Toward Defining the Intersection of Forensics and Information Technology

Gregory A. Hall
Wilbon P. Davis
Texas State University-San Marcos

Abstract

Information technology has become infused with many aspects of everyday life. Naturally, aspects of information technology have become more common in both civil and criminal law. However, the people called upon to address these technological issues are typically versed exclusively in either law or technology. The authors contend that a new discipline is emerging from the intersection of forensics and information technology. They argue for a formalization of the process by which this discipline evolves and a broadening of the definitions currently in place. A case study is presented to illustrate the shortcomings of current, narrowly defined terms and the need for individuals with broader expertise. The authors conclude by proposing some of these areas of expertise.

Introduction

In Section I, the authors call for a concerted effort to produce a working taxonomy of the fields comprising what is known to many as digital forensics. In Section II, they discuss the mechanism by which disciplines often come to be defined. Section III projects this mechanism to a likely evolution of the taxonomy under discussion, and in Section IV a case study that crystallizes the issues involved and gives a concrete example for the need for a taxonomy is presented. Section V summarizes the arguments supporting the goals outlined in Section I.

Section I

Although there are myriad definitions of digital forensics, network forensics, software forensics, computer forensics, etc., each is a sub-discipline of forensics, that is, "The use of science and technology to investigate and establish facts in criminal or civil courts of law" (American Heritage Dictionary of the English Language, 2000). However, the connection of these sub-disciplines with computing appears in two ways: as investigative tools and as evidence. The authors contend that it is time to begin defining an umbrella discipline, a branch of knowledge, incorporating the listed areas, and others, as sub-disciplines, while acknowledging the limitation of the scope to information technology connections. It is also time to begin formulating and standardizing definitions. While the authors believe that it is premature to impose such definitions or

even the choice of terms, for the purposes of this discussion, the phrase "information systems etiology[1]" (ISE) will be used to refer to this umbrella discipline. Note that the definition of digital forensics adopted in Caloyannides (2001) had abstraction and unification as a goal; however, ISE will be much more inclusive.

In academia, physics, chemistry, and biology are considered to be sub-disciplines of science. Loosely speaking, their association is a result of the recognition of the commonalities in their methods, in particular, the scientific method, while their distinctions from one another result from the problems they address. Their organization as well as their respective sub-disciplines(cosmology, biochemistry, genetics) and other disciplines, such as mathematics, forms a hierarchy essential to their study. This hierarchy also is essential to accessing the knowledge the disciplines embody, just as the taxonomy of biology is essential to guiding its research. The establishment of such a classification scheme and relevant definitions might be a boon to research in ISE.

The above definition of forensics consists of two parts. The first refers to the use of science and technology in investigation and the second to the requirement that facts established must be presentable in a court of law. We propose both a generalization and specialization of this definition to constitute a definition of ISE.

The specialization of forensics to ISE comes from analyzing the characteristics of the definitions of disciplines and from the definition of forensics. First, definitions of disciplines imply criteria that enable a person to determine if a methodology or problem belongs to the discipline. Inclusion in ISE requires an element of investigation. Second, these definitions allow the distinction of disciplines. ISE is distinguished from other sub-disciplines of forensics by the involvement of an element of information technology as either a tool or an element of investigation.

The generalization portion of modifying the definition of forensics to define ISE removes the emphasis on admissibility in court of all findings. The objective of an investigation of a digital incident often is not a prosecution; rather, it may be to determine a root cause leading to recurrence prevention or process improvement (Stephenson, 2003 & 2004). And, relaxing the strict legal requirement on the definition may lead to the adoption of methods invented for law enforcement for broader domains as described by Ciardhuáin (2004).

Agreement on a definition of ISE will promote cooperative research among sub-disciplines, provide accessibility to results, and provide a classification hierarchy that will provide guidance for educators in defining a common body of knowledge to support a variety of sub-disciplines. A natural result of the hierarchy will also be the definition of a skill set and knowledge base enabling possessors of same to coordinate and manage ISE activities across a broad range of application domains.

[1]The definition of etiology we choose is "the study of causation" to emphasize the investigative component.

Section II Evolving a Definition for the Discipline

Examining the evolution of a specific discipline is enlightening. Software Engineering has its roots in computer programming. Very early, the software development industry faced problems related to the quality of software. Some of the first problems were associated with the mental burden that expressing computer instructions in low-level programming languages imposed on programmers and which caused them commonly to make mistakes when coding algorithms. The creation of higher level programming languages allowed programmers to operate at a higher level of abstraction, shifting more of the burden of translating an algorithm into computer instructions onto the computer. However the introduction of higher level programming languages did not completely solve the problem of poor quality software. Researchers then turned their attentions toward the design or architecture of software applications. Poorly structured software suffered from low reliability, poor efficiency, and high maintenance costs. New design methodologies emerged to improve the internal structure of the systems. Improving the quality of the software's design brought about larger improvements in software quality than that brought on by the shift to high level languages. However, significant problems still persisted in software development. Eventually new mechanisms for eliciting and specifying the requirements of software systems were developed. This addressed the fact that it was entirely possible to craft a well designed and brilliantly implemented solution to the wrong problem. At every stage, the next substantive improvement came from the recognition of commonalities in a set of problems, the abstraction of techniques developed to solve them, and a broadening of the definition of Software Engineering.

There is still considerable ongoing discussion concerning the body of knowledge that defines Software Engineering. This debate is inevitable in light of the *ad hoc* manner in which the discipline came about. It began as a group of independent researchers each focusing on a low level problem that had to have an immediate solution. Solving the low level problem made the higher level problems visible. Research then went after the higher level problems. The initial focus on low level "symptoms" caused many researchers to initially define this new discipline in terms of their specific sub-disciplines.

One of the earliest definitions for the discipline of software engineering came from F. L. Bauer in 1972. The definition emphasizes the creation of computer programs.

> Software Engineering deals with the establishment of sound engineering principles and methods in order to economically obtain software that is reliable and works on real machines (Bauer, 1972).

Bauer's definition emphasizes the economics, reliability and practicality of computer programs. One of the driving forces in establishing software engineering as an engineering discipline was the high cost of creating computer programs. Lack of disciplined development processes led to cost over-runs and unreliable software. The stipulation that the software work on real machines distinguished software engineering from theoretical computer science. Real world constraints had to be considered.

While Bauer's definition addressed the most obvious critical issues facing the software development community, it did not cover all of the bases. While the ultimate product of software development is a computer program, there are additional products that drive the engineering process and relate directly to the final quality of the computer program. In 1976, B. W. Boehm extended the definition of software engineering to include these additional development artifacts.

> Software Engineering is the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them. It is also known as software development or software production (Boehm, 1976).

The main contribution of Boehm's definition of software engineering is that it has been extended to include the documentation associated with the computer program. It is the documentation that encapsulates what the program will do and how it will do it. In a sense, the computer program is merely an effect of this documentation. While the computer program may contain a fault in its coding, it is the documentation (or lack of documentation) that led to that fault. Focusing solely on the computer program in cases of potential negligence will almost always miss the true source of the problem.

Boehm's definition also addresses the fact that computer programs have to be maintained. Maintenance activities involve such things as correcting faults, adapting programs to new hardware and operating systems, adding new features, and even preventing potential problems.

A standard definition for what software is and what software engineering means was provided by the IEEE in the IEEE Standard Glossary of Software Engineering Terminology in 1990.

> Software. Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.
>
> Software Engineering. (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

The IEEE definition of software establishes it as more than just a computer program. Software also refers to the documentation and data for the program. The IEEE definition for software engineering adds the notion that the development, operation and maintenance of software needs to be quantifiable. The ability to measure and validate software engineering processes and products is what establishes software engineering as an engineering discipline.

Even in light of a standard definition for the field of software engineering, the duties of a software engineer are still vast and varied. Some "software engineers" are

programmers, others are system architects, still others focus on requirements gathering and specification. Some software engineers work entirely in management or leadership roles with no direct contact with the technical aspects of development. Indeed, the field is so broad that the most accurate definition may well be that software engineering is "what software engineers do." What is most important is that a software engineer is aware of the many facets and interrelationships of the sub-disciplines that comprise software engineering and that the software engineer knows how to access the literature of these sub-disciplines as needed. There is still need for people with very specific expertise, such as writing device drivers in assembly language, but software engineers are needed to oversee the construction of the final product to ensure success.

The boundaries of a new discipline are often hard to define. The commonalities among research areas help to determine that a new discipline is forming, but just how many of these research areas there are and whether or not they belong within the context of the new discipline is often unclear. As Software Engineering grew, several research areas became a part of it. Among these were requirements analysis, software design, human factors, programming languages, algorithms, testing, software measurement, project management, and software security. Many of these research areas are heavily related to other disciplines. In fact, as more was learned about the need for security in computer systems, software security came out from under the umbrella of software engineering and became part of the larger discipline of information assurance. Basically then, the discipline grows to fill a void, with its boundaries defined when it meets the boundary of another discipline. In some cases, a sub-discipline may have found a home in another discipline, simply because it was "ahead of its time." When the real home for that sub-discipline is constructed, it moves to where it is most comfortable.

Historically, the formation of a new discipline has been a slow process that happens in a very ad hoc manner. Only after years of growing pains does the discipline become defined to the level that real progress in research can be made. Without formalizing the process, multiple factions arise, each of which claims to be defining the true discipline. Establishing a widely agreed upon common body of knowledge for the discipline becomes a political battle, with various sides arguing for their specific process (or product) to be a part of it.

It seems clear that a new discipline is forming and is currently defined in terms of the specific problems that need to be addressed. The authors believe the term "information technology etiology" is descriptive, but what the discipline will eventually be called is still not clear. Many different terms related to the area have been used in the literature to date. Some of these terms and their various proposed definitions will be examined. The implications of these definitions will be discussed. If steps are taken now to formalize the process by which the definition of the discipline evolves, many of the growing pains which other disciplines have had to endure may be avoided or at least lessened to a substantial degree.

In summary, some of the lessons learned from the evolution of Software Engineering applicable to ISE include:

- The discipline must acknowledge real world constraints,
- Abstraction is a powerful tool and should be foremost in the minds of researchers,
- The equivalent of the maintenance process must be addressed, that is, methodologies must be codified, standards must be set, and processes must be improved.
- The ISE discipline will evolve naturally and achieve a form of stasis but we can greatly influence the rate of progress toward that goal.

Section III Toward a broader definition

An apt analogy providing an inspiration for this proposal appeared when the authors posed to themselves the question "What would be the current state of Software Engineering if there had been an insistence that all work products would have to meet published military standards?" The field might well have failed to discover and incorporate the breadth of tools available to it and the contributions of software engineering research to a variety of fields may never have materialized. Similarly, the restriction of the term "forensics" to pursuits involving the courts may prove equally stifling. However, legal admissibility of investigative artifacts should not be abandoned; rather, it should become a differentiating characteristic among lower level subfields of ISE and should be considered a "gold standard."

A broader definition should lead to significant synergy with a variety of other fields. For example, a cursory sample of methods to be exported to a broader domain outside the legal system might include:

- Interrogation and testimony skills;
- Chain of custody formalisms;
- Data recovery techniques;
- Investigation techniques providing input to process improvement; and
- Investigation techniques providing input driving security research.

Currently, techniques from other fields are incorporated in an *ad hoc* manner into computing related forensics. But, with study and understanding of the broader picture, higher level techniques such as modeling and analysis from software engineering may be brought to bear on outstanding forensics problems in a rigorous manner.

Ultimately, the skill set of its practitioners defines the application domain of a discipline. A broader definition should invite the participation of computer scientists, officials of criminal and civil courts, law enforcers, lawyers, psychologists, forensic accountants, sociologists, educators and many other professions, all of whom would bring a rich set

of skills, varied viewpoints, and a collection of problems to the forum. The benefits of a broader definition are many, while the costs are few.

Section IV A Case Study

Drawing on the example of Software Engineering, the authors contend that the evolution of a new discipline can occur more smoothly if its emergence is recognized early and steps are taken to formalize the process and to establish the boundaries of that new discipline. If the process is left to occur in an unstructured manner, commercial and political pressures can hinder and misdirect the growth of the discipline.

The authors of this paper were contacted to provide expert assistance in a civil lawsuit. In conducting research into the current state-of-the-practice in digital forensics and digital evidence, it became apparent that there is a need for a broader approach to the problems of this discipline. The case study of that particular law suit is presented here to justify broadening the scope of digital forensics to a more inclusive ISE.

(In the following narrative, names, dates, and locations which do not contribute to the understanding of the fact have been withheld.)

A road contractor was using a piece of heavy, self-propelled machinery. An operator had the machine in "park." He placed a control in neutral and released the parking brake. The machine immediately leapt forward causing life threatening injuries and permanent disability to a man standing several feet in front of the machine.

The authors were contacted by the attorney of the victim to help sort out what had happened. They soon became acquainted with the actions of many players in the incident, referred to here as company C, the contractor, company S, the seller of the machine, company M, the manufacturer, company H, the designer and supplier of the hydraulic systems, and P, the programmer of the control system.

The class of equipment was capable of relatively fast speeds, was very large and powerful, and required precision in some of its functions. Company M and its competitors realized that computer control would allow for greater precision and operating convenience at a decreased cost, so company M decided to introduce its first "fly by wire" model. This model was introduced and dealer incentives were used to speed its introduction into the market place.

A mechanical engineer consultant to the lawyer had identified the most likely location for system failure and had been able to reproduce the anomalous behavior. M had chosen to use a hydrostatic drive. Briefly, in this system an engine drives a pump which circulates hydraulic fluid. This fluid can be used to drive motors. The engine can run at a constant, optimum speed, while the motors can be controlled electronically by using servos to tilt an internal plate. Generally, there are two controls involved in such a drive

train: an engine throttle and a control that governs the rotational speed and torque of the motors as they respond to the circulating fluid. Although this latter control is continuous, it is often labeled like a gear shift lever; that is, forward, neutral, and reverse. This lever be called “velocity.” One characteristic of this system is that there is little slippage. If the motors stop, there is no “coasting.” Because wheel skid is a disaster for a new road surface on still molten tar, in a digital control system, the controller senses a new position of the velocity setting and then gradually changes the machine's velocity to the desired value by ramping up or down the response of the motors to the circulating fluid over an appropriate time period.

In this particular case, the operator from company C needed to clean part of the mechanism which could only be opened via the controller. The controller would open it only when the machine was moving forward. There was a switch marked “park” on the operator's console which blocked the signal from the control system to the motors. The operator discovered that placing the velocity lever in a forward position with the throttle set to a high RPM while activating the park switch allowed the controller to open the device needed. In this state, the hydraulic fluid was circulating rapidly, but the motors were insensitive to its flow. Coincidently, a mechanical drum brake was also set by the system in response to the park switch.

After completing the cleaning operation, the operator placed the velocity lever in the neutral position and released the parking brake by pressing the “park” switch as he prepared to back up. To the surprise of everyone, the machine lurched forward and, tragically, caused serious injury.

The problem was that when the parking brake was released, the control system was still applying its signal to the wire leading to the motors and, although that signal was interrupted by the parking switch, it was still being “ramped down.” Because of the short time between repositioning the velocity lever and reapplying the signal to the motors, the motors were almost instantaneously set to a high RPM response. The machine leapt forward.

This was a rare, but instructive, instance of extreme division of responsibility in development. Couching the description in terms of an elementary software life cycle model, company M had an engineering staff with little experience in digital control. They built a requirements document addressing issues such as steel strength and wheel sizes, but they left design of the digital control of movement and functionalities to their supplier of hydraulic systems, company H. In turn, company H designed the hydraulic controls down to the control system level and passed this design on to programmer P for implementation.

The root cause of the accident was the failure of the design to take into account the parking brake switch. The fact that the interruption of the ramp down signal caused the vehicle to move forward while in neutral was determined by digital control systems consultants with whom the victim’s attorney had conferred. The question the attorney posed to the authors was “Had the vehicle been tested sufficiently and should this

defect have been found during testing?" To attempt to answer this question, it was necessary to begin at the source code level.

Our first request was for a copy of the source code to the control program and for system diagrams. While the wiring diagrams of the machine were delivered by M, the code was a different matter. An agreement over the ownership of the source code between M and H delayed our access. The lawyers and the court seemed woefully ignorant of the nature of our request. At first, unbeknownst to us, the attorneys agreed to deliver a printed copy of the code. The one spot of humor in the situation arose when the attorney advised us that the defendant's attorney had printed several reams of paper containing illegible characters. Of course, they had been trying to print the executable binary.

After more negotiation and the signing of nondisclosure agreements, the source code was delivered on CD. This source code had source snapshots of versions both older and newer than the installed version; the specific version of interest had been lost. The authors were assured, and of necessity assumed, that there had been no changes to the relevant portion. After a few days of analysis aided by standard Unix utilities such as cflow, the critical section that provided the ramping of the signal to the motors was identified. At this point, the culpable party appeared to be the programmer, and if we had had to rely solely on evidence existing in digital form, this would have been the only conclusion we could have reached.

However, following Nancy Leveson's dictum that safety considerations must be part of the requirements phase of the development cycle and knowing that requirements are a prerequisite to a test plan, we asked for the requirements documents that company M had produced and which had directed H in their design. After defining such documents to a specificity acceptable to the courts and the attorneys, we learned (to our complete surprise) that no such documents existed. The only communication between M and H had been a series of emails relating to minor details.

The next step was to request the design document delivered by H to the programmer P. Two were received. The first, internal to H, was a standard control diagram indicating the routing and conditioning of signals and referring only to the hydraulic components of the machine. The second, which was shared with the programmer, consisted solely of required voltage ranges and other values read from a test harness in response to a variety of inputs into an evaluation board. In other words, the programmer had no idea of the specifics of the application.

At this point, the programmer was exonerated, save for his lack of judgment in failing to refuse a job with such weak direction.

The only parts of the development cycle left to investigate were the deployment and testing portions. Documentation and training are significant parts of deployment. In the documentation relating to the machine, the procedure followed by the operator is not

addressed. Training was to be supplied by the seller S, but was informal at best and did not address safety issues.

Students learn that a requirement is valid only if it is testable and that a test plan is an essential part of the specification writing process. So, testing related materials were defined to the court and attorneys and requested from all parties. We received from H and P a document specifying voltage and timing responses required when measured from the hardware test harness. M supplied no testing documents other than ground speed performance reports; H supplied only what was given to P.

When challenged, M claimed that their engineers had run the machine in their parking lot. They also claimed that they had an organized field testing program in which initial customers were compensated with discounts for evaluating the machine's performance in the field. No specific example reports were forthcoming.

At this point, reading depositions from M's engineering staff led to another discovery. The most pressing question was how the potentially disastrous interaction of the parking switch and the velocity lever slipped by everyone. The simple fact was that the parking switch was a holdover from an old analog design and M decided to add it as a safety precaution against the velocity lever being accidentally bumped. Although immaterial to the current discussion, M insisted that H had been informed of the modification while H denied any knowledge of the switch. The official medium of communication would normally have been the requirements document.

It should be clear from the description of the case that sound engineering principles were overlooked at several stages of development. For the purposes of the lawsuit, the authors had to describe what the industry standard practices that should have been known by the defendants and that were not followed. Figure 1 depicts what was considered to be the minimum set of such practices with respect to the specification and testing of the product.

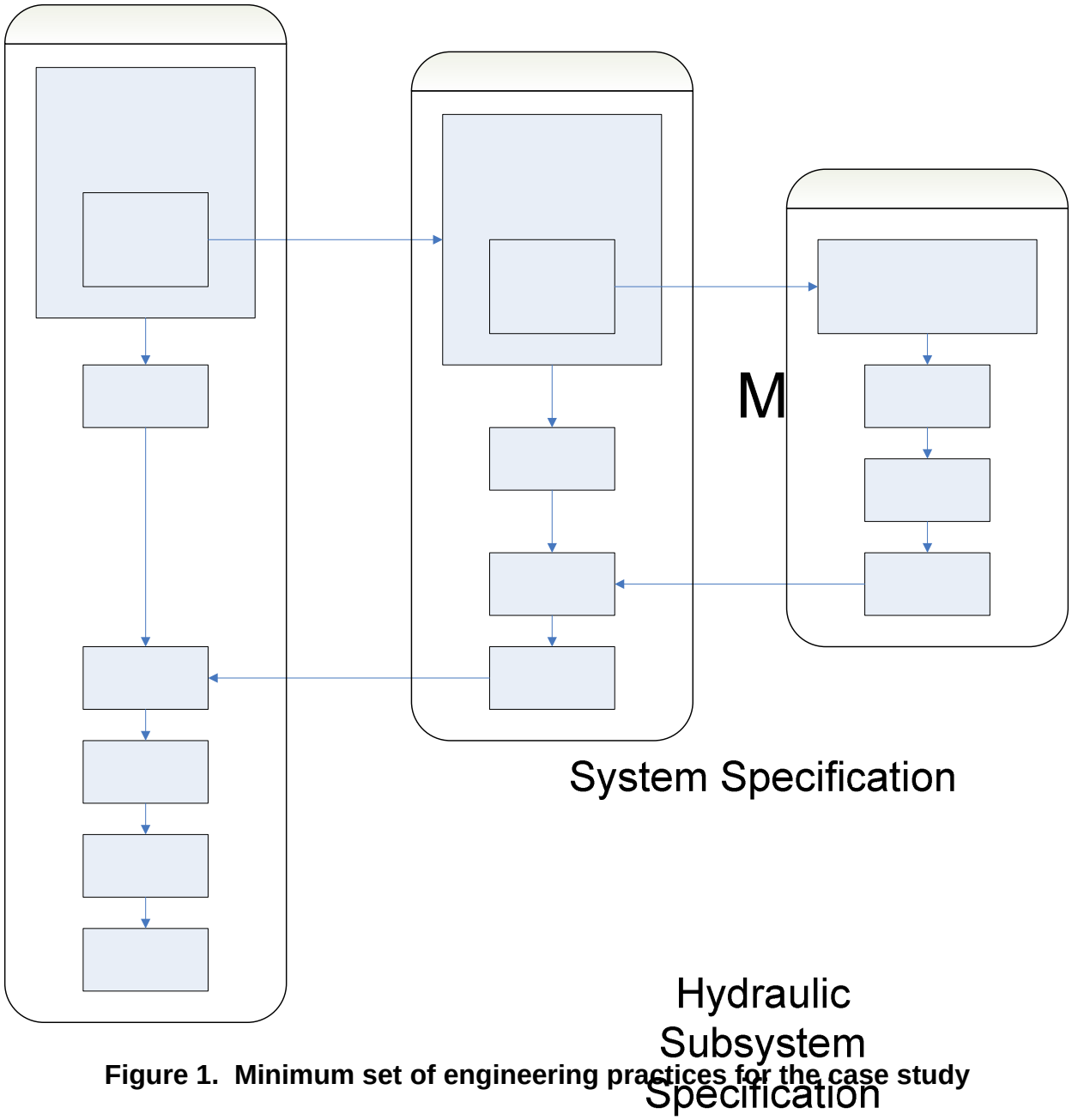

Figure 1. Minimum set of engineering practices for the case study

The first step in the development process should have been the formulation of a detailed system specification for the overall product. Because the hydraulics system was to be subcontracted to a third party, that portion of the system should have been specified in terms of its overall requirements and its interface to other parts of the system. The hydraulics subsystem specification would have represented a contract between M and H and, as such, would have provided the conditions required to satisfy that contract and would have manifested in the form of acceptance test criteria for the hydraulics subsystem. The system specification would have been the input to an overall system test plan. System integration testing would assemble all of the subsystems, testing their interactions. Overall system tests would have been conducted

to verify the system's conformance to the system's specification, under both normal and exceptional conditions. Field testing would have been the final stage, in which the final product was fine tuned, based on feedback from actual use in the field. Given the inherently potential dangers in such machinery, it was reasonable to expect sections of both the requirements and the test plan to address safety.

For H, they would reasonably have been expected to design a hydraulics system that conformed to the requirements provided by M. Because H intended to subcontract the development of the control system software to run the hydraulics subsystem, a software requirements specification for the control system would have been developed to serve as the contract between H and P. As stated before, a part of this contract would stipulate the acceptance testing criteria for the control system software. H would also have been expected to develop a test plan for the hydraulic subsystem.

The programming company, P, would reasonably be expected to create a design based on the specification provided by H. That design would be translated into a source code implementation. That implementation would then be tested to ensure that each piece of code worked as intended (unit test) and that the pieces of code worked correctly when interacting with each other (integration test).

The root cause of the defect that led to the victim's injury was the presence of the parking brake switch of which H claimed to have had no knowledge. No documented specification for any part of the system was ever produced. Such documentation would necessarily have addressed the systems with which the hydraulics subsystem would have to interoperate. The existence of the switch would then have been included in the design of the hydraulic subsystem and referenced in the control system specification. No software design was produced by the programmer, but had one been created from a complete specification, it would have addressed interactions of the software with the switch.

Once the software was constructed, it would have been tested according to documented test plans, created using complete specification documents at each stage. In the absence of specifications, all testing efforts were ad hoc and woefully inadequate. The programmer had no test plan, no documented test cases, no record of any defects found. The hydraulic subsystem was tested without a test plan and no acceptance testing was documented for the code received from the programmer. No system test plan was produced nor was hydraulic subsystem acceptance testing performed. System testing consisted of driving the vehicle through a parking lot "obstacle course" consisting of hills and curves in order to tune the vehicle so it operated more smoothly. No testing was conducted for potential safety hazards or anomalous behaviors. Field testing consisted of a crew observing the vehicle under normal use on a job site.

Several process failures occurred that let the initial oversight of the parking brake switch go undetected. It was the authors' expertise in software engineering that eventually revealed where the real problem originated. However, the authors did not have the background expertise in control systems that originally identified the interaction of the

signal ramp down with the parking brake switch. The lawyers involved did not have any understanding of the software development process or other technical issues. Had there been someone with background in each of these areas (not necessarily expertise), it would have greatly facilitated the investigation into what really led to the defective operation of the machine. For example, even with expertise in software engineering, it was very difficult to describe technical documents in appropriate legal terminology during the discovery process.

When the authors returned to the victim's lawyer to answer his question as to whether or not the system was tested adequately, the lawyer was initially disappointed with the response that his question could not be answered. His mood changed, of course, when the reason was revealed.

Clearly, none of the companies involved in the production of this large, powerful and ultimately dangerous piece of equipment observed industry standard practices for the engineering of machine control systems. While the root cause of the problem was inadequate specification of the product at the highest level, the effects of the specification oversight permeated the entire development process.

While the hydraulics system of the machine was computer controlled, it was software that governed the computer controller. The authors, therefore, considered the investigative work done to acquire evidence about this system to be an aspect of "software forensics." Table 1 provides four definitions acquired from the literature on software forensics.

Table 1. Software Forensics Definitions

Term	Definition
1. Software Forensics	A. "Software Forensics is the area of Software Science aimed at authorship analysis of computer source code. Over the past few years there has been a renewed interest in the science of software authorship identification; this area of research has been termed 'Software Forensics '" (Sallis, Aakjaer, & MacDonell, 1996).
	B. "Source code is the textual form of a computer program that is written by a computer programmer in a computer programming language. These programming languages can in some respects be treated as a form of language from a linguistic perspective, or more precisely as a series of languages of particular types, but within some common family. In the same manner as written text can be analyzed for evidence of authorship, …, computer programs can also be examined from a forensics or linguistics viewpoint … for information regarding the program's authorship. … Similarly, techniques used in forensics for handwriting and linguistic analysis can also, in some cases at least, be transferred in some respect to what is referred to here as software forensics" (Gray, Sallis, & MacDonell, 1998).
	C. "In this paper, we detail some of the features of code remnants that might be analyzed and then used to identify their authors. We further outline some of the difficulties involved in tracing an intruder by analyzing code. We conclude by discussing some future work that needs to be done before this approach can be properly evaluated. We refer to our process as software forensics, similar to medical forensics: we are examining the remains to obtain evidence about the factors involved" (Spafford & Weeber, 1992) .
	D. "Software forensics involves the analysis of evidence from program code itself. Program code can be reviewed for evidence of activity, function, and intention, as well as evidence of the software's author" (Slade, 2004).

Each of the definitions for software forensics presented in Table 1 emphasizes the determination of code authorship. Issues related to intellectual property law drove the research projects that established this definition. While the definitions for software engineering presented earlier in this paper evolved to include all documents related to the development of software as part of the software product, the definitions of software forensics address only the source code listing. As the real fault in the case study lies with the other documents related to the engineering of this system, the state of the practice in software forensics would not help us in this case. The author of the code was a defendant and acknowledged his authorship.

A more commonly encountered term in the literature for this growing discipline is "computer forensics." Table 2 presents eight definitions obtained from a literature survey of computer forensics. Definition 2A severely limits the applicability of computer forensics to the case study. There was need to scour digital media for evidence. Furthermore, this was not a criminal case. Definition 2B emphasizes analysis of data after a computer crime. While this definition is broader than 2A, it still applies only to criminal investigation and data recovery after a computer crime. Definition 2C removes the constraint that computer forensics apply only to criminal investigation, but it still focuses narrowly on examining the contents of a computer system (similar to 2A). Definition 2D returns to computer crimes and begins to address issues of admissibility for digital evidence. Definition 2E focuses only on digital data, while the artifacts

associated with the computer system in the case study were not digital themselves (though they did bring about the development of a digital system). Definition 2F addresses the fact that computer forensics is often focused on preservation of evidence from a computer crime. 2G is another definition that focuses on examining computer systems for evidence of criminal activity. Finally, definition 2H narrows computer forensics to only criminal investigation and gathering of digital evidence.

None of these definitions addresses the problem faced in the case study. While hard drives and other digital media could have been examined, nothing would have been found that was not already available through disclosure. More evidence concerning the specific version of the software that was running on the machine involved in the accident might have been forthcoming, but that would not have done anything to address the root cause of the problem or to determine which party was responsible. Each of the definitions of computer forensics seems to over-emphasize criminal law, analysis of evidence in digital form, and admissibility of digital evidence in court. Civil cases or cases involving digital systems in which the actual evidence is not in digital form are not addressed by these definitions. The case study clearly involved computers, software, and issues of negligence, yet none of the definitions for computer forensics addresses these issues.

Table 2. Computer Forensics Definitions

Term	Definition
2. Computer Forensics	A. "Computer forensics is the analysis of hard drives, CDs, floppy disks, Zip and Jazz disks, flash memory cards, and other forms of storage media to uncover evidence of crimes" (Garber, 2001).
	B. "Computer Forensics undertakes the *post-mortem*, or 'after-the-event' analysis of computer crime. Of particular importance is the requirement to successfully narrow the potentially large search space often presented to investigators of such crimes. This usually involves some form(s) of guided processing of the data collected as evidence in order to produce a shortlist of suspicious activities. Investigators can subsequently use this shortlist to examine related evidence in more detail" (Abraham & de Vel, 2002).
	C. "Computer forensics is the process of conducting an examination into the contents of the data on a computer system using state of the art techniques to determine if evidence exists that can aid in internal or legal investigations. Forensic specialists use a wide array of methods to discover data and recover deleted, encrypted, and damaged files" (Digital Forensics of Texas, 2005).
	D. "The investigation of computer crime, including the collection, analysis and presentation in court of electronic evidence. Also called: Cyberforensics, Forensics, Security Forensics, Digital Forensics, and Forensic Analysis" (Bitpipe, 2005).
	E. "People, processes, tools and measures to gather, analyze and interpret digital data to support or refute certain allegations of misuse involving digital systems" (Cybex, 2005).
	F. "Computer forensics is commonly described as the study of evidence derived from computers. However, the literature on the topic concentrates primarily on the recovery and preservation for presentation as evidence, of data from computers that may have been used in the commission of some criminal activity. In fact, most books on the subject focus on the preservation aspects, fastening on documentation of the chain of evidence, and almost ignoring the technical aspects of the field" (Slade, 2004).
	G. "The term *computer forensics* has many synonyms and contexts. It originated in the late 1980s with early law enforcement practitioners who used it to refer to examining standalone computers for digital evidence of crime. (Some prefer to call this aspect of computer forensics *media analysis*.) " (Yasinsac, Erbacher, Marks, Pollitt, & Sommer, 2003).
	H. "Computer forensics, is the application of scientifically proven methods to gather, process, interpret, and to use digital evidence to provide a conclusive description of cyber crime activities. Cyber forensics also includes the act of making digital data suitable for inclusion into a criminal investigation. Today cyber forensics is a term used in conjunction with law enforcement, and is offered as courses at many colleges and universities worldwide" (ISP Webopedia, 2005).

The terms "digital forensics" and "digital evidence" are both broader terms in that they do not imply any specific issues of computer hardware of software authorship. However, the definition for digital evidence in Table 3 limits the definition to evidence stored in binary form and alludes to digital evidence only being applicable in criminal cases.

Table 3. Digital Evidence Definition

Term	Definition
3. Digital Evidence	A. "Digital evidence is any information of probative value that is either stored or transmitted in a binary form (SWGDE 1998). This field includes not only computers in the traditional sense but also includes digital audio and video. It includes all facets of crime where evidence may be found in a digital form" (National Center for Forensics Science, 2005).

The definition for digital forensics is broader. As presented in Table 4, the definition of digital forensics covers both evidentiary and root cause analysis. Root cause analysis is very much what was needed in the case study. The definition does not tie digital forensics solely to criminal cases.

Table 4. Digital forensics definition

Term	Definition
4. Digital Forensics	A. "Preservation, identification, extraction, documentation, and interpretation of computer media for evidentiary and/or root cause analysis" (Kruse & Heiser, 2002).

The definition of "Forensic Engineering" comes the closest to addressing what we had to do in the case study. While this definition addresses the need to investigate failures and to address the process by which computer systems are built, it does not address the other established areas of digital forensics. The discipline still needs people expert in recovering data from hard drives and determining code authorship. Therefore, while the definition in Table 5 is the closest to addressing the issues presented in the case study, it fails to cover the breadth of the discipline.

Table 5. Forensic Engineering Definition

Term	Definition
5. Forensic Engineering	A. "Forensic systems engineering is the discipline investigating the history of Information Technology failures. It therefore focuses on the post-mortem analysis and study of project disasters. The work involves a detailed investigation of the project, the environment, decisions taken, politics, human errors and the relationship between subsystems. The work draws upon a multidisciplinary body of knowledge and assesses the project from several directions and viewpoints. The concept of systems is a central tool for understanding the delicate relationships and their implications in the overall project environment" (Dalcher, 2001).

From the definitions acquired from the literature in the field of digital forensics, it would seem clear that each definition evolved from trying to solve a very specific problem. Most of these problems have either related to issues of chain of custody and admissibility of digital evidence in criminal courts or to issues of intellectual property. Only the definition of Forensic Engineering addresses the use of forensic techniques to investigate technology failures.

Section V Conclusion

In the case discussed, the authors were struck repeatedly by the lack of the most basic mutual understanding of the fields of the practitioners involved. The lawyers struggled with simple concepts such as a requirements document. The authors wasted time on pursuits that were not feasible because of legal naiveté. The participation of any party with rudimentary knowledge of both legal procedure and software engineering would have facilitated the investigation significantly. But, other than ad hoc knowledge gained from experience and self-directed surveys of published materials, the authors found no source for acquiring the appropriate range of skills. However, the authors' newly acquired experience may be more broadly applicable than this particular case. As outsourcing of parts of the development process becomes increasingly common, similar investigations to identify root causes of system failures and determine responsibility may proliferate.

This case easily could have taken quite different paths with only minor changes in circumstances. For example, the victim might not have been standing where he was and the unexpected motion could have been reported as a serious hazard. An investigation may have resulted that was very similar to the one that occurred, but the focus might have been on identifying which of the companies involved would be responsible for fixing the problem and improving the processes that led to the fault. In this case, there may well have been no legal issues raised. At the other extreme, the operator may have been charged with a criminal offense based on negligence. The rules of investigation and evidence would have changed tremendously. The authors' expertise in software engineering and embedded systems would have still been brought to bear; however, the contexts would have varied greatly. If a similar case were presented in a criminal law context, the authors would likely find little of their experience peripheral to computing in the case studied to be of use.

More generally, the skills of many practitioners that intuitively would fall under the umbrella of ISE do not clearly relate to any subfield currently defined. For example, ISE would benefit greatly from inclusion of members of the Association of Certified Fraud Examiners (ACFE) with their insights into such matters as money laundering via networks and their interrogation skills. Psychologists and sociologists might provide insights into motives underlying computer crime.

An argument has been made and a supporting case study presented to call for the broadening of current definitions or agreement on a new definition to accommodate the full range of skills needed for a practitioner to understand and direct investigations having computing aspects, to categorize problems, to prescribe appropriate techniques, and to identify and effectively employ experts. Such a broader definition should be inclusive of relevant methodologies from the areas of the law, criminal justice, forensic accounting, computer science, computer engineering, software engineering, psychology, sociology, and other cognate areas and the definition should provide a forum for collaborative research.

About the Authors

Gregory A. Hall is an assistant professor at Texas State University-San Marcos. His areas of research include digital forensics, software engineering, software measurement, and software testing. His home page is accessible at http://www.cs.txstate.edu/~gh10 and his electronic mail address is gh10@txstate.edu.

Wilbon P. Davis is a Professor of Computer Science at Texas State University in San Marcos, Texas. He has taught computing for 35 years, has served as a consultant in industry in several roles, and has published in a range of areas. His main academic interests lie in software engineering and graphics. His electronic mail address is wd@cs.txstate.edu.

References

Abraham, T., & de Vel, O. (2002). Investigative Profiling with Computer Forensics Log Data and Association Rules. *Proceedings of the 2002 IEEE International Conference on Data Mining*.

American Heritage Dictionary of the English Language (4th ed.). (2000). New York, NY: Houghton Mifflin.

Bauer, F. (1972). Software Engineering. *Information Processing*, 71.

Bitpipe (2005). Retrieved from http://www.bitpipe.com/tlist/Network-Forensics.html

Boehm, B. (1976). Software Engineering. *IEEE Transactions on Computers*, C-25.

Caloyannides, M. (2001). *Computer Forensics and Privacy*. Artech House, Inc.

Ciardhuáin, S. (2004). An Extended Model of Cybercrime Investigations. *International Journal of Digital Evidence*, 3(1).

Cybex (2005). Retrieved from http://www.terena.nl/tech/task-forces/tf-csirt/meeting11/CYBEX-Bevilacqua.pdf

Dalcher, D. (2001). *Forensics* ECBS: The Way Forward, *Proceedings of the Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*.

Digital Forensics of Texas, Inc. (2005). Retrieved from http://home.swbell.net/txkidd/forensics.html

Garber, L. (2001). Computer Forensics: High-Tech Law Enforcement. *IEEE Computer*, January 2001.

Gray, A., Sallis, P., & MacDonell, S. (1998). IDENTIFIED (Integrated Dictionary-based Extraction of Non-language Dependent Token Information for Forensics Identification, Examination and Discrimination): A Dictionary-based System for Extracting Source Code Metrics for Software Forensics. Paper submitted to Software Engineering: Education & Practice.

ISP Webopedia (2005). Retrieved from http://isp.webopedia.com/TERM/C/cyber_forensics.html

Kruse, W., & Heiser, J. (2002). *Computer Forensics: Incident Response Essentials*, Addison-Wesley.

National Center for Forensics Science (2005). Retrieved from http://www.ncfs.org/home.html

Sallis, P., Aakjaer, A., & MacDonell, S. (1996). Software Forensics: Old Methods for a New Science. *Proceedings of the 1996 International Conference on Software Engineering: Education and Practice*.

Slade, R. (2004). *Software Forensics: Collecting Evidence from the Scene of a Digital Crime*, New York: McGraw Hill, pp. 2 - 5.

Spafford, E., & Weeber, S. (1992). *Software Forensics: Can We Track Code to its Authors?* Purdue Technical Report CSD-TR92-010, 19 February 1992.

Stephenson, P. (2003). Modeling of Post-Incident Root Cause Analysis. *International Journal of Digital Evidence*, 2(2).

Stephenson, P. (2004). The Application of Formal Methods to Root Cause Analysis of Digital Incidents. *International Journal of Digital Evidence*, 3(1).

Yasinsac, A., Erbacher, R., Marks, D., Pollitt, M., & Sommer, P. (2003). Computer Forensics Education, *IEEE Security & Privacy*, July/August 2003.