The Software Quality Advisor Online

Rice Consulting Services, Inc.
P.O. Box 891284
Oklahoma City, OK 73189
405-793-7449
405-793-7454 FAX

March 2001 Newsletter

Testing Middleware and Firmware

by Randall W. Rice, CQA, CSTE

Every software application in use today relies on some level of middleware and firmware to deliver the functionality the user eventually sees. This places middleware and firmware in a very important, yet often overlooked, layer of architecture. When approaching the test of middleware and firmware, people sometimes struggle to find effective strategies and techniques. The purpose of this article is to outline a testing strategy and to point you to further resources.

Definitions

Middleware

Middleware is the layer that resides between the hardware layer and the application layer to provide services such as database management. From the CNET glossary, middleware is defined as:

"This software manages the communication between a client program and a database. For example, a Web server connected to a database can be considered middleware--the Web server sits between the client program (a Web browser) and a database. The middleware allows the database to be changed without necessarily affecting the client, and vice versa."

The Software Engineering Institute (www.sei.cmu.edu) defines middleware as:

"Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. Middleware is essential to migrating mainframe applications to client/server applications and to providing for communication across heterogeneous platforms."

Middleware can handle transaction processing (TP) for distributed applications, Remote Procedure Calls (RPCs) for extending application logic across a network, Object Request Brokers (ORBs) which allow for the distribution of application objects across heterogeneous networks, and Message-Oriented Middleware, which allows the passing of messages between objects and applications across a network.

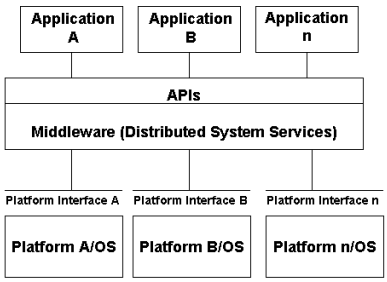


Figure 1 – Middleware

Firmware

Firmware can take on many forms, but is basically the embedded software that is stored in hardware memory to control hardware functions. Firmware controls PCs, printers, and every other electronic device (remember Y2K?).

The Issues

As you can see, middleware carries quite a responsibility in delivering quality information. In some recent projects, I have had the opportunity to delve into the testing issues surrounding middleware. There are significant testing differences required for middleware and firmware, however, most testing literature assumes the ability to test through a user interface. While this

article might not have a broad audience, I often tell people that the information they think is not applicable today may become very important tomorrow.

Some of the issues surrounding both middleware and firmware are:

- Complex processing due to extremely high numbers of interface combinations
- High criticality in the operations performed
- The difficulty in testing from an external perspective
- The importance of developer testing at the unit level
- The uniqueness of the types of test tools needed
- The difficulty in building a test environment.
- An emphasis on structural testing
- An object-oriented focus

All of the above concerns shape an overall test strategy that is:

- Performed early
- Performed by developers first
- Leveraged by structural test tools
- Realized by using test harnesses and stubs
- More object-oriented than procedural
- Involves testing large numbers of interfaces

We will deal with each of these points in the test strategy for middleware and firmware.

A Test Strategy for Middleware and Firmware

We have defined middleware and firmware and understand that they are different, yet have many characteristics in common when it comes to testing. The discussion of test strategy for these types of software will include both middleware and firmware, and can be extended to test any software which is not accessed by a user interface.

Early Testing

Early testing will multiply the testing effectiveness of any software application, regardless of technology. However, in the world of middleware and firmware early testing is most critical because finding defects at later stages carries a higher penalty of rework. This is due to the extent of integration with hardware and other software.

The problem with early testing in this environment is that with so many integration dependencies, how does someone create test harnesses and stubs that allow for an accurate test? Manually, the job is possible, but can be overwhelming when there are many interfaces involved. If you are developing in a language that has tool support for structural test case design and testing, you may find that the job can be very easy. Specifically, for C++ and Java, Parasoft (www.parasoft.com) has a great toolset to design and perform structural tests, with a feature to automatically create a test harness and test stubs. Similar tools are available from International Software Automation (ISA) www.softwareautomation.com.

Developer Testing

Developer testing is essential to avoid high rework costs. To the testers, the software is a black box. Only the developers have the view and access to the code to test all conditions. In addition, not only are functional cases at stake, but also the structural tests for memory boundary violations and memory leaks.

My experience is that developers can test software if the have a good process to follow, standards to show what is expected of them in terms of testing, and a way to hold developers accountable for the quality of their work. Management must also be making the message loud and clear that testing is part of the job and that quality is a shared responsibility between developers, testers, QA, and management.

An Object-oriented View of Testing

In the object-oriented view of testing, tests are isolated at a smaller scope, yet can have high complexity due to the interfaces with other objects. The object-oriented view of testing must be able to deal with classes, methods, and attributes and to validate those at a high level of coverage.

In Shel Siegel's book, "Object-Oriented Software Testing," he describes the Hierarchical approach to O-O testing.

"The hierarchical approach is at the heart of the object-oriented testing system. This test approach uses and builds upon several well-understood testing techniques, tying them together into a comprehensive testing system.

The hierarchical approach leverages the fact that "everything is a system." It defines and applies testing standards for several levels of software component: objects, classes, foundation components, and systems. The hierarchical approach designates as SAFE those components that meet the testing standards for that kind of component. Once you designate a component as SAFE, you can integrate it with other SAFE components to produce the next-level component. In turn, you test this component to the level of safety associated with the component level it represents. SAFE is always a relative state. It depends entirely on the standards you choose to enforce, your application, your attitude toward risk, and the specific risks and risk management practices you adopt in your project. The hierarchical approach provides guidelines for minimum safety; you decide what is right for you."[1]

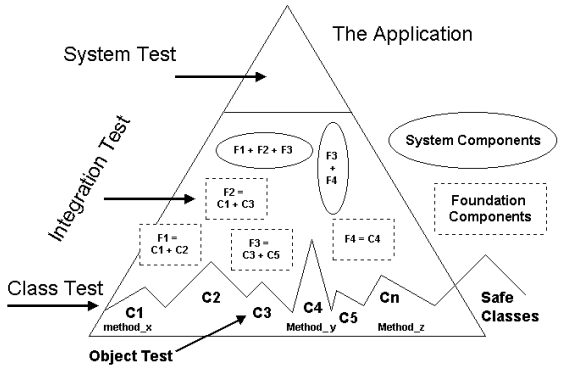


Figure 2 – The Hierarchical Test Approach

Summary

Much more could be said about testing middleware and firmware. This article is intended to provide a strategy for testing and a starting point to start developing detailed techniques. My best advice to you is to read the books on Object-oriented testing by Shel Siegel and Robert Binder (I review Shel's book in this issue of the newsletter), investigate the available test tools (new ones are emerging all the time) and work at the developer level to reinforce the criticality of testing.

[1] Siegel, Shel, Object-Oriented Software Testing, Wiley and Sons, 1996

Frequently Asked Questions

by Randall W. Rice, CQA, CSTE

Q: I would like to know the current status of software testing e.g. how much attention and time is spent on software testing.

And testing is a way to solve problems of buggy software, however, software today is still released in buggy states, why this is so?

A: It varies from project to project, but it's not uncommon to see one-third to one-half of a project's time and cost going to testing.

You are right that testing helps, but it's only one filter among many. Good testing can't totally make up for bad design and coding! Why is software so buggy? Easy. People keep buying it that way. The software companies know exactly the levels of quality their customers will buy. Therefore, they never deliver over that mark.

Q: Where can a person find information specific to "function points"? Are there books that deal specifically with function points (i.e. what they are, how to test for them, how to define them in the requirement)?

A: There are two places I would look.

1) The international Function Points User Group (IFPUG) web site at www.ifpug.org. However, about the most useful thing you will find for free there are the links.

2) Carol Dekker's web site at http://www.qualityplustech.com/. She has an articles page that you can request an article to be mailed to you for free in PDF format. This would probably be the best starting point.

Yes, there are books. I would look for those by Capers Jones to get the most recent information. As for the relationship to software testing and QA, all function points do for you is give you the ability to tell the size of a system or application - period. They are a better way to measure size than lines of code

because they are easier to count and more transportable between technologies. So, for example, you could base defect measurements by function points. Ex: 5 defects per function point in one system vs. 3 defects per function point in another would tell you that the overall quality in the second application is higher than the first.

Here are some books that look good on the topic:

Function Point Analysis: Measurement Practices for Successful Software Projects (Addison-Wesley Information Technology Series)
by David Garmus and David Herron

Software Assessments, Benchmarks, and Best Practices (Addison-Wesley Information Technology Series) by Capers Jones

Estimating Software Costs
by Capers Jones

Q: I do not have the much-needed knowledge in the very complex field of software testing. I would like to get your advise on how to educate myself and where.

A: My advice is to start reading books. The three that I recommend starting with are:

Surviving the Top Ten Challenges of Software Testing by Randall W. Rice and William E. Perry

Effective Methods of Software Testing, 2nd Ed. by William E. Perry

Testing Computer Software by Cem Kaner

Q: We have a WEB application that allows our customers to view their orders and to download a file. We are trying to stress test this product utilizing software. Can you please tell me how many users should test my infrastructure?

A: That's one of the toughest questions to answer in planning a load test. Here are a few thoughts.

1. You need to test at normal usage levels, peak usage (the highest levels in a day, week or month) and maximum usage (the highest levels you have reached thus far).

2. How to predict these levels are another story. If you have a history to work with, that's great. If not, about the best you can do is predict the expected load. Just about everyone I have heard speak on this, everyone I have read, and our own experience bears this out, that at this point load testing is a dark art.

3. Keep in mind that the concurrent users actually performing actions on your web site will be a fraction on your web site visitors. So, if you have 1,000 concurrent users, perhaps only 100 will actually be concurrently viewing orders, etc.

Quotes of the Month

"'Average' has become so bad that a person can just show up to go to the head of the class." John Maxwell

"To err is human…but when the eraser wears out ahead of the pencil, you're overdoing it." Jerry Jenkins

Rice Consulting Services, Inc.
P.O. Box 891284
Oklahoma City, OK 73189
405-793-7449
405-793-7454 FAX

Building a Testing Foundation
by Carl Chandler

To have a successful test team all stakeholders must be involved. If anyone from requirements to final user acceptance testing is left out of the process then "reworks" in the software are going to increase.

Many studies have been done on the effects of reworks on the economics of testing. The bottom line is:

- Most defects are created in the early stages of a project
- Most defects are found in the later stages of a project
- It costs 10 to 100 times as much to fix a defect in the later phases of a project

With the involvement of all stakeholders these pitfalls can be avoided. How do we involve all stakeholders?

Rice Consulting Services, Inc.
P.O. Box 891284
Oklahoma City, OK 73189
405-793-7449
405-793-7454 FAX

Obstacles when involving all stakeholders in the process of testing are from the human side of testing – not the technical. These obstacles can be overcome. The following is a list of the top ten challenges to software testing based on a survey taken over a three-year period of 1,000 testers.

1. Having to say "no"
2. Testers are in a "lose/lose" situation
3. Rapid change
4. Over-reliance on independent testers
5. Not enough time for testing
6. Lack of customer and user involvement
7. Lack of management understanding/support of testing
8. Lack of test tools
9. "Us vs. Them" mentality
10. Not enough training

Rice Consulting Services highlights these obstacles and ways to overcome them in our 2-day course "Basic Training in Software Testing".

Basic Training in Software Testing will help you become more comfortable and confident in testing software applications at just about any level of detail: unit, integration, system, and user acceptance. You will emerge from this one-day session knowing how to develop test cases and test plans. You will also leave with a knowledge of how tools can help you perform testing.

Sometimes people feel intimidated by the technical aspects of software testing and lack the confidence they need to be credible test leaders in their organization. Learn the issues and processes for effectively testing software by attending this hands-on course.

Rice Consulting Services' Course Offerings

If you would like to learn more about the information covered in both Randy's and Carl's articles we here at Rice Consulting Services, Inc. offer two excellent courses that will enhance your companies software quality process.

Unit Testing – 2 days

This course is designed for testers and software developers who want to learn how to test software at a detailed level. The process taught in this course can be applied to many different technologies and development

environments. The course covers both functional and structural testing, with numerous examples and templates.

Basic Training in Software Testing – 2 days

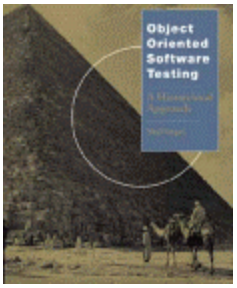
This is a quick start course in software testing for people just getting into the field, or for people who just need a refresher course or validation for their current testing techniques. This is a practical hands-on seminar to cover the critical path of testing. Your instructor will be a certified instructor in the QA and testing field. You will learn the terminology, process, and challenges of testing in the real world. As a result of attending this seminar, you should have a good working knowledge of software testing and what it takes to design and conduct an effective test of software, regardless of the technology.

For more information on these courses or one of our many other offerings please contact Carl Chandler at (405) 414-6759

Book Review

by Randall W. Rice, CQA, CSTE

Object-Oriented Software Testing by Shel Siegel



Paperback - 528 pages 1 edition (July 13, 1996)
John Wiley & Sons; ISBN: 0471137499 ; Dimensions (in inches): 1.16 x 9.07 x 7.47

I had the privilege of knowing Shel Siegel during the time he was formulating the ideas for this book and I can attest that he brings a lot of real-world testing experience of complex O-O applications to the table. It may be a bit of a stretch for some people, but this book is organized as the system of which it describes the testing. If you approach this book with a procedural mind-set, you may have to reorient your thinking process to adapt to the way the book is written.

I found this book very helpful in conveying the concepts of building a testing hierarchy and building a test repository to my clients. This is not the only book out on this topic and would certainly recommend also reading other books to get recent insights. However, I still can recommend this book.

I have read some of the other reviews on Amazon.com and BN.com, and I can say that I think it has received some low reviews from people that simply have different preferences and standards in testing. I can tell you that the approach to testing described in this book is probably more well-defined than most people who do ad-hoc testing would care to perform. However, if you are in an environment where risk is high, you will appreciate the rigor of the process.

All I can say is that I got a lot of good ideas for testing object-oriented software from this book and I'm glad that Shel wrote it.

Scoring

Readability - 4
Breadth of coverage - 4
Depth of discussion - 4
Accuracy - 5
Credibility - 5
Organization – 3

Overall Score - 4

March Links

Help Identify and Manage Software and Program Risk
http://www.stsc.hill.af.mil/CrossTalk/2000/nov/baldwin.asp

Writing an Effective IV&V Plan

http://www.stsc.hill.af.mil/CrossTalk/2000/nov/walters.asp

April 2001 Issue:

- **Problem Analysis – The First Step in Software Development**
  by Randall W. Rice, CQA, CSTE
- **Automated Testing**
  by Carl Chandler