Realtime publishers

The Definitive Guide™ To

Building Code Quality

sponsored by

Don Jones

Chapter 7: Benefits of Automated Debug, Analysis, and Test 98
Selecting the Right Tools 98
Heightened Developer Productivity 99
Improved Code Quality and Reliability 102
Superior Manageability 105
Better Performance and Real-World Behavior 108
Improved Maintainability 110
Access to Modern Methodologies 113
Conclusion 114

Realtime publishers
MICRO FOCUS Leading the Evolution™

Copyright Statement

© 2009 Realtime Publishers. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtime Publishers (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtime Publishers or its web site sponsors. In no event shall Realtime Publishers or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtime Publishers and the Realtime Publishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtime Publishers, please contact us via e-mail at info@realtimepublishers.com.

[**Editor's Note:** This eBook was downloaded from Realtime Nexus—The Digital Library for IT Professionals. All leading technology eBooks and guides from Realtime Publishers can be found at http://nexus.realtimepublishers.com.]

Chapter 7: Benefits of Automated Debug, Analysis, and Test

Much of this book has been about improving your processes and procedures, adopting tools for automation, and improving practices for both development and testing. But what are the real-world *advantages* of these improvements? In other words, after investing time and money in adopting changes and tools, what kind of return can you expect on that investment? That's what this chapter is all about: Wrapping up everything that has come before with a focus on the actual business benefits.

However, before I dive into that, let me remind you that this book has never been simply about tools. It is extremely unlikely that simply purchasing and using tools will gain you the many benefits I'll be writing about. These benefits come primarily from changes in your practices and procedures; tools are—as I've written previously—simply a way to accomplish those practices and procedures more quickly. Tools can also make it a bit easier to stick with certain practices and procedures by offering to help enforce them for you, but it's ultimately your dedication to improved practices and procedures that will set you on the path to achieve the various benefits I'll be describing.

Tools are obviously beneficial in that they provide automation and enforcement, so you'll definitely want to add good development and code quality tools to your environment. Which tools will you select? I have a couple of strong opinions that I'll share.

Selecting the Right Tools

First and foremost, I'm a huge believer in the "right tool for the right job." That is, figure out exactly what features and capabilities you'll need—this chapter will actually help with that—and then adopt tools that provide those. As this book is primarily focused on .NET development, I expect that you're working in Microsoft Visual Studio. Most third-party tool vendors supplement Visual Studio and integrate with it in various ways. Integration in this manner that is essential because it keeps developers within the development environment that they're accustomed to, making it easier for them to use these new tools and features.

Microsoft itself offers supplements for Visual Studio in the form of the high-end "Team System" version of the product. I have mixed feelings about this version; I'm glad to see Microsoft incorporating better code quality and analysis tools into its product line, but at this stage, those tools are fairly primitive compared with offerings from more mature tools from third-party vendors. Team System costs a premium over the more commonly-found editions of Visual Studio, so if you elect to go with Team System, exercise due diligence and carefully compare its features, capabilities, and extra price with the features, capabilities, and pricing of third-party products. Select the candidate that best meets your needs and budget.

Realtime publishers
MICRO FOCUS
Leading the Evolution™

Heightened Developer Productivity

There's little question that automation can improve developer productivity. In fact, I really wanted to highlight this benefit first because some of the practices I've recommended—naming conventions, coding practices, and so forth—can definitely be perceived as *lowering* developer productivity. To be frank, I think you can *expect* a lot of those practices to *initially* lower developer productivity as developers tend to slow down a bit when they're diligently observing new practices and standards for the first time. Automation—through the use of various tools—can help offset that initial productivity hit.

One improvement is in unit testing. I've seen developers waste a lot of cycles testing the same code over and over, while neglecting other sections of the code. A good unit testing tool (see Figure 7.1) can help developers focus by indicating which code has been tested and which hasn't. This denotation helps ensure that developers are spending the right amount of time on unit testing rather than "chasing their tails."

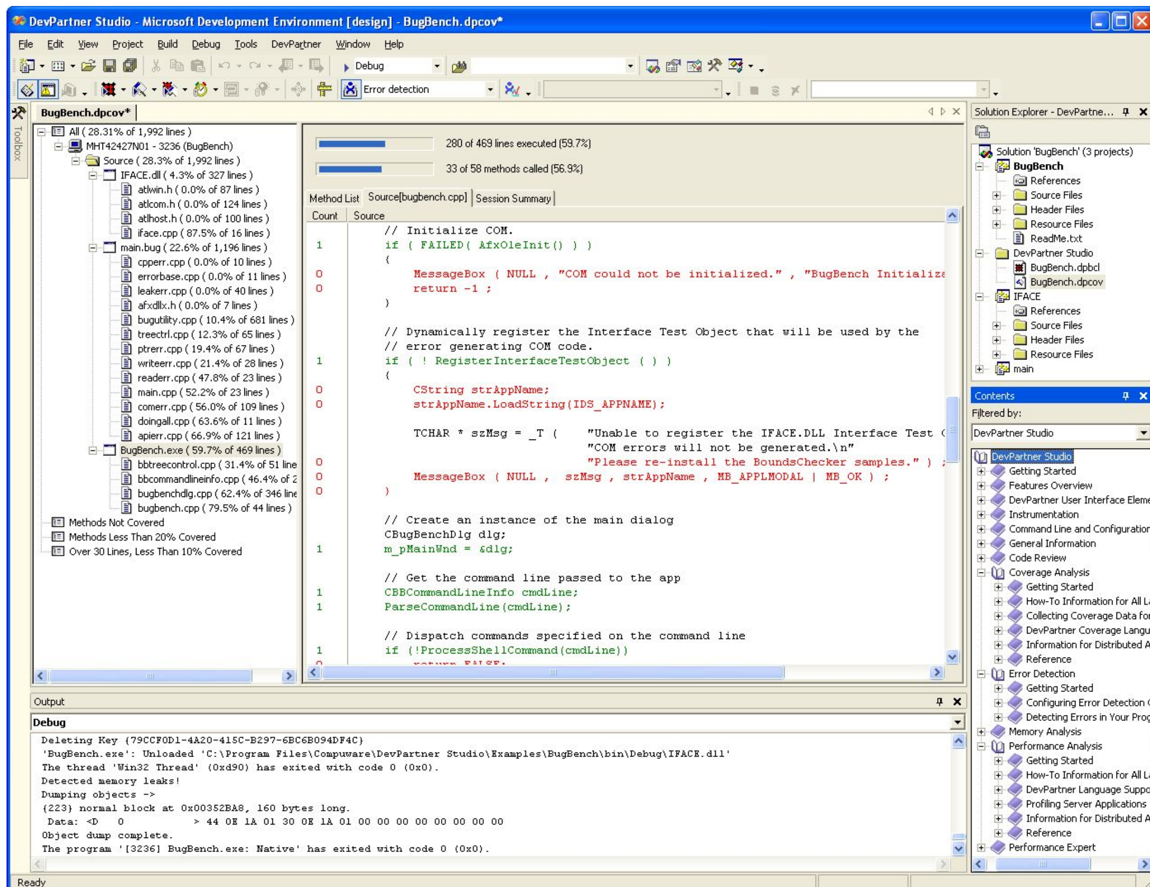


Figure 7.1: Identifying code that has not yet been tested.

Realtime publishers
MICRO FOCUS Leading the Evolution

Another improvement can be gained from tools that help developers during debugging. For example, tracking down a difficult-to-reproduce bug can be an enormous waste of time, especially when the difficulty in reproduction comes from differences between two systems. Tools can help compare two systems at a very detailed level, as Figure 7.2 shows, helping developers arrive at the root cause of the problem much more quickly.

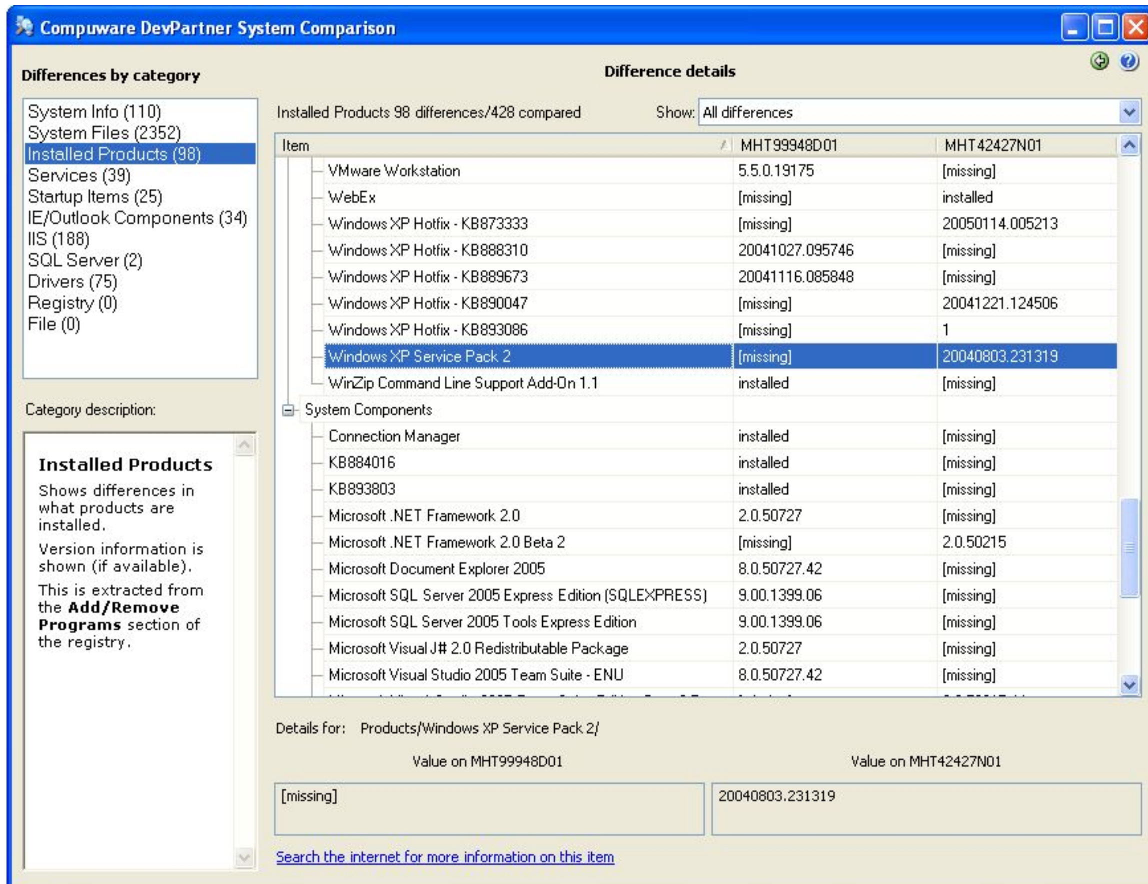


Figure 7.2: Finding differences between systems.

When it comes to identifying performance problems in code, you're typically looking at more wasted time. In fact, without the right tools, it can be impossible to determine exactly where in an application performance is being lost. With the right tools (see Figure 7.3), performance problems can often be traced to a few lines—or even a single line—of code, letting the developer quickly shift focus to the code that's actually causing the problem.

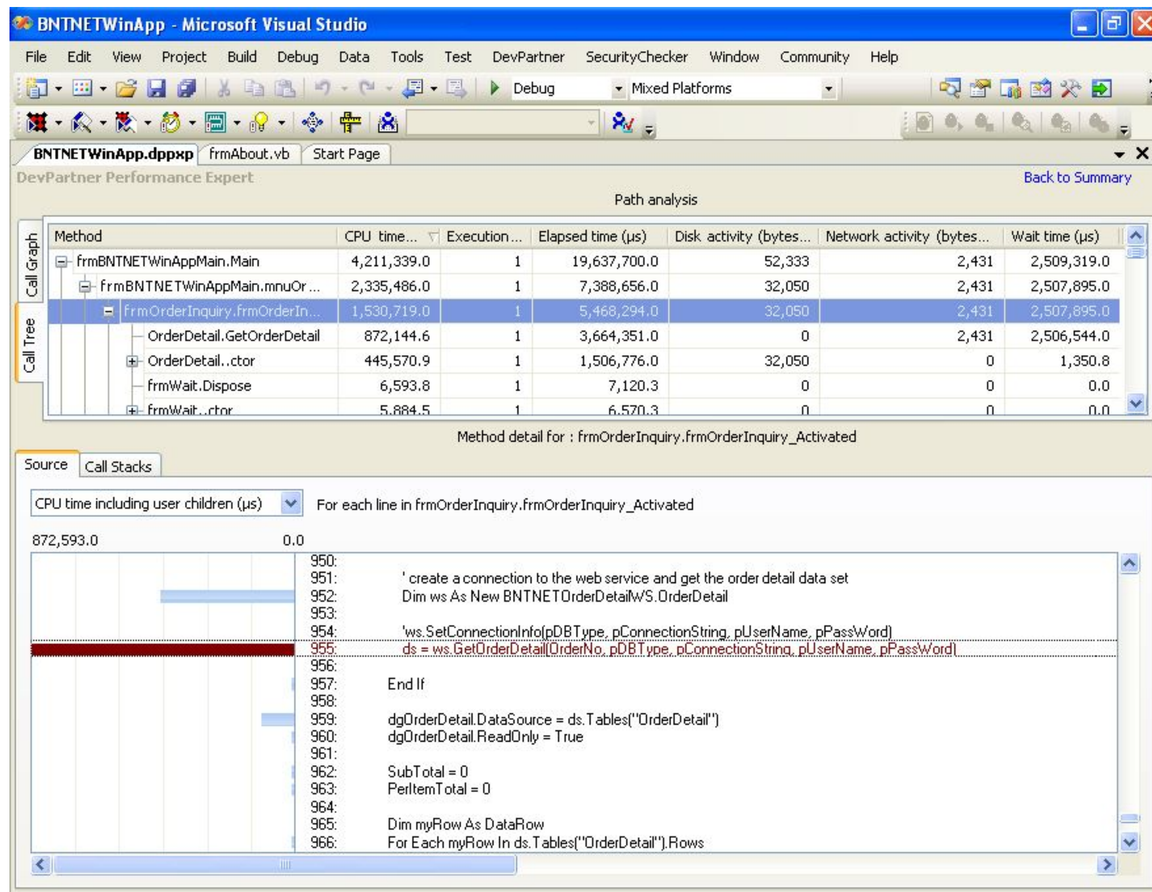


Figure 7.3: Identifying poorly-performing lines of code.

Finally, performance tools can help developers profile their code's performance, giving them insight into resource consumption. Figure 7.4 shows an example, with a breakdown of the time spent in given methods, the amount of memory consumed, and other performance data. This information helps developers quickly gauge overall performance and promptly focus on problem areas—without creating endless cycles of less-direct tests and fix attempts.

Realtime publishers
MICRO FOCUS Leading the Evolution

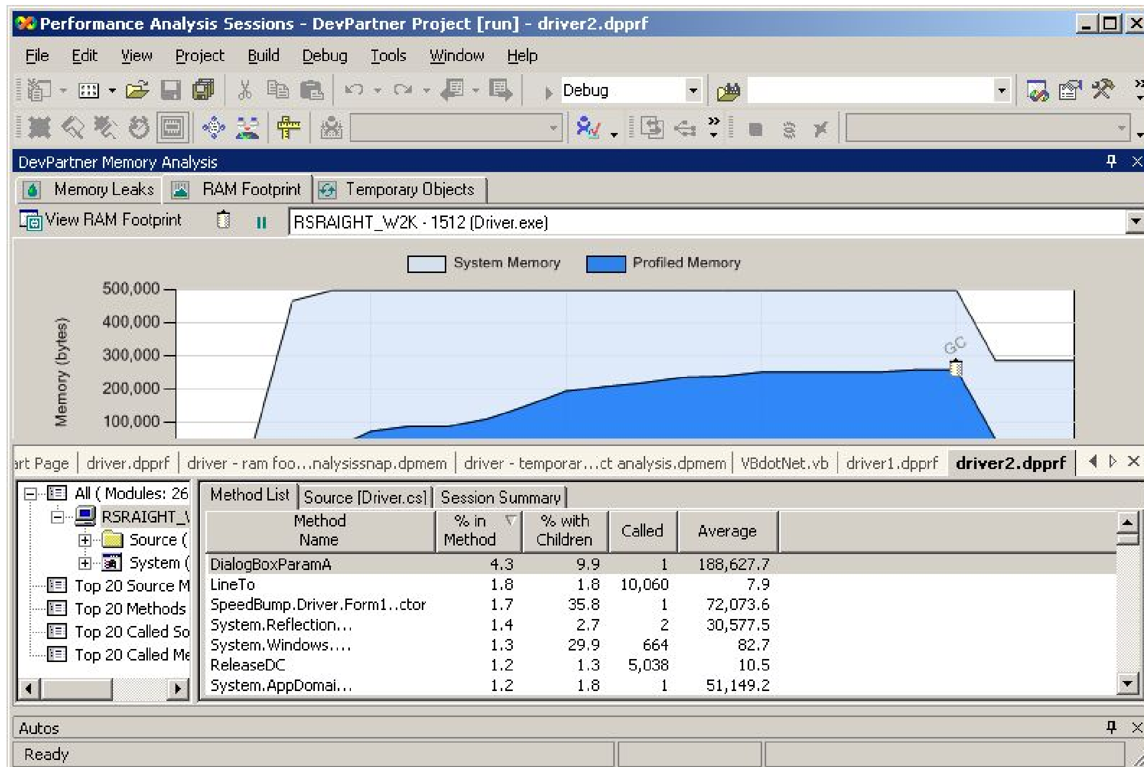


Figure 7.4: Performance profiling.

Frankly, using these tools doesn't require much additional effort for most developers; these tools simply speed up tasks that developers are (or should be) already accustomed to doing. That's a benefit because it means more immediate productivity gains.

Improved Code Quality and Reliability

Tools won't necessarily help improve code quality and reliability directly, but by using tools to enforce good coding practices, you can encourage and assist developers so that they're producing better-quality code all the time. For example, one feature of Visual Studio Team System is the Team Foundation Server's ability to enforce code quality guidelines during source code check-in. Figure 7.5 shows how these rules are defined. The product gives you the ability to define different rule sets for different projects.

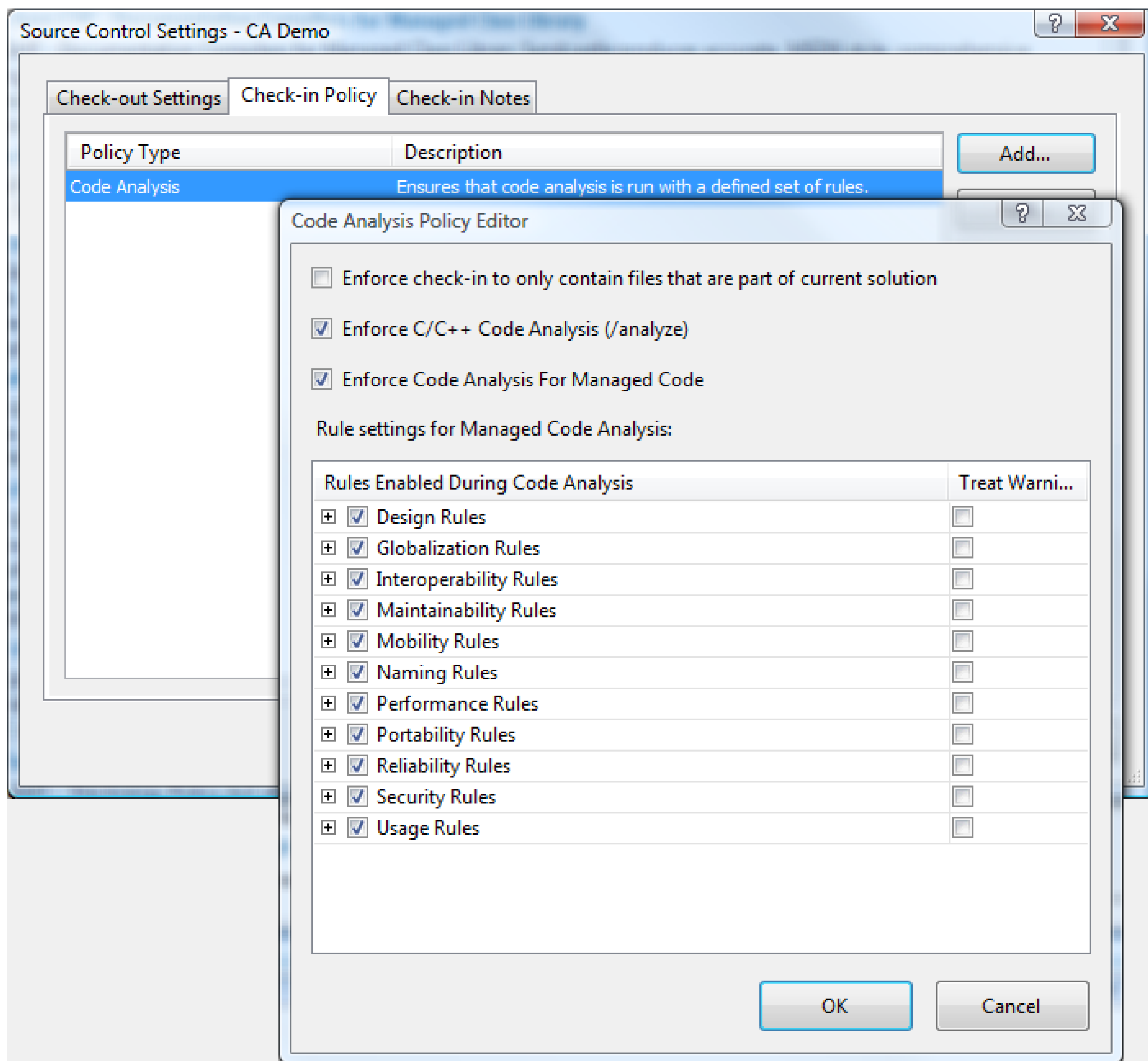


Figure 7.5: Defining check-in rules.

Using their copy of a Visual Studio Team System product, developers can apply the same rule sets to their code for local analysis. If a developer attempts to check-in code that hasn't passed the rule set defined at the source control level, developers are presented with a message like the one in Figure 7.6. This feature helps the developer realize the need to use the top-level set of analysis rules, and helps to enforce a specific level of code quality across all the developers in the project.

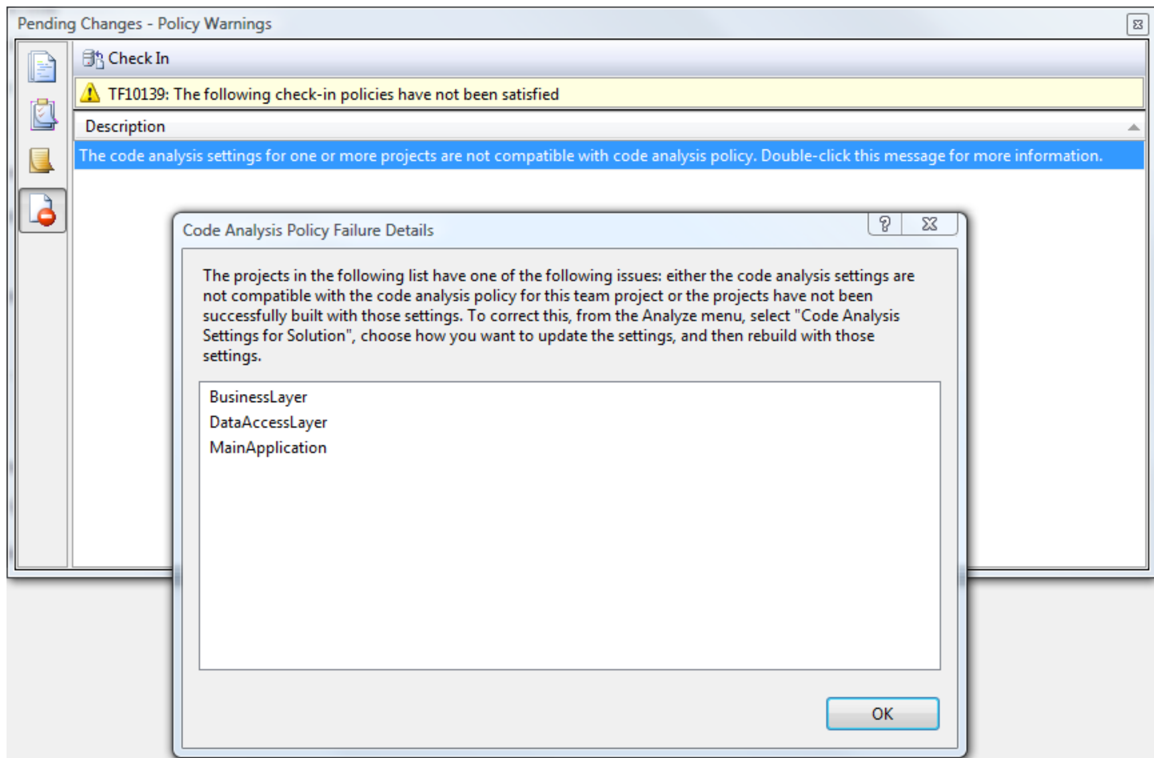


Figure 7.6: Enforcing code quality at check-in.

Third-party tools can also provide local code analysis using rule sets, all of which are designed to improve overall code quality. Figure 7.7, for example, shows Micro Focus DevPartner's code analysis rules in action. Here, the rules defined are a bit more complex than mere coding style and are actually looking for potential security and other functional flaws. To keep developers productive, the analysis should not only highlight problems but also offer an explanation and guidance—and examples—for fixing the problem and improving the code.

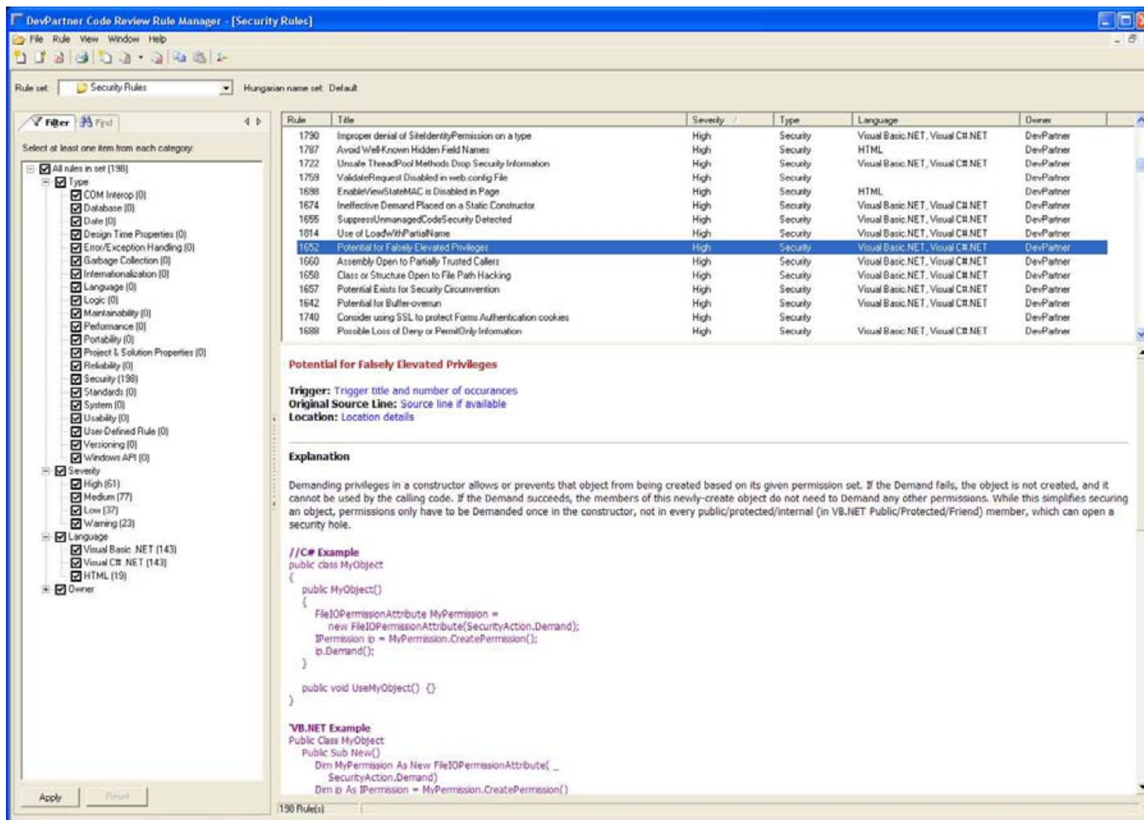


Figure 7.7: Code analysis in action.

Essentially, tools alone can't improve code quality. What they can do, however, is encourage developers to improve code quality by observing standards, conventions, and best practices. Code analysis tools in particular can help developers learn about these standards and best practices by providing developers with clear guidance when a standard or practice has not been followed and by detecting that problem early on in the coding process—ideally, before the code is even checked-in to a source control repository.

Superior Manageability

Management often shares too little of the code quality burden, often because management has so little insight into any kind of actionable quality data. In other words, it's tough to make smart decisions when you have no data to work with.

Provided your development managers are willing to make the tough calls, the right tools can give them the data they need to do so. For example, one simple set of metrics can help managers understand the general complexity and overall risk associated with a given portion of code. Figure 7.8 illustrates a simple, dashboard-style view that focuses on *maintainability*: Portions of the code with a low maintainability index will usually require more ongoing maintenance, create more risk, and definitely require more extensive testing. These numbers help managers decide where to focus resources: either more testing and more maintenance or more development time simplifying these components to raise their index.

Realtime publishers
MICRO FOCUS
Leading the Evolution™

Code Metrics Results

Filter: None

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 7.8: Code maintainability metrics.

Tools can also help make it easier to develop, maintain, and track application requirements—something that's absolutely critical in making project decisions. Figure 7.9 shows one way that requirements can be tracked. This tool also tracks how many tests have been performed, how much of the code the test covered, and how many tests passed and failed—crucial information for someone in charge of the project. This view essentially tells a manager how much testing is yet to be done, and how much of the code is currently in compliance with the requirements.

View: Coverage

Requirements

Favorite Folders

No Favorite Folders

Requirement Folders

Default Requirement Folder
CAF Requirement Folder
Integration Cycle 2
Lawson Interfaces
Master Backup
Master Surginet cycle 1
SC2 Billing Cycle 2
SC2 Case Tracking Cycle 2
SC2 Clin Doc Cycle 2

Reports

List of Projects

Project

Requirements

Default Requirement Folder

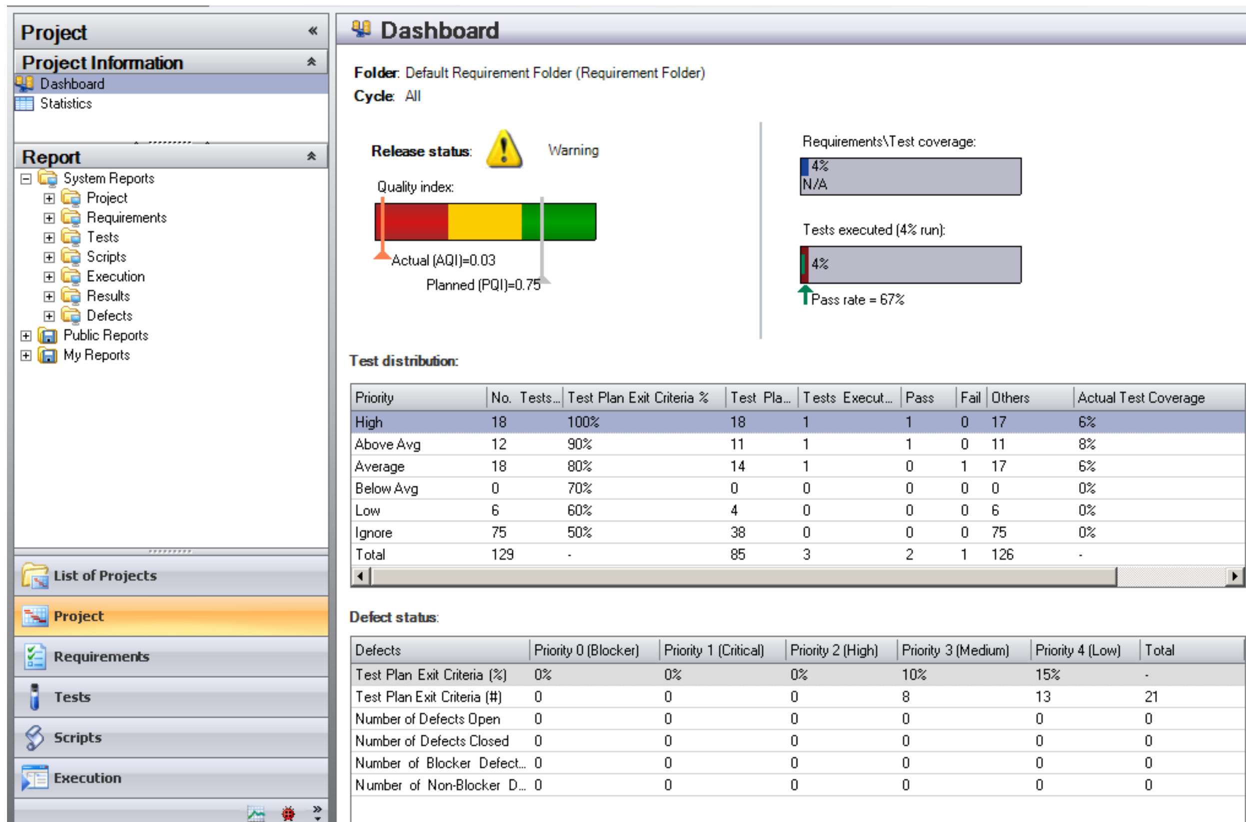
Name	Display ID	Tests (...	Coverage (...	Passed	Failed	Not Executa
Default Requirement Folder		117 (10...	79%	73	20	
Surginet Test Management	SURGIF000	117 (10...	79%	73	20	
Surginet Functional Testi...	SURGIF100	49 (42%)	96%	40	7	
SURGIF006 Billing	BILLNG01	7 (6%)	100%	4	3	
BILLNG01.01 Endo...	BILLNG01.01	1 (100%)	100%	0	1	
BILLNG01.02 OB C...	BILLNG01.02	1 (100%)	100%	1	0	
BILLNG01.03 LiftBr...	BILLNG01.03	1 (100%)	100%	0	1	
BILLNG01.04 Tonsi...	BILLNG01.04	1 (100%)	100%	0	1	
BILLNG01.05 Total...	BILLNG01.05	1 (100%)	100%	1	0	
BILLNG01.06 Urod...	BILLNG01.06	1 (100%)	100%	1	0	
BILLNG01.07 Virtua...	BILLNG01.07	1 (100%)	100%	1	0	
SURGIF005 Orders	ORDERS1	1 (1%)	0%	0	0	
SURGIF003 CASE TR...	CASETRK01	3 (3%)	100%	1	2	
SURGIF001 SCHEDU...	SCHED001	18 (15%)	100%	18	0	
SURGIF002 PREFER...	PREFx001	4 (3%)	100%	3	1	
SURGIF004 CLINICA...	CLINDOC1	16 (14%)	94%	14	1	
SURGIF009 CARE MOB...	CAREMOB01	1 (1%)	0%	0	0	
SURGIF007 INTERFAC...	INTRFA01	13 (11%)	100%	13	0	
SURGIF008 INTEGRATI...	INTEGRATIO...	43 (37%)	77%	20	13	
SURGIF010 SYSTEM Is...	SYSTEM000	11 (9%)	0%	0	0	

Figure 7.9: Tracking requirements.

Realtime publishers

MICRO FOCUS
Leading the Evolution™

Managers can also use higher-level views to get a feel for the application's overall quality. For example, Figure 7.10 shows a very simplified view with a very straightforward "quality meter." This perspective is perfect for executives concerned about the status of the project. In this example, the project's quality is shown, compared with the planned quality level, and a simple indicator of the "release status" makes it easy to see whether the project is ready for release.



Priority	No. Tests...	Test Plan Exit Criteria %	Test Pla...	Tests Execut...	Pass	Fail	Others	Actual Test Coverage
High	18	100%	18	1	1	0	17	6%
Above Avg	12	90%	11	1	1	0	11	8%
Average	18	80%	14	1	0	1	17	6%
Below Avg	0	70%	0	0	0	0	0	0%
Low	6	60%	4	0	0	0	6	0%
Ignore	75	50%	38	0	0	0	75	0%
Total	129	-	85	3	2	1	126	-

Defects	Priority 0 (Blocker)	Priority 1 (Critical)	Priority 2 (High)	Priority 3 (Medium)	Priority 4 (Low)	Total
Test Plan Exit Criteria (%)	0%	0%	0%	10%	15%	-
Test Plan Exit Criteria (#)	0	0	0	8	13	21
Number of Defects Open	0	0	0	0	0	0
Number of Defects Closed	0	0	0	0	0	0
Number of Blocker Defect...	0	0	0	0	0	0
Number of Non-Blocker D...	0	0	0	0	0	0

Figure 7.10: Code quality dashboard.

Dashboards like this can offer a deeper level of management, too. For example, if code quality is poor and a limited amount of time and other resources is available to improve it, where should those resources be focused? By allowing managers to play "what if" scenarios, a tool can show where development and testing resources can be best deployed to achieve the highest quality improvement. Figure 7.11 shows an example: Here, two scenarios are shown, each with different levels of testing devoted to different levels of code. The tool shows the change in overall quality that will be achieved by each action. This data makes it easier for a manager to achieve the desired balance between code quality and available resources.

Realtime publishers
MICRO FOCUS® Leading the Evolution™

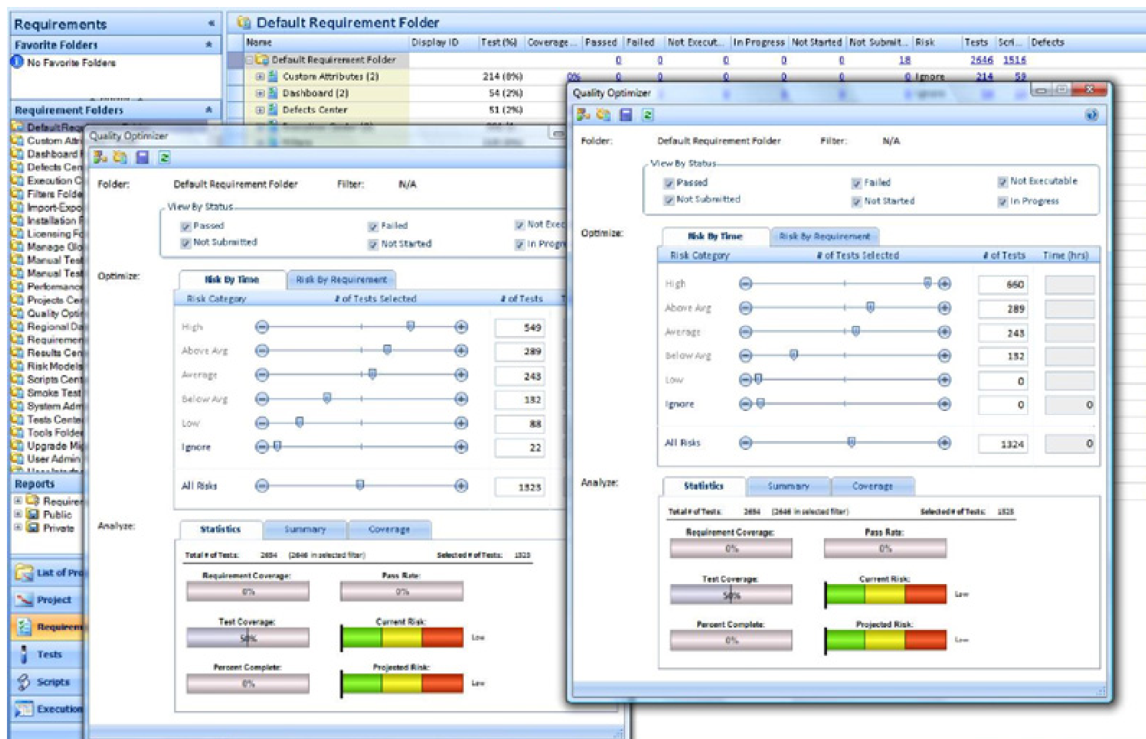


Figure 7.11: "What if" scenarios with code quality.

Much of code quality comes from smart, informed management decisions. With the right tools to provide the right information, management decisions can focus on the desired level of code quality and drive decisions throughout the project that align to help achieve that level of quality.

Better Performance and Real-World Behavior

Of course, one of the things you're really looking for when pursuing better code quality is a better experience for your end users, both in terms of performance and behavior. By *behavior*, I'm referring to both crashes (you want fewer of them, of course) as well as an application that works the way users need it to.

Again, practices and procedures are the way to start improving performance and behavior, but tools can help automate and enforce your goals. Figure 7.12, for example, shows how tools can help profile an application during development, assigning performance values to individual methods and displaying a sort of *call tree* to help developers visualize the execution paths of their code and spot code that is contributing to poor performance. In this example, one method is contributing more than 70% of the application's run time, suggesting that this method could use some re-thinking. This type of visualization helps developers improve performance during development and unit testing, when making changes is more straightforward and less disruptive to the project in general.

Realtime publishers

MICRO FOCUS
Leading the Evolution™

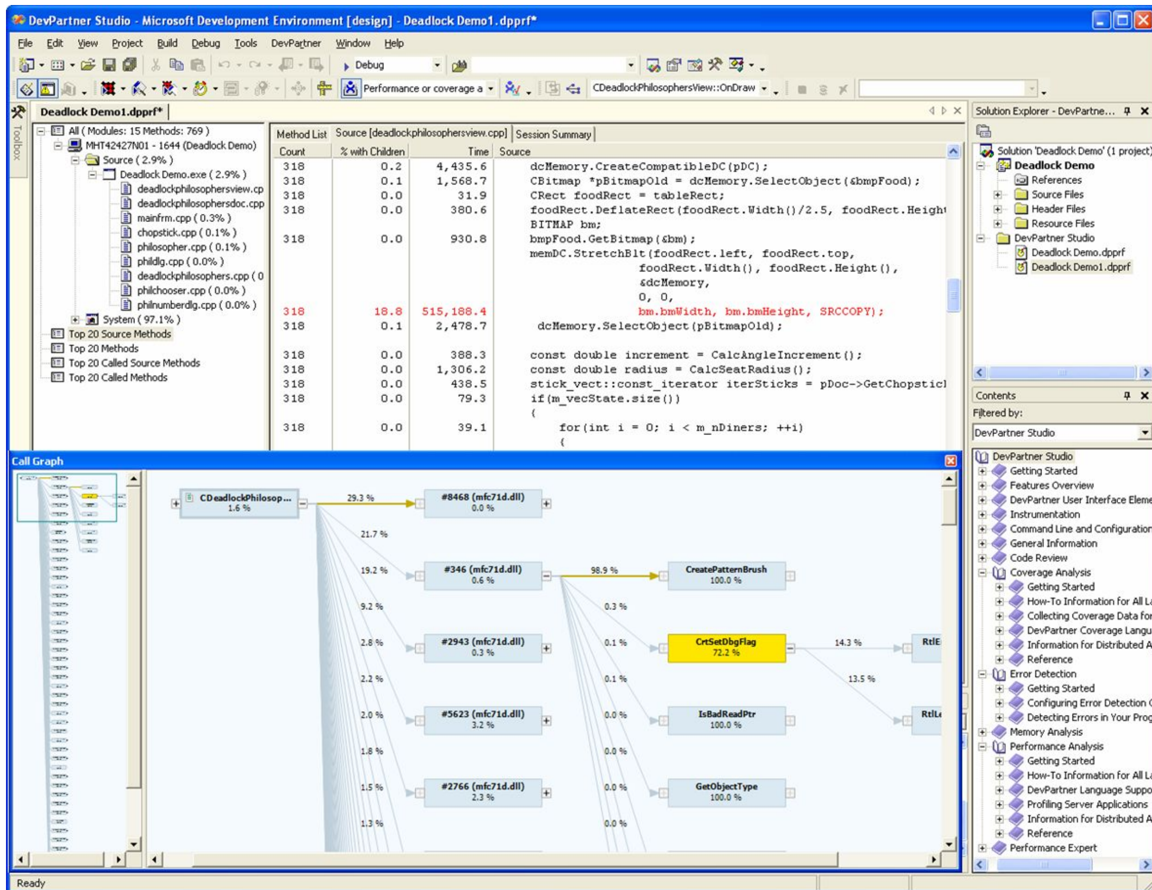


Figure 7.12: Tracking down performance problems in code.

Another advantage of using automation tools is that they can help do the things you *know* you should be doing but probably can't afford to do manually. Workflow-based testing, for example, is a form of functional testing (generally done by a QA team, not developers) that duplicates the way a user interacts with the application. In order to be useful, this type of testing needs to follow fairly precise "scripts" so that the desired portions of the application are tested consistently. Undertaking this task manually is tedious, time-consuming, and frankly, inefficient: People aren't robots and don't typically excel at sticking to a strict script again and again and again. Computers, however, really *are* robots and are great at running through a script over and over. Tools like the one shown in Figure 7.13 help test developers create those testing scripts by allowing them to walk through an application (in this example, a Web application), designate input, indicate desired output, and so forth. The testing tool then executes the script against the real application, over and over—as many times as needed.

Realtime publishers

MICRO FOCUS Leading the Evolution™

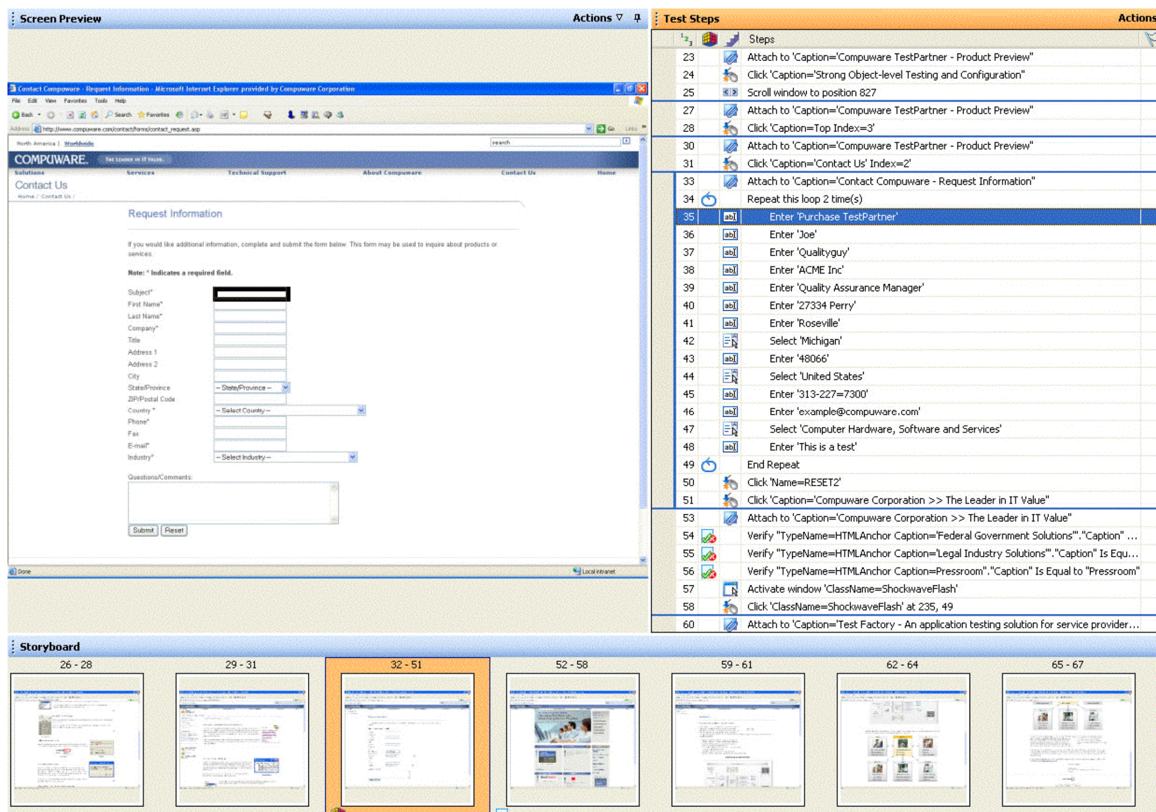


Figure 7.13: Developing workflow-based tests.

These same scripts can often be used to enable other forms of testing, such as load testing and stress testing, which both help to ensure the desired level of performance from the application under real-world, production scenarios involving multiple end users working simultaneously.

Improved Maintainability

Better quality code means code that can be maintained more easily over time. Typically, one component of "better maintainability" is "less complex." Here again, tools—and the right procedures for using them—can help. Code analysis tools that can produce complexity reports—like the one illustrated in Figure 7.14—can help developers and project managers reduce code complexity as the project is underway. High-level reports like the one shown, as well as the drill-down reports that typically come with them, help pinpoint areas of high complexity and offer an opportunity to re-engineer the code to improve its chances for better long-term maintenance.

Realtime publishers

MICRO FOCUS®
Leading the Evolution™

Complexity and Completeness Report

Project	Order System
Description	An application used to allow Order Creation/Editing and general aspects of a Business Ordering system

Summary		
Total number of Interaction Points. (Sum of all packages, requirements, scenarios and steps.)	171	
Total number of packages.	4	
Total number of requirements.	13	
Total number of scenarios.	33	
Total number of steps.	121	
Total number of actors.	3	
Total number of Non-Functional Requirements (Project Level):	3	
Total number of Non-Functional Requirements (Requirement Level):	9	
Average number of requirements per package:	3 (13/4)	
Average number of steps per scenario:	3 (121/33)	
Maximum requirements in a single package:	6	
Minimum requirements in a single package:	1	
Order Fulfillment		
Maximum nested depth:	1	
Steps per actor:		
System	82	
Administrator	0	
User	31	
Number of bad links:	0	
Number of empty glossary definitions:	0	✓
Number of empty actor definitions:	0	✓
Number of empty packages:	0	✓
Package: Order Processing		
Total number of Interaction Points. (Sum of all requirements, scenarios and steps.)	116	
Total number of requirements.	6	
Total number of scenarios.	21	
Total number of steps.	89	
Number of bad links:	0	

Figure 7.14: Code complexity report.

Remember, complex code isn't inherently bad; it can be, however, more difficult and risky to maintain. You can't necessarily eliminate all highly-complex portions of code in every situation, but when you can reduce complexity, doing so will help save time, money, and effort in the long run. Reports like this simply offer management information so that you can make better decisions during your project.

Another aspect of better maintainability is better testing. Think of it this way: If you release version 1.0 of an application but haven't *really*, completely tested the code, you're that much closer to *needing* v1.1—because untested code is very likely to have defects. However, code that has been fully tested before release is likely to have fewer defects, need fewer immediate fixes (including the ones that keep developers up late at night), and instead allow v1.1 to focus on revised features and capabilities that help meet changing business needs. Fully-tested code, in other words, is code that requires less maintenance. Actually testing code completely, however, can be difficult simply because most applications offer thousands of potential code paths that each needs to be tested. Tools, of course, can help.

In addition to automating the testing process—thus allowing you to accomplish more testing in less time—testing suites can work with code analysis tools to help identify every possible code path and track each one that has been tested. Similar to the test coverage tools used by developers during unit testing (which I wrote about earlier in this chapter), reports like the one shown in Figure 7.15 help QA teams see how much of the code they've covered in their full, functional, and integration testing.

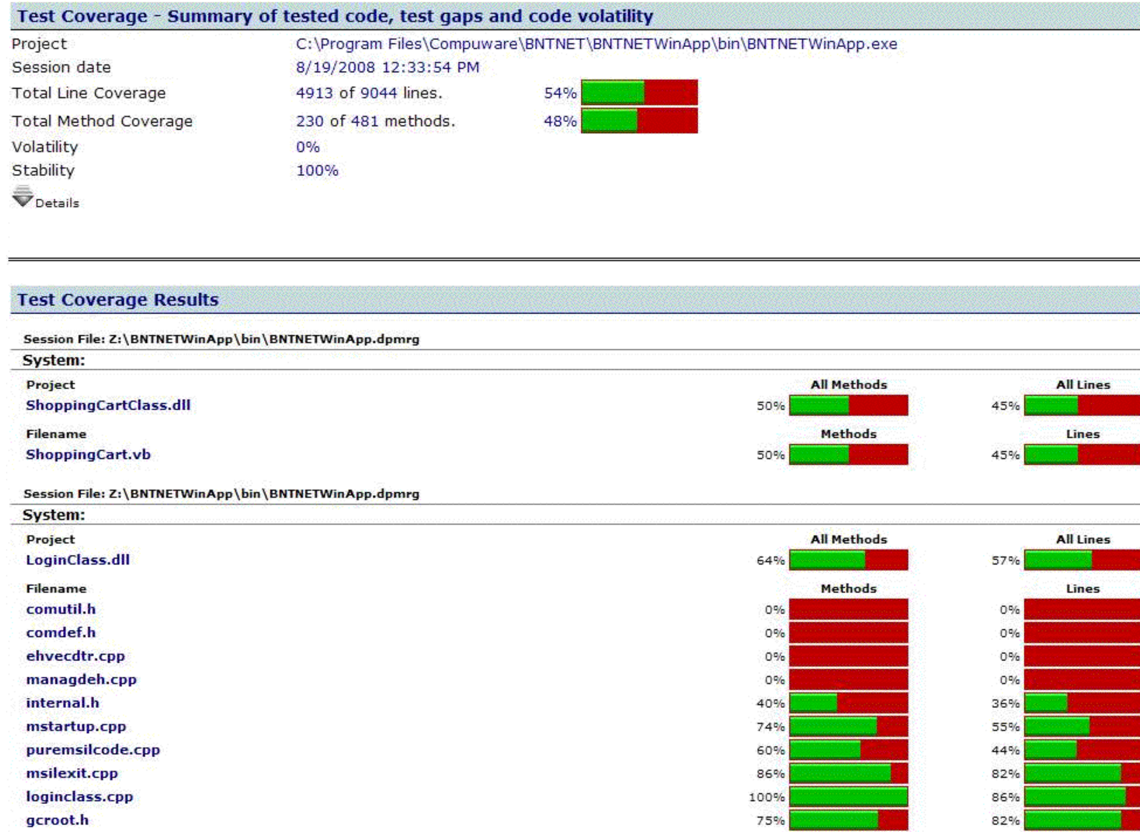


Figure 7.15: Testing coverage report.

Every team might aim for 100%, and if you have the resources to test to 100%, you'll have a better application for it. But even if 100% isn't reachable, this type of report helps QA teams ensure that they've applied the needed testing to the riskiest portions of the code, intelligently applying the available resources. The right tools can even help identify the riskiest portions of code, either by complexity or—as shown in Figure 7.16—by identifying the portions of code that are utilized the most in the application (or that consume more of the application's run time).

Realtime publishers

MICRO FOCUS
Leading the Evolution™

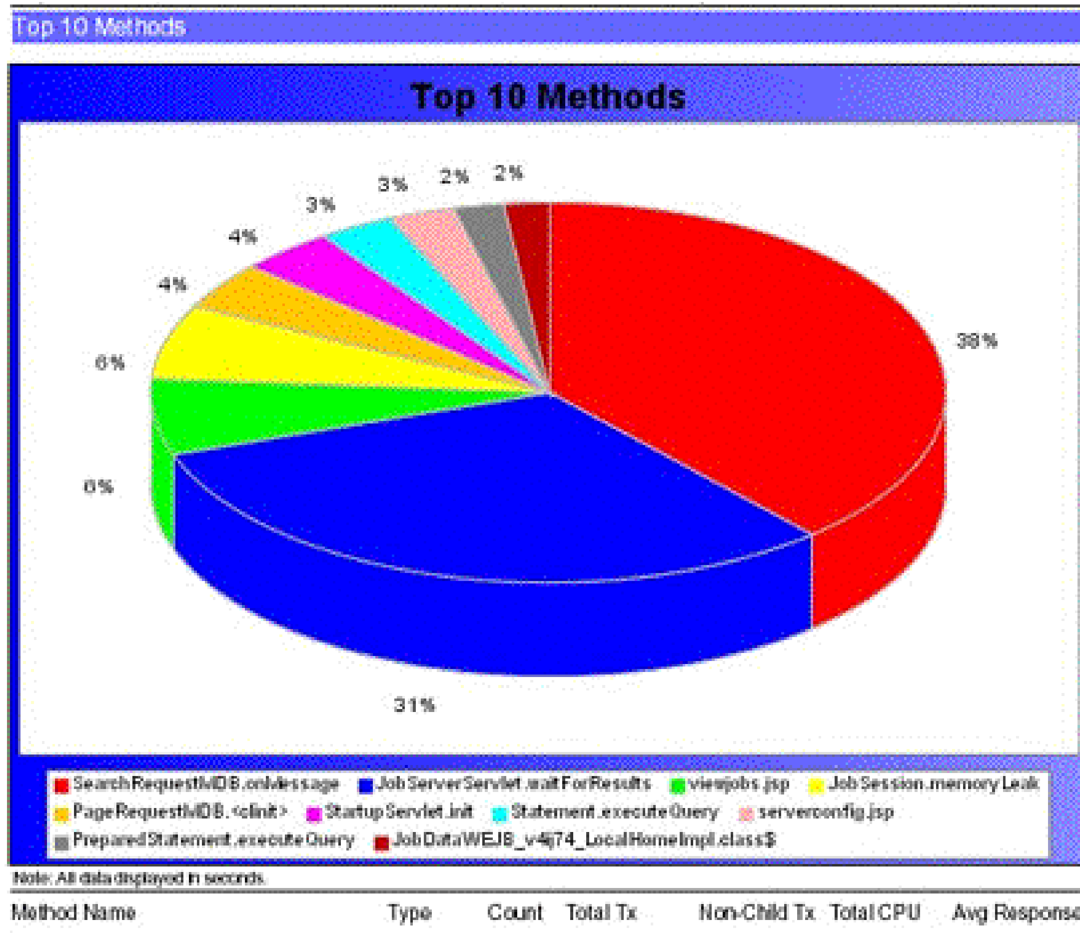


Figure 7.16: Identifying the most-used code in the application.

Access to Modern Methodologies

With the improved code quality that comes from using the right practices and procedures, and with the streamlining and efficiency that code quality tools can contribute to the development process, your development team may find themselves able to utilize more modern development methodologies. For example, teams using a monolithic, top-down development approach such as Waterfall may now find themselves able to work in smaller, shorter project cycles using a newer development methodology such as Extreme Programming, Agile, Scrum, and so on. Many of these methodologies are specifically intended to work with high-efficiency teams on smaller, incremental development projects, and they help teams to release smaller revisions of their applications more frequently. That approach helps increase the application's ability to respond to changing business needs. However, the approach is difficult to execute successfully with a team that isn't used to producing high-quality code with the help of code quality tools.

The bottom line: Better code quality not only brings direct benefits from having better applications but also offers development teams new opportunities for flexibility and change.

Conclusion

Well, there you have it: *The Definitive Guide to Building Code Quality*. In this book, I've touched on numerous aspects of quality and examined ways to improve the quality of the code in your applications. Together, we've covered code analysis and quality metrics, looked at various aspects of testing and QA, and examined ways to track performance issues and implement performance improvements in code.

Code quality is a journey. Cliché as that might sound, it's true. The idea is that there is always some room for improvement, and you tackle a small bit of quality at a time, gradually improving code quality and educating developers, testers, and managers. With the right processes and procedures—and with the right tools to help enforce them and speed things along—you'll begin to realize significant business benefits.

Good luck!

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.