tests the application; the application creates defect data, which in turn improves both the test data and the test suites.

Test Assets

At this point, let's evolve from the phrase *test data* to *test assets,* which is a broader term that allows for testing elements beyond mere input data. With many applications, the bulk of your test assets *will* be input data sets, but as you move into automation, you might develop assets that describe the user's actions in a graphical user interface (GUI) or you might create scripts that run background processes, and so forth.

The idea is for these varied assets to be organized and reusable, and for you to have corresponding outputs that the test assets *should* create. In other words, given a fixed set of inputs, what output do you expect to achieve? Software that can take the input, produce the correct output, and not experience any problems in the middle is as close to bug-free as your testing process can make it.

Finally, the big move you'll need to make is establishing a quality team: Dedicated people in charge of ensuring application quality. For now, they will focus mainly on testing the code that your developers create, but once they're in place, you're going to start building them into a quality machine that can help ensure quality for every application you build.

Quality as an Effort

Your organization has a quality team, but they've not reached a high level of quality maturity, yet. They're working hard, and producing good results—sometimes. Consistency is a problem, and an awful lot of effort seems to go into getting consistent results. In fact, you might worry that the amount of *additional* effort required to achieve consistent quality is…well, unachievable.

The Scenario

You have a dedicated team of QA folks, many of whom might have migrated from your development teams. They've got the right ideas, including documented test cases and reusable test assets. You might wonder if all the time they spend developing and maintaining test cases is actually worth it, in fact, because they're still struggling to define effective testing goals and haven't yet hit what you would consider a consistent level of software release quality. They're not able to convey "quality" in meaningful, business-related terms. You've invested some money in getting them automated testing tools, and you've put a defect-tracking system into place, but you're not sure that you're getting the value you had hoped for from those investments. You're in what is perhaps the most common quality scenario in the IT industry, and you're running up against a very common set of hurdles that can seem insurmountable.

The Quality Bar

Businesspeople are familiar with a number of "tiers" in building a business. For example, it's said that any sufficiently disciplined individual can build a business that makes a million dollars, or perhaps even two. Making that business into a five million dollar company, however, is where many entrepreneurs fail, because they have to learn an entirely new and uncomfortable set of skills—like managing people. Going from a five million dollar company to a twenty million dollar company requires even more unfamiliar skills, like how to manage a sales team, how to build an IT staff, how to deal with HR issues—all things the smaller company didn't have to deal with.

Quality works in much the same way. A given quality skill set will get you to a certain point but then you have to make a revolutionary change—change attitudes, change skill sets, change processes—to move to the next level of quality maturity.

At this level of quality maturity, the QA team is struggling to find their identity. Resentment between them and the developers may have already started, as the QA team is viewed as the "guys who try to break our code." You might have established competitions—intended to be friendly and motivating—that reward developers with the most defect-free code and reward testers who find the most defects. But the QA team is still struggling to test the right things, to efficiently update test cases and test assets to reflect real-world defects that they're tracking, and to keep the software aligned to the business.

Communicating quality is also difficult for the QA team. Oh, they have statistics—number of test cases passed, number of defects found, and so forth—but none of that speaks directly to quality in a business sense of the term. In other words, you're well aware of how much money you're spending on quality, but you're not seeing hard numbers on a return on your quality investment. Quality is being discussed as a statistical exercise, not as a contribution to the business' bottom line.

It's difficult to get IT innovations through the pipeline, and sometimes the QA team comes across as a bottleneck rather than a benefit. QA seems to hold things up, to slow things down, and you're not able to put your finger on any numerical, business-related benefit from that.

The Problem

The problem here can often be traced to the beginning of your software development process, and in order to move to the next level of quality, *everyone in the business* needs to change the way they think about software development. Let's look at this problem from a purely business standpoint:

- We've implemented a quality team
- They're telling us that they're catching bugs
- Software releases are coming slower
- QA is costing us a lot of money

You see what's missing? *The benefit.* And the statistics your QA team is producing are not regarded as a benefit. Tell a business leader that the QA team prevented 52 bugs from making it into production and the likely—and perfectly reasonable—response will be, "okay, so we need developers that won't produce those 52 bugs in the first place."

Return on investment. ROI. That's what the business wants to know: What benefit or gain is all that QA money getting us? Many QA managers simply throw their hands up in the air, unable to explain any better than via the statistics they've produced. Many business managers throw their hands up in the air, too, knowing that they need QA but not able to really comprehend whether they're spending too much, not enough, or just the right amount on quality.

The ultimate problem at this phase of quality maturity is that quality and the business don't have a common set of terminology. QA likes statistics on defects caught; the business wants ROI numbers. Interestingly, the solution to this problem also helps to solve the problem of QA being perceived as a bottleneck in the development process, and even helps developers produce higher-quality code in fewer revision cycles.

The Solution

The solution? *Business requirements.* Examine Figure 5.3, which presents a slightly different view of a fairly generic software development life cycle.

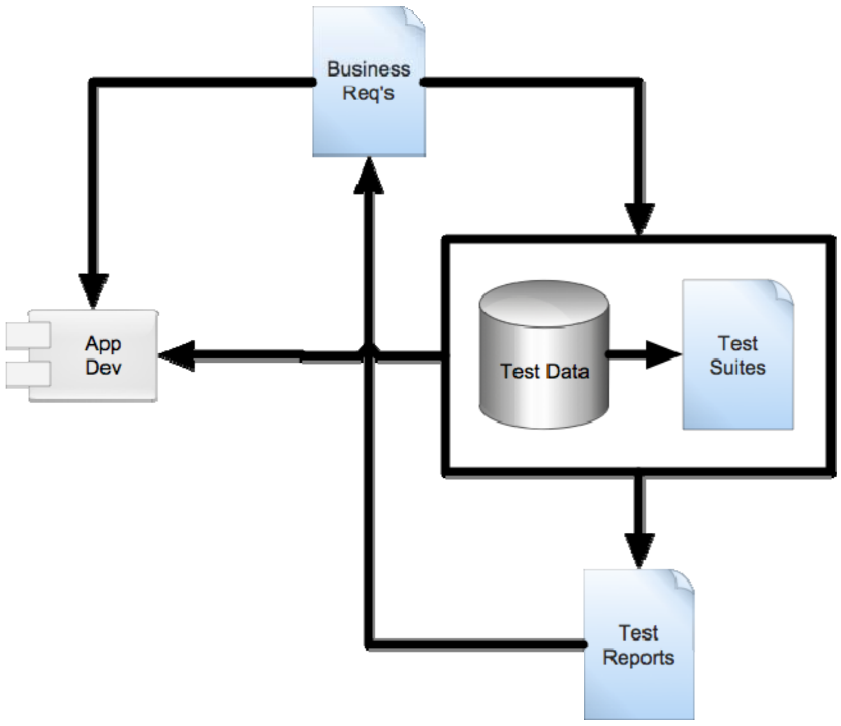


Figure 5.3: The role of business requirements in determining quality ROI..

Here's how it all needs to fit together:

- Business requirements drive both the software's design and its development
- Business requirements also drive the test assets and the test suites—the idea being that testing is intended to see how well the software is meeting the business requirements—bugs per se are incidental; a bug is merely a symptom of the software not meeting a given business requirement. "Bug free" isn't a business requirement; it's a technical means of achieving business requirements.
- Business requirements therefore drive the test reports coming out of QA: "The software has passed 80% of the business requirements and has failed to meet 20% of them."

By having a strong, clear set of business requirements to begin with, and by tying *every* point in the software development life cycle back to those business requirements, suddenly you can communicate quality ROI much more easily.

Think about it: The business presumably knows the business value of the business requirements (if they don't, that's beyond what you can fix). They can then look at the time it takes for the software to be developed, and the amount of time (and other resources) that QA requires to ensure 100% compliance with the business requirements. If the combination of development and QA exceeds the original value of the business requirements, you have negative ROI; if the combination of development and QA is less than the original value of the business requirements, the difference is your ROI. Figure 5.4 illustrates this idea in a simple chart.

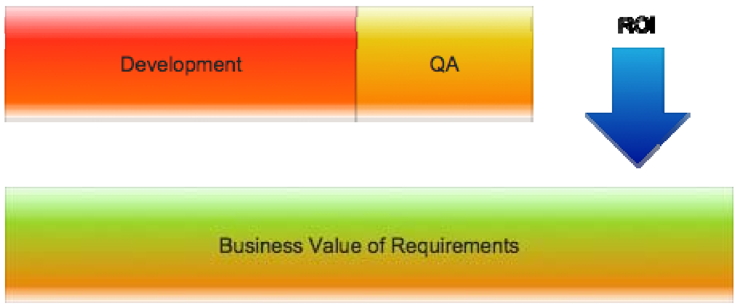


Figure 5.4: Calculating ROI by understanding the value of the business requirements.

Is this an oversimplified explanation of software development ROI? Certainly. But the point is that many businesses trying to calculate ROI *don't actually know the value of the project in the first place.* Without business requirements, QA isn't creating ROI, and QA can't tell you how good of a job they're doing because there are no milestones, no benchmarks, and no measurements that will make sense. The business has to drive the requirements, and the business has to be able to put a value on those requirements. If the business gets a piece of software that meets the requirements, what benefit does the business expect to see?

- Will the software increase sales by 200%?
- Will it eliminate the need to hire 20 more people?
- Will it enable a partnership that results in $20 million in additional revenue?

If the business can communicate a *dollar value* (ideally) or in some other way express the benefit this project is expected to realize for the business, QA can communicate quality ROI: Your return is whatever benefit you expected the project to achieve, less what it cost you to achieve it.

ROI at Home

Let's put this into different terms: You decide to build a house. The new house will be in a better neighborhood and will include a home office for your spouse, who plans to launch a part-time side business. Your spouse will also be home with the kids more often, and the kids will have access to a better school—which you anticipate will help them get into a better college and possibly even a scholarship. The house is going to cost $500,000 to build (a nice round number), but you anticipate it bringing you a benefit of $1.5 million over 5 years in terms of lifestyle improvements and all that money you'll save on college—thanks to those scholarships. A third of your anticipated return is going to be taken up by the house itself. Now you have to decide how much of your time you're going to spend on the job site inspecting the work in progress and keeping it on track.

You could just focus on the major items—layout and so forth—and spend relatively little time, say $100,000 worth of your life. You could also choose to micromanage every little detail, which might take up $500,000 of your life. Or you could aim somewhere in the middle, showing up twice a week to review everything and initiate corrections where necessary—say, worth $250,000 of your time.

That middle road brings your total investment to $750,000, and will probably do a good job of ensuring you get your full $1.5 million in benefits—for a net ROI of $750,000. The lesser approach means you'll only invest $600,000 total in the house, but spending less time on the job site might mean more and bigger mistakes, which might drive up the cost of the house or reduce the house's benefit to you—meaning you wouldn't achieve that full benefit you anticipated. You could spend a lot more time, investing $1 million total (your time plus the house), and you'll definitely get your $1.5 million in benefit—but the net return would be lower, at $500,000.

There's no wrong answer when it comes to ROI and quality. In this example, your home inspection time is the QA time on a software project, while the cost of the house itself is the software development time; the $1.5 million is your "business benefit." But the only way you could calculate all of this is by having a clear set of requirements—a home office, for example—that drove the development, gave your QA efforts something to focus on, and let you judge the proper balance for your QA efforts.

Incidentally, a strong, clear set of business requirements will also reduce cycles for software developers. With a clear goal that everyone can work toward, software can be developed more quickly, tested more accurately, and deployed in less time—simply because everyone's on the same page, there is less back-tracking, and everyone agrees on what the result should look like and how the final product should behave. QA becomes less of a bottleneck because QA and the developers both know what's expected and what the final product should look like. Developers can focus on that more strongly from the beginning, and QA and developers start to have a less adversarial relationship and more of a partnership.

A QA team without a clear set of requirements can test what they're given only to see whether it breaks—not to see if it's what the business needs. If the app turns out to *not* be what the business needs, the business loses faith in QA's ability to deliver a quality app—but it's not QA's fault.

Here are your next steps to move on to the next level of quality maturity:

- Start using clear, strong business requirements to drive development and quality assurance.
- Continue to grow your body of reusable test assets, ideally deriving them from your business requirements.
- Begin automating your test cases by using testing automation tools. You should have sufficiently mature test suites, at this point, to effectively use automated testing tools, and automation will help make QA more efficient—meaning you'll lower your overall QA costs and have a more positive impact on ROI.
- Firmly align quality to business goals. Communicate quality in business terms, relating quality back to defined business requirements.
- Create executive-level visibility into quality by rolling up business requirements into broad categories and aligning quality metrics to those categories. Give upper management insight into what quality is achieving in terms that they can understand and relate back to the business benefit they expect the project to achieve.

Quality as a Profession

You've got a QA team—possibly even more than one. Quality is a serious part of the business, and multiple projects are underway at any given time. You're still seeing inconsistent release quality, though, and you're starting to feel that you've invested in a lot of different quality tools that aren't all being used consistently.

The Scenario

When an organization gets serious about quality, develops reusable test assets, starts to automate their testing, and starts to really tie things back to a top-level set of business requirements, they're treating quality as a profession. There's a dedicated QA team, and they may be involved in several projects or you may even have different QA teams.

A problem is that each team or project isn't using the same tools consistently, and in many cases, might be trying out completely different toolsets. That right there is a problem: If you take your car to the shop, you expect to see each mechanic working from a more-or-less identical toolbox so that they can achieve more-or-less identical results. With your QA teams relying on different tools, or not using them in the same way, you should *expect* to see inconsistent results.

Beware the Kingdom

Having different QA teams working with different tools, or using the same tools in different ways, is a common sign of political problems within the business. The situation often arises when separate development teams spawn their own QA groups, establish their own methods, and acquire their own tools, and then become intent on building their own little "kingdoms" within the organization—and become dedicated to defending their kingdom, its assets, and its processes, against all comers.

This is an issue that can only be solved through top-level management intervention. Quality is quality; it should be implemented consistently throughout the organization. Having multiple quality teams is of course acceptable, especially in larger organizations, but they all need to use a common set of processes, procedures, tools, and techniques; ideally, this means they'll all be organized into a "Software Quality" department that exists in parallel to your software development department. Achieving this level of organization may involve shuffling people around, merging groups, and enforcing some cross-functional friendliness, but this effort is well worth it.

The end result of this type of quality maturity is that some software releases have great quality while others have lesser quality. Customers—the end users of this software—can't develop a reliable expectation for quality, and executive management typically has minimal (if any) visibility into what makes quality work, and how it's being achieved. Business units can sense that IT is capable of greatness, but they don't always get it; business leaders are wary of new projects because they know they can trust IT to delivery quality—but not always. ROI is still difficult to calculate, and business leaders may start to instinctively micro-manage both software development and QA in an attempt to ferret out inconsistency and create dependable quality.

The Problem

The problem comes down to two main things: consistency and communication. Simply stated, it can be tremendously complicated to fix because the problems typically stem from political/cultural ones within the organization and not from any lack of capability or desire to succeed.

The Solution

The solution, of course, is more difficult than merely stating the problem or even accepting that the problem exists. There are a number of steps you'll need to take to further mature the quality level in your organization. The main ones include:

- **Establish best practices.** Document how the quality team works, including their tools and processes. Draw flowcharts and create the "Quality Manual" for your organization. Require every quality project to utilize this manual and follow its procedures. Keep in mind that the manual isn't written in stone, though; be open to changes that improve quality processes. Each new project is a learning opportunity—continue to fine-tune the manual with each completed project.
- **Align to the business.** Continue to align testing results to business requirements. No quality project should begin until the final requirements are clearly stated; without those requirements, you can't know what to test!
- **Automate.** Rely more and more on automation for testing, tracking business requirements, tracking defects, and so forth. Automation equals consistency, and with the right tools, automation can remove the very human tendency to "innovate on the fly." Although automation tools can of course accommodate changes and improvements, they tend to force those changes and improvements to occur in a more ordered, managed fashion—meaning beneficial changes become a permanent part of the process rather than a happy accident that can't be later replicated.
- **Involve the business.** The business leaders who create a software project's initial business requirements should also be involved in developing key quality metrics. How will you look at the application and know whether it is high quality? Only the people who anticipate a business benefit from the application can answer that question, and by answering it, they inherently create an opportunity for both consistency and for more straightforward management of ROI.
- **Improve visibility.** Continue to focus on quality metrics that are meaningful to company executives and use those metrics to drive everything about the quality process. Ultimately, the company's executives are the first and most important customers of any software project: It is their business, after all, that the software seeks to improve or enhance; they should be able to tell at a glance how well the software is doing that.

Quality should ultimately become not a happy accident that can't be predicted but rather a consistent, predictable, repeatable science that works the same, reliable way across the entire organization—not just on specific projects.

Quality as a Science

When quality becomes a science, business requirements drive everything, as Figure 5.5 shows. Business requirements set the stage for the software design, which drives development; business requirements specify what will be tested, while the software design drives how the testing will be physically achieved. The ultimate quality output—quality reports—connect directly back to the business requirements, telling business leaders (and software developers and designers) how well the application is meeting their requirements. With that kind of information in hand, business leaders can more easily calculate ROI, balancing the need for further quality against the anticipated business benefits that the application is meant to deliver.

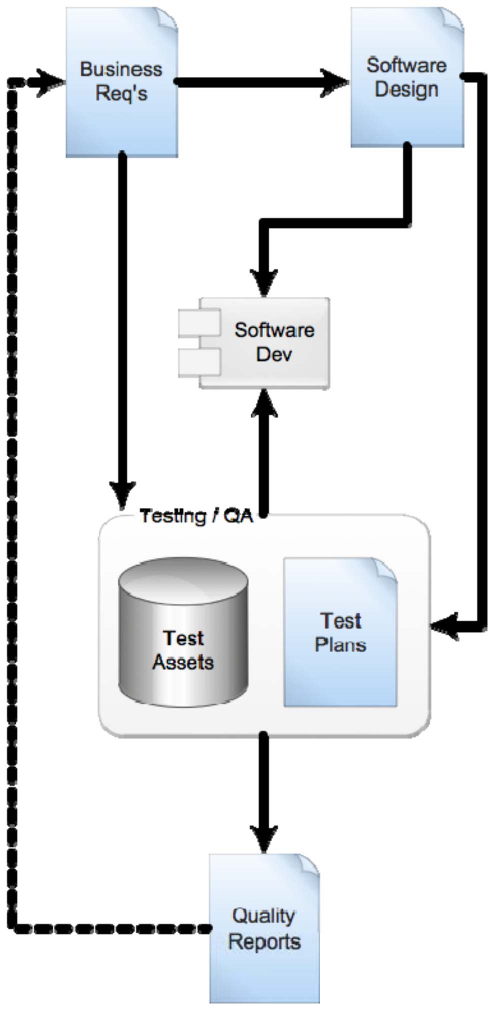


Figure 5.5: Driving the entire process from business requirements.

With automation and reusable test assets, quality becomes repeatable, consistent, and reliable. Everyone in the organization knows what to expect, and they can be assured that final software releases *will* meet their needs because those needs are the primary focus of design, development, and QA.

Quality from the Beginning

In the next chapter, we'll start exploring ways to make quality a science, beginning with a well-structured, business-focused requirements-definition phase. We'll move on, in future chapters, to the design phase, looking at fundamental areas where quality can be ensured, and then moving through the development and testing cycles.