



**White Paper:**

**Introduction to Software Auditing**

**with Harkins Audit Software's**  
*Real-Time Program Audit*

**by Tom Vernon**

**United States Patent No. 6,775,827**  
**Australian Patent – Patent No. 778165**

Copyright © 2003, All Rights Reserved  
Harkins Audit Software, Inc.  
816 Daisy Lane  
West Chester, PA 19382  
[www.harkinsaudit.com](http://www.harkinsaudit.com)

Telephone: 888.350.9148 or 610.431.1755  
Fax: 610.436.1249

Technical Support: [paulhark@aol.com](mailto:paulhark@aol.com)

Sales: [paulhark@aol.com](mailto:paulhark@aol.com)

# Table of Contents

I. Introduction.....	1
II. Typical Problems in Today’s Software Development .....	2
Reengineering Legacy Software .....	2
Debugging.....	3
Validating New Programs .....	4
III. Traditional Solutions .....	4
Reengineering legacy programs.....	4
Debugging.....	4
Validation/new programs .....	5
IV. Limitations of Traditional Solutions .....	5
V. What is Real-Time Program Auditing (RTPA)? .....	7
VI. Productivity Gains from RTPA .....	8
Reengineering Existing Programs.....	9
Debugging.....	9
Validating New Programs .....	9
VII. Case Studies.....	10
Using RTPA to Enhance Software.....	10
Using RTPA to Find Bugs.....	12
Using RTPA to Verify Development.....	14
VIII. Conclusion .....	16

## I. Introduction

In the short span of 50 years, software engineering has evolved from a profession mainly of interest to those working in research labs, large manufacturing firms and computer science departments to one that impacts on virtually all aspects of 21st-Century life. Air traffic control systems, telephone networks, the Global Positioning System, cell phones and even household appliances all depend on reliable source code to function properly.

Despite many advances in software engineering such as extreme programming, CASE (Computer-Aided Software Engineering), Open Source, and 4th generation analysis tools, many in the industry feel that rather than improving, developing and supporting software is actually getting more difficult.

The cost of bad software is staggering. It is estimated that defective programs cost corporate America \$293 billion a year in lost productivity<sup>1</sup>. Research reveals that 31% of software projects will be cancelled before they are completed, and that almost 53% of projects will cost 189% of their original estimates. Even more alarming are the lost opportunity costs, which are not directly measurable, but may be in the trillions of dollars.<sup>2</sup>

Critics of the industry point to a series of recent events where software glitches have grabbed international headlines<sup>3</sup>:

- the opening of the largely computer-controlled Denver airport has been delayed for over a year
- a NASA Mars probe crashes on the planet's surface
- a U.S. Navy ship accidentally shoots down a civilian airliner
- the ambulance system in London is shut down, resulting in as many as 30 deaths

---

1 Koerner, Branden I. "The Bugs in the Machine." Wired Aug., (2002): 28.

2 "CHAOS Report," The Standish Group,  
[www.standishgroup.com/sample\\_research/chaos\\_1994\\_1.php](http://www.standishgroup.com/sample_research/chaos_1994_1.php)

3 Mann, Charles C. "Why Software is So Bad" Technology Review July/August (2002): 34.

Skeptics reflect on these headlines and similar problems and predict that litigation and legislation may be the only things that will force the software industry to release more thoroughly tested programs<sup>4</sup>. More positive-thinking analysts believe that new practices and technological tools will spearhead an industry-wide movement to create better software.

While the challenges for software engineers are great, so are the opportunities. This white paper will briefly discuss the state of the art in software engineering, then introduce Harkins Audit Software and Real-Time Program Audit (RTPA), a cutting-edge software tool that allows programmers to watch the execution of their software. Using RTPA enables programmers to easily create and enhance software more quickly and with fewer defects.

## **II. Typical Problems in Today's Software Development**

Whether they are reengineering existing software, debugging programs, or creating new ones, software developers who want to create quality code are faced with a number of challenges during a typical workday. How well they respond to these challenges affects not only their own performance, but also the ability of their businesses to achieve their goals.

### **Reengineering Legacy Software**

Maintenance of legacy programs is a Herculean chore. While much of today's software is developed in object-oriented languages such as C++ and Java, a vast amount of legacy source code is written in older procedural languages such as RPG and COBOL. Current estimates suggest that there are roughly 250 billion lines of source code in existence which need to be maintained.<sup>5</sup> Most of this code is written in procedural languages, and much of it still runs on mainframe computers. The challenge for programmers is to

---

4 Koerner, Branden I. "The Bugs in the Machine" Wired Aug., (2002): 28.

maintain and enhance this software at a reasonable cost while avoiding interruptions to vital business services.

Reengineering legacy programs is difficult because it is necessary to first understand existing code before revisions can be made. Documentation of these legacy programs is usually sparse or non-existent, and with the high turnover rate in many IT shops, the original programmer or development team is often long gone. Even with good documentation, finding the location for making enhancements in a large program can be difficult.

### **Debugging**

Debugging large and complex programs is a major component of IT costs. Bugs not only add to the cost of software maintenance, but they can also result in business expenses in terms of unhappy customers, penalties and employee downtime.

It has been estimated that a professional programmer makes 100 - 150 errors for every 1000 lines of code that they write<sup>6</sup>. The challenge with debugging is to quickly identify the faulty logic in a program before making the necessary changes.

Some bugs are easily discovered and fixed, while others such as intermittent failures or unexpected results can be very hard to trace. Abnormal terminations are often the result of straightforward logical failures, such as “divide by zero” or a memory leak, while intermittent failures are often the hardest to debug. These failures can include an unexpected result or abnormal termination that occurs infrequently and is not easily replicated in a test environment. Unexpected results can occur when test sets are bad, conditions are improperly coded, fence conditions are improper, conditions are insufficient for all cases and scope-related problems (accidentally resetting variables).

---

5 Sommerville, Ian. Software Engineering, 6<sup>th</sup> Edition. Essex: Person Education Limited, 2001. 623.

6 Humphrey, Watts S. Winning With Software: An Executive Strategy. Reading MA: Addison-Wesley Professional, 2001. 73.

## **Validating New Programs**

Testing and validating software functionality is a major component of the software development process. New programs may include complex logic, critical calculations or extensive manipulations of data which require validation before being put into service. The challenge for programmers is to create software at a pace that matches today's rapidly changing business environment while ensuring the quality of the software through testing and validation. Programmers must find innovative ways to shorten delivery times for large, complex systems without compromising quality.

## **III. Traditional Solutions**

Programmers currently have at their disposal a wide variety of tools and techniques to assist them in the everyday tasks of reengineering legacy programs, debugging, and the creation of new software.

### **Reengineering legacy programs**

The most time consuming part of reengineering or enhancing legacy programs is the process of understanding the program. Programmers may use a visual code review (also known as a *structured walkthrough*) to analyze how a program works. With this technique, programmers read the code line by line, running through the cases in their mind to follow the potential execution flow.

4th Generation analysis tools are also available for analyzing programs. These programs review source code to create theoretical maps of their execution flow. They provide graphical representations of software structure, identify poorly-written code sections and support testing programs.

### **Debugging**

Debugging is a complicated process, and a skill that most programmers develop with time and practice. Programmers look for common programming

---

mistakes such as improperly declared variables, and attempt to match these with observed patterns.

Interactive debugging tools are included in most software development suites, and usually include a run-time environment that allows access to the compiler symbols table, as well as values of program variables. Users can run a program by “stepping” through it, statement by statement. Values of variables can be examined after each step, allowing the programmer to find the causes of errors.

Another common method of catching bugs is to insert *ad hoc* statements manually into the source code with instructions to write debugging state information to a file or buffer. This allows programmers to see if the program execution reaches certain points and potentially any state information that the programmer wants to see.

#### **Validation/new programs**

Once a software package has been developed, validation and verification (V&V) is performed to confirm that the software meets specifications, as well as meeting the needs of the customer. This includes extensive evaluations with test sets which are similar to the actual data that will be processed by the program. The developer will then examine the outputs of the program and check for anomalies.

### **IV. Limitations of Traditional Solutions**

While there have been many advances in the tools that developers use to reengineer, debug software, and validate software, these tools also have some limitations in certain situations.

**Structured Walkthroughs** can provide a good insight into the structure of a program, but they are inefficient with a programmer’s time and are prone to error. Because programmers do not know how the execution flows before the

walkthrough, they often waste time analyzing portions of the program that are not important to the project at hand. Further, programmers are required to guess about cases and conditions when following execution flow, which is prone to error and may result in the programmer following false paths, wasting valuable time and energy.

**4th-Generation Analysis** tools can help a programmer understand the overall flow of a program. However, these tools are often very expensive, require substantial training to use and are limited in their ability to show the potential execution flow of programs that are not highly modular. Further, these analyses are unable to tell a programmer what actually happened; they only show what could happen.

**Interactive Debuggers** are a mainstay for programmers, although they have some limitations that reduce their efficiency. One problem with interactive debuggers is that they require a programmer to interact with the program for each and every line of execution. For large programs, this can make the process tedious or even impractical. To help resolve this problem, programmers can insert breakpoints in their software to engage the debugger only where necessary. However, programmers often do not know where to place the breakpoints in the first place, making the process hit-and-miss. Interactive debuggers are generally not practical for intermittent problems. Finally, debuggers only operate on objects in test environments, requiring a programmer to create a suitable environment for testing real-world data, which can be time-consuming or impractical.

**Inserting Debugging Statements**, while effective, is tedious and time consuming. In order to add or remove debugging statements, the programmer must do a complete edit-compile-load cycle. Generally, programmers must create these statements from scratch, potentially leading to new errors. Finally, all of this code needs to be removed before the program is released.

**Unit Tests** allow a programmer to ensure that each component of a new application works correctly prior to putting the entire program together. However, the developer may invest considerable time and software development in setting up the components for testing, ensuring that the program is reading the test sets correctly and ensuring that the test output corresponds to the program's actual function. In a batch environment, waiting for results adds even more time to this process.

## **V. What is Real-Time Program Auditing (RTPA)?**

RTPA is a technique developed by software developer and IT consultant Paul Harkins. Paul discovered that too much of his time was being spent on three repetitive tasks: learning other people's code to reengineer programs, tracking down bugs in production systems, and validating new programs.

Most programmers handle these tasks by using common tools and techniques such as structured walkthroughs, manually inserting debugging statements into their code, and using interactive debuggers. The real-time program auditing method that Paul devised combines the best of all these techniques, resulting in a technique that is easy to use and extremely effective.

RTPA is a software utility that can theoretically work on any source code, in any environment, to create a line-by-line audit of that code's actual execution. Using RTPA, programmers can capture just about anything that they want to know about a program's execution. An audit-enabled program captures source statements, comments, variable contents, compile listing statement sequence number, change ID of the source statement, statement change date, and time of execution.

RTPA is a precompiler that converts a normal source program into an audit-enabled source program which is then compiled normally. The RTPA process is simple:

1. RTPA analyzes the source code and then creates a new temporary source code file containing both the source code and audit statements.
2. RTPA compiles the audit-enabled source code with the regular compiler. The resulting object is an audit-enabled object program.
3. When the audit-enabled object program is initiated (interactive or batch), the executable itself produces an audit output file, which we normally refer to as an audit file, or audit.

Audit-enabling a program is simple. Using the RTPA interface, a programmer selects a source file for auditing and optionally sets other options. Once the software and options are selected, RTPA creates an expanded object program, which it places in a user-specified library for execution. The original source files and object programs are not modified.

There are several advantages to using RTPA. One is hands-free operation. Once a programmer compiles RTPA into a program, there is no need to set program breakpoints or stop the normal flow of the program as it executes.

Also, programmers can make your audit as in-depth or high level as they like, choosing to audit every executable source statement in the source program and the value of every variable, or just focusing on selected program files, records, operations, variables and subroutines. Audit output may also be started, stopped, and resumed based on the data contents of variables, or the relationships among variables in the audit-enabled object program.

RTPA has a very slight impact on overall system performance, allowing programmers to test and validate their software while it runs at near-normal speeds. In real-world tests, auditing a program adds as little as 20% to total CPU time on an IBM iSeries (AS/400) Midrange Computer.

## **VI. Productivity Gains from RTPA**

Auditing of software with RTPA techniques can lead to substantial gains in productivity for many kinds of programming activities. For large projects,

these gains can often be measured in months or millions of dollars. RTPA takes hours, days and months off a programmer's development and debugging time because it lets the programmer see exactly what is happening in the object code. RTPA takes away the guesswork and lets programmers be more productive.

RTPA is very intuitive and easy to use. In most cases new users are up and running in within minutes.

### **Reengineering Existing Programs**

The most difficult and time-consuming part of reengineering legacy programs is usually understanding someone else's code. Auditing can shave days or weeks off this process by identifying key units in a program and illustrating program flow.

### **Debugging**

Some of the most perplexing software bugs to catch are those which are too intermittent to find with interactive debuggers or difficult to replicate in a lab environment. Auditing is a uniquely powerful tool for locating these problems. RTPA users have saved hours of work and months of elapsed time waiting for these intermittent bugs to occur.

### **Validating New Programs**

RTPA speeds the development process by allowing programmers to validate execution of code while developing it, thus insuring the integrity of all inputs, logic and outputs. Auditing saves considerable time over traditional methods, where the programmer may spend hours working through the logic, only to discover that the test input data was corrupt.

Programmers are able to confirm that the test sets are what they want them to be as they are read into the program. Auditing can also shorten large batch times. Since auditing is done in real time, the programmer is able to use a real-world set that might take hours to process, and using RTPA, look at the

first few records of input to confirm that they are correct. Without RTPA, a programmer would need to wait for the processing to complete before reviewing the output.

## **VII. Case Studies**

Whether they are tasked with enhancing existing software, finding bugs, or writing new programs, many programmers are finding that Real-Time Program Audit from Harkins Audit Software enhances their productivity, increases the quality of their software and solves problems that may have been virtually unsolvable with conventional tools.

### **Using RTPA to Enhance Software**

Enhancing pre-packaged software can be a thorny task. Often these large applications were created by multiple programmers over many years. The logic is frequently hard to follow, and there may be lots of calls to external programs at different levels. The challenge for a programmer is to understand the application before making changes and to ensure that these changes don't result in unwanted consequences. These two tedious, laborious stages are often where programmers spend the majority of their time.

Because of the difficulty and risk of enhancing existing software, most companies have backlogs of software enhancement requests, resulting in large numbers of frustrated, unhappy customers. To add to the pressure, many IT departments are unable to make vendor-supplied software upgrades because of the difficulty in reintegrating prior enhancements.

Despite pressure to get the job done quickly, programmers must understand the existing program before attempting any modifications. This was the challenge facing Dan Goldstein, Southern Applications Support Manager for Fishman & Tobin, Inc., who has been programming for 35 years. He was tasked with taking an existing distribution system and migrating it into PkMS,

Manhattan Associates' extended supply chain execution platform, bringing his functionality into MA's package.

Many problems can arise from this type of complex interaction between in-house programs and vendor supplied software. Dan's earliest attempts at integrating the two applications were not successful, due largely to the difficulty in interpreting the logic of the program. "I could have spent days, weeks or months just looking at their flow, and there were hundreds of different conditions that branched out. After using RTPA," Dan explains, "I was able to determine where in the logic I could exit their application and insert my program, and then return to their logic without destroying any of the integrity of their data. This allowed us to implement our new warehouse system much faster than we could have without RTPA."

In another instance, RTPA helped Dan with a problem of tracking down a faulty result in a vendor-supplied package after he made an enhancement to it. "I thought the problem was with the code that I had written, because theirs was pure vanilla. I spent an entire day going over my logic and could not find a bug," Dan explained. "By using RTPA, I was able to determine that it really wasn't the program that I had written, it was about six levels down in their application, which called about 35 different programs depending on different scenarios that you never would have been able to find by just reading logic. But by using RTPA and following the data flow, I actually found a problem caused by the pure vanilla program with a bug that only certain conditions exposed. I was able to fix their program and take the onus off the code that I had written."

Dan uses RTPA to audit labels, file I/O, external references, branches and conditional operations in order to get an immediate overview of the actual flow of an unfamiliar program. This allows Dan to eliminate time spent reviewing unnecessary sections of the source listing: by seeing the actual flow, without the need to guess about which conditions or branches are followed in the execution, he is able to save time and maximize his efficiency.

Another way that Dan uses RTPA is to audit arithmetic operations in order to understand complex calculations. Following the actual execution of the operations and seeing all of the intermediate values of the variables in the operations eliminates guessing about how a program calculates its results, and gives him greater confidence in verifying changes.

RTPA can also be used to audit individual variables to identify where key variables are changed by the program. This is much faster than working with breakpoints and stepping. The extra time left over for testing reduces the likelihood of unintended consequences when modifying programs.

Dan's experiences show that by using RTPA to enhance software, a programmer can shave days, weeks or even months off the time it normally takes to understand, enhance and test enhancements to legacy software.

### **Using RTPA to Find Bugs**

When it comes to maintaining complex production systems, the stakes are usually high. Downtime can cost thousands of dollars for each hour that a system is off-line. Similarly, mistakes can cause customer charge-backs or other penalties, and even a single bug may cost as much as a programmer's annual salary. Not surprisingly, programmers are under the gun to find and fix problems quickly.

Solving bugs in production systems can be difficult for several reasons. Failures can be intermittent and therefore not easily replicated in a test environment; reviewing the source code's logic may not reveal where or why the bug occurs; the bug may be the result of a complex interaction between home-grown code and the vendor-supplied software; or the failure may occur only in response to real-world data sets that are difficult or impossible to replicate in a test setting.

This is the working environment of Jerry Hollo, Senior Programmer/Analyst for a large clothing wholesaler, who has been programming for over 22 years, and has worked with every IBM midrange system to date. Much of his

work is with PkMS, a complex supply chain execution platform with multiple program calls to numerous sub-levels.

Jerry is often faced with the challenge of testing new code which has been added to in-house programs that perform functions specific for his company. These new programs ride on top of the production program environment. Because of the complex interaction between the in-house programs and the vendor-supplied programs, when bugs occur, it is often difficult to determine what is causing the problem. The error could be a result of a side-effect of a new business rule, where data changed in the in-house program inadvertently affects the function of the vendor-supplied program. Regardless of the cause, these problems must be handled quickly. If a developer spends too much time finding bugs, it could adversely impact on deadlines for other corporate projects.

Jerry notes that the debugging process took much longer before he had RTPA. “Sometimes I used before and after data analysis,” he explained, “I would also insert calls to a temporary program that would print out limited pre-selected data. I would use debug if the program was interactive, although this involved the additional step of submitting jobs to batch processing. With RTPA, I am able to see everything that happens in all of the programs. I can immediately see everywhere that a piece of data is used, which business rules are put into operations, their results and all of their intermediate values.”

Often, debugging PkMS involves following the program’s execution flow through numerous levels based on the values of various parameters. One program may or may not get called depending on the value of one of the parameters. One bug that Jerry encountered was the result of placing code in the wrong program in a stack of eleven programs. “Everything worked fine under one condition,” Jerry explains, “but failed to work under another. Using RTPA helped me to understand that I needed to place my code in a different program so that it would execute under both circumstances.” Jerry credits RTPA with

dramatically improving his productivity. “In this case,” he notes, “I was able to save about 40 hours of debugging/testing/analysis time.”

Such time savings are not unusual. Jerry states that he is now able to solve most problems within 1-2 hours. Without RTPA, he says that it can often take from one day to two weeks, depending on the complexity of the programs involved.

One strategy that works for Jerry is to use RTPA to audit a unique variable, such as a customer number, to search the audit file. This saves time by allowing him to skip over normal cases. Another tactic involves auditing the time of execution along with the source statements. Doing this allows Jerry to match the audit result with the time of failure.

Jerry uses RTPA to audit-enable the programs in the production system to resolve critical production problems. By audit-enabling all of the programs that are being executed, he can readily isolate and correct bugs.

Some schemes are easy with RTPA, but difficult or impractical with others such as interactive debuggers. One strategy that Jerry employs is setting up a test system and engaging the audit until a bug occurs. This ‘hands off’ approach allows him to get other work done while waiting for errors.

RTPA also assists in maintaining higher levels of customer satisfaction. Jerry says, “Our Information Technology department endeavors to provide users with software that does not require correction, and RTPA has been a tremendous assist in that effort.”

By using these techniques with RTPA, a programmer can dramatically reduce the time that it takes to isolate and fix production bugs. The end result is a system with fewer failures and shorter outages.

### **Using RTPA to Verify Development**

The ‘do more with less’ maxim prevails in most IT departments these days, where the pressure is on to complete more projects in the same amount of time, while maintaining or improving reliability and quality. Developing new

software however, is a time and labor intensive process. New applications can behave erratically, requiring the programmer to spend time tracking down exactly what is happening. New programs frequently requires novel solutions, and novel solutions require that time be invested in validation.

Patrick Weaver has been in the business for over 20 years, and works as a senior programmer at Certegy, a check collection company. He is faced with the difficult task of creating one-time conversion programs to bring in and convert new business into his company's collection system. Before he can write new code, Patrick must first understand how the existing program works.

DEBUG is not well-suited to this type of task. "With DEBUG, the assumption is made that you are aware of what the program is or should be doing. You're stepping through one thing at a time, and when you find a problem you stop and fix it," Patrick notes. "But what if it's program that you want to add functionality to? You really need to know how a program works and what the flow is. RTPA is much better suited to this task, as you can run the program and get a flow view of the logic".

Patrick adds that RTPA cuts down on the number of steps normally taken using a stepwise debugger feature. "RTPA gives such a sweeping overview of a program that you can see 2 or 3 situations that can be addressed all at once without having to step it, get to a place where you had a problem, then fix that problem. Typically, you go through an iteration process where you do 5 or 6 recompiles of the program. I was eliminating several recompiles at a time every time that I used RTPA, and finished a conversion in about half the time that it would normally take. RTPA is a simple concept with profound implications."

Patrick uses RTPA to audit conditional statements in order to validate reliable operation in "fence conditions". By using this strategy, he is able to confirm that his code works exactly as expected, and he also reduces the need for additional test functions.

When an application terminates unexpectedly, Patrick audits all the source statements to find out why. While this could be done with a interactive debugger, auditing with RTPA saves time because he doesn't have to guess where to put the breakpoints. He can see the execution flow from the failure back to its initiation in one file, eliminating guesswork and recompiles. Finally, by auditing all I/O, Patrick ensures that the right data sets are being used, and that they are being read correctly.

By making dramatic reductions in the time it takes developers to design and validate new software, Harkins Audit Software empowers IT departments to do more with less, while improving quality at the same time.

## **VIII. Conclusion**

Over a comparatively short span of time, software has become embedded in nearly every aspect of modern life. Programmers in the new millennium will be dealing with more complex code along with more stringent demands for productivity and quality control of that code than at any time in the past. Many of the traditional software development tools such as the interactive debugger have severe limitations that restrict the programmer's ability to efficiently create such code.

New technologies and techniques are being developed to meet the demand for better quality software in less time. One innovation is Harkins Audit Software's *Real-Time Program Audit*, which combines the power of the structured walkthrough and the interactive debugger into a single, easy-to-use package. It simplifies some of the most tedious and time-consuming tasks in a programmer's day: learning other people's code so that enhancements can be made, tracking down bugs in production systems, and validating new programs. RTPA creates a line-by-line record of the executing source code and variable values, enabling the programmer to see exactly what is happening in the program.

