Realtime publishers

The Definitive Guide™ To

Quality Application Delivery

sponsored by

MICRO FOCUS®
Leading the Evolution™

Don Jones

Copyright Statement

Chapter 10: Quality Application Delivery

In this chapter, I'd like to reflect on some of the major points we've covered. I'd also like to provide some real next-step actions that you can take in your environment to start immediately improving application quality and to start tracking your progress toward increasingly-better application quality. I'll focus especially on how tools—once you've adopted an attitude and environment of quality within your organization—can help expedite the process of creating higher-quality applications.

Finding Lost Quality

Let's review: Where is quality lost? What hurdles must project teams overcome in order to achieve better application quality? What areas of the project life cycle deserve the most attention, and what specific activities can be improved upon to introduce more quality into the project? Although lost quality is often *detected* most easily in testing, testing is rarely where the quality is actually *lost*. In fact, although testing can often detect a wide range of missing quality—bugs, poor performance, bad security, and so forth—many of these *are not fixable* (without essentially starting over to some degree), so it pays to catch the lost quality as early as possible.

In Requirements

Most quality, I find, is lost—forever—in requirements, simply because no requirements are offered or because the requirements that are provided aren't business-focused. Here's my final thought on this subject:

> Requirements are the business' only opportunity to define "quality" in business terms. Any application that does exactly what the requirements state is, by definition, a high-quality application. If an application meets those requirements but is still perceived as being of low quality, the problem is in the requirements—not the application.

There's no simpler way to say it: You get what you ask for. If a business isn't specific about what it wants, it won't get it, but it has no reason to be disappointed with that fact later. I know that some projects get infinitely bogged down in "requirements definition" phases, and so there's this feeling that requirements are something to be avoided. But think about that for a moment.

Let's say you and your spouse agree to purchase a new set of living room furniture. Neither of you really has strong feelings about what you want, so you hire a designer to come over and make a recommendation. You give the designer a few vague suggestions like "wood is nice" and "we don't like blue," but not much else. The designer tries to get more detail from you, but you and your spouse start arguing about what you want, and so ultimately you decide it's easier to just let the designer use his or her best judgment. The designer purchases something that should look great in your house. You love it, but your spouse hates it and feels that you've wasted the money.

What happened? You and your spouse decided not to spend enough time defining your requirements because doing so was too difficult, politically. Given your differences in aesthetic, nothing the designer purchased would have suited both of you. The moral here is that if your organization can't manage to put together a clear and complete set of requirements in a reasonable amount of time, you're not likely to get a quality application no matter what you do. Recognize that fact and expect poor-quality applications or get management under control and learn to produce solid requirements in a reasonable period of time.

> **Note**
>
> There are many good books and classes on producing application and project requirements, and these may be of help in educating managers and other stakeholders to help streamlining the requirements phase of your project.

In Design

Most quality loss attributable to the design phase comes from incomplete requirements; the rest most commonly comes from what I like to call the "elegance syndrome." Designers don't always work directly with code or even with the resulting applications, and tend to favor designs that are elegant, follow textbook rules and best practices, and may not be the best for performance, security, or usability. Designers simply need to accept the difference between practical reality and the textbook, and design accordingly. If a database design with 20 tables is textbook-perfect but won't perform well under real-world conditions, it needs to be fixed.

Designers also need to be willing to look literally outside the box—outside the box that their chosen platform came in, specifically. For example, selecting Microsoft's .NET Framework or Sun's Java as a development technology is fine—but don't think that you're stuck using every component of those platforms. Database drivers, visual components, and more may need to be swapped out to meet specific project requirements, and designers must feel free to do so if necessary.

In Development

The development process offers plenty of opportunity for lost quality, which I've written about in previous chapters and I explore at length in *Definitive Guide to Building Code Quality* (http://nexus.realtimepublishers.com). Some highlights:

- Not following best practices for coding, including formatting, naming conventions, modularization, and so forth
- Building overly-complex code that is difficult to debug and maintain
- Insufficient and/or poorly-executed unit testing that doesn't provide complete coverage of the code or uses poor data
- Not understanding how to debug complex processes

And, of course, developers working without a design that maps back to a complete, clear set of business requirements—especially requirements related to performance and security.

In Testing

If you're testing to a complete, clear set of business requirements—using those requirements as a sort of checklist to determine whether the application is compliant—then you've probably got as much quality as you need in testing. If you're testing to some other specification, you'll never have any real quality to look forward to. Testing doesn't actually add or remove quality; it simply verifies whether you've achieved the quality that was asked for in the beginning.

That said, testing *can* do a poor job of verification. It can be inconsistent when performed manually, and it can use poor data that doesn't reflect real-world conditions. It can be incomplete, not covering all the code, and it can fail to mimic the real-world businesses processes the application will see in production. Testing might ignore security or performance, as well; however, if testing is being driven by requirements, the requirements *should* address all those considerations.

The Path to Better Quality

So how will you know when you've reached the maximum level of quality that is practical and feasible in your organization? Referring back to the "quality quiz" in an earlier chapter, let me suggest some milestones and actions you can look for and take.

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 is an extremely common scenario 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. Look for:

- Few or no tools designed to improve quality
- No documented best practices and procedures for coding or testing
- No dedicated QA or testing personnel (relying entirely on developers to conduct testing)
- No documented processes for conducting development (everything is ad-hoc)
- Few or no business requirements at the outset
- Minimal or missing technical designs
- Few or no reusable test assets
- No formal testing goals
- No metrics for measuring quality

So how can you improve?

- Establish a defect tracking system that identifies and classifies every defect—not just bugs but everything that the end users regard as a defect
- Begin to create sets of test data that reflect actual production data and scope
- Start to build a framework for testing automation; start documenting what needs to be tested, writing out test "scripts" that will be performed manually, and so forth
- Begin tracking testing statistics: How often is code being sent back due to failed testing?
- Establish a dedicated quality team—even if it's one person doing testing
- Document everything: best practices, processes, and procedures; *documented* means *repeatable* and *consistent.*

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 impractical. Some key indicators:

- Processes and procedures either not documented or documented (or followed) inconsistently
- Only a few reusable test assets (like test data), which are inconsistently used
- May still not be working from clear, complete requirements that reflect business needs and goals
- A few tools in use but still working largely on manually-performed processes
- Developers have few (if any) tools that give them insight into code quality

Here are some ideas for moving forward:

- 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

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. Some of the signs include:

- May have multiple QA teams working from different procedures and processes
- Inconsistent tools across projects
- Some projects have higher quality than others
- Few "dashboards" to reveal simple quality "grades" for projects

How do you take that last, final step? It's mostly a question of fine-tuning and committing to your processes. Some ideas:

- Establish best practices—Document how the quality team works, including their tools and processes; draw flowcharts; and create the "Quality Manual" for your organization
- Align to the business—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"
- Involve the business—The business leaders who create a software project's initial business requirements should also be involved in developing key quality metrics
- 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

Once you have these things in place, quality will be a science for you.

Quality as a Science

When quality becomes a science, business requirements drive everything. 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.

Quality Tools: A Shopping List

I've written it many times throughout this guide: Tools alone can't improve your quality level. What tools can do is help automate tasks that would otherwise be repetitive or impractical when performed manually. In other words, if you *know* how to achieve better quality, tools *can* make it faster and easier to do so. Also, tools can help make quality more consistent by allowing you to follow the same practices and patterns all the time.

I'm not going to recommend any specific tools or vendors. Instead, I'm going to provide a shopping list of features, and you can use that as you start to explore and evaluate tools on your own. I do want to point out that my "baseline" for features is the feature set included in Microsoft Visual Studio, Standard and Professional editions. In my experience, they're the most widely-used editions of Visual Studio, and they're the editions that most third-party tool makers target for integration and support. Microsoft does offer a higher-end "Team System" edition of Visual Studio, but that competes in many regards with the third-party tool manufacturers because it bundles tools like code analysis, performance reporting, and so forth. In many cases, those extra "Team System" tools are valuable but less mature than what you'll find from third-party vendors, and they still come with a significant additional price with relation to the Professional and Standard editions of Visual Studio.

Tools for Requirements

Because everything in a high-quality project tracks back to the project's original requirements, *tracking* those requirements becomes really crucial to success. You need to be able to manage requirements as standalone entities, including managing changes and tweaks over time. You also need to be able to manage requirements as drivers for things like development and testing so that test cases (for example) can be mapped to specific requirements. Ideally, you want to be able to manage your current level of compliance with each requirement so that reporting and "dashboards" can help you see exactly where you're at in achieving the requirements and thus meeting the specified level of quality.

There are a few key tasks that a requirements management solution should offer:

- **Definition.** Tools should not only capture requirements but also help prompt for the capture of clear and complete requirements. Tools should be accessible and intuitive for non-technical business users, utilizing flow diagrams and other visualizations (see Figure 10.1). Tools should help maintain a project glossary so that specialized words can be unambiguously defined and common words, such as "should" and "must," can also be clearly and unambiguously defined.

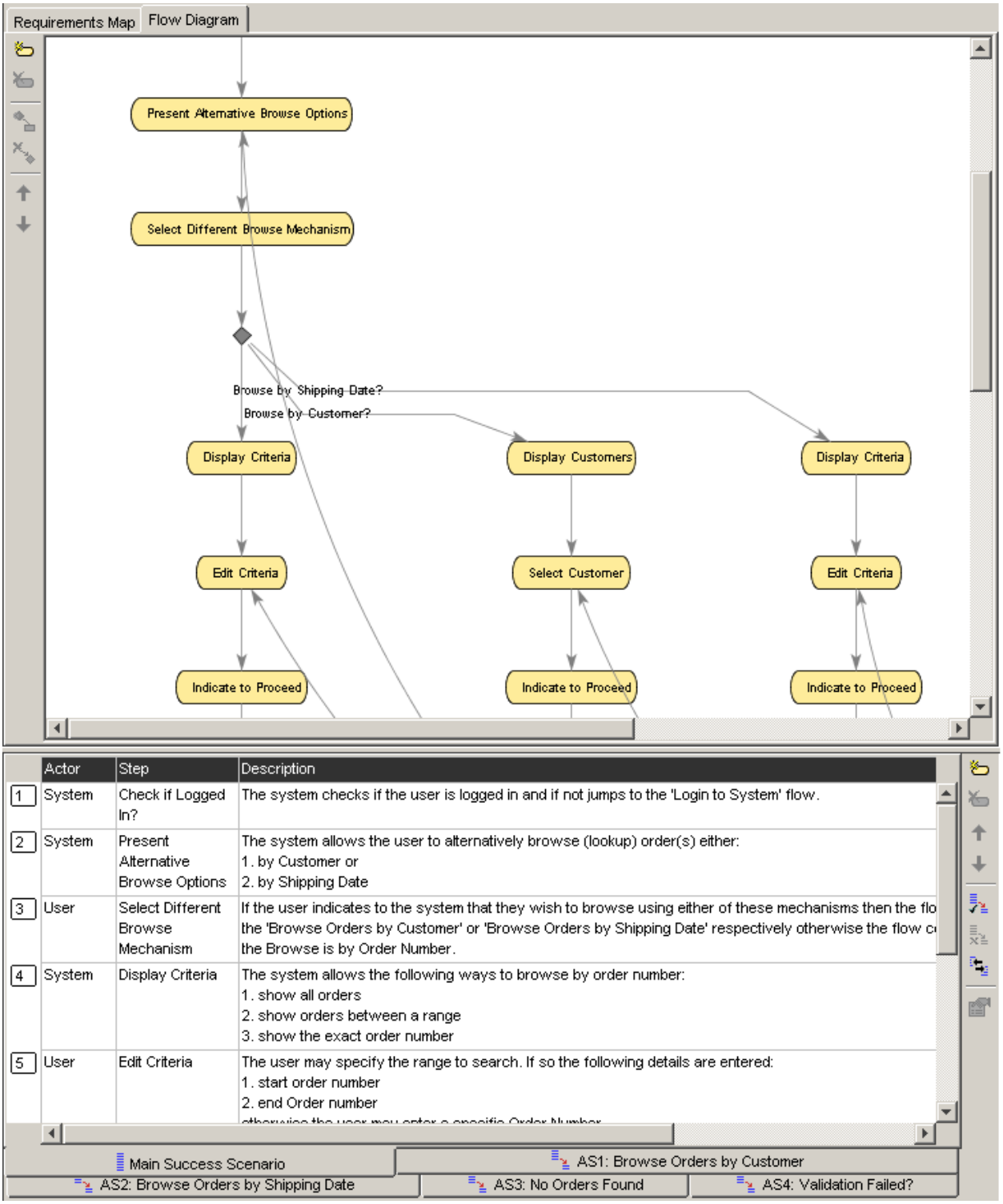


Figure 10.1: Using workflows to display project requirements.

- **Documentation.** Tools should be able to take defined requirements and predefined or custom templates and produce a set of project documentation that facilitates communication across the business. Common formats (such as Microsoft Office, as Figure 10.2 shows) as well as project management formats (such as Microsoft Office Project or CSV) should be available.

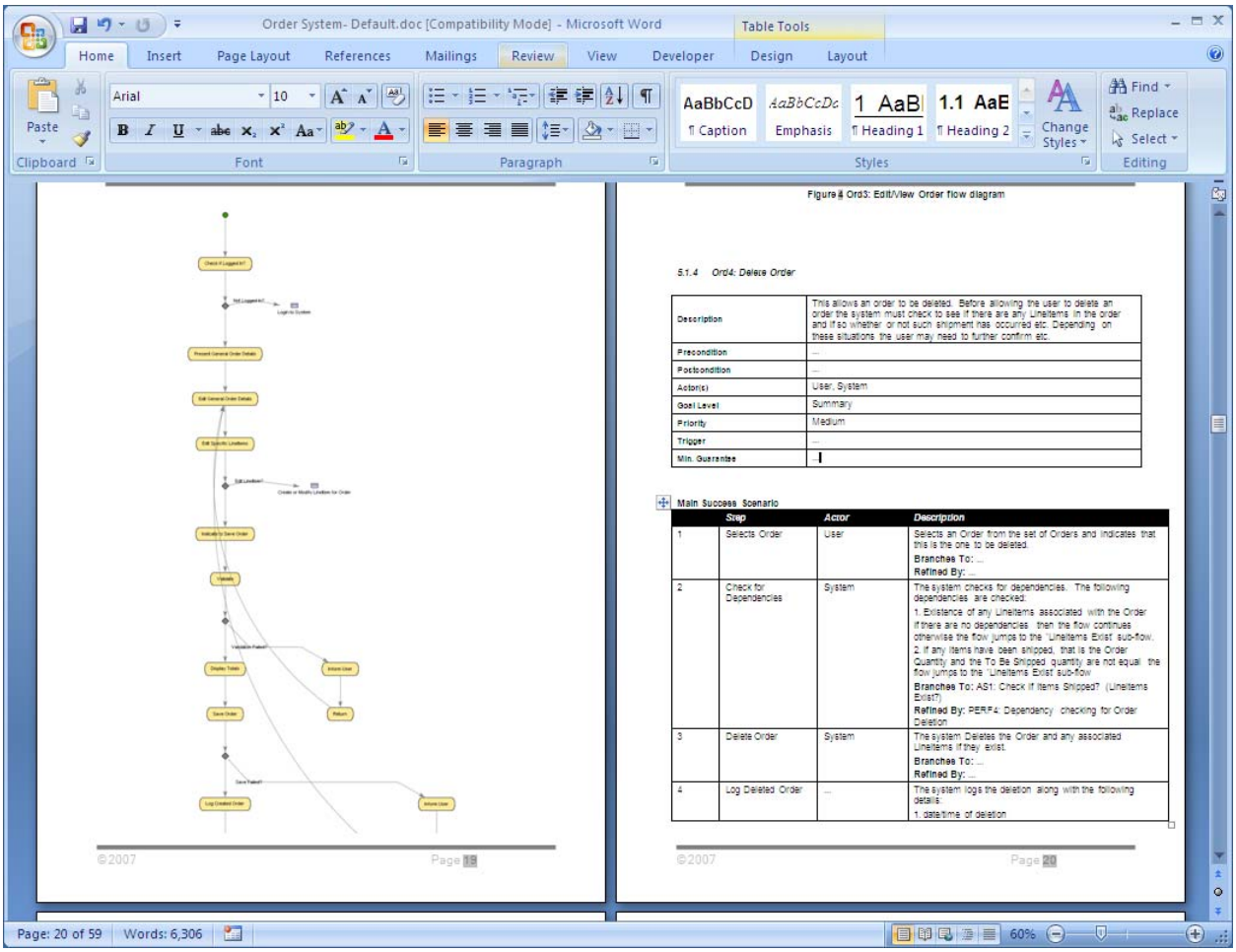
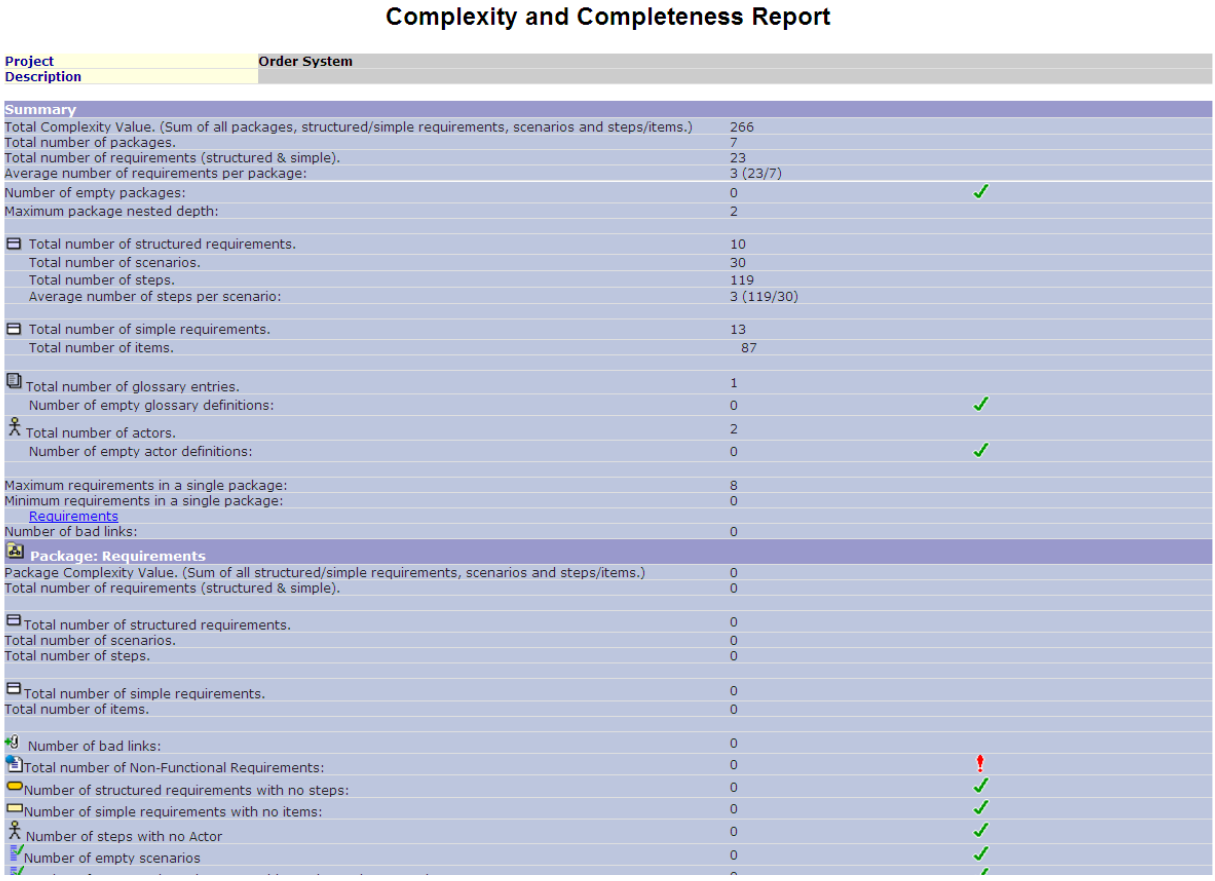


Figure 10.2: Exporting requirements to a Microsoft Word document.

- **Collaboration.** Perhaps most importantly, tools should enable collaboration across the business, acting as a central repository for business requirements and any supporting documents or media so that everyone can see the current, live state of the project's requirements at all times. Capabilities for upper-level review and sign-off should also be available, as management may choose to deliberately forgo certain requirements given by users or other contributors.

- **Validation.** This is where a requirements-tracking tool can contribute directly to the project's actual production by generating written test cases automatically. This helps drive acceptance testing as well as other aspects of the project and ensures that everyone downstream is working from the same set of requirements. Integration (often via Universal Modeling Language—UML) with other test and design tools can help drive database design and other aspects of the project's design, as well.
- **Management.** Everyone should be able to manage and track the project in real time. Changes should be captured and retained and coupled with email alerts and other notifications to signal to stakeholders when elements change. Changes to dependent elements should trigger alerts for upper-level dependent elements (called *traceability* by many tools). Comprehensive reporting (such as the complexity report in Figure 10.3) must allow views of the projects for various levels of participants, such as managers, designers, developers, and so forth.



Complexity and Completeness Report

Project	**Order System**
Description	

Summary		
Total Complexity Value. (Sum of all packages, structured/simple requirements, scenarios and steps/items.)	266	
Total number of packages.	7	
Total number of requirements (structured & simple).	23	
Average number of requirements per package:	3 (23/7)	
Number of empty packages:	0	✓
Maximum package nested depth:	2	
Total number of structured requirements.	10	
Total number of scenarios.	30	
Total number of steps.	119	
Average number of steps per scenario:	3 (119/30)	
Total number of simple requirements.	13	
Total number of items.	87	
Total number of glossary entries.	1	
Number of empty glossary definitions:	0	✓
Total number of actors.	2	
Number of empty actor definitions:	0	✓
Maximum requirements in a single package:	8	
Minimum requirements in a single package:	0	
Requirements		
Number of bad links:	0	
Package: Requirements		
Package Complexity Value. (Sum of all structured/simple requirements, scenarios and steps/items.)	0	
Total number of requirements (structured & simple).	0	
Total number of structured requirements.	0	
Total number of scenarios.	0	
Total number of steps.	0	
Total number of simple requirements.	0	
Total number of items.	0	
Number of bad links:	0	
Total number of Non-Functional Requirements:	0	!
Number of structured requirements with no steps:	0	✓
Number of simple requirements with no items:	0	✓
Number of steps with no Actor	0	✓
Number of empty scenarios	0	✓

Figure 10.3: Example report from a requirements-tracking system.

In many cases, these tools will integrate with other quality-focused tools, both from the same vendor as well as other leading vendors. You may find integration with Microsoft Visual Studio Team System, for example, which lacks strong requirements-tracking capabilities of its own. Having integration across your entire tool set, all the way through to final testing, is critical to ensure that your requirements can flow clearly throughout the entire project and that results can flow back and be mapped to those requirements.

Tools for Design

Specific design tools are fewer in number. Generally speaking, the design tools that designers are accustomed to using—modeling tools, documentation systems, and so forth—are sufficient, especially if you're using a requirements-tracking solution that can export modeling information (in UML or another interchange format) to the tools your designers already use. The key is to make sure that each aspect of the design maps back to a specific requirement. It's less important that the code and tests that come from the design map back to portions of the design than it is for those elements to map back to portions of the *requirements*. Remember: It's the requirements that should be driving everything, and the design is merely a technical explanation of how to implement those requirements.

Tools for Development

Development tools are designed to perform several tasks during the development process to help developers produce higher-quality code. Some tools will be found integrated into an Integrated Development Environment (IDE) like Visual Studio or Eclipse. Features such as code snippets, syntax highlighting, automatic formatting, and code completion and other elements can help encourage better code quality by supporting good coding practices. Other tools help with aspects of quality that are often impractical to achieve manually.

Static Code Analysis

Also called *source code analysis,* this is where a tool reviews the source code without running it and compares it with an often-extensive set of rules that help preserve best practices, avoid common problem situations, and so forth. For example, Figure 10.4 shows an analysis that has detected a situation that often causes errors at runtime. The tool explains the problem and has prescriptive guidance to help fix it.

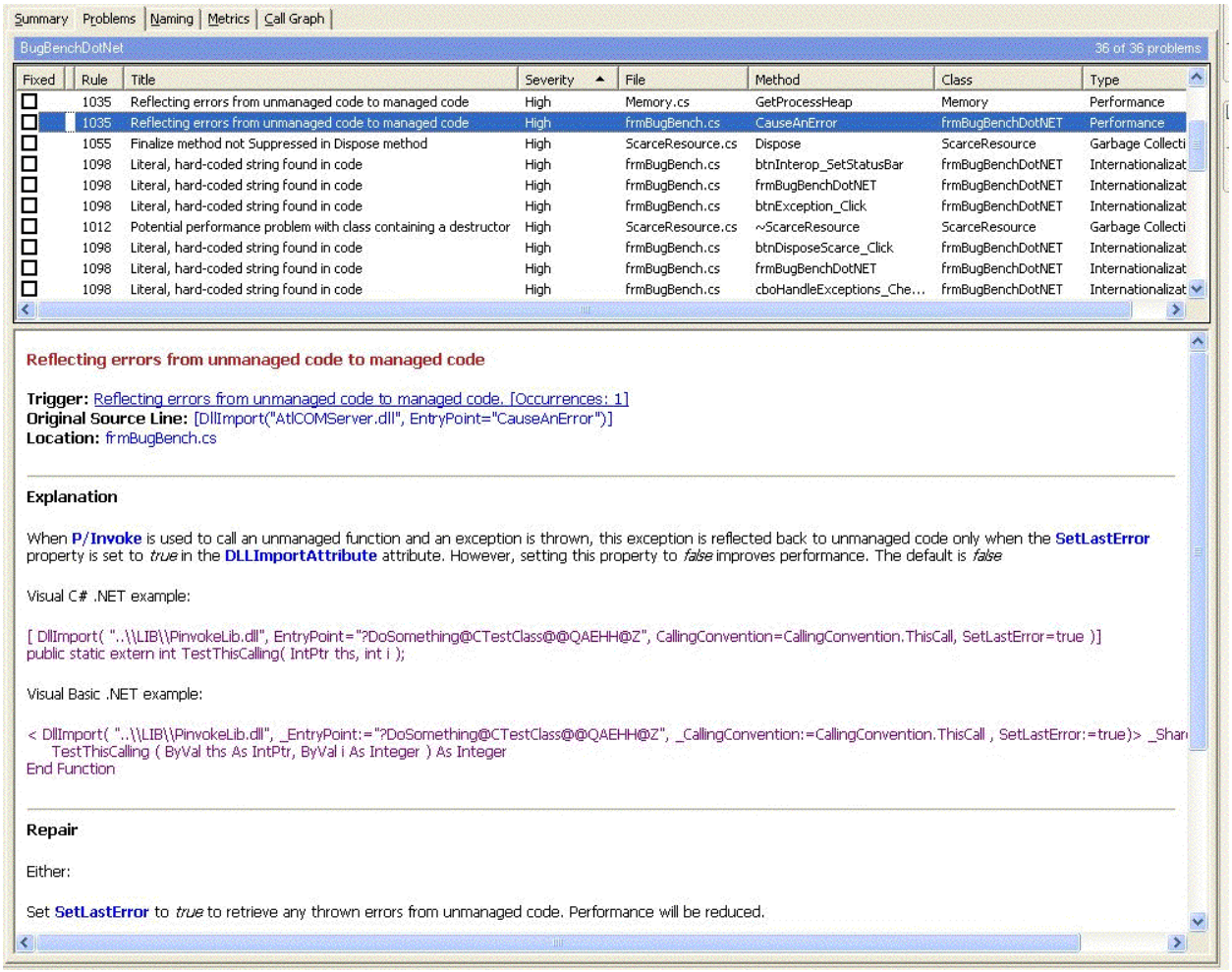
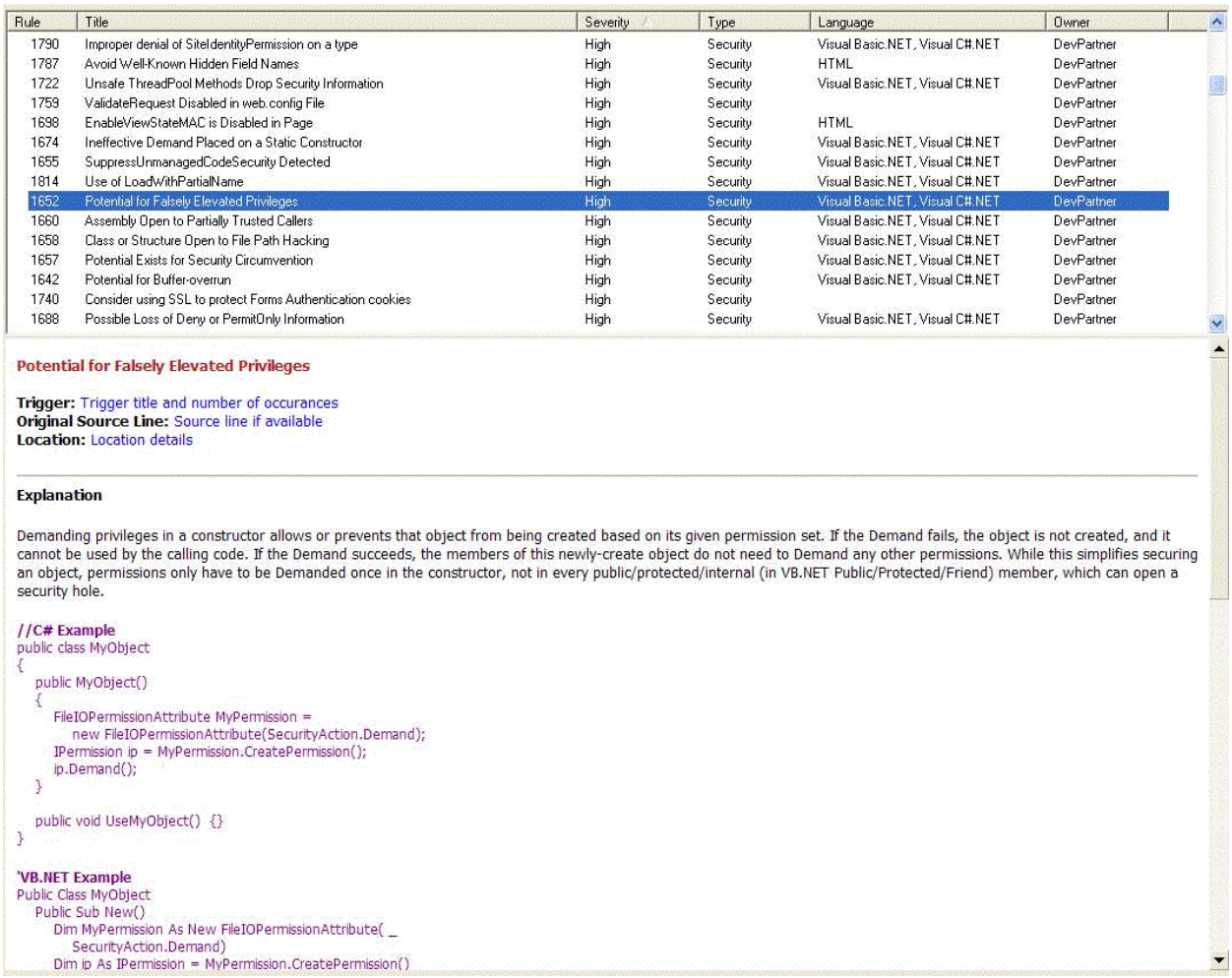


Figure 10.4: Code analysis can reveal problems and should recommend solutions.

Security Analysis

Similar in many ways to static code scans, security scans focus on situations involving potential security issues. Using rule sets and typically working while the application is being compiled, rather than just running a scan of the static code, security analysis tools can often help spot common security problems before they become too deeply embedded in the application to fix easily. Figure 10.5 shows an example where a tool has detected the potential for falsely elevated privileges, displays an example of code to illustrate the problem, and recommends a fix.



Rule	Title	Severity	Type	Language	Owner
1790	Improper denial of SiteIdentityPermission on a type	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1787	Avoid Well-Known Hidden Field Names	High	Security	HTML	DevPartner
1722	Unsafe ThreadPool Methods Drop Security Information	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1759	ValidateRequest Disabled in web.config File	High	Security		DevPartner
1698	EnableViewStateMAC is Disabled in Page	High	Security	HTML	DevPartner
1674	Ineffective Demand Placed on a Static Constructor	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1655	SuppressUnmanagedCodeSecurity Detected	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1814	Use of LoadWithPartialName	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1652	Potential for Falsely Elevated Privileges	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1660	Assembly Open to Partially Trusted Callers	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1658	Class or Structure Open to File Path Hacking	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1657	Potential Exists for Security Circumvention	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1642	Potential for Buffer-overrun	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner
1740	Consider using SSL to protect Forms Authentication cookies	High	Security		DevPartner
1688	Possible Loss of Deny or PermitOnly Information	High	Security	Visual Basic.NET, Visual C#.NET	DevPartner

Figure 10.5: Security analysis can help spot potential security problems.

Properly using this type of tool is critical. In addition to each developer using it frequently to spot security problems, developers as a group should review the security problems they've been finding using the tool, and help each other watch out for poor security practices. Many developers are not well-educated in secure coding concepts and practices, so using a tool like this can be a learning experience; sharing that experience across the development team can help improve security for the entire application.

Performance Analysis

Good tools should provide a variety of performance capabilities. For example, Figure 10.6 shows how a tool can provide not only a method-by-method breakdown of where time is being spent when running your application but also a "code path" that visually illustrates where performance is being consumed. The latter is especially useful for catching smaller modules of code that, on a single execution, don't consume much time but are called repetitively and add up to a significant performance hit.

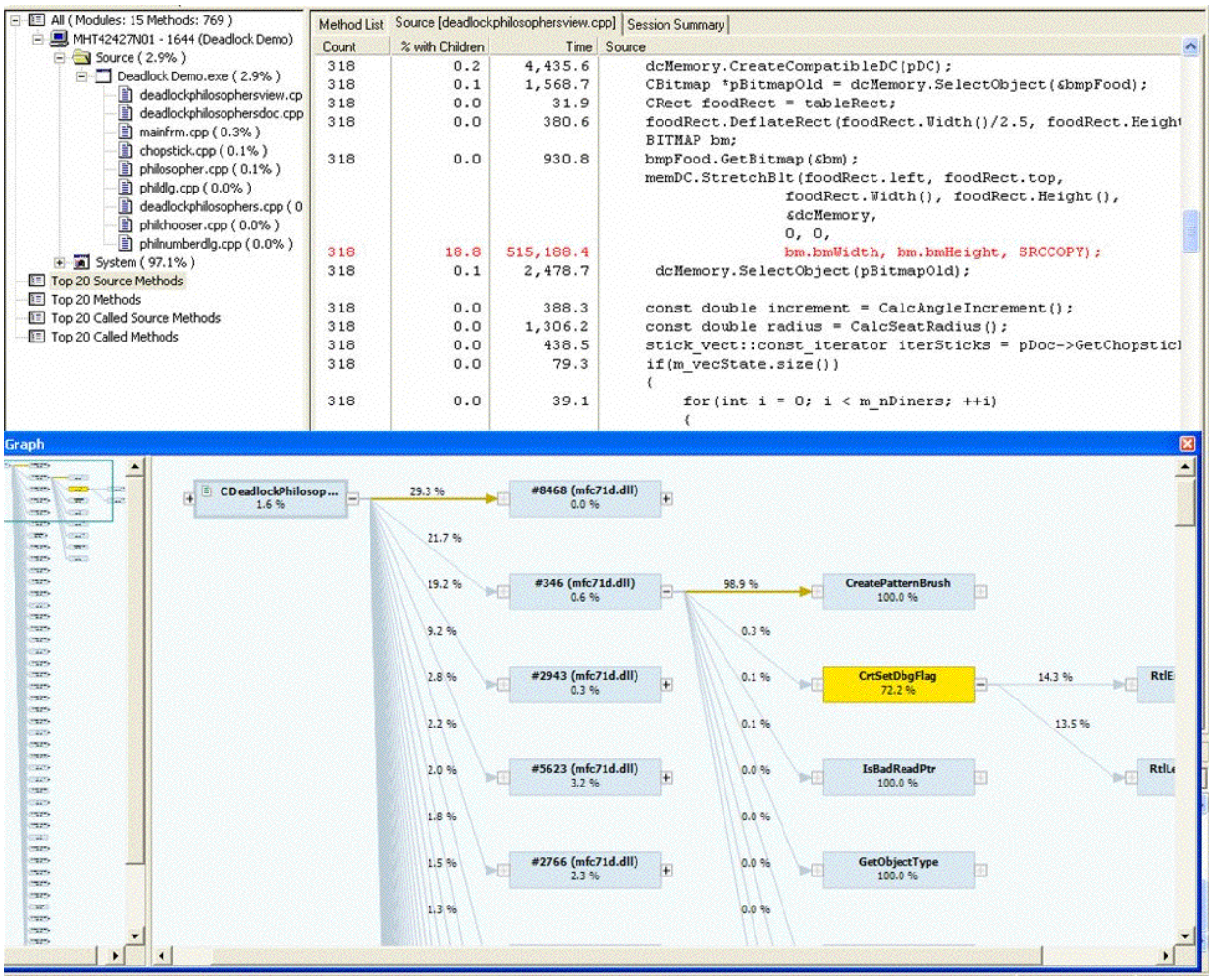


Figure 10.6: Example performance analysis.

Tools may also be able to help with multi-tier testing, helping developers to spot performance problems on a middle-tier or database tier that can't be easily detected in a single-machine development environment. Finally, tools can also help with difficult performance problems resulting from system resource consumption, such as memory leaks, overall memory footprint, and so forth. Figure 10.7 shows how a tool can display memory consumption, helping developers visualize memory leaks and other resource-related problems.

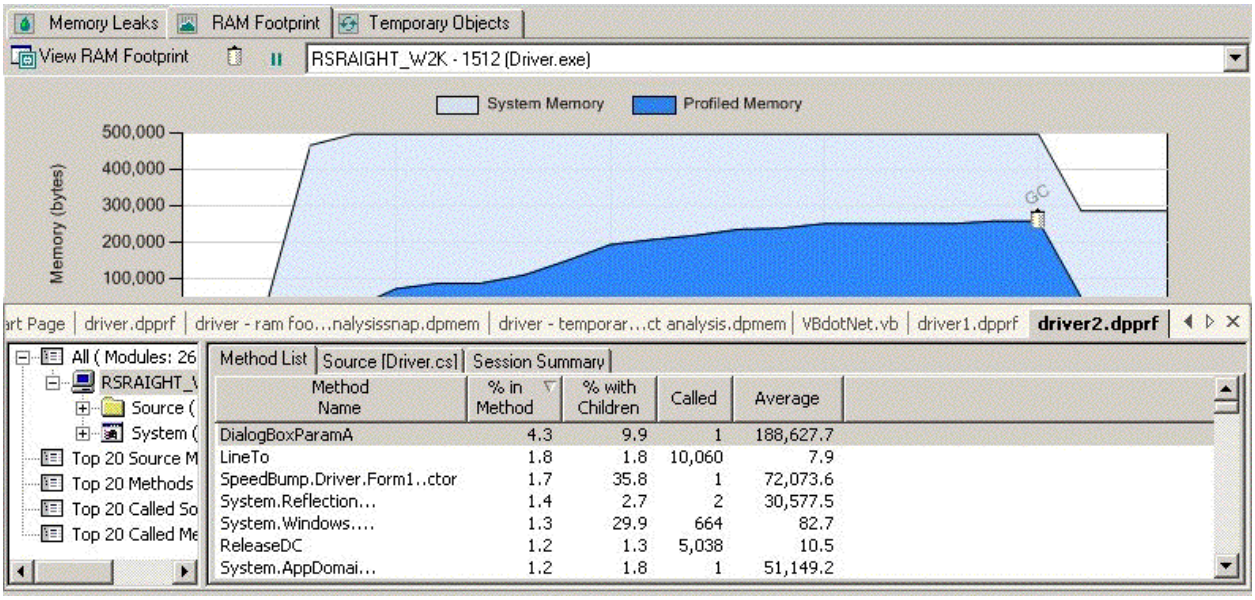


Figure 10.7: Visualizing resource consumption during development.

Unit Testing

Good development tools can also help encourage unit testing. This helps ensure that *every* possible code path—not just the ones the developer thinks of ad-hoc—is thoroughly tested. Figure 10.8 shows an example of this kind of "code coverage" report, which makes it clear which portions of the application's code have never been tested so that those portions can be unit-tested and any bugs caught and found before the code moves on to more resource-intensive integration testing.

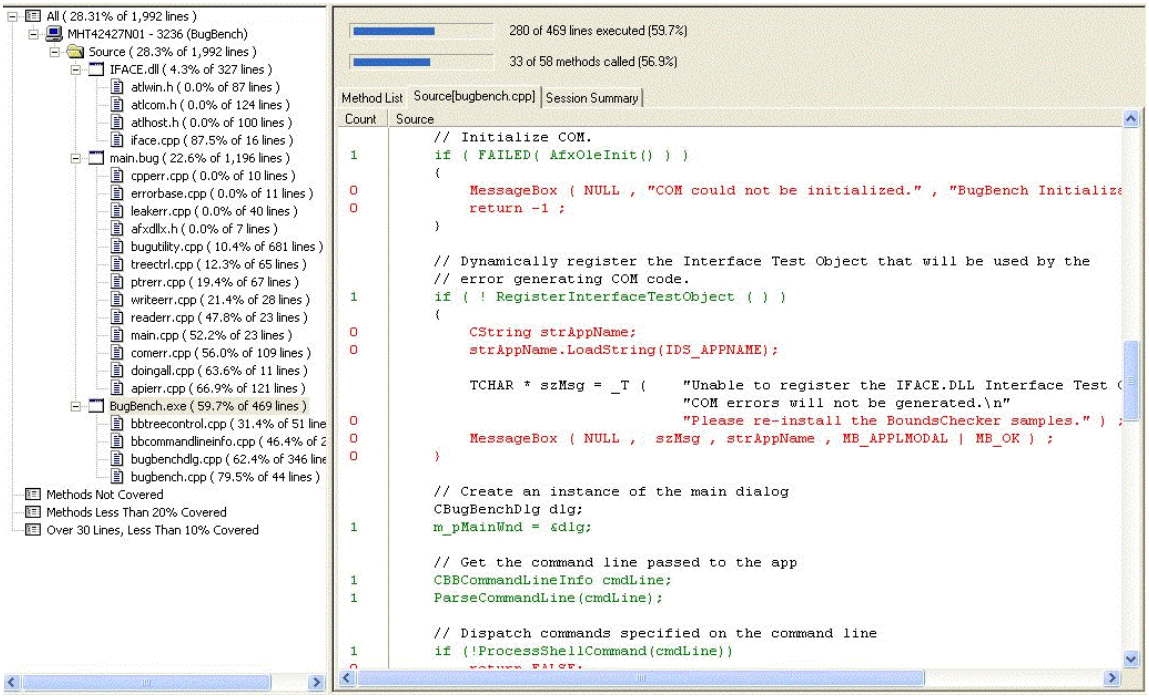


Figure 10.8: Code coverage reports help ensure more thorough testing.

Easier Debugging

Some of the most difficult-to-solve bugs involve differences between the machine where the bug was detected and the developer's machine. On Windows systems in particular, these changes can be subtle and difficult to detect; good tools can help by providing machine-comparison capabilities that help developers easily spot differences that might be preventing them from duplicating—and solving—a problem locally. Figure 10.9 shows what a difference report might look like.

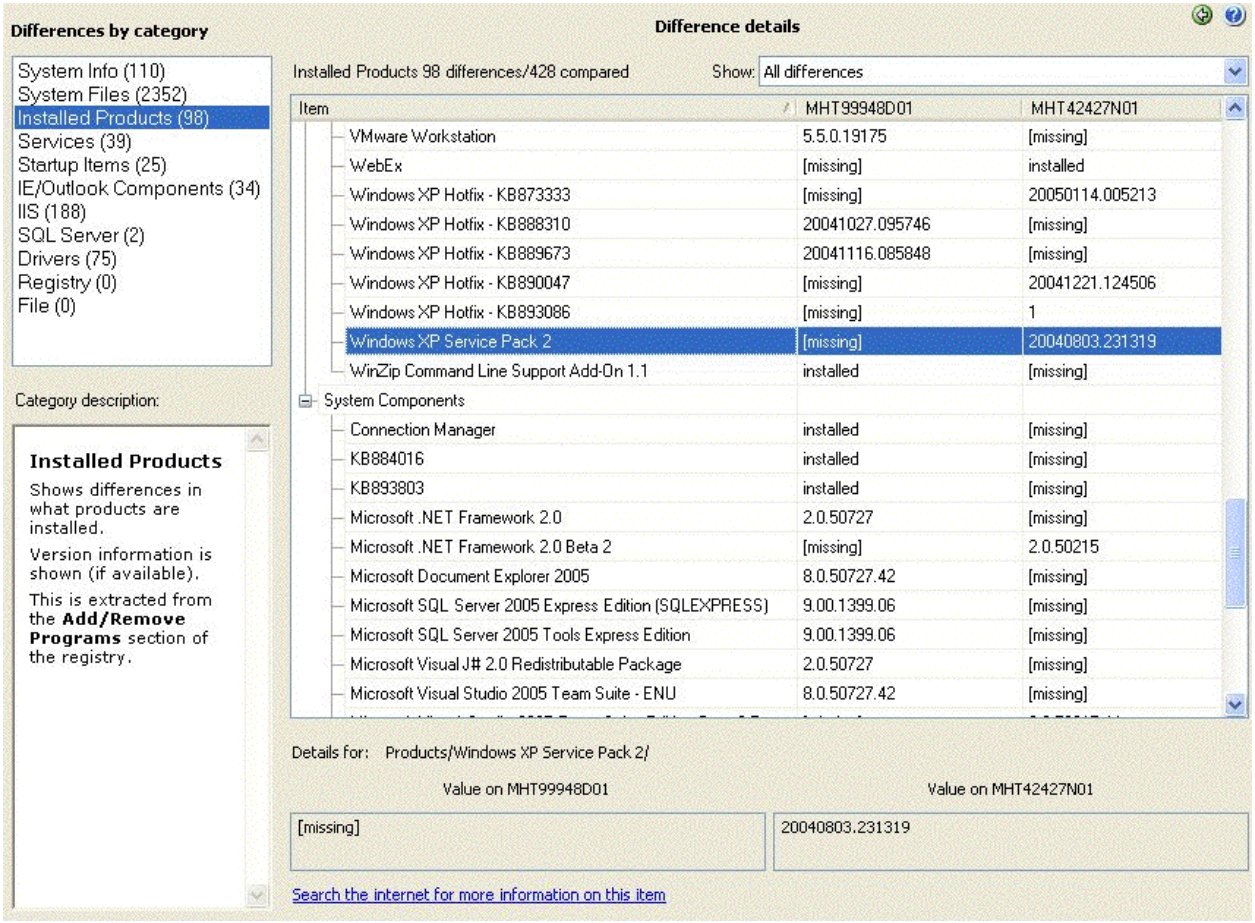
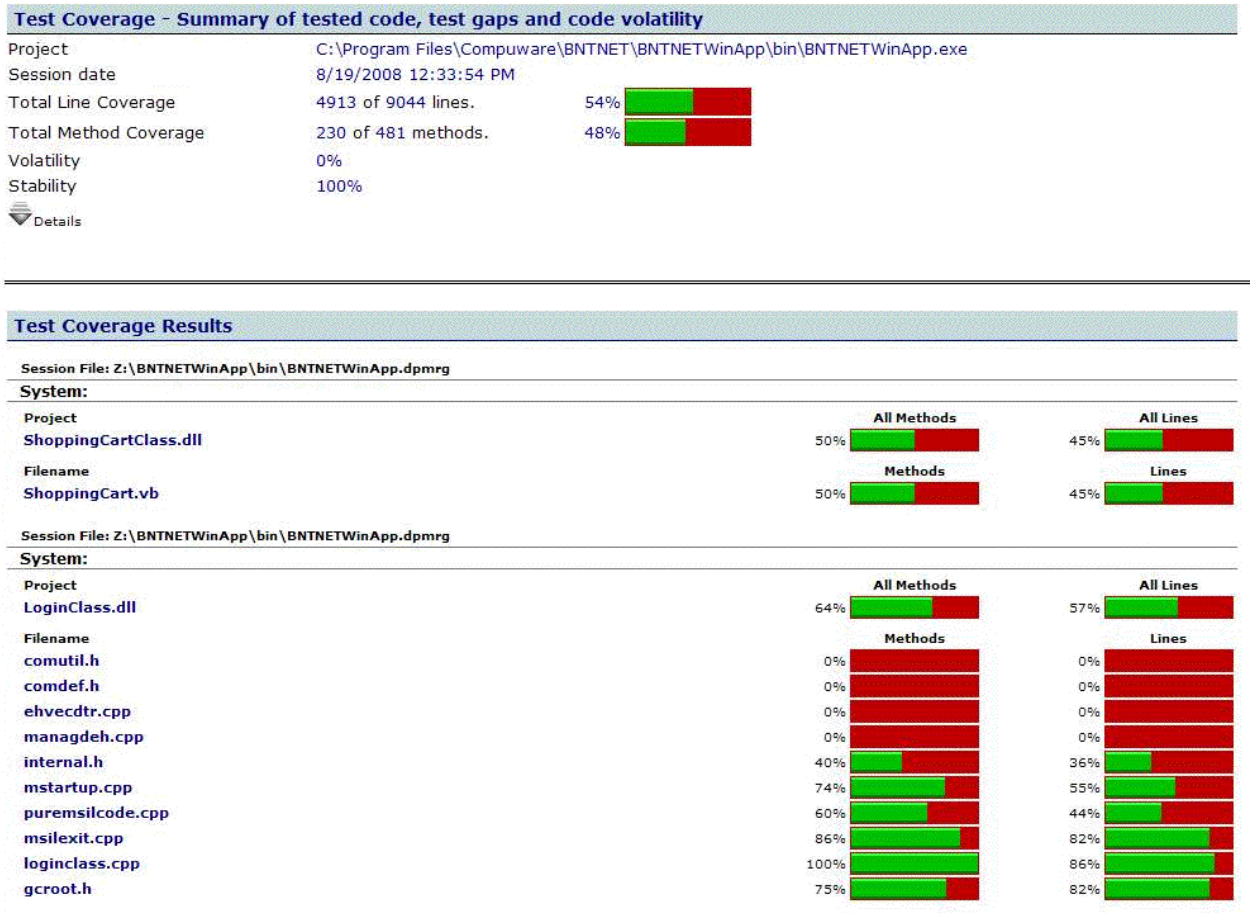


Figure 10.9: Machine difference report.

Any difference can be responsible for a big problem occurring in one place but not another; by detailing these changes in this fashion, a tool can help developers focus their efforts and solve the problem more quickly.

Tools for Management

Management needs good tools, too, and these often come bundled as part of development tools and testing tools. For example, a development tool suite can include management-focused reports that help chart developer progress, as in the code coverage report that Figure 10.10 shows.



Test Coverage - Summary of tested code, test gaps and code volatility

Project	C:\Program Files\Compuware\BNTNET\BNTNETWinApp\bin\BNTNETWinApp.exe	
Session date	8/19/2008 12:33:54 PM	
Total Line Coverage	4913 of 9044 lines.	54%
Total Method Coverage	230 of 481 methods.	48%
Volatility	0%	
Stability	100%	

Details

Test Coverage Results

Session File: Z:\BNTNETWinApp\bin\BNTNETWinApp.dpmrg

System:

Project	All Methods	All Lines
ShoppingCartClass.dll	50%	45%
Filename	**Methods**	**Lines**
ShoppingCart.vb	50%	45%

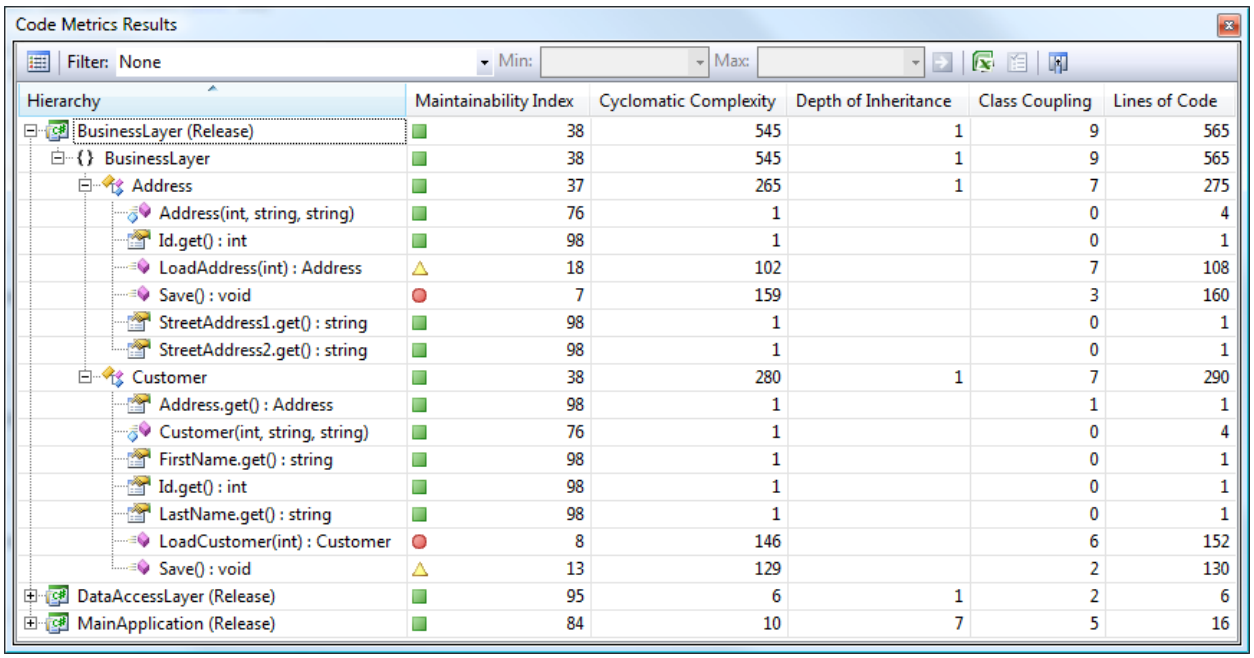
Session File: Z:\BNTNETWinApp\bin\BNTNETWinApp.dpmrg

System:

Project	All Methods	All Lines
LoginClass.dll	64%	57%
Filename	**Methods**	**Lines**
comutil.h	0%	0%
comdef.h	0%	0%
ehvecdtr.cpp	0%	0%
managdeh.cpp	0%	0%
internal.h	40%	36%
mstartup.cpp	74%	55%
puremsilcode.cpp	60%	44%
msilexit.cpp	86%	82%
loginclass.cpp	100%	86%
gcroot.h	75%	82%

Figure 10.10: Management reporting from a quality development tool.

Other tools can include can include code complexity measurements and other metrics, as the example in Figure 10.11 shows. These metrics can help managers know where to focus additional testing efforts, more peer reviews, and other measures to help mitigate the risk of more complex portions of the application.



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
BusinessLayer (Release)	38	545	1	9	565
BusinessLayer	38	545	1	9	565
Address	37	265	1	7	275
Address(int, string, string)	76	1		0	4
Id.get() : int	98	1		0	1
LoadAddress(int) : Address	18	102		7	108
Save() : void	7	159		3	160
StreetAddress1.get() : string	98	1		0	1
StreetAddress2.get() : string	98	1		0	1
Customer	38	280	1	7	290
Address.get() : Address	98	1		1	1
Customer(int, string, string)	76	1		0	4
FirstName.get() : string	98	1		0	1
Id.get() : int	98	1		0	1
LastName.get() : string	98	1		0	1
LoadCustomer(int) : Customer	8	146		6	152
Save() : void	13	129		2	130
DataAccessLayer (Release)	95	6	1	2	6
MainApplication (Release)	84	10	7	5	16

Figure 10.11: Code metrics reports can be used to help manage and mitigate risk.

Tools for Testing

Testing—more specifically, integration testing that involves the entire application rather than individual units—is the last chance to ensure that an application meets its originally-planned requirements. Again, tools can help both by automating tedious tasks and performing more consistently as well as by easing the overall management burden that QA testing can involve.

Test Case Management

What do you test? Obviously, you need to test to make sure that an application meets its requirements. Tight integration between a test management tool and your requirements management tool can ensure that test plans are created to act as a kind of checklist against your original business requirements, saving time and helping testing keep in sync with changing requirements. The results of these tests are meaningful to the entire business, and act as a gauge of the application's completeness from a business perspective. As Figure 10.12 shows, tools can help the business determine how many requirements have been met in the code to date, what tests (and therefore, which requirements) have not yet been completed satisfactorily, and so forth.

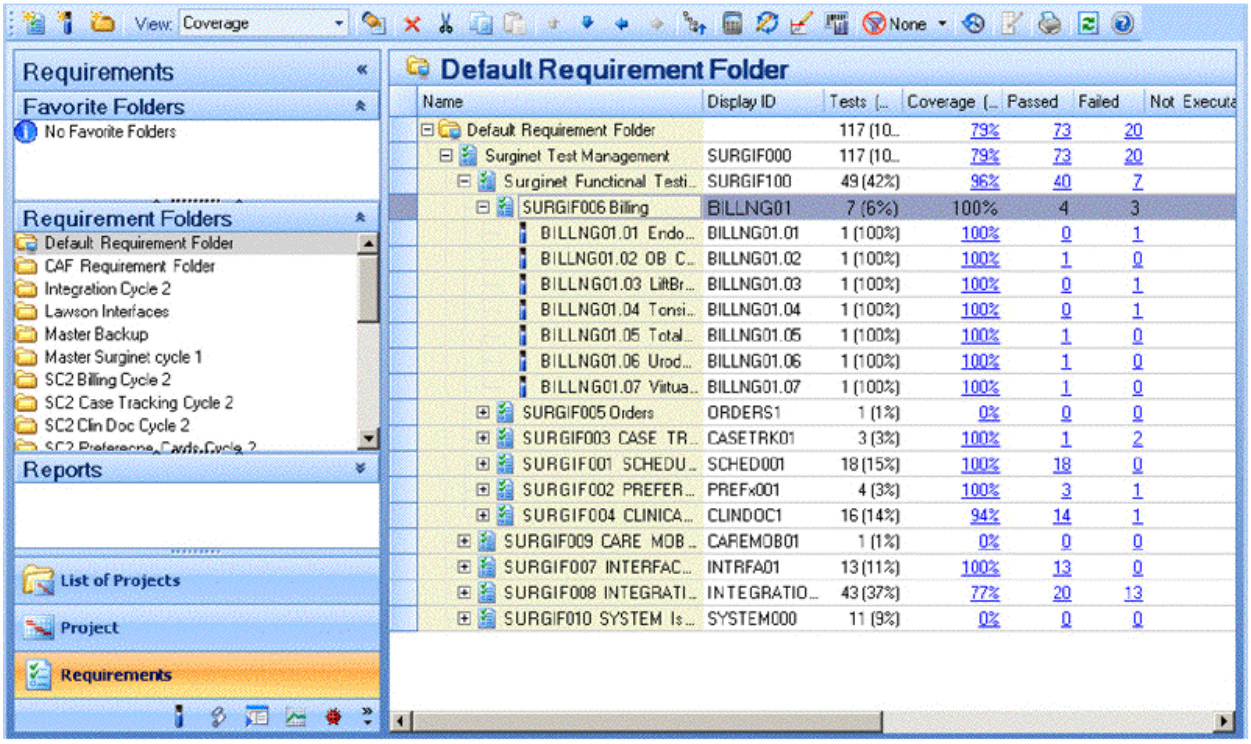


Figure 10.12: Requirements-based testing helps the business understand how complete the application is.

"Dashboard" views can be especially useful to executives concerned about the progress of an application. As Figure 10.13 shows, the entire body of testing information, combined with application requirements, can be rolled up into a single view that shows managers exactly how much quality—with regard to business requirements—the application currently achieves.

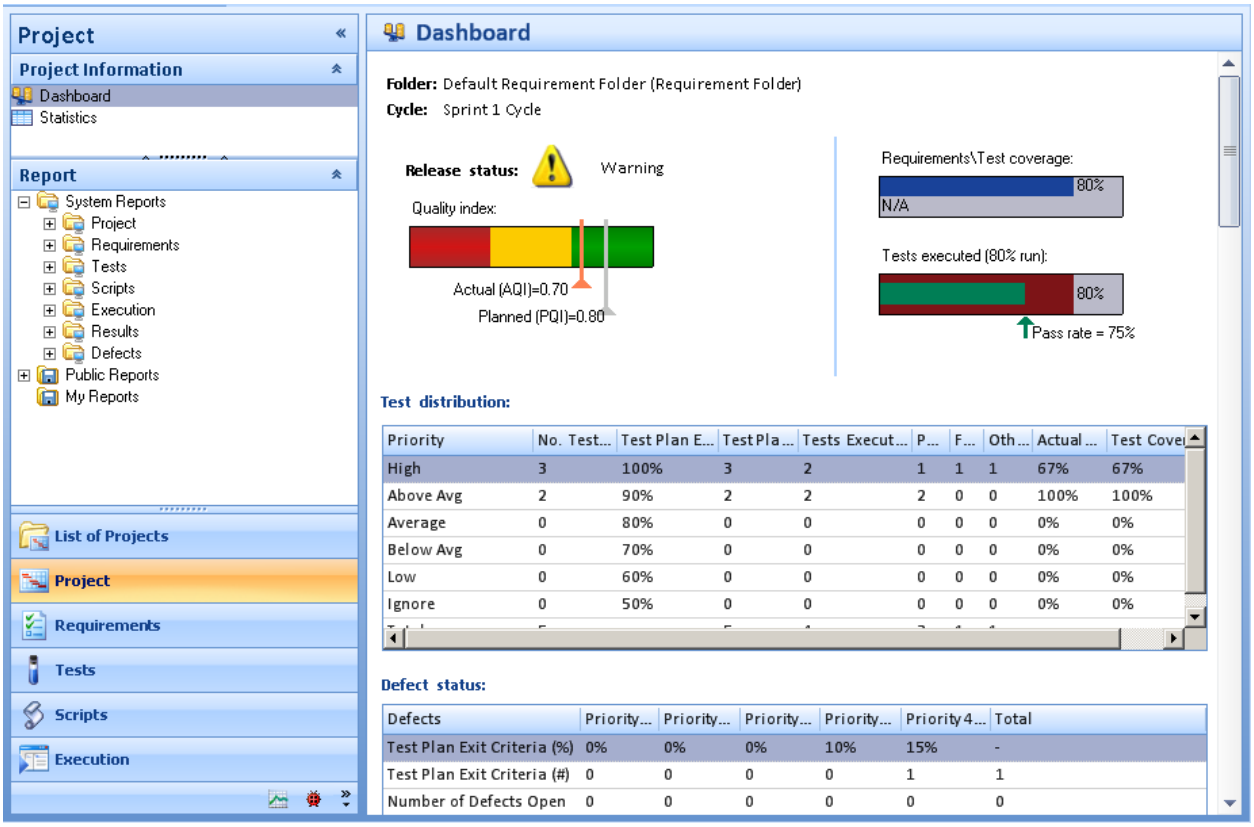


Figure 10.13: Management dashboards roll up complex test results and requirements into a simple overview.

But none of this means that testing is purely about requirements. As a form of risk mitigation, complex portions of code are often scheduled for more frequent and comprehensive testing, and a test management tool should accommodate this type of risk-based testing activity. As Figure 10.14 shows, the right tools can help managers plan the appropriate amount of testing given the limited time available by helping managers visualize the risk and quality associated with various tests, portions of code, and so forth. By getting the optimal coverage of testing over riskier portions of code, the application's quality can be maximized within the limitations of the project's resources.

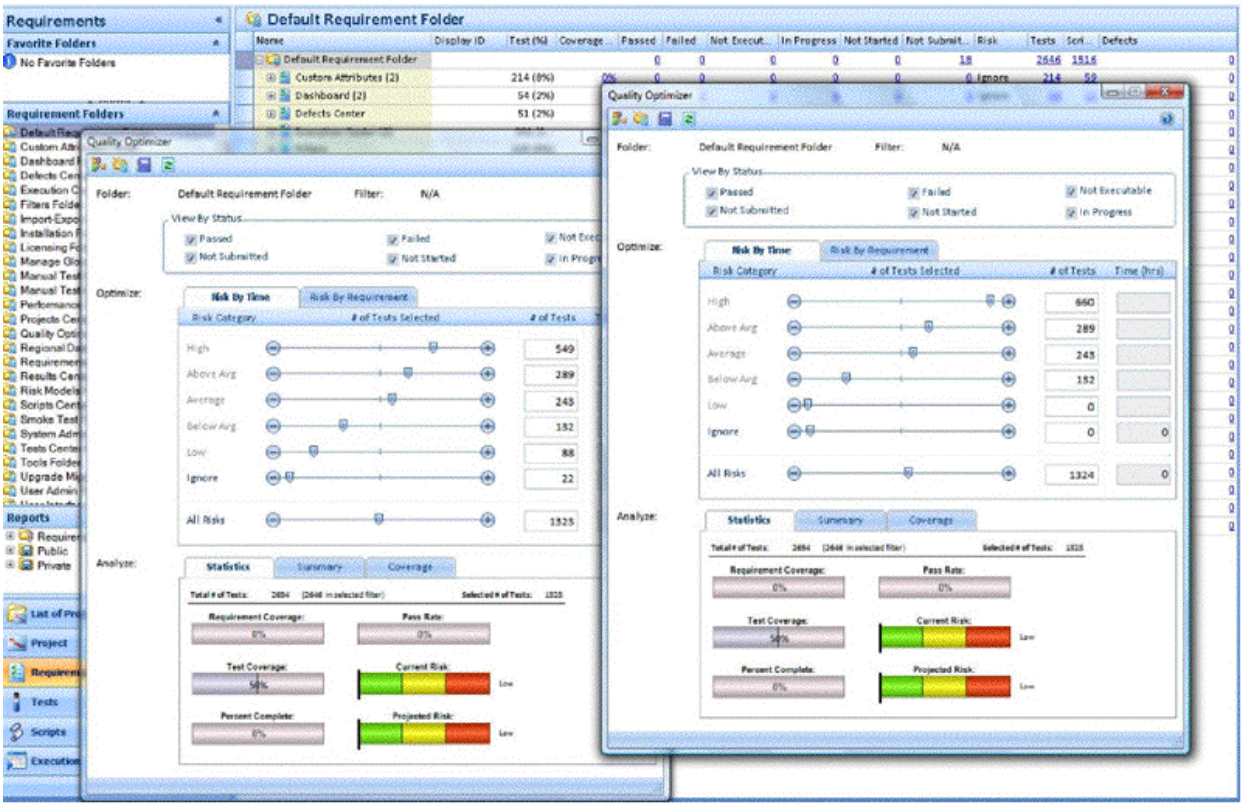


Figure 10.14: Optimizing application quality by selecting appropriate test coverage for the time available.

Test Data Management

Managing test data is difficult, but *not* managing it leads to poor-quality data, which results in poor-quality tests, which result in poor-quality applications. Test data can be large and unwieldy, may involve security concerns, and are the perfect candidate for management tools. These tools often provide graphical interfaces so that developers and testers can utilize quality test data without resorting to manual scripts. And with *everyone* on the project working from the same set of test data, more bugs are caught earlier in the development and testing cycle, which saves time and money.

Test Automation

Most IT professionals who think about "testing tools" are thinking about automation. Although automation is certainly a valuable aspect of testing, it must flow from good, requirements-based test management and must utilize good, quality test data.

Test automation often involves building complex test scripts that execute an application, feed it data, and look for specific results—all automatically. These scripts may involve a complex and/or proprietary scripting language or, as Figure 10.15 shows, may utilize a graphical user interface (GUI) to develop "scripts" without additional programming (typically, such tools also allow the resulting scripts to be manually tweaked if necessary).

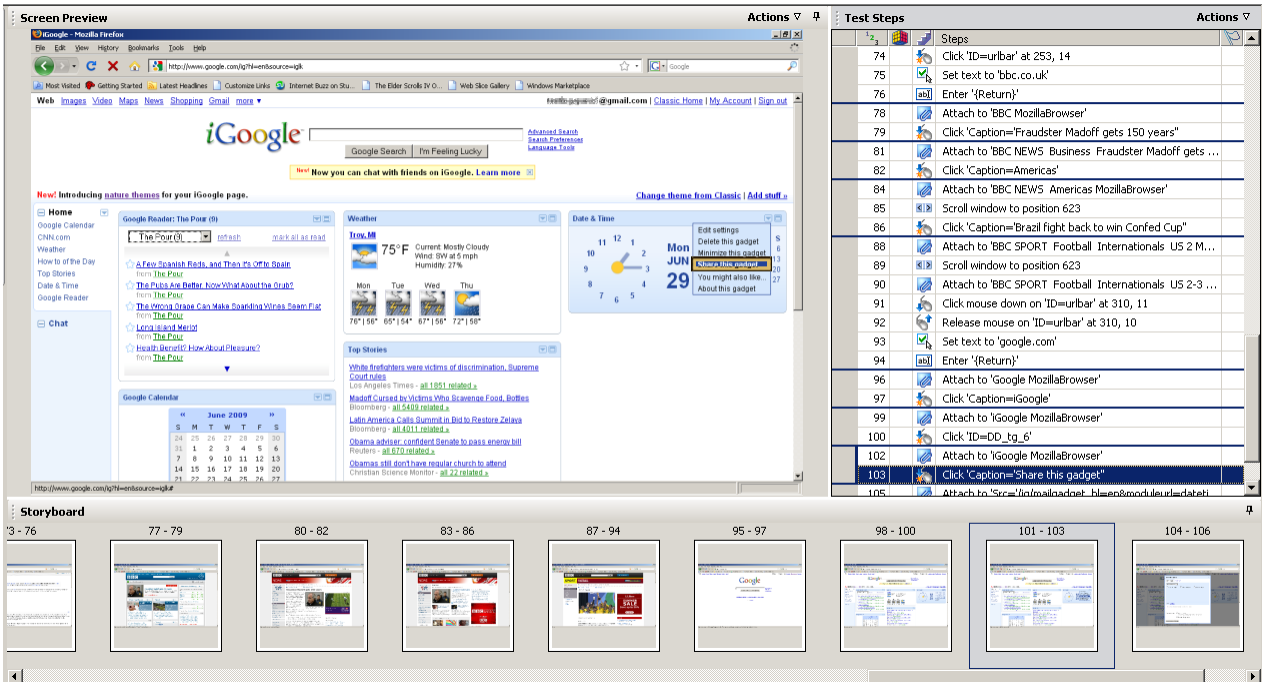


Figure 10.15: Building test scripts using a visual environment.

Such tools often allow test developers to create a "storyboard" that illustrates the flow of the application, allows test developers to specify the data that will be fed into user input forms and so forth, and allows them tospecify which results to look for. Some visual tools require that the application itself be available in order to capture its screens and workflow; others allow test developers to work from screen captures (bitmaps) provided by developers. This functionality is especially useful, as it allows test development to proceed in parallel with coding, helping to shorten project timelines and better utilize team resources. Tools should also provide an easy way to update application screens without starting test development from scratch, as the application will almost certainly change over the course of its development.

When testing tools do allow or require scripting or programming, they should ideally rely on independent, well-known languages rather than proprietary ones. Common ones include variations of Microsoft's Visual Basic language, Perl, ECMAScript, and so forth.

Collaboration is also key. Imagine a scenario in which:

- End users can use an existing application to capture a business process, submit it to the testing system, and flag it for a test developer to include in the application's future testing.
- Test developers can reproduce a defect in the application, flag the exact point in the process where it occurred and the data that was used to create it, and assign that to a developer for resolution.

This type of collaboration is provided by larger, enterprise-class test automation and management tools and can significantly reduce the effort needed to obtain higher levels of quality.

Performance Testing

Another key capability offered by many toolsets is automated performance testing and load testing. Those are really two different faces of the same coin. *Performance testing*—often incorporated into automated testing suites—helps ensure that the application operates at a specific level of performance for specific tasks and workflows. *Load testing*, which may be a separate toolset, is designed to simulate the workload of many users performing specified workflows within the application to ensure that the application can handle a certain maximum number of users or to test and see how many users the application can actually handle while delivering acceptable performance.

Both of these key tasks are pretty difficult to complete manually. It's not practical to have five hundred real-live users pounding away at an application, for example, and it's extremely difficult for a human tester to accurately gauge application response times and other performance metrics. Tools can not only make these kinds of testing practical but also make them easier to repeat as often as necessary ensure that each test run is consistently performed.

In Conclusion

So this is quality: Starting with a set of clear, unambiguous requirements that communicate the business' expectations for the application; moving on to a design that maps to those requirements and specifies clear metrics that can be tested throughout the development life cycle; development that encompasses best practices and continual unit testing to verify compliance with requirements; testing that focuses on the requirements as a checklist for acceptability. Throughout, an organization can utilize the right tools—in an environment that has matured to an advantageous quality level—to greatly enhance the process of high-quality application development.

Download Additional eBooks from Realtime Nexus!

Realtime Nexus—The Digital Library provides world-class expert resources that IT professionals depend on to learn about the newest technologies. If you found this eBook to be informative, we encourage you to download more of our industry-leading technology eBooks and video guides at Realtime Nexus. Please visit http://nexus.realtimepublishers.com.