

Chapter V

Validation of Digital Forensics Tools

Philip Craiger, University of Central Florida, USA

Jeff Swauger, University of Central Florida, USA

Chris Marberry, University of Central Florida, USA

Connie Hendricks, University of Central Florida, USA

Abstract

An important result of the U.S. Supreme Courts Daubert decision is that the digital forensic tools must be validated if the results of examinations using those tools are to be introduced in court. With this audience in mind, our chapter describes important concepts in forensic tool validation along with alternative just-in-time tool validation method that may prove useful for those who do not have the capability of conducting extensive, in-depth forensic tool validation efforts. The audience for this chapter is the law enforcement agent and industry practitioner who does not have a solid theoretical background—from training or experience—in software validation, and who is typically time-constrained in the scope of their validation efforts.

Introduction

As with all other forensic disciplines, digital forensic techniques and tools must meet basic evidentiary and scientific standards to be allowed as evidence in legal proceedings. In the United States, the requirements for the admissibility of scientific evidence and

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

expert opinion were outlined in the precedent setting U.S. Supreme Court decision *Daubert vs. Merrell Dow Pharmaceuticals, Inc.*, 509 U.S. 579 (1993). The U.S. Supreme Court found that evidence or opinion derived from scientific or technical activities must come from methods that are proven to be "scientifically valid" to be admissible in a court of law. The term "scientifically valid" suggests that the tools and techniques are capable of being proven correct through empirical testing. In the context of digital forensics, this means that the tools and techniques used in the collection and analysis of digital evidence must be validated and proven to meet scientific standards.

Traditional software validation testing is performed as a routine part of any software development effort. Software validation has been well studied, and the basic tenets of a successful validation approach have been codified in numerous standards accepted by such international bodies as the IEEE. There are significant references and standards covering the role of validation testing during software development, as illustrated in the references to this chapter.

There is often some confusion between the terms validation and verification as applied to software testing. The definitions provided in "General Principles of Software Validation; Final Guidance for Industry and FDA Staff" (<http://www.fda.gov/cdrh/comp/guidance/938.html>):

- **Software verification** provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase. Software verification looks for consistency, completeness, and correctness of the software and its supporting documentation, as it is being developed, and provides support for a subsequent conclusion that software is validated. Software testing is one of many verification activities intended to confirm that software development output meets its input requirements. Other verification activities include various static and dynamic analyses, code and document inspections, walkthroughs, and other techniques.
- **Software validation** is a part of the design validation for a finished device...considers software validation to be 'confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled.' In practice, software validation activities may occur both during, as well as at the end of the software development life cycle to ensure that all requirements have been fulfilled. ...the validation of software typically includes evidence that all software requirements have been implemented correctly and completely and are traceable to system requirements. A conclusion that software is validated is highly dependent upon comprehensive software testing, inspections, analyses, and other verification tasks performed at each stage of the software development life cycle.

Validation of Digital Forensic Tools

If the developer or manufacturer validates software, one would presume that it should address requirements as specified in Daubert. The problem is that end users of the software rarely receive information as to the methods or results of the validation testing performed on the software. Consequently, end users are not capable of offering evidence or testimony in court to support the assumption that the software used in an investigation worked as intended. Some companies will provide representatives to give expert testimony about the validity of their software if required during a court case, but that is not something the average examiner—local or state law enforcement agent or industry practitioner—can rely upon or expect.

A good deal of forensic software is developed on an ad hoc basis, often by small labs or individuals who recognize a need and provide a product to address it. Because of its ad hoc nature the software tools often do not undergo extensive development testing or planning. This software is sometimes shared among practitioners, or provided to the public as open source software. Practitioners who will use this software in examinations will be required to perform their own validation testing in order to assure both themselves and the courts of the suitability of their tools and results.

Our experience suggests that most practitioners have little or no training or experience in software validation. Consequently, there are several practical matters that limit the law enforcement agents or industry practitioner's ability to perform validation at the same level rigor as the professional software engineer or developer. Foremost is that law enforcement agents and industry practitioners would need documentation tailored to their level of expertise in the field of digital forensics. Second is that in practice there are typically time constraints that limit the scope of the practitioners validation efforts to only a subset of the functions of the tool that will be used in the current examination.

In practice, there are few opportunities for digital forensic practitioners to conduct thorough validation tests. One reason is time: several law enforcement agencies, including, local, state, and federal agencies, have informed us of several months to years of backlogged cases involving digital forensic examinations, some as long as two years. Clearly need is outstripping production. Any process that will reduce the amount of time spent examining a system, while maintaining a high level of quality control, is advantageous to the forensic practitioner as well as to the judicial system. Below we describe a more efficient method of test validation that meets the pressing needs of forensic practitioners.

Limitations in Organized Digital Forensics Validation Efforts

The National Institute for Standards and Technology's (NIST) Computer Forensics Tool Testing (CFTT: www.cftt.nist.gov) division is one government entity that formally tests computer forensics software. CFTT performs extremely rigorous scientific tests to validate software tools used in digital forensic examinations. CFTT has and continues to perform testing on numerous computer forensic software applications, and has identified various problems that have been addressed by the software vendors. Unfor-

unately, the ability of one organization to examine all forensic software products and their variations is limited due to the sheer magnitude of the task (Craiger, Pollitt, & Swauger, in press).

The digital forensics community cannot rely on a single certifying body to test and evaluate all forensic software, as the sheer pace of change and number of software products is overwhelming (Craiger et al., in press). In addition, software tools written by forensic examiners—that are not commercial products—often provide additional functionality examiners find useful (such as EnCase scripts). Such software, unless it is widely distributed and used, will not rise to the attention of major validation organizations.

Validation Testing Methods

In the following sections, we describe two software validation methods that are appropriate for our practitioner audience: white- and black-box testing. These methods meet the needs of practitioners because: (a) they are simple yet effective methods that require little in-depth knowledge of software validation testing and (b) they are efficient in that they allow the examiner to quickly test only those functions that will be used in the current case.

White-Box Testing

White-box testing (WBT) involves examination of the source code on which the application is built as well as tests comparing the performance of the software against requirements. A requirement is a specification of something that the software must do. Requirements are developed during the software design requirements phase, one of the first phases in the software engineering process.

A formal examination of the source code is called a code walkthrough and has two major requirements. First, the source code on which the application is built must be available for review. Most commercial vendors are reluctant to make source code available to external reviewers due to intellectual property concerns. Thus, code walkthroughs conducted by parties external to a vendor are not common.

The second requirement is that team members conducting the walkthrough ideally consist of individuals with solid technical skills and expertise in two areas. Some members, such as programmers and software engineers, will have expertise in programming and software engineering. In addition, a code walkthrough requires participation by parties with domain knowledge of the tasks to be performed with the software. In the context of digital forensics this will include forensic experts with knowledge about media composition, file systems, forensic tasks, and so forth.

Code walkthroughs are sufficiently labor intensive—moderate size applications may contain hundreds of thousands or even millions of lines of code—which they may take months or even years to complete. Code walkthroughs, while thorough, are of limited use to members of the computer forensic community dealing with the rapidly changing software environment associated with digital evidence recovery tools.

Black-Box Testing

Black-box testing (BBT) evaluates software by comparing its actual behavior against expected behavior. Unlike WBT, BBT assumes nothing about the internal structure of the application (i.e., the source code). In BBT the software serves essentially as a "black box" and the performance of the application is evaluated against functional requirements.

In a digital forensics context, BBT is performed using a tool to perform forensics tasks under various conditions, such as; different file systems, various digital artifacts, different hardware, and various software parameters (switches and settings, etc.). The results of these tests across different conditions are compared against the software design requirements. If the tool performs as specified in the requirements then we have a level of confidence that the tool will work as expected under similar conditions. A positive outcome indicates we have validated the tool for the current task and conditions only; however, this confidence in the tool does not extend to conditions not covered in the test validation. For instance, a validation study may demonstrate that a tool passes a test for searching for non-fragmented ASCII encoded keywords. This result does not generalize to other text encodings, such as UNICODE, UTF-8 or even to ASCII text fragmented across non-contiguous clusters. Representations about a tool's capability only extend as far as the conditions covered during tool testing.

BBT can be performed more quickly than WBT because it does not include a code walkthrough; however, it can still be a time consuming process as a thorough validation test may include several dozens to hundreds of test scenarios, each of which includes different combinations of hardware, test media, and software parameters. In a typical thorough validation it is crucial to exercise a tool over its full range of user selectable parameters and against a number of different data sets or test samples. Although one or two tests may produce positive results, there can always be situations where the tool will fail, situations that are unusual enough to have not been tested or addressed by the designers. Some peculiar combination of set-up parameters, operating criteria, and so on, may reveal a hidden error (i.e., software bug) that, while rarely occurring, may invalidate a tool's functionality for a particular set combination or variables.

Just-in-Time Validation

Just-in-time validation is a testing methodology that involves testing software tools using only those parameters (file systems, file types, software switches, hardware, etc.) that will be used in the actual collection and/or analysis of the evidence. For instance, if a forensic examiners task is to use tool X to identify graphical images on an NTFS formatted volume, then the tool validation test should use only those parameters (file system=NTFS, file types=graphics, etc.) that duplicates the actual task. The set of parameters used in the test will be a subset of the total set of parameters available to be tested. However, only testing those conditions that are required at the time can save effort that would otherwise go into testing test scenarios that are irrelevant for the current case.

Just-in-time validation may be conducted using either a validated reference data source or using a comparative analysis, each of which we describe below. First we describe tool validation procedures as promoted by the Scientific Working Group on Digital Evidence that will serve as the basis for our tool tests.

SWGDE Guidelines for Validation Testing

The best source for guidance for digital forensic tool validation is from the Scientific Working Group for Digital Evidence (SWGDE) Recommended Guidelines for Validation (Scientific Working Group for Digital Evidence, 2004). SWGDE is composed of members from law enforcement (local, state, federal), industry, and academia whose goal is to create standards for digital evidence (www.swgde.org).

SWGDE's guidelines for validation testing describe the procedures one should follow in validating digital forensics software. The guidelines specify that tool validation includes creating a test plan, performing the tests specified in the test plan, and documenting the results. Below we will use SWGDE's guidelines to demonstrate just-in-time validation of a tool's capability for identifying and recovering deleted files on a floppy disk.

Using the SWGDE guidelines our first step is to develop our test plan. A test plan specifies the tool and its functionality to be tested, as well as how the tool will be tested. The test plan includes a description of the purpose and scope of the test, the requirements (tool functionality to be tested), a description of the testing methodology, the expected results, a description of the test scenarios, and a description of the test data.

In our example we will test tool X's capability to identify and recover deleted files, a very common forensic task. Our purpose and scope might be written as: "To validate tool X's capability to identify and recover deleted files on a FAT12 formatted floppy disk." Next we specify three requirements that the tool must exhibit:

1. The tool must be able to identify deleted files and mark them in an unambiguous fashion so that the examiner may differentiate deleted from non-deleted files.
2. The tool should be able to identify and display metadata for deleted files, to include the files size, modified, access, and created times.
3. The tool must be able to recover, and export, the logical contents of the deleted file to the host file system.

Based on the requirements we can then specify the expected results for the test:

1. The tool shall mark each deleted file to differentiate deleted from non-deleted files.
2. The tool shall display and unambiguously label the deleted files metadata.
3. The tool shall write the contents of the deleted file to the host file system using a unique name for each file recovered.

Table 1. Example test plan

Test #	Environment	Actions	Requirement	Expected Result
001	1. 1.4MB Floppy 2. FAT12 3. File A in directory A	Recover and Export Deleted File (logical file only)	Recover and Export Deleted File (logical)	1. Tool shall recover and export each file to the host file system. 2. Hash of the recovered file shall match the hash of the original file.

4. The hash of the recovered file shall match that of an original copy of the file. (This ensures that the file recovered is exactly the same as the original.)

Next we specify the test scenarios. A test scenario specifies the conditions under which the tool will be tested, as well as the pass/fail criteria for each test scenario. For instance, a test scenario for recovering deleted files might look something like Table 1.

Finally, we describe our test data. In this case our test media is a 1.4MB floppy disk, formatted with the FAT12 file system. Our digital artifacts (files) to be recovered include two sets of files: a non-deleted and a deleted version of File A (a small file < 1K), and a non-deleted and a deleted version of File B (a moderately sized file of ~ 60K). Our next step is to prepare the test media that will be used in our testing.

To ensure a scientifically rigorous test we must first sterilize our media to ensure no file remnants remain on the test media, which could bias our results. The test media preparation methodology occurs as follows:

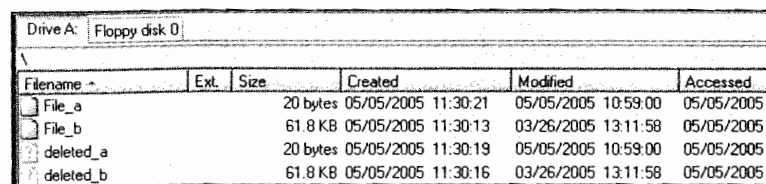
1. "Sterilize" the media by writing a series of characters over the entire writeable area of the media, from the first to the last sector. Typically, 0s (zeros) are written to the entire disk. (In our experience, using 0s make its easier to determine whether a complete sterilization of the media was accomplished). Sterilization is accomplished easily using Linux command line utilities (see Craiger, 2005). Most commercial tools provide this capability.
2. Format the floppy disk to create a file system. The choice is important as we wish to extrapolate from our test to the real-world media we will use. In this case, it is a FAT12 formatted floppy.
3. Copy our test files to the media.
4. Delete some of the files.
5. Write block the floppy to prevent from changing the contents inadvertently.
6. Create a *forensic duplicate* (exact copy) of the image. Again, Linux command line utilities may be used (Craiger, 2005), or any commercial tool that provides that capability.

7. Calculate a hash (e.g., MD5 or SHA-1) for both duplicate and original. These hash values should match.

Running the Test Scenarios

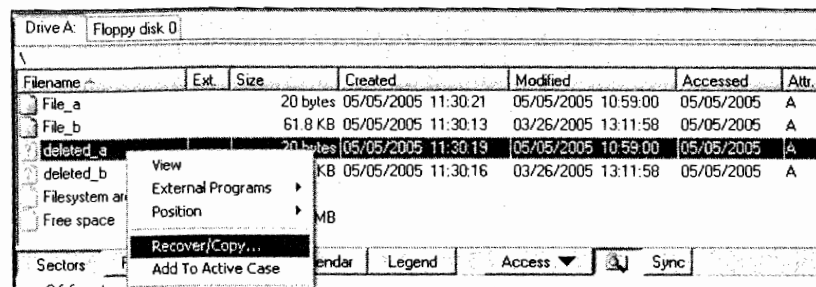
The forensic duplicate now constitutes our validated reference data source. We are now prepared to run the test according to our test plan. Figures 1 through 3 demonstrate a test scenario using X-Ways Forensics (www.x-ways.net or www.winhex.com) capability of identifying and recovering deleted files on a FAT12 floppy.

Figure 1. Unique marking of deleted files



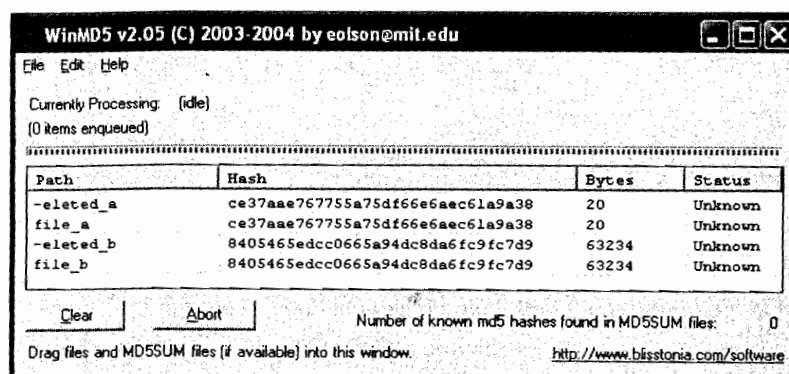
Filename	Ext.	Size	Created	Modified	Accessed
File_a		20 bytes	05/05/2005 11:30:21	05/05/2005 10:59:00	05/05/2005
File_b		61.8 KB	05/05/2005 11:30:13	03/26/2005 13:11:58	05/05/2005
deleted_a		20 bytes	05/05/2005 11:30:19	05/05/2005 10:59:00	05/05/2005
deleted_b		61.8 KB	05/05/2005 11:30:16	03/26/2005 13:11:58	05/05/2005

Figure 2. File recovery menu item



Filename	Ext.	Size	Created	Modified	Accessed	Attr.
File_a		20 bytes	05/05/2005 11:30:21	05/05/2005 10:59:00	05/05/2005	A
File_b		61.8 KB	05/05/2005 11:30:13	03/26/2005 13:11:58	05/05/2005	A
deleted_a		20 bytes	05/05/2005 11:30:19	05/05/2005 10:59:00	05/05/2005	A
deleted_b		61.8 KB	05/05/2005 11:30:16	03/26/2005 13:11:58	05/05/2005	A

Figure 3. Hashing of original and recovered files



Path	Hash	Bytes	Status
-eleted_a	ce37aae767755a75df66e6aec61a9a38	20	Unknown
file_a	ce37aae767755a75df66e6aec61a9a38	20	Unknown
-eleted_b	8405465edcc0665a94dc8da6fc9fc7d9	63234	Unknown
file_b	8405465edcc0665a94dc8da6fc9fc7d9	63234	Unknown

al. These hash

ce. We are now
demonstrate a
(com) capability

essed
05/2005
05/2005
05/2005
05/2005

essed	Attr.
05/2005	A
05/2005	A
05/2005	A
05/2005	A

Status	
Unknown	
Unknown	
Unknown	
Unknown	
files:	0
com/software	

orms without written

Figure 1 shows that we open our forensic duplicate in the tool and note that it displays our four known files on our validated reference data source. Figure 1 indicates that the tool unambiguously identifies deleted files using a "?". The tool thus passes our first requirement.

The next requirement specifies that the tool allow for the recovery of deleted files. Figure 2 demonstrates that the tool provides a menu selection to recover deleted files. We select the menu item, specify the location of the file, and the tool writes the recovered file to the host file system. Our tool thus passes the second requirement.

Requirement three is important as it determines whether the logical portion of the file was successfully recovered. To be forensically sound, the hash of the recovered files *must* match the hash of the original files. Figure 3 demonstrates that the original and deleted files are hashed using the MD5 cryptographic hash, a 128-bit hashing algorithm. Note that the deleted and original files hashes match, indicating that the tool successfully recovered the file, and thus, it passes the final requirement.

The results of our tests were consistent with the expected results, indicating a positive outcome of the tool validation test.

This example was a simple test of a tool's requirement for identifying and recovering deleted files using the SWGDE guidelines for tool testing using BBT. Next we discuss a second method of testing that can be performed without a validated reference data source.

Comparative Analysis

The example above illustrates the use of a validated reference data source (i.e., test media with known contents) to validate the functionality of a software tool using BBT. A second method, what we call a comparative analysis, is useful when a validated reference data source is either unavailable, or the creation of which would require a significant investment of time and resources that would imprudently delay the actual examination of the evidence. Note that comparative analysis also uses BBT as the test design method.

The key to a comparative analysis is to compare the results across multiple independent tools. Tools are independent in the sense that they are written by independent teams of programmers, and are usually from different commercial vendors or are open source alternatives. Versions 1.0 and 1.1 of the same tool would not constitute independent tools. If all three tools return the same result, then we have supporting evidence that the software functions as intended. We have a stronger claim for validation when one or more of the tools have been validated using a reference data set. For instance, if tools Y and Z were previously validated using a reference data set, then we have stronger evidence of validation when tool X produces the same results as tools Y and Z. The claim is weaker if only one of the other tools has been validated. If none of the other tools have been validated, the confidence is the weakest, although the fact those three tools created by three separate programming teams returned the same result can be interpreted as triangulating on those results.

Table 2. Comparing results of tools

	Find Keyword		
	UNICODE	ASCII	UTF-8
Tool X	Y	Y	Y
Tool Y	Y	Y	Y
Tool Z	Y	Y	Y

The actual testing procedure is the same as that described previously in the BBT section. A test plan is created, and then each of the tools is tested, following the test plan, and using the test media. After the tools have been tested, we compare the results for each tool as demonstrated in Table 2.

Table 2 illustrates the simple case where three tools are used to recover three files differing in the type of encoding (A = UNICODE, B = ASCII, C = UTF-8). Each of the tools successfully identified the keyword in the different encodings. The results suggest that we can have a measure of confidence in the three tools given that they triangulated on the same result. We have more confidence in our results if one of the tools had been validated previously using a validated reference data source.

What if the tools do not produce the same results? Reality may not be so clear-cut for the simple reason that even the best designed software will contain bugs (as demonstrated by the prevalence of service packs and interim patch releases one sees on a weekly basis). Below we discuss software errors and how to calculate error rates.

Metrics and Errors

There are several validation metrics against which software may be tested, two of the most common of which are performance (speed) and errors. Typically speed will not be of utmost importance to the forensic practitioner. For the digital forensics practitioner the most significant metric will be whether the software performed as expected, as measured by the error rate of the tool.

In the Daubert decision, known or potential rates of error, and error type should be considered when evaluating a scientific technique. Two statistical error types of interest are false positive (Type I) and false negative (Type II) errors. False positive errors occur when a tool falsely identifies a positive result when none is present. For instance, using a validated reference data source, Tool X identifies a file as deleted when in actuality it is not. False negative errors occur when a tool fails to identify results that are actually there. For instance, Tool X fails to identify a file as deleted when in actuality it is.

As an example, consider a forensic tool whose purpose is to scan digital media to detect .jpg graphic image files. The primary design requirement of the software, from a forensic point-of-view is to detect obfuscated jpg image files, for example, when a user changes

a jpg extension as a means of hiding the files true type. The tool works by scanning file headers and footers (i.e., the first and last few bytes of a file that determine the files real type) and comparing the files true type with its extension. Normally, the file header/footer and extension will match. However, a simple way of hiding a file is by changing its extension.

For test media we use a hard drive with 50 files, five of which are jpg files. Of these five jpg files, two have an extension other than a normal jpg file (.jpg or .jpeg). The hard drive constitutes our reference data set.

The tool's performance is evaluated by comparing the tool's results with the expected results: which is the tool's ability to detect extensions that do not match the files signature. One expected result is that the tool should identify all instances of .jpg image files, regardless of the extension, using header information. A second expected result is that the tool should not misidentify any non-jpg files as .jpg image files.

Table 3 shows the result of a test where the tool found all instances of jpg files on the hard disk.

In this example, the tool successfully passed the test by (a) detecting all instances of the jpg images, both with and without the correct extensions, and (b) not identifying non-jpg files as jpg files. Out of the 50 files on the test hard drive, all 50 were correctly identified by type. In this case, the tool has proven 100% accurate (correctly identified divided by the total number) with an error rate of 0%.

Now let us consider the case where the tool missed several jpg files as illustrated in Table 4. In this example, the tool failed to detect some jpg files on the test hard drive. Of the 50 files, only 48 were correctly identified. In this case, the tool has displayed an accuracy of 96 percent and displayed two false-negative, or Type II, errors.

Table 3. Search results for JPG detection tool

Known JPG Files	Tool X Discovered JPG Files
Test1.jpg	Test1.jpg
Booty.jpg	Booty.jpg
Hidden.txt*	Hidden.txt
Byebye.zip*	Byebye.zip
Test2.jpg	Test2.jpg

Table 4. Search results for JPG detection tool

JPG File	List of Discovered JPG Files
Test1.jpg	Test1.jpg
Booty.jpg	Booty.jpg
Hidden.txt*	(FAIL)
Byebye.zip*	(FAIL)
Test2.jpg	Test2.jpg

Table 5. Search results for JPG detection tool

JPG Files	List of Discovered JPG Files
Test1.jpg	Test1.jpg
Booty.jpg	Booty.jpg
Hidden.txt*	Hidden.txt
Byebye.zip*	Byebye.zip
Test2.jpg	Test2.jpg
	Document.doc (FAIL)

Now, let us consider the results as seen in Table 5. In this example, the tool successfully identified all jpg format files on the test hard drive, however it misidentified a Microsoft Word file (with the .doc extension) as a jpg image. In this case, the tool correctly identified 49 of the 50 files, resulting in a correct score of 98%, or a 2% error rate, and returned a false-positive, or Type I, error.

(Although not relevant for just-in-time validation, full-blown validation tests would include the above test run using other test media in order to generate more reliable statistics. For example, if the tool was run three times with the results as indicated above, the average accuracy would be $(100 + 96 + 98)/3$, or 98%, with a standard deviation of 2 (2%) and displayed both Type I and Type II errors. The larger the number of test samples, or the larger the number of relevant data in the test sample, and the more times the tool is tested against different test media, the higher the confidence in the results of the test.)

Identifying Error Causes for Validation Testing

When a test of a software application results in test failures the most important task is to attempt to determine the cause of the test failure. In the examples above, the bit patterns of the files that were not correctly identified should be examined, and their location relative to disk sector or cluster boundaries should be reviewed. It could be that the tool is coded in such a way that it is not looking at the entire header or footer field, or has a coding error that allows it to misread the header, footer, or extension information. In addition, it may be possible that the tool has a problem accurately identifying these fields if they lay across cluster/sector boundaries, or if they lie in non-contiguous clusters. In the example used in this chapter above, further testing and analysis of the test hard disk should be performed to determine if any identifiable cause for the failures could be found. Further testing based on this and other scenarios should be performed to gather further data.

It should be noted that a limited number of failures does not necessarily completely discredit the use of the test tool software. The failure needs to be interpreted with respect to both the entirety of the test results and the nature of the failures. Depending on the manner in which the tool is to be used, a certain error rate may be acceptable if that error rate and the error types are known and taken into account in the analysis of the data recovered.

Test Scenarios

Note that just-in-time validation is efficient because of the judicious selection of test scenarios. Just-in-time validation only includes test scenarios that are immediately relevant to the current task. Contrast this with full-blown validation testing, the purpose of which is to draw inferences about an entire population of tasks, some of which include highly improbable boundary conditions.

Selecting or creating test scenarios for full-blown validation testing is one of the most challenging aspects of validation testing. The set of test scenarios should consist of a number of heterogeneous examples that duplicate conditions that will be found in real world forensic tasks. In addition to common types of data, the test scenarios must include boundary cases. Boundary cases are conditions or examples of things the tool must be capable of detecting even if they are rarely found in most situations. Recovering a 100GB file is an example of a boundary condition for the task of recovering a deleted file. If the tool correctly reports the results from real-world examples as well as boundary cases, then we can say with some authority that the software functions as expected.

A test scenario would ideally include test media containing a complete set of variables and data to thoroughly exercise the application. The advantage of running the tool against a known standard is that the results are known a priori given that the examiner knows what exists on the test media. The disadvantage is the time and effort to create the test media, which can be extensive, and the potential lack of knowledge about the range of variables that can exist. For example, consider the case of a test of a simple keyword extraction software package, which searches a hard disk for the presence of a keyword or keywords. To perform even a moderately extensive test of this application, the following conditions must be tested, with corresponding test cases produced: (1) five different HD sizes must be used that fall within the traditional hard disk size boundaries; (2) each drive must be presented with both the default and non-default cluster/sector size; (3) the disks must be partitioned in a variety of common formats (FAT 32, FAT 16, NTFS, and EXT3); (4) the keyword(s) that are to be searched for should be present in various formats, including at a minimum: Unicode, ASCII, UTF-7, UTF-8, and RTL; (5) the keyword(s) to be searched for should be placed on the disk in such a way that various locations relative to the physical clusters are presented for test, in other words, lying entirely within one cluster, and crossing cluster boundaries for both the contiguous and non-contiguous cluster cases; (6) and the keyword(s) that are to be searched for should be placed on the hard disk embedded in other characters with no leading or trailing spaces, embedded in other characters but with one leading and trailing space (e.g., null character), and alone with no leading or trailing characters.

This is only a partial, although fairly comprehensive, approach to performing a validation test of a keyword search and extraction algorithm/software package. Certainly additional encodings and other disk partitioning and cluster sizes could be tested. In addition, to more fully test the software, a wide variety of different keywords could be tested, as the algorithm may always find a specific combination of characters that it might not detect (though one can carry this to extremes if one is pedantic enough). As it is, even testing for only one keyword using the above approach, 1800 different individual test cases must be prepared, and if only one partition type is used on each hard disk, 20 hard drives must

be prepared for testing. This represents a significant amount of time and effort for both test preparation as well as test performance, with test preparation taking significantly more time than it takes to perform the test.

In the creation of test scenarios for computer forensic applications, this requires that an expert with extensive knowledge of both computer hardware and operating system standards is involved with the test scenario and test media creation. This expertise is required to ensure that the test scenario and media does not overlook important conditions or data that would diminish the validation tests thoroughness.

Conclusions

The Daubert decision will continue to have a major impact on the practice of computer forensics practice as courts and litigants become more technically savvy. The case will also serve to modify expectations of scientific testing of computer forensics tools used to create evidence in these court cases. The thrust of this chapter was to provide an overview of tool validation for digital forensics examiners with limited training and experience in tool testing. Unfortunately, there is very little literature that directly and specifically addresses digital forensic tool validation. The best sources are the SWGDE Guidelines (2004) and documents at National Institutes for Standards and Testing Computer Forensic Tool Testing site (www.cftt.nist.gov).

As the number of forensic software applications continues to increase, and the environment that the tools must operate in continually evolves with the development of new operating systems and computer applications, traditional, intensive software validation will continue to be unable to keep pace with the requirements of the forensic community for validated software. Individual forensic practitioners, as well as major labs and accrediting bodies, must be capable of performing validation of tools to some degree of rigor if the results of such tools are to continue to be accepted as evidence in legal proceedings. The approaches presented in this chapter, when applied with due diligence and documentation, will be called upon more and more to provide the required validation and assurance that forensic software applications perform as required.

References

- Center for Biologics Evaluation and Research, U.S. Food and Drug Administration. U.S. (2002). *General principles of software validation; Final guidance for industry and FDA staff*. Retrieved from <http://www.fda.gov/cdrh/comp/guidance/938.html>
- Craiger, J. (in press). Computer forensics procedures and methods. To appear in H. Bidgoli (Ed.), *Handbook of Information Security, Volume III*. New York: John Wiley & Sons.

effort for both
significantly

quires that an
rating system
is expertise is
book important
ss.

ce of computer
y. The case will
sics tools used
s to provide an
ed training and
that directly and
are the SWGDE
ds and Testing

and the environ-
lopment of new
tware validation
nsic community
major labs and
o some degree of
vidence in legal
ith due diligence
quired validation
1.

ministration. U.S.
ance for industry
aidance/938.html

To appear in H.
New York: John

Craiger, J., Pollitt, M., & Swauger, J. (in press). Digital evidence and law enforcement. To appear in H. Bidgoli (Ed.), *Handbook of Information Security, Volume III*. New York: John Wiley & Sons.

IEEE Computer Society. (2004). IEEE 1012 Software Verification and Validation Plans. Retrieved from <http://standards.ieee.org/reading/ieee/std/se/1012-2004.pdf>

IEEE Standards Association. (1993). IEEE 1059 Guide for Software Verification and Validation Plans. Retrieved from http://standards.ieee.org/reading/ieee/std_public/description/se/1059-1993_desc.html

IEEE Standards Association. (1997). IEEE 1074 Standard for Developing Software Life Cycle Processes. Retrieved from http://standards.ieee.org/reading/ieee/std_public/description/se/1074-1997_desc.html

Scientific Working Group for Digital Evidence. (2004). Recommended Guidelines for Validation Testing. Retrieved from www.swgde.org

ms without written