03 43020X Ch02.qxd 3/18/03 8:48 AM Page 23

CHAPTER 2

Maintaining Quality Assurance in Today's Software Testing Environment

The quality assurance person said, "You are not following the process in the quality manual. We worked hard to write these procedures and you are changing it in the middle of the project. You can't do that."

The developer answered, "I am not changing the process; I am *automating* the process you published in the quality manual."

"But you *are* changing it," the quality assurance person insisted.

The developer took a deep breath and tried again, "I'm not changing it; I'm putting it *online*. See, this electronic form is exactly like your paper checklist form except it's online. This way we can all see it and it is really easy to fill out. No more wondering what happened to it, whose desk it's stuck on. Most of the information is already filled in by the system." The developer beamed. Certainly the quality assurance person would finally understand.

"No, you are changing it," repeated the quality assurance person. "The way you are doing it is not what's described in the quality manual. You just can't do it now."

"But why? We already know this automation will save about 40 person hours and 3,000 pages of paper each week. It will pay for itself in about eight weeks." The developer was becoming exasperated.

"Yes, that may be, but we would have to get together and rework the whole process. Then we'd have to change the quality manual, get it approved, redistribute it, call in all the old copies. You have no idea how much is involved. We are in the middle of the delivery schedule.

"No, you will have to wait until the next release. We just do not have time right now," concluded the quality assurance person.

What's Wrong with This Picture?

Before anyone gets offended, I want to say that I have found myself on both sides of this debate. What people perceive to be wrong with this picture will depend largely on the type of development process they are following. In a carefully planned and controlled development effort, quality assurance would win and the innovation would be held up until it could be fit into the schedule. In a RAD/Agile shop, a dynamic creative process will not be stifled simply because it is difficult and inconvenient to update *paper* documentation. In a RAD/Agile shop, the innovation would most likely be put in place as quickly as it is available, and the quality assurance procedure that cannot keep up with technology and innovation is likely to be discarded.

For the past several years, I have hosted special collaborative Web sites that provide features such as single-source documentation, proactive notification to subscribers of changes, interactive discussions and live chats, task lists, and schedules with automatic reminders and notifications. I have trained several teams on how to use these Web sites to minimize the problems described here and to mediate disagreements. However, even after training, these same problems persist, even when updates are instant and notification is automatic. These problems are caused by our culture, and culture is difficult to change. Even though the technology exists to solve the problem, or at least improve the situation, it will not succeed unless the culture can be changed.

To determine how to achieve and maintain proper quality assurance, now and in the future, we need to evaluate what's lacking in the current quality assurance environment. The following section looks at some common, faulty perceptions of what quality assurance is and then

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 25

examines six faulty assumptions of what quality is and how it should be measured with respect to software systems.

Problems with Traditional Quality Assurance

Let's consider the traditional definition of quality assurance. The following definition is taken from the British Standard, BS 4778:

> ***Quality Assurance:*** *All those planned and systematic actions necessary to provide adequate confidence that a product or service will satisfy given requirements for quality.*

Testers and managers need to be sure that all activities of the test effort are *adequate* and properly executed. The body of knowledge, or set of methods and practices used to accomplish these goals, is *quality assurance*. Quality assurance is responsible for ensuring the quality of the product. Software testing is one of the tools used to ascertain the quality of software. In many organizations, the testers are also responsible for quality assurance—that is, ensuring the quality of the software. In the United States, few software development companies have full-time staff devoted to quality assurance. The reason for this lack of dedicated staff is simple. In most cases, traditional formal quality assurance is not a cost-effective way to add value to the product.

A 1995 report by Capers Jones, "Software Quality for 1995: What Works and What Doesn't," for *Software Productivity Research*, gives the performance of the four most common defect removal practices in the industry today: formal design and code inspections, formal quality assurance, and formal testing. The efficiency of bug removal for these methods used individually is as follows:

Formal design inspections	45%–68%
Formal software testing	37%–60%
Formal quality assurance	32%–55%
No formal methods at all	30%–50%

When taken in combination:

Formal design inspections and formal code inspections 70%–90%

The best combination:

Formal design inspections, formal quality assurance, formal testing 77%–95%

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 26

When used alone, formal quality assurance does only 5 percent better than no formal methods at all. It is not possible to determine its relative worth when used in combination with other methods. However, it can be argued that considering the following problems, the contribution of formal quality assurance is minimal.

Traditional Definitions of Quality That Are Not Applicable

Quality assurance defines quality as "the totality of features or characteristics of a product or service that bear on its ability to satisfy stated or implied needs." The British Standards 4778, and ISO 8402, from the International Standards Organization (ISO), definitions cite "fitness for purpose" and "conformance with requirements."

Quality is not a thing; it is the measure of a thing. *Quality is a metric*. The *thing* that quality measures is *excellence*. How much excellence does a thing possess? Excellence is the fact or condition of excelling; of superiority; surpassing goodness or merit.

The problem is that the methods put forward by the experts of the 1980s for achieving quality didn't work in the real market-driven world of the 1990s and probably won't be particularly useful for most commercial software makers in the coming decade. For example, Philip B. Crosby is probably best remembered for his book, *Quality Is Free* (Mentor Books, 1992). In it he describes in nontechnical terms his methods for installing, maintaining, and measuring a comprehensive quality improvement program in your business. The major emphasis is on doing things right the first time. Crosby maintains that this quality is free and that what costs dearly is the rework that you must do when you don't do it right at the get-go.

According to Mr. Crosby's teachings:

- The definition of quality is "conformance with requirements."
- The system for achieving quality is "prevention, not cure."
- The measure of success is "the cost of quality."
- The target goal of the quality process is "Zero defects—get it right the first time."

These concepts are most certainly laudable, but they require a very high level of discipline and maturity to carry out. The fact is that this set of concepts doesn't fit the commercial software development process. The

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 27

reason for this is that the assumptions that they are based on are inaccurate in today's software development process. This situation is especially true in an environment where no one has ever gone before, and so no one knows what "right the first time" means.

Metaphorically speaking, the folks writing the definitions of quality and the procedures for achieving it were all from some major department store, but the market demand was going toward volume discount pricing. At the time of this writing, Wal-Mart is the dominant player in this field. Wal-Mart developed its own definitions for quality and invented its own methods for achieving it. It did its own market research and tailored its services to meet the actual (real) needs of that market. It didn't just leave it to the designers to guess. The other major point of distinction is Wal-Mart's overwhelming commitment to customer satisfaction. This sets it apart from most commercial software makers. Notice that there is nothing about customer satisfaction in Mr. Crosby's points. By the way, Wal-Mart is bigger than Microsoft.

Fact: If all you have is a hammer, then everything looks like a nail.

Get the right tool for the job. Overplanning and underplanning the product are two of the main failings in software development efforts today. While a safety-critical or high-reliability effort will fail if it is underplanned, in today's market, it will also fail if it falls into the trap of overplanning—trying to build too good a product for the technology environment and the market. The entrepreneurs are more concerned with planning to make money. They are not going to be bogged down by cumbersome quality assurance procedures that might give them only a marginal improvement.

So, on one end of the spectrum, we have the PC-based commercial software developers who have successfully marketed all manner of semifunctional and sometimes reliable products, and on the other end, we have the high-reliability and safety-critical software developers who must always provide reliable, functioning products. Over the years, consumers have come to expect the price and rapid release schedule of the entrepreneurial commercial software systems. The real problem started when they began to demand the same pricing and release/update schedule from the high-reliability folks. Mature companies like Boeing and Honeywell have faced a terrible challenge to their existence because they must maintain best-practice quality assurance and compete with the shrink-wrappers at the same time.

Some sobering thoughts . . . I found it a truly terrifying experience when I realized that the software monitoring system I was testing on the Microsoft Windows platform would be monitoring critical systems in a nuclear power plant. This was the same operating system that would let my fellow testers lock up the entire air defense system network of a small but strategic country by moving the mouse back and forth too fast on an operator console. These are only a couple of examples of the types of compromises software developers and the market are making these days.

Some Faulty Assumptions

Formal quality assurance principles are based on a number of precepts that are not a good fit for the realities of commercial software development today. The following six precepts are among the most prevalent—and erroneous—in the field today.

✔ *Fallacy 1: Quality Requirements Dictate the Schedule*

The Facts:

- For most software systems, market forces and competition dictate the schedule.

Traditional development models cannot keep up with the demand for consumer software products or the rapidly changing technology that supports them. Today's rich development environment and ready consumer market has sparked the imagination of an enormous number of entrepreneurs. Consequently, this market is incredibly competitive and volatile. Product delivery schedules are often based on a first-to-market strategy. This strategy is well expressed in this 1997 quote from Roger Sherman, director of testing at Microsoft Corporation:

> *Schedule is often thought to be the enemy of quality, but at Microsoft it is considered to be part of the quality of the product.*

(Microsoft studied their market and made their own definitions of quality based on the needs of that market.) Most software developed in RAD/Agile shops has a life expectancy of 3 to 12 months. The technology it services—PCs, digitizers, fax/modems, video systems, and so on—generally turns over every 12 months. The maximum desirable life expectancy of a current hardware/software system in the commercial domain is between 18 and 24 months. In contrast, traditional quality

assurance principles are geared for products with a design life expectancy measured in *decades*.

✔ *Fallacy 2: Quality = Reliability*

This equation is interpreted as "zero defects is a requirement for a high-quality product."

The Facts:

- Reliability is only one component of the quality of a product.

The commercial software market (with a few exceptions) is not willing to pay for a zero-defect product or a 100 percent reliable product.

Users don't care about faults that don't ever become bugs, and users will forgive most bugs if they can work around them, especially if the features are great and if the price is right. For example, in many business network environments in 1994 and 1995, users religiously saved their work before trying to print it. The reason: About one in four print jobs submitted to a certain type of printer using a particular software printer driver would lock up the user's workstation and result in the loss of any unsaved work. Even though many thousands of users were affected, the bug was tolerated for many months because the effects could be limited to simply rebooting the user's workstation occasionally.

Safety-critical and mission-critical applications are the notable exceptions to this fact. Consumers are willing to pay for reliability when the consequences of a failure are potentially lethal. However, the makers of these critical software systems are faced with the same market pressures from competition and constantly changing technology as the consumer software makers.

✔ *Fallacy 3: Users Know What They Want*

The Facts:

- User expectations are vague and general, not detailed and feature-specific. This situation is especially true for business software products. This phenomenon has led to something that we call *feature bloat*.

For example, if you asked several banking customers if they would like to be able to pay their bills online, many would say yes. But that response

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 30

does not help the designer determine what type of bills customers will want to pay or how much they will use any particular type of payment feature. Consequently, in a well-funded development project, it is common to see every conceivable feature being implemented.

I once ported a client server application to the Web that produced 250 different reports on demand. When I researched the actual customer usage statistics to determine which reports were the most requested and therefore the most important to implement first, I discovered that only 30 of these 250 reports had *ever* been requested. But each one had been implemented to satisfy a customer request.

✔ *Fallacy 4: The Requirements Will Be Correct*

This fallacy assumes that designers can produce what the users want the first time, without actually building product or going through trial-and-error cycles.

The Facts:

- Designers are commonly asked to design products using technology that is brand new and poorly understood. They are routinely asked to guess how the users will use the product, and they design the logic flow and interface based on those guesses.
- Designers are people, and people evolve good designs through trial-and-error experimentation. Good requirements also evolve during development through trial-and-error experimentation. They are not written whole at the outset. A development process that does not allow sufficient time for design, test, and fix cycles will fail to produce the right product.

✔ *Fallacy 5: Users Will Accept a Boring Product If the Features and Reliability Are Good*

The Facts:

- To make an excellent product, we must consistently meet or exceed user expectations. For example, text-based Web browsers in cell phones have failed to captivate the consumer (up till now), even though they provide fast and efficient use of the slow data transmission rates inherent in the current cellular networks.

- Software must be innovative in order to compete. The software leads the users to new accomplishments. Some examples of competitive innovations in the home consumer market include digital video editing, 3D animation, imaging, and video conferencing.
- As corollaries of the preceding facts, the software must provide a competitive advantage to the business user and it must educate users.

For example, let's consider color and graphics. DOS, with its simple black-and-green appearance on screen, was very reliable compared to the first several Windows operating systems, yet it passed away and became extinct.

Color printers have come to dominate the printer world in only a few short years. The cost to purchase one may be low, but the life expectancy is short. The cost of ownership is high (color ink is very expensive), yet they have become the status quo, successfully supplanting the tried-and-true, fast, reliable, and economical black-and-white laser printer.

Third Generation (3G) cell phones don't have 3G networks to support them yet in the United States, yet because of their brilliant color displays, their ability to use picture screen savers, and their ability to play tunes, they are outselling excellent 2G cell phones that offer superior feature sets that work reliably in today's cellular networks.

✔ *Fallacy 6: Product Maturity Is Required*

The Facts:

- Product maturity has little to do with the consumer's buying decision. Price and availability are far more important considerations in most business scenarios.

The very mature premier high-end digital video creation software system has been supplanted by two new software editing systems that provide about 10 percent of the features it does, at 10 percent of the price. In addition, the new systems can be purchased and downloaded over the Internet, whereas the premier system cannot. We are also seeing this trend in large system software. The typical scenario involves dropping a massive, entrenched, expensive client/server system and replacing it with a lightweight, Web-based, database-driven application.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 32

This relates also to *Fallacy 3: Users know what they want*. When analysis is performed on the current system, a frequent discovery is that the customers are paying for lots of features they are not using. Once the correct feature set has been determined, it can often be implemented quickly in a new Web-based application—where it can run very inexpensively.

Feature maturity is a far more important consideration than product maturity. As I have already pointed out, most consumers have realized that the "latest release" of software is not necessarily more reliable than the previous release. So product maturity is most often a myth. A mature product or system is typically overburdened by feature bloat.

What's Not Working: The Traditional Tools for Controlling Quality

So, now I have pointed out some of the problems associated with the ideas underlying what quality assurance is. But how do we control quality and bring about effective testing to ensure effective products and services? To determine that answer, we need to examine some of the tools used by traditional quality assurance people and see where there is room for improvement.

What, then, is quality control? The British Standard 4778 defines quality control as "the operational techniques and activities that are used to fulfill requirements for quality."

We try to *control* quality and defects in software by instituting processes to control design, coding, testing, delivery, and so forth. The idea is that if we standardize the process by which something is done, we can establish *repeatability* in the process and reliable *uniformity* in the product of the process. Further, we can automate a standardized process, eliminating the need to have people participate in it at all. People are nonuniform by nature. Process is a good tool for minimizing the impact of (nonuniform) people on a project. But, when all is said and done, software is written by people to be used by people. It is this fact that may ultimately be our salvation even though it is proving to be a challenge during this time of transition.

Software development is a creative, innovative process. Quality assurance principles were developed for manufacturing, which is based on

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 33

repeatability and is rarely innovative. Traditional quality assurance principles are a poor fit for the needs of today's fast-paced software development environment.

Traditional quality assurance principles equate quality primarily with reliability. This definition is too simplistic for today's market, which means that the priorities of traditional quality assurance are out of synch with the needs of the software development community.

Manufacturing has not been excessively concerned with human factors. Manufacturing was founded in an environment where the largest asset was machinery, where repetitive mechanical processes could be institutionalized by managerial decree, and where the role of people was to service the process.

In entrepreneurial software shops, the largest asset is the people and the intellectual property that they generate. Only a process that is accepted by the people doing the work will succeed. And only a process that treats people as assets will be accepted.

Traditional QA and Testing Tools Can't Keep Up

The following are traditional tools used by quality assurance and software testers:

Records. Documentation that keeps track of events, answering the questions when, where, who, how, and why.

Documents. Standards, quality plan, test plan, process statements, policy.

Activities. Reviews, change management, version control, testing.

These are primarily paper-dependent tools. Most traditional quality assurance tools rely on *paper*. It is virtually impossible to perform work that is more precise than the tools used to create the work. How can quality assurance and test groups be expected to be more effective than their best tools?

The Paper Problem

Paper is the biggest single impediment to software quality today. It is no coincidence that document inspection and reviews are the most effective ways to take bugs out of our software. Inspections and reviews test the

paper documentation. *Paper documentation is the single biggest bug repository in software development*. In addition to the number of design errors, miscommunications, ambiguities, and fallacies we introduce and entrench in our products, the number of errors introduced by outdated or discrepant paper documentation is a major quality problem. Furthermore, the creation and compilation of paper documents is expensive and slow.

The main problem with traditional quality control in RAD is that the productivity-enhancing tools used by software developers have far outdistanced the paper-producing tools used by quality assurance groups, testers, and documentation groups. The development of software proceeds at a pace that is faster by several orders of magnitude than the knowledge transfer, composition, layout, and review of paper documentation. The result is paper-producing groups not keeping up with the pace of development.

The distribution of information through paper documents is expensive and slow. Paper documentation is typically at least somewhat out-of-date by the time it is printed. When we need to distribute information to more people, we make more paper copies. When we need to update information, we must make enough copies to replace all existing earlier versions and try to distribute these new paper copies to everyone who had a copy of the earlier version. This is a manual version control process. It cannot hope to keep all distributed information fresh.

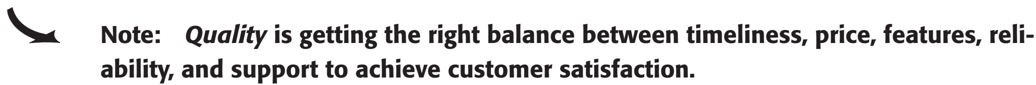
In the time it takes to explain the paper problem, I can make major changes in the functionality of a software application, recompile it, and have it ready for testing. The paper documentation is now out-of-date.

In general, development takes hours to build a release, but change management needs days to approve the release and the testers need weeks to test it. Meanwhile, the design changes daily, and documentation simply cannot keep up.

Solution: Improving the Quality Process

To improve a quality process, you need to examine your technology environment (hardware, networks, protocols, standards) and your market, and develop definitions for quality that suit them. First of all, quality is only achieved when you have balance—that is, the right proportions of the correct ingredients.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 35



Note: ***Quality*** **is getting the right balance between timeliness, price, features, reliability, and support to achieve customer satisfaction.**

Picking the Correct Components for Quality in Your Environment

The following are my definitions of the fundamental components that should be the goals of quality assurance.

- The definition of quality is *customer satisfaction.*
- The system for achieving quality is *constant refinement.*
- The measure of quality is *the profit.*
- The target goal of the quality process is *a hit every time.*

Note: Quality can be quantified most effectively by measuring customer satisfaction.

My formula for achieving these goals is:

- Be first to market with the product.
- Ask the right price.
- Get the right features in it—the required stuff and some flashy stuff that will really please the users.
- Keep the unacceptable bugs to an absolute minimum. *Corollary*: Make sure your bugs are less expensive and less irritating than your competitor's bugs.

As far as I am concerned, this is a formula for creating an *excellent* product.

A Change in the Balance

Historically, as a market matures, the importance of being the first to market will diminish and reliability will become more important. Indications show that this already happening in some cutting-edge industries. For example, I just completed a study of 3G communications devices in the Pacific Rim. Apparently, the first-to-market service provider captures the early adopter—in this case, young people, students, and professionals. However, the higher volume sales of hardware and services go to the provider who can capture the *second* wave of buyers. This second wave is made up of members of the general public who need time to evaluate the new offerings. The early adopters

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 36

serve to educate the general public. Further, the second wave of buyers is non-technical and they are not as tolerant of bugs in the system as the early adopters.

Elimination by Market Forces of Competitors Who Fail to Provide Sufficient Quality

The developers and testers who are suffering the most from the problems of outdated and inefficient quality assurance practices are the software makers who must provide high reliability. A market-driven RAD shop is likely to use only the quality assurance practices that suit their in-house process and ignore the rest. This is true because a 5 percent increase in reliability is not worth a missed delivery date. Even though the makers of firmware, safety-critical software, and other high-reliability systems are feeling the same pressures to get their product to market, they cannot afford to abandon quality assurance practices.

A student said to me recently, "We make avionics. We test everything, every time. When we find a bug, we fix it, every time, no matter how long it takes." The only way these makers can remain competitive without compromising reliability is by improving, optimizing, and automating their development processes and their quality assurance and control processes.

Clearly, we must control quality. We must encourage and reward invention, and we must be quick to incorporate improvements. What we are looking for is a set of efficient methods, metrics, and tools that strike a balance between controlling process and creating product.

Picking the Correct Quality Control Tools for Your Environment

Earlier I mentioned the traditional tools used by quality assurance to ensure that quality is achieved in the product: records, documents, and activities (such as testing). Improving these tools is a good place to start if you are going to try and improve your quality process. We need all of these tools; it is just a matter of making them efficient and doable.

I also talked about the failings and challenges associated with using paper in our development and quality processes. And, in Chapter 1, I talked about the fact that trained testers using solid methods and metrics are in short supply. In this section, I want to talk about some of the techniques I have used to improve these three critical quality assurance tools.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 37

Automating Record Keeping

We certainly need to keep records, and we need to write down our plans, but we can't spend time doing it. The records must be generated automatically as a part of our development and quality process. The records that tell us who, what, where, when, and how should not require special effort to create, and they should be maintained automatically every time something is changed.

The most important (read: *cost-effective*) test automation I have developed in the last four years has *not* been preparing automated test scripts. It has been automating the documentation process, via the inventory, and test management by instituting online forms and a single-source repository for all test documentation and process tracking.

I built my first test project Web site in 1996. It proved to be such a useful tool that the company kept it in service for years after the product was shipped. It was used by the support groups to manage customer issues, internal training, and upgrades to the product for several years.

This automation of online document repositories for test plans, scripts, scheduling, bug reporting, shared information, task lists, and discussions has been so successful that it has taken on a life of its own as a set of project management tools that enable instant collaboration amongst team members, no matter where they are located.

Today, collaboration is becoming a common theme as more and more development efforts begin to use the Agile methodologies.[1] I will talk more about this methodology in Chapter 3, "Approaches to Managing Software Testing." Collaborative Web sites are beginning to be used in project management and intranet sites to support this type of effort.

I have built many collaborative Web sites over the years; some were more useful than others, but they eliminated whole classes of errors, because there were a single source for all documentation, schedules, and tasks that was accessible to the entire team.

Web sites of this type can do a lot to automate our records, improve communications, and speed the processing of updates. This objective is accomplished through the use of team user profiles, role-based security,

[1] The Agile Software Development Manifesto, by the AgileAlliance, February 2001, at www.agilemanifesto.org.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 38

dynamic forms and proactive notification, and messaging features. Task lists, online discussions, announcements, and subscription-based messaging can automatically send emails to subscribers when events occur.

Several vendors are offering Web sites and application software that perform these functions. I use Microsoft SharePoint Team Services to perform these tasks today. It comes free with Microsoft Office XP, so it is available to most corporate testing projects. There are many names for a Web site that performs these types of tasks; I prefer to call such a Web site a collaborative site.

Improving Documentation Techniques

Documentation techniques can be improved in two ways. First, you can improve the way documents are created and maintained by automating the handling of changes in a single-source environment. Second, you can improve the way the design is created and maintained by using visualization and graphics instead of verbiage to describe systems and features. A graphical visualization of a product or system can be much easier to understand, review, update, and maintain than a written description of the system.

Improving the Way Documents Are Created, Reviewed, and Maintained

We can greatly improve the creation, review, and approval process for documents if they are (1) kept in a single-source repository and (2) reviewed by the entire team with all comments being collected and merged automatically into a single document. Thousands of hours of quality assurance process time can be saved by using a collaborative environment with these capabilities.

For example, in one project that I managed in 2001, the first phase of the project used a traditional process of distributing the design documents via email and paper, collecting comments and then rolling all the comments back into a single new version. Thirty people reviewed the documents. Then, it took a team of five documentation specialists 1,000 hours to roll all the comments into the new version of the documentation.

At this point I instituted a Microsoft SharePoint Team Services collaborative Web site, which took 2 hours to create and 16 hours to write the

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 39

instructions to the reviewers and train the team to use the site. One document specialist was assigned as the permanent support role on the site to answer questions and reset passwords. The next revision of the documents included twice as many reviewers, and only 200 hours were spent rolling the comments into the next version of the documentation. Total time savings for processing the reviewers' comments was about 700 hours.

The whole concept of having to call in old documents and redistribute new copies in a project of any size is so wasteful that an automated Web-based system can usually pay for itself in the first revision cycle.

Improving the Way Systems Are Described: Replacing Words with Pictures

Modeling tools are being used to replace descriptive commentary documentation in several industries. For example, BizTalk is an international standard that defines an environment for interorganizational workflows that support electronic commerce and supply-chain integration. As an example of document simplification, consider the illustration in Figure 2.1. It shows the interaction of a business-to-business automated procurement system implemented by the two companies. Arrows denote the flow of data among roles and entities.

Microsoft's BizTalk server allows business analysts and programmers to design the flow of B2B data using the graphical user interface of Visio (a drawing program). They actually draw the process flow for the data and workflows. This drawing represents a business process. In BizTalk Orchestration Designer, once a drawing is complete, it can be compiled and run as an XLANG schedule. XLANG is part of the BizTalk standard. This process of drawing the system is called *orchestration*. Once the annalist has created this visual design document, the programmer simply wires up the application logic by attaching programs to the various nodes in the graphical design image.

Figure 2.2 shows the movement of the documents through the buyer and seller systems as created in the orchestration process. It also shows the interaction between the XLANG schedule, BizTalk Messaging Services, and the auxiliary components.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 40

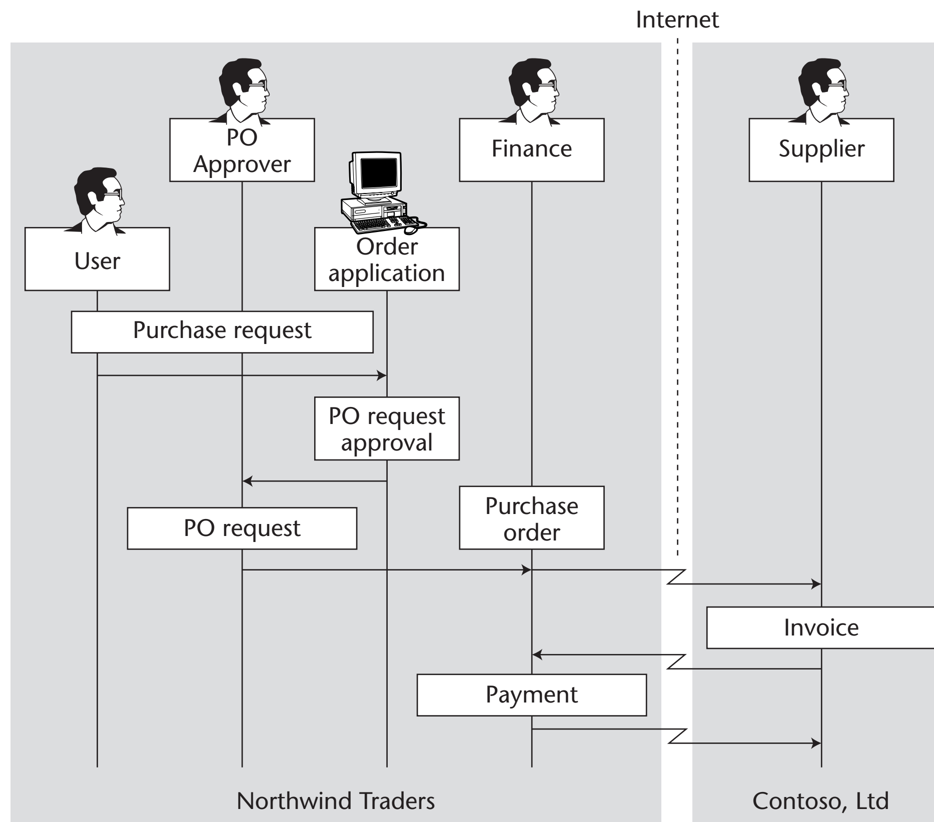


Figure 2.1 Business-to-business automated procurement system between two companies.

Fact: One picture (like Figure 2.2) is better than thousands of words of commentary-style documentation and costs far less to create and maintain.

It can take anywhere from 15 to 30 pages of commentary to adequately describe this process. Yet this graphic does it in one page. This type of flow mapping is far more efficient than describing the process in a commentary. It is also far more maintainable. And, as you will see when we discuss logic flow maps in Chapters 11 and 12, this type of flow can be used to generate the tests required to validate the system during the test effort.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 41

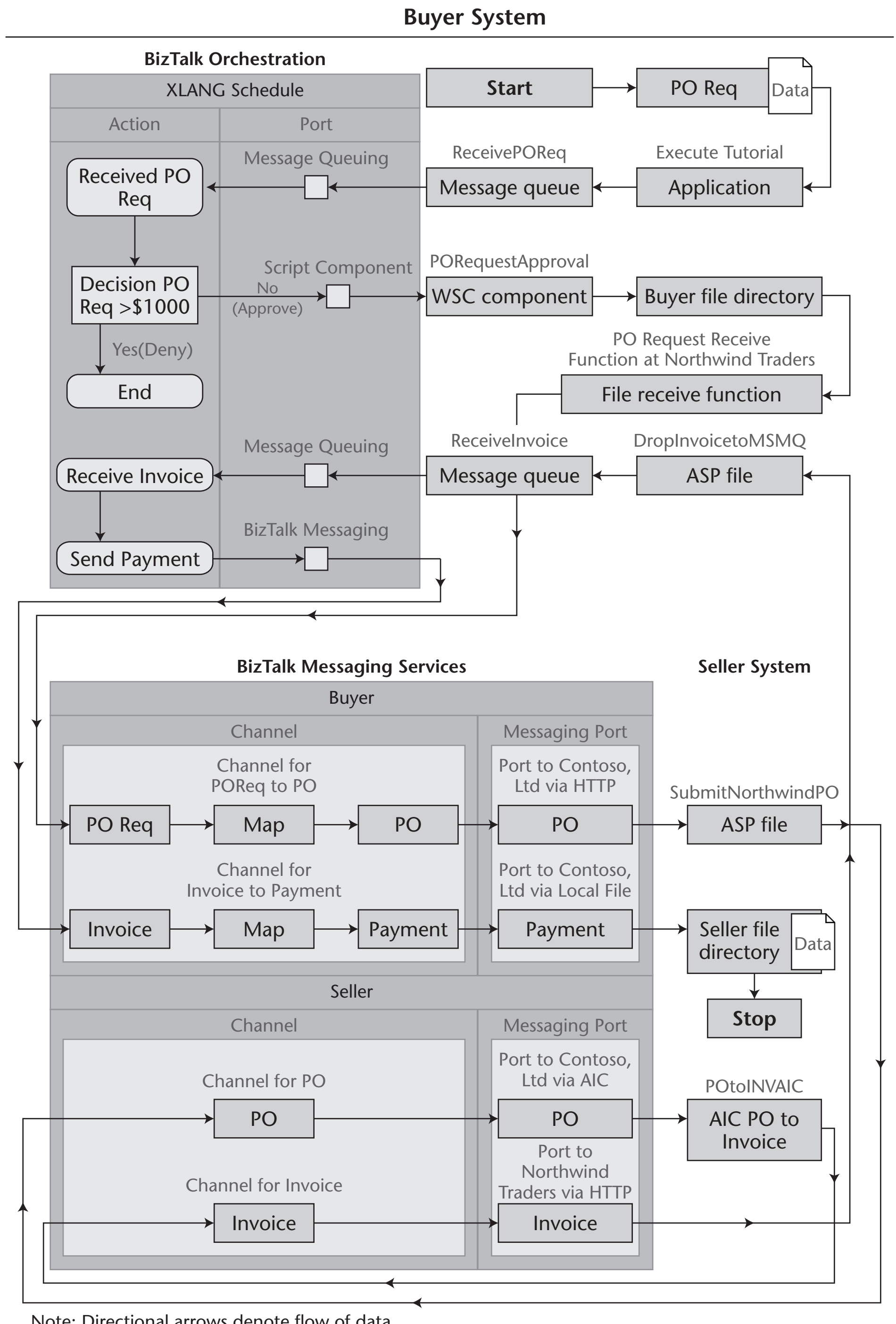


Figure 2.2 The movement of documents through the system.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 42

Visualization techniques create systems that are self-documenting. I advocate using this type of visual approach to describing systems at every opportunity. As I said in Chapter 1, the most significant improvements in software quality have not been made through testing or quality assurance, but through the application of standards.

Trained Testers Using Methods and Metrics That Work

Finally, we have the quality assurance tool for measuring the quality of the software product: testing.

In many organizations, the testers are limited to providing information based on the results of verification and validation. This is a sad under-use of a valuable resource. This thinking is partly a holdover from the traditional manufacturing-based quality assurance problems. This traditional thinking assumes that the designers can produce what the users want the first time, all by themselves. The other part of the problem is the perception that testers don't produce anything—except possibly bugs.

It has been said that "you can't test the bugs out." This is true. *Test* means to verify—to compare an actuality to a standard. It doesn't say anything about taking any fixative action. However, enough of the right bugs must be removed during, or as a result of, the test effort, or the test effort will be judged a failure.

Note: In reality, the tester's product is the delivered system, the code written by the developers minus the bugs (that the testers persuaded development to remove) plus the innovations and enhancements suggested through actual use (that testers persuaded developers to add).

The test effort is the process by which testers produce their product.

The quality of the tools that they use in the test effort has a direct effect on the outcome of the quality of the process. A good method isn't going to help the bottom line if the tools needed to support it are not available.

The methods and metrics that the testers use during the test effort should be ones that add value to the final product. The testers should be allowed to choose the tools they need to support the methods and metrics that they are using.

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 43

Once the system is turned over to test, the testers should own it. After all, the act of turning the code over for testing states implicitly that the developers have put everything into it that they currently believe should be there, for that moment at least, and that it is ready to be reviewed.

As I observed in Chapter 1, there is a shortage of trained testers, and practicing software testers are not noted for using formal methods and metrics. Improving the tools used by untrained testers will not have as big a benefit as training the testers and giving them the tools that they need to succeed.

Summary

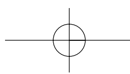
Traditional quality assurance principles are not a good fit for today's software projects. Further, the traditional processes by which we ensure quality in software systems is cumbersome and inflexible. Even more important, the traditional tools used by quality assurance are not able to keep up with the pace of software development today. Testers constrained to follow these outmoded practices using these cumbersome tools are doomed to failure.

The quality process must be reinvented to fit the real needs of the development process. The process by which we ensure quality in a product must be improved. A company needs to write its own quality goals and create a process for ensuring that they are met. The process needs to be flexible, and it needs to take advantage of the tools that exist today.

Several new technologies exist that can be used to support quality assurance principles, such as collaboration, which allows all involved parties to contribute and communicate as the design and implementation evolve. These technologies can also help make quality assurance faster and more efficient by replacing traditional paper documentation, distribution, and review processes with instantly available single-source electronic documents, and by supplementing written descriptions with drawings.

Replacing tradition requires a culture change. And, people must change their way of working to include new tools. Changing a culture is difficult. Historically, successful cultures simply absorb new invading cultures, adopt the new ideas that work, and get on with life. Cultures that

03 43020X Ch02.qxd 3/17/03 11:21 AM Page 44



resist the invasion must spend resources to do so, and so become distracted from the main business at hand.

Software quality is a combination of reliability, timeliness to market, price/cost, and the feature richness. The test effort must exist in balance with these other factors. Testers need *tools*—that is, methods and metrics that can keep up with development—and testers need the knowledge to use those tools.

Traditionally, testing has been a tool to measure the quality of the product. Today, testing needs to be able to do more than just measure; it needs to be able to add to the value of the product. In the next chapters, we discuss various approaches to testing and some fundamental methods and metrics that are used throughout the rest of the book.