Gartner

Research

Publication Date: 3 August 2010 **ID Number: G00173432**

Q&A for Sizing the Test Team

Thomas E. Murphy

IT organizations struggle with the issue of software quality, trying to understand how much testing is enough and how many testers are required. Automation can provide some assistance, but quality is best achieved by a holistic approach.

ANALYSIS

What is the proper ratio of testers to other resources in a software project?

Boiled down to the ratio of testers per developer, there is a wide range. Primarily in software companies, where the software is the business, ratios from 1:1 to 1:3 are often found. In nonsoftware companies, the high end tends to be around 1:3 (see Table 1). Companies that focus on packaged solutions and tend to have a ratio nearer to 1:5 and down to 1:10. On this lower end of the scale, we tend to see a higher rate of project failure or a focus on outsourcing. We have seen companies with ratios as low as 1:20, but these companies have software that is undergoing very little change or on which a lot of testing work is being done by the end user and the user acceptance testing (UAT) process is overloaded. The users performing testing are trying to fill in the gap of not enough trained resources earlier in the project. This results in increased overall cost, because defects are found later in the life cycle. In general, companies with fewer testing resources are trying to reduce the cost of testing and they succeed in this, but end up spending more on maintenance and have a higher defect cost because defects are found later in the development cycle (see "Toolkit: Defect Containment and Quantitative Defect Management").

Table 1. Per-Developer Ratios

Development Type	Average	Best in Class	Lagging
ISV	1:3	1:1	1:5
Web/E-commerce	1:3	1:1	1:5
Enterprise Custom Development	1:5	1:3	1:10
Departmental Custom Development	1:10	1:3	1:20
Package Implementation	1:10	1:3	1:15

Source: Gartner (August 2010)

Overall, spending on testing and quality is driven by the perceived risk involved. Software companies know the value of software — it is their business. Recognizing the value of software and the cost of downtime is a key determinate in how much a company expends on testing and quality activities. New technologies, government regulations and competitive advantage all feed into risk. For many companies, the risks of most concern are cost and time. Unfortunately, less quality generally means greater long-term cost.

Software testers are not the only ones involved in software quality efforts. We are generally asked about developer-to-tester ratio, but this does not measure the overall percentage of effort placed on quality during a software project. This may include testers, users, developers and business analysts all taking on a portion of the testing and quality role. Organizations invested in agile practices are engaging many roles in the quality equation.

There are two types of test teams in enterprise application projects. First is the set of testers performing what's usually called a system test, which is typically done by IT and, often, developers. The second is what's usually called an acceptance test, which is typically performed by a dedicated test team in IT or a team of non-IT personnel (hence, UAT). Overall, these testing activities usually comprises between 20% and 35% of the overall project's effort, split between the two types of testing.

Does the ratio vary over the project's lifetime?

The ratio can vary depending on the type of project and the project methodology. Projects that are new custom developments require a lot of testing, but also require a lot of development or, in the case of a package implementation, a lot of customization and integration. Once an application is in production and projects shift toward fix-oriented maintenance and updates, there may be fewer developers involved but the testing load won't reduce as much (especially if automation support is weak). The bigger shift in ratios comes with the project method (e.g., waterfall, agile) and when resources are used. In waterfall and many traditional iterative methods, testing lives in its own silo and each functional team has its turn at being the primary resource: business analysts, architects, developers, testers and operations. In this format, early in the project the percentage of the team that is from quality assurance (QA) will be none to very limited, and it will stay this way until the project makes it to the testing phase, at which point most or all of the resources will be from the QA group. With agile practices in place, the quality team is involved more directly throughout the life of the project. This is in keeping with the goals of finding defects as early as possible, thus reducing fix costs and keeping the state of the project in "shippable code" for every iteration. Many of the concepts of early involvement can happen in a waterfall as well as in agile. Thus, having the QA silo involved during requirements and signing off (before requirements move to the next phase) is a best practice, no matter what the life cycle is. Over the life of the project, the total resources required may be the same with either project style, but it has been shown that agile teams are generally more productive in large part because defects are located sooner, thus dramatically reducing rework costs.

If tester role is not part of business analyst role, how many testers should you have?

Especially with package applications, the business analyst is playing more of a role in quality. For many companies, the primary testing staff is made up of these analysts and end users. The value of using business analysts is knowledge of the application rules. However, it doesn't cause a change to the ratio or number of people involved in testing. You may use fewer people with the primary job role of tester, but the overall effort will be the same. The key is that even if business analysts or end users are involved in performing the tests, there is still a requirement to have test architects involved in building the test plan and working with the business analysts to define the test cases (see "Mix in the Right Test Skills to Achieve Quality"). Regardless of role, skill is the major thing to consider when planning how many testers you need. Also, the value of using end users is the ability to have a large number of people with varying skill levels, a variety of system configurations and potentially different usage patterns using the software. This is why vendors have beta test cycles. It opens the software up to ad hoc testing and configurations. However, it doesn't mean you will have good focused efforts, and trained testing staff should have a higher productivity ratio both in finding defects and in the accuracy and detail of defect reports. Thus, while using users to perform testing may provide a large pool of resources, it may not reduce the cost of quality. Engaging users and business analysts in testing is a good idea, but does not replace the need for people formally trained in software quality/testing architecture, processes and tools.

What is the impact of developer involvement in quality to the tester/developer ratio?

Developers are always ultimately responsible for the quality of an application solution, but with agile methods, there are several practices that affect how the developer may participate in software testing. These practices, in general, don't affect the tester/developer ratio, but should shorten the overall development life cycle by identifying defects earlier, thus reducing their costs.

The main driver for this question comes from the idea that if developers are utilizing test-driven development and building and executing unit tests, then perhaps there is less need for testing downstream. While this is an effective way to drive quality earlier in the project, it doesn't mean a reduction in the number of testers (see "Driving Quality Upstream by Using Unit Testing"). If your testers do unit testing, the test team will have less need around this task; however, developers use unit tests in a different way, and generally all applications are shipped with less than an optimal amount of testing performed, leading to many defects found in production, where they are the most expensive to fix. So, rather than looking at developer unit testing and other quality activities reducing the amount of testers required, look at it as a way to improve quality and reduce maintenance costs, along with improving the ability to deliver software on time and to contain defects earlier in the project. This can reduce overall time in testing and overall project length, in general, because there will be fewer downstream defects, and less need for rework and the associated retesting that goes with it, but the real focus is that defect costs will be lowered (see "Toolkit: Defect Containment and Quantitative Defect Management").

As the automation of scripts develops, how many testers will you need?

The challenge of automation is that it produces an application to test the target application. Most test automation tools are still based on a framework and scripting language that have test engineers producing a test script (a piece of code) that will drive the application to perform the test. This is true whether the vendor has covered this over with "record and playback" or whether the test creates the script by hand. These scripts need to be maintained as the application changes. In general, organizations with successful test automation have approximately 20% of their test cases automated. Testers are still required to build and maintain these scripts. The value generally isn't a need for fewer testers, but potentially shorter test cycles. This can translate to a need for a reduction in the number of people needed to perform testing; however, for most companies, given the minor percentage of automation results in a reduction, an overall 5% to 10% reduction in labor would be the most seen.

Generally, the time it takes to create an automated test represents about six times over the cost of manually executing the test. This is improving in some of leading- edge tools or in specific domains, such as package applications. We are seeing, especially in package applications, cases where automation is becoming successful at hitting 50% of the test suite and above. This is creating a dramatic drop in test cycle time, but often this reduction in labor need is sucked up by increased application complexity and a need to perform more types of testing, such as application security, that haven't been performed in the past. In addition, performing the test is only a small part of the overall process; there is still test data management, design of tests, defect reports, etc. Thus, we will see an improvement in test productivity during the next five years, but it will require organizations to migrate to more-complete application life cycle management solutions (see "Application Life Cycle Management Has Found a Home"), and there will be a time period to build an effective automation foundation. The end result isn't a change in the ratio of testers to developers, but improved overall productivity. In general, we see that automation efforts are hampered by a lack of time available to the test team, and that overall companies focus testing efforts not on quality, but on time. The team is given a specific window, and this window often collapses as upstream work slips and the end date remains fixed. This, in turn, means that tasks like automation are not attended to, and the cost "out the door" is contained, but traded off for much higher maintenance costs.

Does process automation affect the need for testers?

Like test automation, process automation doesn't really affect the ratio of needed testers; it drives the productivity of the entire team. There are good strides being made here, especially with the

use of virtualization software. One of the biggest time wasters for developers and testers is the loop between a tester finding a bug and the developer trying to reproduce the bug so that it can be debugged and fixed. Often, configuration changes create this problem. In a virtual environment, the application state can be captured by taking a snapshot of the virtual machine (VM). This can be added to the defect report, enabling the developer to start the VM, attach a debugger to the process and see specifically what occurred. In addition, utilization of virtual environments fits nicely with the use of centralized hardware and lab management software. Utilizing this software, prebuilt configurations can be created, scheduled and loaded for testing use.

Another area that process automation is helping in is traceability from requirements to validating test cases; however, this is reliant on good, accurate requirement documents. In general, improvements are available now for many areas of the overall test process — test case generation, test data management and data masking, test platform management, and test creation and execution are all areas where visionary companies are providing solutions to make the process run more smoothly. Again, these generally don't affect the ratios as they are just keeping the tester up with the increasing productivity of the developer and rising application complexity. These solutions make the overall process more productive, which speeds the time to find and fix defects (the most important task) and better enable compliance and the discovery of deeper issues. These will reduce overall software maintenance costs and enable a better shot at driving reuse. The number of testers required is based on the complexity of the application, and this is also the main driver of how many developers are required. Anything that increases tester productivity is also increasing developer productivity, but most of this increase in productivity is being absorbed by increasing complexity. It doesn't mean the tools, processes, etc., are not useful, but that the idea should not be to focus on the ratio, but rather on the end results of higher customer satisfaction, faster time to market and reduced maintenance costs.

RECOMMENDED READING

"Mix in the Right Test Skills to Achieve Quality"

"Improved Collaboration Drives Productivity and Software Quality"

"Using Continuous Build to Drive Quality"

"Requirements Form the Foundation of Software Quality"

"Using Metrics Effectively: Proving and Improving the Business Value of IT"

REGIONAL HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
U.S.A.
+1 203 964 0096

European Headquarters
Tamesis
The Glanty
Egham
Surrey, TW20 9AW
UNITED KINGDOM
+44 1784 431611

Asia/Pacific Headquarters
Gartner Australasia Pty. Ltd.
Level 9, 141 Walker Street
North Sydney
New South Wales 2060
AUSTRALIA
+61 2 9459 4600

Japan Headquarters
Gartner Japan Ltd.
Aobadai Hills, 6F
7-7, Aobadai, 4-chome
Meguro-ku, Tokyo 153-0042
JAPAN
+81 3 3481 3670

Latin America Headquarters
Gartner do Brazil
Av. das Nações Unidas, 12551
9° andar—World Trade Center
04578-903—São Paulo SP
BRAZIL
+55 11 3443 1509

Gartner