Good Enough V&V for Simulations: Some Possibly Helpful Thoughts from the Law & Ethics of Commercial Software

Cem Kaner
Florida Institute of Technology
Department of Computer Sciences
150 West University Blvd.
Melbourne, FL 32901
kaner@kaner.com
321-674-7137

Stephen J. Swenson
AEgis Technologies Group
SSwenson@aegistg.com
401-539-2504

Keywords:
Ethics, software testing, liability, cost-benefit analysis, negligence analysis, cost of quality, VV&A

ABSTRACT:. *Historically, the focus of VV&A research and development has been in the areas of developing new techniques and metrics and developing standards to ensure consistency across a wide range of applications. This work has proven technically beneficial to developers of M&S applications as "how to" guidance for conducting V&V. We all understand that how much V&V is driven by the needs of the target application. The guidance is vague and doesn't, in any substantive way, address the value of V&V in relation to ramifications associated with inadequate testing. The courts have long wrestled with the responsibility of product and service providers to adequately protect consumers and by-standers against failure. The theme of the 2008 Spring SIW is "Innovation at the Intersections." This paper will address the intersection of modeling and simulation VV&A with civil law, suggesting a style of cost/benefit analysis that might be specifically applicable to M&S VV&A, how it might be used to frame discussion of ethical commitment to adequate M&S testing, and how some of the information needed for that analysis might be collected.*

1. Introduction

In 1944, a group of barges including the Anna C broke free of their moorings in New York Harbor and drifted together. They stopped when Anna C bumped a tanker whose propeller broke a hole in her. No one was aboard the Anna C and therefore, no one noticed she was taking on water. She sank. Had someone noticed the water, she could have been saved by the people who were on the scene dealing with this multi-barge situation. The ensuing lawsuit required the judge to determine who should pay for the loss of cargo on Anna C.

Judge Learned Hand first determined there was not yet a general rule in American maritime law for allocating damages when a barge breaks free and the bargee (a person assigned to stay on the barge) is not on the barge. Bargees sometimes leave barges for legitimate reasons, and if the risk of their absence is low enough, this is not unreasonable. Rather than working through the mass of individual, conflicting precedents, Learned Hand laid out fundamental principle for decisions of this type:

> *"Since there are occasions when every vessel will break from her moorings, and since, if she does, she becomes a menace to those about her; the owner's duty, as in other similar situations, to provide against resulting injuries is a function of three variables: (1) The probability that she will break away; (2) the gravity of the resulting injury, if she does; (3) the burden of adequate precautions. Possibly it serves to bring this notion into relief to state it in algebraic terms: if the probability be called P; the injury, L; and the burden, B; liability depends upon whether B is less than L multiplied by P: i.e., whether B less than PL."*

This case, *United States v. Carroll Towing Co.* [1] is one of the most famous in American products liability law.

Learned Hand is widely recognized (on all sides of the political spectrum) as one of the great judges of American history. For example, his work was foundational for the

Law and Economics tradition of legal analysis, which has played a large role in the "tort reform" movement that has largely limited the reach of products liability litigation.

The T.J. Hooper [2] is another key case decided by Learned Hand. This case involves the sinking of two barges in a gale. Learned Hand determined that the tugs that were towing the barges were unseaworthy because they did not have radios and therefore could not call for help. The owners argued that no legislation or regulations required radios on tugs and industry standard practice was that tugs did not have them. Learned Hand's response:

> *"Is it then a final answer that the business had not yet generally adopted receiving sets? There are yet, no doubt, cases where courts seem to make the general practice of the calling the standard of proper diligence; we have indeed given some currency to the notion ourselves…. Indeed in most cases reasonable prudence is in fact common prudence; but strictly it is never its measure; a whole calling may have unduly lagged in the adoption of new and available devices. It may never set its own tests, however persuasive be its usages. Courts must in the end say what is required; there are precautions so imperative that even their universal disregard will [**12] not excuse their omission.…We hold the tugs therefore because had they been properly equipped, they would have got the Arlington reports. The injury was a direct consequence of this unseaworthiness. Decree affirmed."*

Put succinctly, industry standards are not legal standards. The argument that you were following industry-standard practices is no shield for inadequate work.

In this article, we argue that Learned Hand's cost-benefit analysis is a useful approach for VV&A of simulations.

We are ***not*** recommending a ***liability*** standard in this paper. We are ***not*** arguing that courts should adopt a products liability standard for simulations. Most complex simulations are created for government clients, typically for Defense applications. In the United States, developers of custom products for the United States DoD are not subject to civilian products liability litigation. Many factors mitigate against applying civilian risk regulations to military decision-making and we are not considering any aspect of the history or appropriateness of that well established rule in this paper.

What we are suggesting is that practitioners of VV&A for simulations might find this analytical approach useful:

- as an ethical framework in a context (simulations) that is so different from traditional DoD-related VV&A that many of the ethical/analytical touchstones are not applicable;
- as a practical guideline for considering the types of investigation that the V&V practitioner might undertake to determine whether a product is good enough, or to persuasively build an argument that it is not.

2. The Problem

Software products for DoD clients are usually created under a development model in which the key decisions are made early and documented in detail in specifications of the project/product requirements and the internal and external product design. Special care is taken to create applicable oracles, decision rules that can be applied to determine whether the product is operating within expectations or not. Not all projects are done this way, but in the context of a contracting and development culture that assumes this as the normal style, significant deviations from this approach carry significant development, cost, and quality risk.

Verification and validation traditions have evolved in the context of this development culture. V&V analysts vigorously encourage project managers and others to develop clearer and more comprehensive specifications and oracles and to relate them to the specific objectives of the model/simulation [3-5]. To the extent that the specifications can be considered complete, thorough testing tied directly to those specifications is desirable and is often taken as sufficient for system-level testing. In DoD projects, verification also includes assessment of the development process and practices employed [3, 6], comparing them to standards-driven expectations.

A diligent, ethical practitioner in traditional V&V has a broad collection of standards and cultural support for doing her or his work in a way that will be seen as competent and respectable.

Simulations are different.

A simulation is an implementation of a model over time. [6] "All models are wrong, but some are useful" [7, p. 424]. An essential part of the evaluation of a simulation is its utility, its value to its intended stakeholders. It is not only the actual implementation that is important, but the conceptualizations behind it. [8, 9] A perfectly implemented simulation that starts from the wrong theory will yield perfectly wrong results.

The situations that simulations model are typically not perfectly understood. That is, in a typical simulation, there are many major unknowns. If everything important was known, we wouldn't need a simulation. Therefore we are unlikely to have a strong set of specifications (requirements or design) or reliable oracles for evaluating the simulation. This requires us to rethink the role and strategy of verification and validation.[10]

If part of a simulation specification is thoroughly and credibly developed, this probably means that the development team (defined broadly to include subject matter experts and the customer) have such a good understanding of this aspect of the situation being simulated that they can provide clear instructions. Coding errors and misunderstandings of the specification are certainly possible, but we should expect *more* problems, and *worse* problems, with all of those aspects of the simulation that were developed *without* the benefit of a good specification, because these are probably the parts of the system that are not well understood and do not benefit from stabilized human expertise.

Another complication is that the user's understanding of the system or situation being simulated will probably evolve over time, partially through interaction with draft versions of the simulation. In other words, requirements are likely to drift significantly.[11]

These characteristics of simulations make it difficult to rely on traditional stopping rules for software testing, such as:

- **Structural coverage:** Execution of every statement, or every statement and branch, or similar simple coverage criteria. The code that is there might work, but is it the right code? Does it do the right tasks? What about the conditions not considered, therefore not included in the code?
- **Coverage of the specification:** Execution of one or a few test cases for each statement or condition or data item listed in the specification. If we already know that the specification is significantly incomplete and incorrect, then we know that tests focused on the specification might be useful, but they are far from sufficient even for a superficial test of the software application.
- **Adherence to a traditional development process:** Effective simulation development might look much more like an agile development project, an approach that relies on rapid iteration and close interaction with users or their subject-matter-expert representatives, rather than implementation of a well-understood solution to a well-understood problem.
- **Low failure rates from testing:** The rate of failures from a suite of regression tests is not very informative if you don't know whether the tests in the suite are representative of the usage in the field [12]. The specification cannot be an effective surrogate for understanding usage in the field because it is incomplete and incorrect.

3. An Analogy: Testing Commercial Software Products

In two important ways, commercial software projects are often like simulations:

1. A software product attempts to automate some task or system, but the humans who do the tasks that are being automated often use personal knowledge and heuristics that are hard to capture. In effect, the specification (if there is one) is an imperfect model of the human system being automated, and the implementation of that specification is a simulation of that human system.
2. These projects often have incomplete, evolving specifications and multiple influential stakeholders who have conflicting interests (and therefore conflicting success criteria for the project).

In practice, developing stopping rules for commercial software testing has been very difficult. Summarizing the results of several working meetings of test managers and consultants, Kaner [13] listed a few hundred different measures of testing progress that are in use in the field. There is remarkably little agreement on which subset of measures, or even which subset of types of measures, should be used to determine how much testing has been done and to estimate how much testing is left.

As with simulations, the oracle problem is difficult for commercial software. In an influential presentation to practitioners, Doug Hoffman [14] argued that all oracles for commercial software products are heuristic (useful decision rules, but not guaranteeably correct). Hoffman made several arguments, but his argument from complexity is the most relevant to simulations:

1. An oracle provides a mechanism for determining whether the program's behavior was correct, given a set of preconditions, a specified action, and the observed results.
2. The first problem is that we never fully specify the preconditions. (How much free memory is in the system? When was the last garbage collection cycle? How fragmented is the hard disk? Did the test just before this one send data to the printer?) One could argue that many conditions like these are (or should be) irrelevant, so long as the values are "reasonable." However, many hard-to-reproduce

bugs turn out to have unexpected values in variables that the testers and programmers didn't expect to be relevant and therefore didn't formally observe and control. The list of potentially-relevant (not probably-relevant, but potentially-relevant) variables is enormous.

3. The second problem is that the user action is often loosely specified. For example, timing of a test (time between keystrokes for example) is sometimes critical to reproducing a failure, but this is rarely specified. On systems (like Windows) that have ongoing background task processing by the operating system, it seems impossible to know what else (what other user demands) compete for CPU attention, memory, and other system resources.
4. The third problem is that the output is not fully specified. For example, if a program's task is to add integers, and it adds 2+7, obtaining 9, is this correct? It might look correct, but what if the program took 6 hours to add those two numbers? What if, along with displaying the answer on the screen (as expected), it also sent the result as an email to everyone in your email client's address book? These examples might seem to obviously violate a rule of reason, but how many test specifications for a function that adds two numbers would instruct the tester to check for timing, memory leaks, file corruption, email activation, or the many other things that could theoretically go wrong (and sometimes do)?
5. Given the unspecified aspects of the test (almost every aspect of the test as it is actually run is unspecified), the program might appear to pass but actually fail, or it might appear to fail but actually be behaving correctly under the circumstances. That is, the decision rule provided by the oracle is probably correct most of the time, but it is not guaranteed. It is a heuristic, rather than a genuine rule.
6. If an oracle is a mechanism for deciding whether a product's behavior is correct or not, an oracle heuristic is an imperfect but useful mechanisms for deciding whether the behavior is correct or not. Hoffman's argument was that all software oracles are heuristic oracles.

James Bach extended Hoffman's idea while he was developing the testing process for Microsoft for qualifying products as Windows 2000 compatible [15]. He identified a collection of patterns in the ways that Microsoft engineers argued that a product was or was not defective. These, he reasoned, were the implicit oracle heuristics that Microsoft's development staff relied on.

Kaner & Bach [16] provide a video lecture that present and discusses Bach's heuristic oracles. Here is the list of heuristics from that video (see [17] for additional course slides):

- ***Consistent within product:*** The behavior of a function should be consistent with behavior of comparable functions or functional patterns within the product.
- ***Consistent with comparable products:*** The behavior of a function should be consistent with that of similar functions in comparable products.
- ***Consistent with history:*** The present behavior of a function should be consistent with past behavior.
- ***Consistent with our image:*** The behavior of a function should be consistent with the image the organization wants to project.
- ***Consistent with claims:*** The behavior of a function should be consistent with documentation or ads.
- ***Consistent with specifications or regulations:*** The behavior of a function should be consistent with claims that must be met.
- ***Consistent with user's expectations:*** The behavior of a function should be consistent with what we think users want.
- ***Consistent with Purpose:*** The behavior of a function should be consistent with the product or function's apparent purpose.

This heuristic approach is relevant to simulations because the exact, correct behavior for the simulation is probably unknown (for many systems, if we knew them completely, we wouldn't have to simulate them) and even if we have a theoretically complete knowledge of the system, the simulation is a simplification for teaching purposes, demonstration purposes, or some other practical purposes and the ultimate criterion is whether the simulation is valuable and not whether it is perfectly correct in all details.

The shared complexities of simulation software and (much) commercial software provide a basis for thinking that decision rules used to manage the complexity of commercial software testing *might* be relevant to simulation evaluation.

We will focus the rest of this paper on one type of decision rule used in commercial projects, the cost/benefit analysis, as applied to project management. We'll consider two variations on this theme:

1. *Quality Cost Analysis* as developed by the American Society for Quality (e.g. [18]).
2. *Negligence analysis* as applied to the American law of products liability (e.g. [19]).

4. Overview of Cost-Benefit Analysis

We use the phrase, "managing a project under uncertainty" as a shorthand for the complexity of the problem space when stakeholder interests conflict, stakeholder requirements shift over time, development starts without full knowledge (let alone specification) of the solution to be achieved, oracles are sometimes informal, unavailable or untrustworthy outside a narrow range of parameters, exhaustive testing is impossible and it is difficult to meaningfully measure how much testing is completed, let alone how much risk has been mitigated.

How can a project manager or a test manager operate effectively and ethically when working with a project under uncertainty?

Rather than focusing on the specifics of the individual decisions or decision criteria, the quality-cost literature and the products liability literature create a framework for thinking about the pattern of decisions as a whole.

5. Quality Cost Analysis

> *"Because the main language of [corporate management] was money, there emerged the concept of studying quality-related costs as a means of communication between the quality staff departments and the company managers."*[20, p. 4.2]

Joseph Juran, one of the world's leading quality theorists, began advocating analysis of quality-related costs 1951, when he published the first edition of his *Quality Control Handbook*. Feigenbaum made it one of the core ideas underlying the Total Quality Management movement.[21] It is a tremendously powerful tool for product quality, including software quality.

Quality costs are the costs associated with preventing, finding, and correcting defective work. For commercial products, these costs are huge, running at 20% - 40% of sales [20, p. 4.2. We are not aware of credible data on quality costs for commercial software] Many of these costs can be significantly reduced or completely avoided.

One fundamental objective of quality engineering is to drive down total cost of quality associated with a product or product line. Notice that this is full lifecycle cost, not just cost of development.

Here are six useful definitions, as applied to software products. Figure 1 gives examples of the types of cost. Most of Figure 1's examples are (hopefully) self-explanatory, but I'll provide some additional notes on a few of the costs. (These are our translations of the ideas for a software development audience. More general, and more complete, definitions are available in [18] as well as in Juran's and Feigenbaum's works).

- ***Prevention Costs:*** Costs of activities specifically designed to prevent poor quality. Examples of "poor quality" include coding errors, design errors, mistakes in user manuals, as well as badly documented or unmaintainably complex code. (Note that most prevention costs don't fit within a Verification Group's budget. This money is spent by the programming, design, and marketing staffs.)
- ***Appraisal Costs:*** Costs of activities designed to find quality problems, such as code inspections and any type of testing. Design reviews are part prevention and part appraisal. (To the degree that you're looking for errors in the proposed design itself when you do the review, you're doing an appraisal. To the degree that you are looking for ways to strengthen the design, you are doing prevention.)
- ***Failure Costs:*** Costs that result from poor quality, such as the cost of fixing bugs and the cost of dealing with customer complaints.
- ***Internal Failure Costs:*** Failure costs that arise before a company supplies its product to the customer. Along with costs of finding and fixing bugs are many internal failure costs borne by groups outside of Product Development. If a bug blocks someone in the vendor company from doing her job, the costs of her wasted time, missed milestones, and overtime to get back onto schedule are all internal failure costs. (Including costs like lost opportunity and cost of delays in numerical estimates of the total cost of quality can be controversial. Campanella doesn't include these in a detailed listing of examples [18, Appendix B]. Gryna [20, 4.9-4.12] recommends against including costs like these in the published totals because fallout from the controversy over them can kill the entire quality cost accounting effort. We include them here because in Kaner's industrial experience in Silicon Valley, as a project manager, test manager, and development director, these were very useful in practice, even if it might not make sense to include them in a balance sheet.)
- ***External Failure Costs:*** Failure costs that arise after a company supplies the product to the customer, such as the costs of customer service, maintenance, warranty repairs, and public relations efforts to soften the impact of a bad failure on the vendor's reputation.
- ***Total Cost of Quality:*** The sum of costs: Prevention + Appraisal + Internal Failure + External Failure.

PREVENTION	APPRAISAL
• Staff training • Requirements analysis • Early prototyping • Fault-tolerant design • Defensive programming • Usability analysis • Clear specification • Accurate internal documentation • Evaluation of the reliability of development tools (before buying them) or of other potential components of the product	• Design review • Code inspection • Glass box testing • Black box testing • Training testers • Beta testing • Test automation • Usability testing • Pre-release out-of-box testing by customer service staff
INTERNAL FAILURE	**EXTERNAL FAILURE**
• Bug fixes • Regression testing • Wasted in-house user time • Wasted tester time • Wasted writer time • Wasted marketer time • Wasted advertisements (1) • Direct cost of late shipment (2) • Opportunity cost of late shipment	• Technical support calls • Preparation of support database • Investigation of customer complaints • Refunds and recalls • Coding / testing of interim bug fix releases • Shipping of updated product • Added expense of supporting multiple versions of the product in the field • PR work to soften drafts of harsh reviews • Lost sales • Lost customer goodwill • Discounts to resellers to encourage them to keep selling the product • Warranty costs • Liability costs • Government investigations (3) • Penalties • All other costs imposed by law

Figure 1. Examples of quality costs associated with software products (4)

Here are a few additional notes on the figure:

(1) Imagine buying advertisements for a product that should release to the public in early 2009 and then releasing it in 2010.

(2) If bug fixes caused late shipment of a product, the *direct cost* of late shipment includes rush shipping fees and lost sales. The *opportunity cost* includes costs of delaying other projects while everyone finishes this one.

(3) Cost of cooperating with a government investigation, including legal fees, whatever the outcome.

(4) Gryna [20] cautions against presenting estimates that include costs that other managers might challenge as not quality-related. A perception that you are padding the numbers for dramatic effect can destroy the credibility of your estimates. Consistent with this, in Figure 1, we omit such costs as high turnover (staff quit over quality-related frustration) and lost pride (people will work less hard, with less care, if they believe the final product will be low quality no matter what they do.) We don't know how to credibly include them in our totals, but that doesn't mean that we shouldn't bear them in mind. They can be very important.

The ***power*** of the quality-cost approach is that it translates qualitative concerns into financial estimates that can attract the attention of more senior managers.

Some companies treat quality-related costs very formally and in great detail, with quality- cost management databases. Others use this concept more tactically. Here's an example of tactical use:

> *Suppose you think the user interface, as designed, will lead to user errors as a person uses a product to control a device or analyze a situation under time pressure. You run some quick usability tests and demonstrate that in your pool of users, people made some characteristic mistakes. Approaching other members of the product development staff, you could build arguments, with them, that helping people avoid these mistakes in real use in the field will require higher documentation costs (get an estimate from the writers), higher training costs (get estimates from trainers or training materials developers), higher post-sales support costs, reputational damage if the mistakes are serious, and goodwill damage with the users if an investigation of mistakes made in the field blames them on "pilot error." The estimates of these costs don't have to be perfect, but each one must come from a credible source and be in some way plausible and justifiable.*

Along with arguing about individual bugs or decisions, this approach opens opportunities to make business cases on broader issues. Here are three examples that verification staff might be involved in:

- What savings could be realized if some programming staff and some verification staff spent time with users in the field, developing an understanding of the subject matter and the complexity of the situations being simulated? If the design is more robust (because the programmers / low-level designers have more intuition about the types of changes that are likely) and the tests are more realistic and more powerful, how much time and money is this worth?
- Could savings be achieved by increasing the programming staff and requiring them to develop and maintain suites of unit tests that check for common errors (such as boundary conditions) in every variable, compared to doing this work through verification at the black box system level later on? What additional code inspection (inspection of the unit tests) would be required and who should do it? With the added programming and inspection costs, is this a cost-favorable, neutral, or cost-wasteful change?
- One of the common assertions from the agile development movement [22] is that intense automated unit testing provides a foundation for refactoring [23] and through that, relatively safe and cost-controlled maintenance. Agile development was created to cope with poorly-developed, rapidly changing user requirements. Would a shift in methodology that included some of the agile practices, such as using extensive unit test suites to improve maintainability, be appropriate for a simulation project? How much conceptual rework is likely on this particular simulation? How much will require complete discarding of implemented parts of the system, rather than rework? How would those impact the cost and value of the unit test suites?

By providing an economic structure for these discussions, quality-cost analysis describes rules of research and engagement for staff who are inclined to vigorously argue for or against choices of broad process or specific practices. That is, people who can made a credible quality-cost argument for a given decision are more likely to be taken seriously, and people (of comparable stature in the company) who cannot or will not develop cost/benefit estimates are more likely to be ignored or bluntly told to quit wasting people's time on subjective hunches about project management and product quality.

The ***problem*** of this approach is that quality cost analysis looks at the company's costs, not the customer's costs. The manufacturer and seller are definitely not the only people who suffer quality-related costs. When a manufacturer sells a bad product, the customer faces significant expenses in dealing with that bad product.

The Ford Pinto litigation provided a classic example of quality cost analysis. Among the documents produced in these cases was the Grush-Saunby report [24, p. 841, 25, p. 225], which looked at costs associated with fuel tank integrity. The key calculations appeared in Table 3 of their report:

Benefits and Costs Relating to Fuel Leakage Associated with the Static Rollover Test Portion of FMVSS 208
Benefits Savings – 180 burn deaths, 180 serious burn injuries, 2100 burned vehicles Unit Cost -- $200,000 per death, $67,000 per injury, $700 per vehicle Total Benefit – 180 x ($200,000) + 180 x ($67,000) + 2100 x ($700) = $49.5 million.
Costs Sales – 11 million cars, 1.5 million light trucks. Unit Cost -- $11 per car, $11 per truck Total Cost – 11,000,000 x ($11) + 1,500,000 x ($11) = $137 million.

Figure 2. The cost of quality analysis for the Ford Pinto

This is an internal cost of quality analysis. Ford estimated that it would cost $137 million to fix the gas tanks of these cars, and if they did not, they would be held accountable for 180 burn deaths, 180 serious burn injuries, and 2100 burned vehicles. (In fact, there would probably be more deaths and injuries, but this is the estimated number of cases that could be *proved* to have been caused by the exploding gas tanks.)

To *Ford*, the estimated cost of causing the death of a customer was $200,000 and the cost of serious burn injuries was $67,000. To Ford's *customers*, the children of the dead father, the people who were disfigured for life with burns, those costs were probably much higher. How much pain would it cause for you if your mother or your daughter died? That's not an economic loss, but it is a huge cost—for the customer. Not for Ford.

From the point of view of quality cost analysis as it is documented by the American Society for Quality and typically used in practice, it doesn't matter how high the cost of a defect is to a customer. That cost is made invisible. All that matters is cost to the company.

SELLER: EXTERNAL FAILURE COSTS	**CUSTOMER: FAILURE COSTS**
These are the types of costs absorbed by the seller that releases a defective product.	These are the types of costs absorbed by the customer who buys a defective product.
• Technical support calls • Preparation of support database • Investigation of customer complaints • Refunds and recalls • Coding / testing of interim bug fix releases • Shipping of updated product • Added expense of supporting multiple versions of the product in the field • PR work to soften drafts of harsh reviews • Lost sales • Lost customer goodwill • Discounts to resellers to encourage them to keep selling the product • Warranty costs • Liability costs • Government investigations • Penalties • All other costs imposed by law	• Wasted time • Lost data • Lost business • Embarrassment • Frustrated employees quit • Demos or presentations to potential customers fail because of the software • Failure when attempting other tasks that can only be done once • Cost of replacing product • Cost of reconfiguring the system • Cost of recovery software • Cost of tech support • Injury / death

Figure 3. Comparing sellers' costs and customers' costs

When costs to the customer are high, cheated or injured customers have an incentive to transfer some of their costs back to the seller. They do this through litigation.

Thus, in cases like Grimshaw v. Ford [26], juries awarded millions in punitive damages to the victims of the exploding gas tanks. Punitive damages are awarded when the defendant's actions are determined to be sufficiently outrageous. They are awarded not to compensate the victim but to punish the defendant, *to change the economics* of the defendant's actions. Big products liability verdicts change the cost of selling dangerously defective cars.

The Ford case is a classic example of a chronic problem. Another example from the automotive industry is the more recent *General Motors Corp. v. Johnston* [27]: A PROM controlled the fuel injector in a pickup truck. The truck stalled because of a defect in the PROM and in the ensuing accident, Johnston's seven-year old grandchild was killed. The Alabama Supreme Court justified an award of $7.5 million in punitive damages against GM by noting that GM "saved approximately $42,000,000 by not having a recall or otherwise notifying its purchasers of the problem related to the PROM."

The well-publicized cases are for disastrous personal injuries, but there are plenty of cases against computer companies and software companies for breach of contract, breach of warranty, fraud, etc.

The fundamental problem of cost-of-quality analysis is that it sets sellers up to underestimate customer dissatisfaction and litigation risks. Many sellers' staff think, when they have estimated the total cost of quality associated with a project, that they have done a fairly complete analysis. But if sellers don't take customers' external failure costs into account, then when those customers resort to litigation to transfer back to the sellers some of the enormous costs the sellers thought they had successfully pushed off on the customers, the sellers' staff can be surprised by huge increased costs (lawsuits) over decisions that they thought, in their incomplete analyses, were safe and reasonable.

6. Negligence Analysis

The greatest strength of quality cost analysis is that it is persuasive. If you can demonstrate to a business that it can reduce its costs by X% by spending less than this to improve the company's quality processes or products, most executives will listen carefully. They will *want* to make these changes.

The greatest weakness of quality cost analysis is that it ignores large categories of costs, such as costs borne by the customer instead of the seller. It creates an incentive for hiding costs and risks by externalizing them (pushing them onto the customer). As another example, technical support costs used to cost software publishers an average of $23 per call [28], creating enormous incentives to improve the quality of consumer software. Those incentives flipped when publishers started charging for technical support. Customer dissatisfaction lost its biggest indicator in the quality cost spreadsheets.

Let's bring this back to simulations: the Ford cases are the more compelling when we think about software used to help people deal with (for example) complexities of battlefield conditions. Lives are at stake. Analyses that focus only on producers' costs are, arguably, a fundamental breach of the trust under which the contract for developing this simulation was created. The objective of the effort is to protect the warfighter, not to protect the profitability of the vendor (although, certainly, that profitability is a justifiably essential requirement for any company that wants to survive in this market space). The solution is not to ignore the vendor's costs but to find a way to also include costs (risks) to the customer / user.

It is hard to take quality cost analysis seriously as an ethical guide for responsible, professional conduct by the verification engineer or the other development staff, because it ignores the risks to the warfighter. The analysis is too deeply self-serving to serve as a guide for how to deal with others with integrity.

Negligence analysis, as laid out by Learned Hand, provides an alternative. It looks almost like quality cost analysis, but is different in one critical way.

Here again are his factors, mapped to failure of a product:

- A potential failure (F), causing a loss (L)
- The probability that this failure will happen in the field (P)
- Measures that can be taken to prevent that loss, with a cost of C

Preventative measures are called for if $C < P*L$. Preventative measures are unreasonably expensive and need not be done if $C > P*L$.

The difference between quality cost analysis and negligence analysis is this:

- Under a quality cost analysis, we estimate L by totaling all of the *manufacturer's costs* associated with the failure.

- Under a negligence analysis, we estimate L by totaling all of *society's costs* (costs of failure to the user and anyone else impacted by the failure).

Unlike quality cost analysis, negligence analysis tells the manufacturer to balance its own costs against the risks that its products create for society.

It's important to distinguish here between the use of this analysis as a personal guide to ethical analysis of a situation and as a legal criterion for liability.

As a tool for determining legal liability, negligence analysis has been criticized on several grounds. Here are some of the concerns raised:

- In the press of multimillion dollar litigation, there are enormous incentives to bias the estimates of costs and risks. Expert witness testimony to juries will conflict and jury opinions may depend more on their perception of credibility of the expert than on the underlying soundness of the estimates.
- In the American justice system, juries are charged with making all decisions about the facts in a case. They decide what is true and what is false. The judge then applies the law to those facts. It is difficult to understand how a jury of people who have little mathematical knowledge and no engineering background can decide what the engineering facts of a case are.
- A design flaw exists whenever all copies of the product have the same flaw. Software bugs are all design flaws. A small flaw can have an enormous cumulative effect. The cumulative cost of 10 million people being slightly inconvenienced can be huge. Unless we change the criterion, to require correction of a problem only when social cost is *much* greater than manufacturer's cost, manufacturers of mass-market products will face so much litigation over tiny defects, involving decisions that are even harder for juries because they are close calls that require more careful judgment, that liability protection will become one of the main concerns of the business (for those manufacturers who don't simply abandon the field).

The American Law Institute [29] published a rebalancing of products liability law to deal with issues like these. While this work has been very influential in the legal community, the process and result were enormously difficult and controversial.

We don't have to express opinions about these very difficult issues in this paper, or join debates about how well the legal system is handling them, because we are not suggesting this approach as a legal framework. We are suggesting it as a way for V&V professionals to think about the implications of their work in a context that strips them of other familiar touchstones for determining whether the decisions and practices around them are responsible and appropriate or not.

In short, our proposal is this:

Suppose that we drove the scope of testing by considering risks and the ways to mitigate them [30]:

- Given a potential failure (a hazard), we could ask what kind of testing would be required to determine whether the software could fail in this way, how likely the failure, and what it would take to fix it.
- What is our best starting estimate (guess) of the cost of doing the testing needed to expose failures of this kind?
- What is our best starting estimate of the severity of such a failure, taking into account **cumulative impact on *all* stakeholders** (the vendor, the warfighter, the agency paying for the software, etc.)?
- What is our best starting estimate of the probability of such a failure?

If the estimated additional cost of testing (and fixing) is significantly less than the estimated risk (cost times probability), we should do the testing. Similarly, if there was some other way to mitigate the risk (such as better programming rather than better testing) that cost less than the testing, we should do that instead.

If the estimated additional cost of mitigation is significantly less than the risk, we should not mitigate the risk. It is too expensive.

If the estimated cost of mitigation is close to the risk, we should make the decision based on other heuristics, such as asking for guidance from the client who is paying for the simulation.

7. Stakeholder Analysis

To estimate the cumulative impact on all stakeholders, we have to identify all of the stakeholders and estimate the potential impact on each.

A stakeholder is normally defined as "anyone who could be materially affected by the implementation of a new system or application" [31, p. 40].

We prefer a refinement to this definition suggested by Gause & Weinberg [32]. Some stakeholders are *favored*, some are *disfavored*, and some we ignore.

- Most discussions of stakeholders are of favored stakeholders—the best system affects these people positively.
- In contrast, the successful system affects disfavored stakeholders negatively. For example, an embezzler is a disfavored stakeholder of a bank security system. An enemy combatant is a disfavored stakeholder of any system designed to protect our troops or improve their combat capabilities.

A complete stakeholder analysis should consider both classes. A system that protects our troops might be an utter failure if, in achieving this, it protects enemy combatants even more.

Robertson & Robertson [33] and Gause & Weinberg [32] provide guidance, including brainstorming suggestions, for identifying the many potential stakeholders for a project or product, classifying them (favored, disfavored), grouping them, and deciding how important they are. If you learn better from videos, Kaner [34] created a worked example on video for students in Computer Law & Ethics courses at Florida Tech and the University of Illinois at Springfield.

Some of the most important lessons of stakeholder analysis are:

- There are more stakeholders than you think
- Disfavored stakeholders may be very important
- A stakeholder can be affected by the same system, or even by the same aspect of the same system, in more than one way
- Favored stakeholders often have conflicts of interest: the same aspect of the system may both benefit and interfere with interests of the same person (for example, a particularly effective development process might reduce a vendor's costs to complete the project, thereby increasing its profit on a fixed-price contract, but might also reduce the need for later maintenance and thus the opportunities for maintenance-related profits)
- The interests of different favored stakeholders may conflict
- A requirements document reflects a balance of power/influence among the stakeholders at the time it was written. Over time, the stakeholders change, their respective power changes, and the agreement reflected in the document no longer reflects the realities of the stakeholder group.

If you try too hard to do a complete stakeholder analysis, you will be paralyzed by the scope of the work: This is an open-ended creative process. It is probably impractical to identify every stakeholder, and it is probably impossible to tell whether you have identified all of them. It is often impractical to treat each stakeholder (such as, every soldier who carries a certain type of weapon) individually. It is probably impossible to tell whether you have identified all of the interests (and thus the roots of all of the relevant costs and benefits) of a given stakeholder or stakeholder group.

Thus, a stakeholder-focused cost/benefit analysis will always be incomplete and uncertain. This is not a precise estimation task, because that is not possible. This is a way of thinking about a system's effects, the tasks that you might consider that could improve those effects, and the likely net cost or net benefit of such a task.

The likely win from this type of analysis is that in doing the brainstorming, and in training yourself and your group to think along these lines, you will sometimes identify situations that will have a big effect on a class of stakeholders, and the impact is obviously significant compared to the cost of mitigating it. These make for easy decisions—once you identify them.

8. Developing Tests for Simulations

Much of traditional verification and validation involves relatively simple tests that focus on some part of a specification (requirements spec, design spec, etc.). Testing coverage is measured by checking that all parts of all of the controlling specifications have tests associated with them, and not necessarily how powerful or insight-provoking those tests might be.[35]

If a product is fully and accurately specified by its collection of specification documents, and if the tests cover all of the parts of the specifications in a way that includes tests for each way that the product might fail to meet one of the specification parts, then one could argue that this testing fully verifies the behavior of the product. Kaner would consider that argument incorrect even in this case, but that is a controversy that we can avoid in this paper.

The problem is that even if "complete" specification-driven testing were sufficient when the specifications are complete and accurate, when they are not (as in simulations), then testing that is solely driven by the specifications cannot be complete, or even adequate, by any reasonable criterion.[36]

When dealing with complex systems, there is certainly value in creating a collection of fairly simple tests. If the program cannot do a specific function correctly, then running a test that involves that one function in combination with many others will involve a lot of work to expose a simple failure, both in running the test and in

troubleshooting the failure to the extent needed to reveal that the problem is as narrow and simple as it is. However, once the individual features seem to work on their own, there is a lot of value in tests that run these features together in ways that stakeholders will find meaningful.

In a simulation, if it is not fully specifiable and the requirements / expected behavior are not fully understood, we won't always know what behavior to expect when we create a realistic test case. By "realistic test case", we mean a test in which the software is used to do something that a stakeholder will actually want to get done. A well-designed test of this type provides a sample of the program's behavior that a stakeholder can respond to and criticize. For example, a subject matter expert might say that it doesn't matter what the specification says, under these circumstances, the simulation should not behave this way. Or a person who will use the equipment or follow the process being simulated might react to a test by saying that it is unworkable, that if the real equipment (or the real process) works this way, it will be too complex (too slow, too confusing, too something-bad) to be useful. Tests of this class thus mix verification (checking the actual behavior of the functions tested in combination against the behavior you would predict from the specifications) and validation (checking whether the behavior is desirable).

Commercial software testers often call this type of testing, *scenario testing*. [37-39].

In a commercial software test, the typical scenario has five characteristics:

- The test is *based on a story* about how the program is used, including information about the motivations of the people involved.
- The story is *motivating*. A stakeholder with influence would push to fix a program that failed this test. (Anyone affected by a program is a stakeholder. A person who can influence development decisions is a stakeholder with influence.)
- The story is *credible*. It not only *could* happen in the real world; stakeholders would believe that something like it probably *will* happen.
- The story involves a *complex use* of the program or a *complex environment* or a *complex set of data.*
- The test results are *easy to evaluate*. This is valuable for all tests, but is especially important for scenarios because they are complex.

The Online M&S Glossary [6] defines scenarios differently: a simulation scenario under that definition might require a setup of the actual conditions being simulated, such as a realistic, live demonstration. That's more extreme than we intend.

Kaner & Bach [40] provide a video and slide set on the design of scenario tests. They identify 16 ways in which people generate scenarios. Each of these questions, taken on its own, might lead a tester to imagine a suite of related scenarios:

- Write life histories for objects in the system. How was the object created, what happens to it, how is it used or modified, what does it interact with, when is it destroyed or discarded?
- List possible users, analyze their interests and objectives.
- Consider disfavored users: how do they want to abuse your system?
- List system events. How does the system handle them?
- List special events. What accommodations does the system make for these?
- List benefits and create end-to-end tasks to check them.
- Look at the specific transactions that people try to complete, such as opening a bank account or sending a message. What are all the steps, data items, outputs, displays, etc.?
- What forms do the users work with? Work with them (read, write, modify, etc.)
- Interview users about famous challenges and failures of the old system.
- Work alongside users to see how they work and what they do.
- Read about what systems like this are supposed to do. Play with competing systems.
- Study complaints about the predecessor to this system or its competitors.
- Create a mock business. Treat it as real and process its data.
- Try converting real-life data from a competing or predecessor application.
- Look at the output that competing applications can create. How would you create these reports / objects / whatever in your application?
- Look for sequences: People (or the system) typically do task X in an order. What are the most common orders (sequences) of subtasks in achieving X?

These are much like the activities / analyses you would expect from a requirements analyst, but there are differences:

- Requirements analysts try to foster agreement about the system to be built. Testers exploit disagreements to predict problems with the system.

- Testers don't have to reach conclusions or make recommendations about how the product should work. They have to expose credible concerns to the stakeholders.
- Testers don't make the product design tradeoffs. They expose consequences of those tradeoffs, especially consequences that are unanticipated or more serious than expected.
- Testers don't have to respect agreements that were recorded in the requirements documents or other specifications. They can report anything as a problem that they think a stakeholder with influence *should* (but does not yet) know.
- The set of scenario tests need not be exhaustive, just useful.

A narrower definition of the "scenario" is an instantiation of a use case—essentially a description of a sequence of tasks or steps that the system should go through, along with the data it should receive and the responses it should generate [41]. In this tradition, the use case focuses on *what the system should do*, abstracting out the individual and her or his motivations [42, 43]. This is often useful for systems analysis, but not for our purposes in this paper.

We *want* a testing approach that helps us imagine why someone would be unhappy with the behavior of a system and how unhappy it would make them, because that is the type of information that helps us estimate the cost of this (mis)behavior, which is exactly what we need for cost/benefit analysis.

References

1. *United States v. Carroll Towing Co.*, vol. 159, 1947 p. 169 (United States Circuit Court of Appeals, Second Circuit (Learned Hand)).
2. *The T.J. Hooper*, vol. 60, 1932 p. 737 (United States Court of Appeals, Second Circuit).
3. S. Youngblood, "DoD Verification, Validation and Accreditation Recommended Practices Guide," *Book DoD Verification, Validation and Accreditation Recommended Practices Guide*, Series DoD Verification, Validation and Accreditation Recommended Practices Guide, ed., Editor ed.^eds., 1996, pp.
4. R.G. Sargent, "An Overview of Verification and Validation of Simulation Models," *Proc. Winter Simulation Conference*, Society for Computer Simulation, 1987.
5. M. Kilikauskas and D. Hall, "The Use of M&S VV&A as a Risk Mitigation Strategy in Defense Acquisition," *Journal of Defense Modeling and Simulation*, vol. 2, no. 4, 2005, pp. 209-216.
6. United States Department of Defense: Defense Modeling and Simulation Office, "Online M&S Glossary (DoD 5000.59-M)," *Book Online M&S Glossary (DoD 5000.59-M)*, Series Online M&S Glossary (DoD 5000.59-M), ed., Editor ed.^eds., pp.
7. G.E.P. Box and N.R. Draper, *Empirical Model-Building and Response Surfaces*, Wiley, 1987.
8. D.K. Pace, "The Value of a Quality Simulation Conceptual Model," 2002; http://www.modelingandsimulation.org/MandS0101/Pace0101.html.
9. R.G. Sargent, "Verification and Validation of Simulation Models," *Proc. 31st Winter Simulation Conference*, Society for Computer Simulation, 2000.
10. P. Castro, ed., Modeling and Simulation in Manufacturing and Defense Acquisition, Enhancements for 21st Century Manufacturing and Defense Acquisition, National Academies Press, 2002.
11. S. Youngblood, *RPG Special Topics: Requirements*, Defense Modeling and Simulation Office, 2004.
12. J. Musa, *Software Reliability Engineering*, McGraw-Hill, 1999.
13. C. Kaner, "Measurement issues and software testing (Keynote address)," *Book Measurement issues and software testing (Keynote address)*, Series Measurement issues and software testing (Keynote address), ed., Editor ed.^eds., 2001, pp.
14. D. Hoffman, "A taxonomy of test oracles," *Book A taxonomy of test oracles*, Series A taxonomy of test oracles, ed., Editor ed.^eds., 1998, pp.
15. J. Bach, "General Functionality and Stability Test Procedure for Microsoft Windows 2000 Application Certification," 1999; http://www.satisfice.com/tools/procedure.pdf.
16. C. Kaner and J. Bach, "A Course in Black Box Software Testing: Fundamental Issues in Software Testing: The Oracle Problem " 2005; http://www.testingeducation.org/k04/video/OverviewPartC.wmv.
17. J. Bach and M. Bolton, "Rapid Software Testing (course slides)," *Book Rapid Software Testing (course slides)*, Series Rapid Software Testing (course slides), Version 2.1.3 ed., Editor ed.^eds., 2007, pp.
18. J. Campanella, *Principles of Quality Costs*, ASQ Quality Press, 1999.
19. W.P. Keeton, et al., *Prosser and Keeton on Torts*, West Publishing, 1984.
20. F.M. Gryna, "Quality Costs," *Juran's Quality Control Handbook*, 4th ed., McGraw Hill, 1988.
21. A.V. Feigenbaum, *Total Quality Control* McGraw-Hill, 1991.
22. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2005.

23. M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 2000.
24. W.P. Keeton, et al., *Products Liability and Safety: Cases and Materials*, Foundation Press, 1989.
25. R.A. Posner, *Tort Law: Cases and Economic Analysis*, Little Brown & Co., 1982.
26. *Grimshaw v. Ford Motor Co.*, vol. 174, 1981 p. 348 (California Court of Appeal).
27. *General Motors Corp. v. Johnston*, vol. 592, 1992 p. 1054 (Supreme Court of Alabama).
28. C. Kaner and D.L. Pels, *Bad Software: What To Do When Software Fails*, Wiley, 1998.
29. American Law Institute, *Restatement of the Law Third, Torts: Products Liability*, American Law Institute Publishers, 1998.
30. United States Coast Guard, "Risk-based Decision-making Guidelines," http://www.uscg.mil/hq/g-m/risk/e-guidelines/hazop.htm.
31. D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach*, Addison-Wesley, 2000.
32. D.C. Gause and G.M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.
33. S. Robertson and J. Robertson, *Mastering the Requirements Process*, Addison-Wesley, 1999, p. 404.
34. C. Kaner, "Interest Analysis," 2006; http://www.testingeducation.org/k04/video/CopyrightInterestAnalysis.wmv.
35. P.K. Davis, "Generalizing Concepts of Methods of Verification, Validation and Accreditation (VV&A) for Military Simulations," *RAND*, 1992.
36. D. Brade, "Enhancing M&S Accreditation by Structuring Verification and Validation Results," *Proc. Winter Simulation Conference*, Society for Computer Simulation, 2000.
37. C. Kaner, "The power of 'What If…' and nine ways to fuel your imagination: Cem Kaner on scenario testing," *Book The power of 'What If…' and nine ways to fuel your imagination: Cem Kaner on scenario testing*, Series The power of 'What If…' and nine ways to fuel your imagination: Cem Kaner on scenario testing 5, ed., Editor ed.^eds., 2003, pp. 16-22.
38. J.M. Carroll, ed., Scenario-Based Design: Envisioning Work and Technology in System Development, John Wiley & Sons, 1995.
39. R.M.B. Young, P.B., "The use of scenarios in human-computer interaction research: Turbocharging the tortoise of cumulative science," *Proc. CHI+GI'87: Conference on Human Factors in Computing Systems and Graphics Interface*, ACM Press, 1987, pp. 291-296.
40. C. Kaner and J. Bach, "A Course in Black Box Software Testing: Scenario Testing," 2004; http://www.testingeducation.org/BBST/ScenarioTesting.html.
41. J. Rumbaugh, et al., *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.
42. A. Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2001.
43. S. Adolph and P. Bramble, *Patterns for Effective Use Cases*, Pearson Education, 2003.
44. C. Kaner, "Quality cost analysis: Benefits and risks," *Software QA*, vol. 3, no. 1, 1996, pp. 23.

Acknowledgements

These notes are partially based on research that was supported by NSF Grant 0629454 "Learning Units on Law and Ethics in Software Engineering" to Cem Kaner. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s)and do not necessarily reflect the views of the National Science Foundation.

Much of the quality cost discussion, including Figure 1 updates work published in [44]. Much of the scenario discussion was previously published in [37] and [40].