

---

# Shipping the Right Products at the Right Time

---

## A View of Development and Testing at Microsoft

By Roger W. Sherman, former director of testingMicrosoft Corp.  
© March 1994 by Microsoft Corporation, All Rights Reserved

### The Microsoft Corporate Environment for Testing

Microsoft is one of the most successful companies in the history of software. It has achieved this success, in part, by developing the capability to ship the right products at the right time. The evolution of this company from a small group of developers to the world's largest software development house resulted from having a clear view of the customer and strong ideas about what the technology could do for users. The success of Microsoft may also be attributed to the maturation of technically oriented testing departments and to the evolution of a product development process called "milestones." A common view of product quality is shared by testers, developers, and program managers. Microsoft builds products that meet customer needs, and it leverages new technologies to put them in the hands of users in a timely manner.

Microsoft is a company that puts people first. Each employee has his or her own office, with a door and a phone. A typical test engineer has two or three Pentium™ or 486 PCs (many have more) or a Macintosh. Procedures are in place to ensure that test engineers are compensated at the same rate as developers with similar technical ability. In the Systems Division, most testers are developers who have chosen to work in testing departments. Testing is a viable technical career path at Microsoft—test engineers are promoted based on technical ability. Management skills or an interest in management positions are not required for testers to advance within the company.

Product teams are small and autonomous. Business units compete with companies many times their size. They are empowered to make

all critical decisions about their product content, pricing, and strategy. Testing groups in business units choose their own tools and own their processes. But communication between groups is frequent, so that if something works in one group, other groups will be quick to plagiarize and improve on it. Test managers meet monthly to share best practices.

## Three Facets of Quality

At Microsoft, the quality of a product is defined in three dimensions. The first is product definition, or what is sometimes called the feature set. The second dimension is “shipping on time,” or schedule. 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. The third dimension is the traditional focus of Quality Assurance: reliability. The term “reliability” is defined at Microsoft as the rate at which a user will encounter anomalies. One goal of implementing the product (coding/testing) is to maintain the reliability of the product while new features are being added and to increase reliability in the system testing phase until it is good enough to release the product. Depending on the market and available technologies, the relative importance of each of these quality dimensions may differ. For example, a low number of defects might be the most important in a mature market where competing products are not otherwise differentiated. In another situation, being first to market may take precedence over reliability. It is an axiom of the personal computer business that the first product to claim a majority market share on a new platform or in a new product category may have that position for a long time. In this case, product definition and schedule may be relatively more important than the number of defects, especially if there is no preexisting installed base. At other times, features might be cut from a product to increase the amount of time available for testing or to increase the certainty of making a ship date.

While the *relative* importance of each quality dimension may change in proportion to the others, the absolute importance of each is high in our business. One cannot succeed unless the capability of the product development group in each of these three areas is very high. The cost of shipping a product with a recall-class defect can be fatal to the profitability of a product group. Microsoft’s profitability is based on high numbers of unit sales and low margins. If a product is recalled, millions of product units may have to be scrapped, inventory from the marketing channels may be returned, and new disk sets may need to be manufactured and distributed. In addition, re-releasing the product may have to be repeated for each foreign language version of the

product as well. The cost of fixing a problem in the field is extremely high. Most defects also have an associated cost in customer support. Even if the defects do not warrant a recall, they may incur high support costs. Product support costs are charged directly to the business unit that produces the product. Finally, if a product is late, it may *never* recapture the market share it loses to a competitor. And if it is lacking in imagination, features, and functionality, it will have a very short life—giving the product team less time and resources for the next version.

At Microsoft, development, testing, program management, and product support groups decide as a team to make carefully considered trade-offs between the three dimensions of quality, from the time coding begins until they agree that the product is ready to be shipped to customers.

## Product Definition (Feature Set)

Every major version (version 1.0, 2.0, and so forth) of every product at Microsoft starts with a vision statement. The vision statement is a document that describes the product positioning, key features, price range, and desired ship date. The vision statement may also describe competitive products' strengths and weaknesses. The purpose of the vision statement is to articulate key product positioning to the entire team, so that the whole development process is focused on the right objectives and that decisions regarding product quality (as defined above) are strategically focused.

The document is produced by the marketing group and has many inputs. If the product is a new version of an existing product, Product Support provides the list of top cost generators. Microsoft also has a call-in phone line for user suggestions on improving the product. Letters from users and competitive reviews in the trade press are also used as input.

Microsoft also engages an outside company to do annual customer satisfaction surveys for its products and those of its competitors. This survey identifies key factors in customer satisfaction and rates Microsoft products in those areas. Individual product teams also conduct their own customer satisfaction surveys and registration base studies.

Another activity that receives public attention is Microsoft's usability testing. Users from the target market are given prototypes of products under development, with a task to accomplish using the prototype. Their actions are videotaped and annotated. This provides product designers feedback on whether the design is easy to use or needs further refinement. Instrumented versions of our products are also used in a similar way. These special versions record user actions, so designers can see how customers used the design to accomplish tasks. Recently, designers used the data from instrumented versions of our Office products to discover that users were not using the toolbar for common tasks, but were going through menus instead. They eventually determined that many users could not remember what the icons on the toolbar meant, and the product design was changed so that an explanatory caption appeared whenever the mouse cursor passed over an icon.

This solution was then tested in the Usability Lab to confirm that users naturally migrated toward using the toolbar, which was quicker and easier than accessing the same functionality through menus.

While these inputs help determine weaknesses in existing products, more is needed to make a product that exceeds customer expectations. Features of this type may come from extensive studies of customer behavior or shifts in available technology. Examples of the latter include OLE (object linking and embedding), new Windows® platforms, reusable code from other projects, or the presence of high-power processors in the target market. Groups often take retreats for creative brainstorming on product ideas.

Once a product vision is defined, it is challenged by peer reviews and by review from the senior staff. One way to guarantee criticism from the very top is to plan a product lacking in creativity or one that does not aim to be best of breed, leap-frogging the competition. The drive for this level of quality in product definition at Microsoft is most aggressive at the top.

The next step in defining the product is the product specification. This is a document that describes in detail how the product will work. The product specification receives line-by-line review from testers, developers, and user education staff. When the product specification is sufficiently detailed, the development team builds a schedule, based on the specification.

## Shipping on Time; Process and Schedule

The development schedule divides the product into features and then estimates the time required to code and unit test each feature. Features are grouped into sets, which will be delivered to testing simultaneously at specified milestone dates. In a typical project there are three to five milestones of this type. While development is working on the code for a milestone, testing creates or writes test cases and test specifications for that feature set. Just prior to formally releasing the code to testing, developers often ask testers to informally test their code to ensure that the release will be solid. In many groups there is a formal buddy system, which assigns each tester to support an individual developer. Prerelease testing and the buddy system help developers rid their code of the most obvious issues in the most efficient manner. When the release is made to testing, development also publishes a Testing Release Document (TRD), which describes exactly what is testable. Testing takes the release and runs a series of tests to determine whether the release is testable. To accept the release, the testing group must be able to test all coded features (no blocking bugs), load test files, and run preexisting automated tests against the product. Testing may reject the release if it is not testable. After acceptance, testing runs its newly developed test cases against

the release, and development fixes the bugs. The bug fix cycle continues until the test cases yield no additional bugs. The milestone is then certified and development, and then begins to work on the next feature set for the next milestone.

After each milestone the team holds a postmortem with its managers. The postmortem is a meeting where the work of the team in coding and testing the milestone is reviewed. The role of the manager is to encourage honest conversation and to make sure that all necessary issues are raised. The team examines its process for the milestone, the test results, and the schedule. If necessary, the ship date is adjusted, features are cut, process is changed, or people reassigned to tasks deemed critical to the project. This is one instance where trade-offs among the three dimensions of quality are adjusted.

The last milestone of this type is called code complete: all features are coded, there are no bugs blocking testing, and all native coding (speed optimization) is done. From code complete until release to manufacturing (RTM), development fixes bugs in releases made once or twice a week. During this period the schedule includes a number of other milestones, such as beta testing, configuration and printer testing, and complete passes of all test cases, automated and manual. At some point development resolves issues faster than testing can generate them. This is called bug convergence. As the active number of issues begins to decline, one can begin to estimate with more precision when the product can be released. It is only a matter of time before development is able to make a release that resolves all active issues. This release is called the zero bug release (ZBR). From this point on, all changes to the code are carefully reviewed. The goal is to stabilize the product. Releases increase in frequency until they are made daily. Gradually the bug find rate drops to less than one per day. Although each team may decide its own exit criteria (called FAT, for final acceptance test), most teams will not release the product until it has withstood five full days of testing without yielding a must-fix bug.

After the product has shipped to customers, the entire team will hold a postmortem to review what went well and what could be done to improve the process, tools, or work environment. The results of these postmortems are published for the rest of the company and are a major means of sharing best practices between groups.

The primary benefit of the milestone process, however, is to maintain code stability. When a code base becomes too unstable, it will not solidify with additional testing and bug fixing. Bug fixing may actually make things worse. The best solution in this case is to throw away the code and rewrite it. Rewriting the code is very costly and a

solution of last resort. Maintaining a stable code base increases the probability that testing and bug fixing will improve the product, rather than destabilize it.

The milestone process also reduces the number of bugs found after code complete. The milestone process makes it possible to fix bugs earlier in the development process, when it is less expensive to do. Low-priority bugs can be addressed earlier in the cycle as well, when it is cost effective to fix them.

## Implementation and Tools

### Making Sure It's Getting Better, Not Worse

Microsoft uses an internal groupware product called SLM (Source Library Manager) to control source code. Each developer has all the code necessary to build the project on his or her own machine. After developing new code and unit testing it, the developer builds the product and runs a suite of automated test cases against it. The cases are often provided by the testing group. The purpose of the quick test is to demonstrate whether the new code breaks functionality in the product. If the new code passes this test, the developer does a check-in, and his or her code is included in the daily build of the project. If the developer checks in code that breaks the daily build, he or she most likely will be given daily build responsibilities until another developer commits a similar transgression. Other playful forms of stigma are also employed.

### Making the Code Testable

Development teams at Microsoft work closely with test engineers to make products testable. Development includes many *asserts* in the debug version of their code. The asserts test the value of data structures while the code is executed, often reporting a problem before it manifests itself through the user interface. Most products also include an invisible menu that can be activated manually or through an automated script. This menu activates code within the application to simulate resource failures (low memory, low disk space, and so forth), to shake the heap, to verify the integrity of data structures, to fill memory with assigned values, and so forth.

### Automation Tools

The most common automation tools are C or C++ (for testing API layers), MS-Test (testing through user interfaces, or managing test drivers), Visual Basic® for Applications (VBA, for interapplication testing) or the macro language that ships with the product

(Microsoft® Excel's macro language, WordBasic for Word, AccessBasic for Access®, and so forth). Verification is done through a variety of means: smart file comparisons, directly querying the application, direct queries to the operating system, and, as a last resort, screen dumps and comparisons.

There are also a number of test-case execution managers in use at Microsoft. Among the most interesting is Teacher/Pupil. When testers leave for the night, they enroll their machines as pupils in school. The Teacher (server machine) downloads an automated suite and the pupil machine executes it, reporting test results back to the teacher machine. The teacher can determine whether a machine has frozen, and it can reboot it if necessary.

Automated cases are used in a variety of ways in Microsoft. Groups that have to test APIs make heavy use of automation since that is the only way to test them. The Windows NT™ test group wrote over 2 million lines of test code (25 percent of the total SLOCs of source code for the shipping product). Most application groups have not found it cost effective to automate everything. Other groups achieve nearly 100 percent statement coverage with their automated tests, but have found that this never tests the code completely, so automation is used for particular tasks. The most efficient use of automation is regression testing. Regression suites are made up for breadth testing nightly builds of the product. They are enhanced and augmented as the project develops. Since regression tests are rerun more than any other kind of tests, automating these tests has the greatest payback for the time invested.

Another application of automated tools is in the use of *monkeys*. A monkey is a tool that drives an application in random fashion, sending keystrokes that may or may not be intended by the product designer or developer. Monkeys of this type are used in an informal way to determine how robust a product is. Further up the evolutionary chain is the intelligent monkey. This kind of automated tool understands the user interface and sends valid user input to the application under test. An intelligent monkey can be calibrated to imitate customer usage if such a profile is known.

When imitating a statistically valid customer profile, the intelligent monkey can be used to determine a true mean time to failure. Most intelligent monkeys in use at Microsoft allow tracing when a fault occurs to determine the set of keystrokes that caused the failure.

## Beta Testing

There are several kinds of beta testing at Microsoft. Marketing runs beta tests that have very little to do with testing the product for defects. Marketing betas are quite useful for getting customer feedback on the way the features have been implemented, and sometimes adjustments are possible before release to the general public.

Technical betas, as they are called in Microsoft, are for the purpose of finding bugs. Internal betas (called alpha testing elsewhere) are popular and effective. For many Microsoft products, internal betas have been far more useful than external beta tests. Beta versions of products are posted on internal servers and everyone in the company is invited to download them and try them. Anomalies are reported via electronic mail. Some developers are so proud of their code that they offer bug bounties—cash—for anyone who can find a bona fide bug in their product.

External betas can be quite large and are run by a centralized group that has developed expertise in managing beta sites. The internal beta group can put together a beta site list that duplicates the user profile of any product's target market. External betas are most important to the Systems group, since they write code that may be the most affected by hardware configuration. In spite of the large number of beta sites and the length of beta tests, bugs from beta testing account for less than 5 percent of all bugs found in most products, including operating systems. While some groups find those remaining 5 percent important enough to continue beta testing, other groups have abandoned external beta tests. Internal testing for these groups has been effective enough.

## Anomalies, Incidents, and Bugs: Tracking and Reporting

An internally developed tracking tool called RAID (Reporting And Information Database) enforces a procedure for resolution of problem reports and bugs. Originally developed for testers and developers for tracking defects, RAID databases have become complete to-do lists for shipping projects because resolution of each issue is enforced and tracked. Once an issue has been opened in the database, it is assigned to a developer or a program manager for resolution. The developer may fix a problem if it is a defect or assign it back to the tester if he or she cannot reproduce the problem. The developer may also assign the issue to a committee of the test lead, development lead, program manager, and product support lead for resolution. This committee then decides how the issue should be resolved. Typically, the committee will choose to fix it if its members believe it is a defect, postpone it if necessary, or do nothing if the product behaves as they

intend. Once resolved, issues are closed. Test engineers are the only people who can close an issue, therefore assuring every issue is completely fixed or appropriately reviewed.

Issues in RAID are ranked by severity and priority, and they are tracked by status (ACTIVE, RESOLVED, CLOSED), assignee, product area, open date, change date, closed date, version number, and how the issue was resolved. Product teams add additional fields to track other items, such as how such an issue could have been avoided in the first place.

RAID provides data for most of the metrics used in the company. Because everyone uses the same tool, conclusions using these metrics are widely applicable. Data from RAID databases is our primary source for analyzing process improvements and comparing best practices between groups. Because individual teams can add their own fields to issue reports in RAID, test engineers also use RAID to test hypotheses on the causes of bugs and ways to improve development efficiency. RAID is an important tool for improving our capability to deliver world-class products, on time.

## Conclusion

From its start 19 years ago, Microsoft has gone through many stages of evolution in its capability for producing the right software at the right time. From small teams developing software for early adopters, through the use of more defined processes today, Microsoft still embodies the organizational values that made it successful: small, empowered, focused teams, producing the best product that they can for their customers.