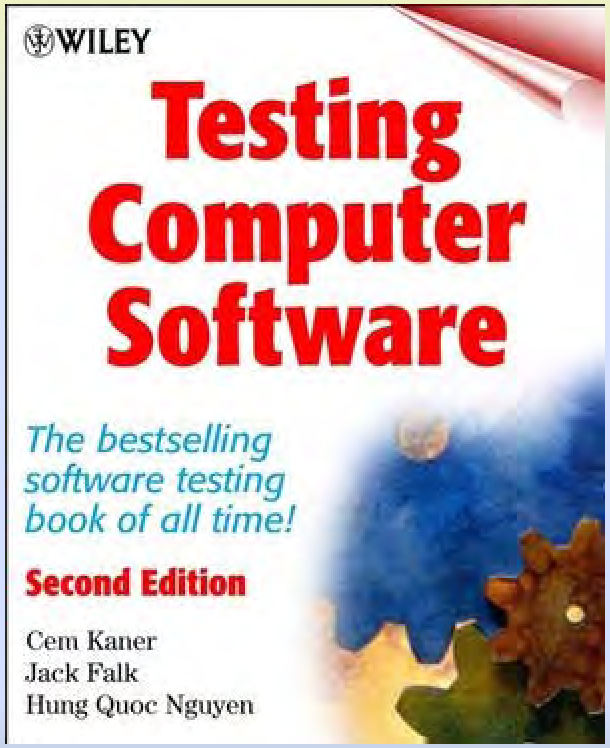
Good Enough V&V for Simulations: Some Possibly Helpful Thoughts from the Law & Ethics of Commercial Software

Cem Kaner, J.D., Ph.D.
Professor of Software Engineering
Florida Institute of Technology

Stephen J. Swenson
AEgis Technologies Group

Hi, I'm Cem Kaner

Best known in software testing



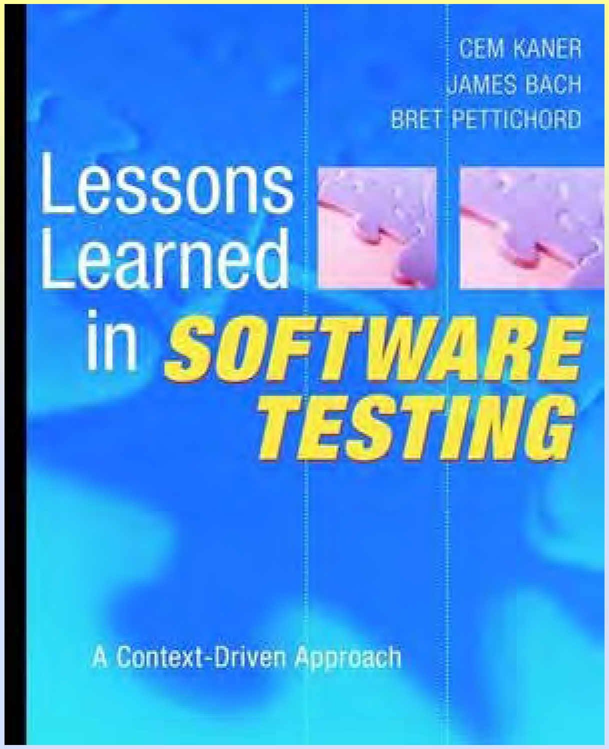
WILEY

Testing Computer Software

The bestselling software testing book of all time!

Second Edition

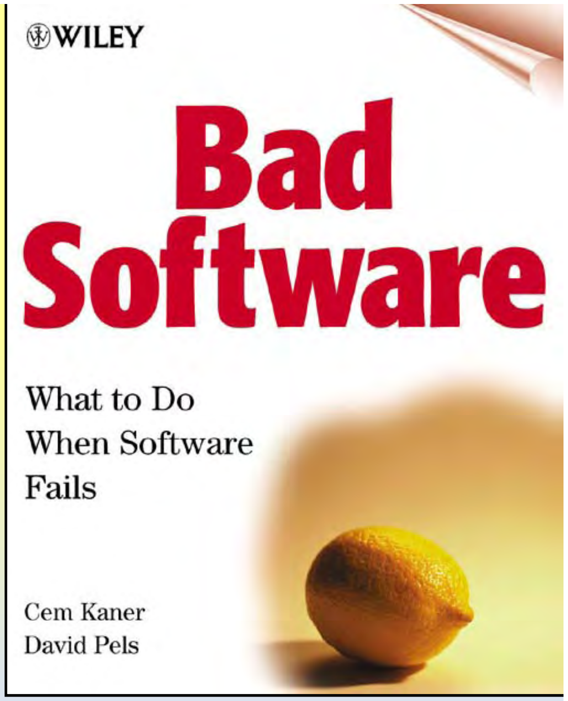
Cem Kaner
Jack Falk
Hung Quoc Nguyen



CEM KANER
JAMES BACH
BRET PETTICHORD

Lessons Learned in SOFTWARE TESTING

A Context-Driven Approach



WILEY

Bad Software

What to Do When Software Fails

Cem Kaner
David Pels

And software-related commercial law

A bit of psychology (human factors) background, too

On the other hand

- NO simulation experience
- NO military experience
- So with respect to your world,

I don't know what I'm talking about

- If this talk is useful to you, it is as a bridge:

  Law / psychology / ethics / street sense of commercial software V&V

  ←→

  Simulation VV&A

I think we might face a similar cluster of problems

- Complex real-world problem domain being imperfectly partially-represented in software
- Potential-solution space infinitely large compared to potential budget
- Multiple stakeholders with conflicting interests and needs and quality criteria
- Evolving requirements
- Incomplete, non-authoritative specifications
- Waterfall-style development is just a way to make sure that you make your most expensive mistakes at the start of the project

- But these are problems, not excuses.
- The challenge is to do well in this type of context

For more information, especially about slides 22-39

1. Free course videos on many aspects of software testing, at www.testingeducation.org/BBST
2. Rework on bug advocacy materials in progress, check the site after May 1, 2008 for new videos on error definition, application of oracles etc. (193 slides, 3+ hours of video, free). These slides reflect those materials.
3. Free practitioner courses (under course mgmt system) offered to members by Association for Software Testing. For access to materials (copy/adapt to your use for free) contact me, kaner@kaner.com

These notes are partially based on research that was supported by NSF Grants EIA-0113539 ITR/SY+PE: “Improving the Education of Software Testers” and CCLI-0717613 “Adaptation & Implementation of an Activity-Based Online or Hybrid Course in Software Testing” and SES-0629454 “Learning Units on Law and Ethics in Software Engineering.” 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.

If the perfect is the enemy of the good enough…

1. What is good enough software?
2. What is good enough software development practice?
3. What is good enough VV&A?

Let's start with some thoughts on standards of accountability (for work that is **not** good enough)

Four approaches to standards of accountability

1. Strict compliance

ALL DEFECTS ARE THE FAULT – AND RESPONSIBILITY – OF THE VENDOR

Characteristic development practices:

- IEEE / CMM standards, heavyweight
- Intense quality control
- Intense documentation

Characteristic problems

- What's a defect?
- Can anyone make zero-defect software?
- In the simulation marketspace?

- Appropriate rule for commodity contracts
- Does it create a reverse lottery for software contracts?

2. Negotiated liability

ALLOCATE RISK BY CONTRACT

Characteristics:

- Thoroughly-negotiated contracts
- Often, much work is time/materials—either for initial work or for "scope change" enhancements
- Verification dominates validation—if the rule for payment is conformance to the contract
- Quality criteria, including any warranties, are specified in the contract. Little additional regulatory oversight.

Common problems:

- May be sensible if all parties have equal sophistication and bargaining power
- Third-party beneficiaries (e.g. troops) may be at risk if they are not well represented by the principals

2. Negotiated liability

ALLOCATE RISK BY CONTRACT

Interesting challenge:

- What if the client cannot specify its needs in the contract?
  - It doesn't know its needs
  - It doesn't know what is possible
  - It doesn't know how to accurately estimate costs and risks
  - It has multiple stakeholders with shifting balances of power
- What if the client wants more than it can afford, and cannot prioritize? (Goal: get the most that it can in the time, cost available. Challenge: insufficient ability to estimate cost / risk)

This creates intense pressure to adopt agile practices:

- Which rely on TRUST and COMPETENCE
- How do you specify THESE in the contract?

3. Balanced optimization

NEGLIGENCE LIABILITY: EXPLICIT COST / BENEFIT ANALYSIS

- No expectation of perfection in any aspect of any process
- The demands are for "reasonable care" (products liability) and "reasonably competent performance" (professional negligence)
- Prospective analysis: Duty / Breach / Causation / Harm
- Retrospective analysis:
  - Given this harm
  - What could have avoided it?
  - How much would avoidance have cost?
- Compare:

Total expected harm to society

Total expected cost of avoidance

3. Balanced optimization

NEGLIGENCE LIABILITY: EXPLICIT COST / BENEFIT ANALYSIS

- Negligence: Expected harm > Expected cost
- Manufacturer / vendor MAY be liable:
  - For following inappropriate or inadequate industry standards
  - For using outdated practices
  - For deferring known bugs
  - For failing to use techniques that would have caught the bugs
  - For inappropriate requirements analysis
  - Etc.
- REALITY ALERT
  - THIS ANALYSIS IS HYPOTHETICAL
  - IN U.S. LAW, THERE IS NO TORT OF SOFTWARE MALPRACTICE

3. Balanced optimization

NEGLIGENCE LIABILITY: EXPLICIT COST / BENEFIT ANALYSIS

- Benefit of this approach
  - Encourages continuous improvement of all aspects of design and development
- Risk
  - Estimation of expected risk / benefit carries cost and is done under uncertainty
    - How reasonable is accountability over small improvements?
    - How trustworthy are the estimators?
    - How angry are vendors held accountable for good-faith estimates later found wrong (especially in close cases)?
    - How much does it cost to win this type of lawsuit?
  - Perceived unfairness was a key factor in “tort reform” movement

4. Self-focused optimization

QUALITY-COST ANALYSIS

Negligence-style analysis that considers only costs to the vendor…

The primary goal of quality engineering is often described as minimization of quality-related costs.

American Society for Quality defines **cost of quality** of a product as the total amount the company spends to achieve and cope with the quality of its product.

This includes the company's investments in improving quality, and its expenses arising from inadequate quality.

- » Total Cost of Quality =
  - » Costs of Prevention +
  - » Costs of Appraisal +
  - » Costs of Internal Failures +
  - » Costs of External Failure.

Quality-Related Costs

Total Cost of Quality =

Prevention + Appraisal + Internal Failure + External Failure costs.

Costs of Prevention

- Preventing software errors, documentation errors, and any other sources of customer dissatisfaction

Costs of Appraisal

- Looking for defects (all types of inspection and testing)

Costs of Internal Failures

- (Typically) failures during development and their consequences

Costs of External Failures

- The impact on the vendor of failures of the product in the hand of external users

For examples, see
http://www.kaner.com/pdfs/Quality_Cost_Analysis.pdf

Categorizing Quality Costs

Prevention	*Appraisal*
• Staff training • Requirements analysis • Early prototyping • Fault-tolerant design • Defensive programming • Usability analysis • Clear specification • Accurate internal documentation • Pre-purchase evaluation of the reliability of development tools	• 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 • Direct cost of late shipment • Opportunity cost of late shipment	• Technical support calls • Answer books (for Support) • Investigating complaints • Refunds and recalls • Interim bug fix releases • Shipping updated product • Supporting multiple versions in the field • PR to soften bad reviews • Lost sales • Lost customer goodwill • Reseller discounts to keep them selling the product • Warranty, liability costs

Quality-Related Costs

Total Cost of Quality =

Prevention + Appraisal + Internal Failure + External Failure costs.

What about costs to the customer?

Quality / Cost analysis doesn't ask us to minimize customer's costs. Can we ignore them?

Remember the Pinto (and the Mustang)?

"Benefits and Costs Relating to Fuel Leakage Associated with the Static Rollover Test Portion of FMVSS 208"

Benefits -- Savings to Ford	
180 burn deaths	$200,000 each
180 serious burn injuries	67,000 each
2100 burned vehicles	700 each
Total Benefit	**$49.5 million**
Costs of Fixing the Problem	
11 million cars	$11 each
1.5 million trucks	$11 each
Total Cost	**$137 million**

Quality-Related Costs

This analysis ignores ***externalized*** failure costs -- the costs absorbed by the customer.

Seller: external 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 • Preparing answer books • Investigating complaints • Refunds and recalls • Interim bug fix releases • Shipping updated product • Supporting multiple versions in the field • PR to soften harsh reviews • Lost sales • Lost customer goodwill • Reseller discounts to keep them selling the product • Warranty, liability costs • Gov't investigations	• Wasted time • Lost data • Lost business • Embarrassment • Frustrated employees quit • Demos or presentations to potential customers fail because of the software • Failure during tasks that can only be done once • Cost of replacing product • Reconfiguring the system • Cost of recovery software • Cost of tech support • Injury / death

Theoretical justifiability of self-focused optimization?

Coase theorem

- Widely cited in "law and economics" approach to jurisprudence
- Highly influential among judges appointed since 1980
- Two parties can allocate risk-related costs in a cost-neutral way
  - (you bear the risk of accidents and I give you a discount equal to the expected liability for accidents)
- Given sufficient competition or other incentive to bargain, and zero transaction costs:
  - it doesn't matter how society allocates liability because the parties will allocate the liability-related costs appropriately
- IF this actually applies, then quality-related costs include all customer losses because they show up as external failure costs

Recommendations for practice?

1. Strict compliance is probably unrealistic for M&S work
2. Negotiated accountability is an interesting option IF you can adopt an agile model
   – The essence of agile development is minimization of cost of change, in the service of the actual needs of the customer. Agile practices are often highly disciplined, but disciplined differently from document-driven practices.
3. Balanced optimization provides an interesting model.
   – Engineering model, not liability model
   – "Good work" not "no lawsuits"
4. Self-focused optimization is easier to justify to vendor executives, and arguments framed in terms of it are persuasive to vendor executives
   – But customers should mistrust the self-focused optimizer because s/he ignores their risks

Suppose we embrace an agile or optimization model…

The presentation (1/2 hour) is probably over at this point, but

- The following slides
- The paper
- And the course videos (especially on the objectives of testing, on bug advocacy, and on test design)

might help you consider the impacts on testing of adopting an agile or optimization model instead of failing in a fruitless effort to apply the IEEE-Software-Engineering-Standards approaches to development efforts that do not, and can not, start with authoritative requirements or specifications.

Given that challenge, two questions that I look at in the next slides are:

- What kinds of problems are we looking for?
- What is the referent?

What kinds of problems are we looking for?

Quality is value to some person

-- Jerry Weinberg

Under this view:

- Quality is inherently subjective
  - Different stakeholders will perceive the same product as having different levels of quality

Testers look for different things …

… for different stakeholders

Software error

An attribute of a software product

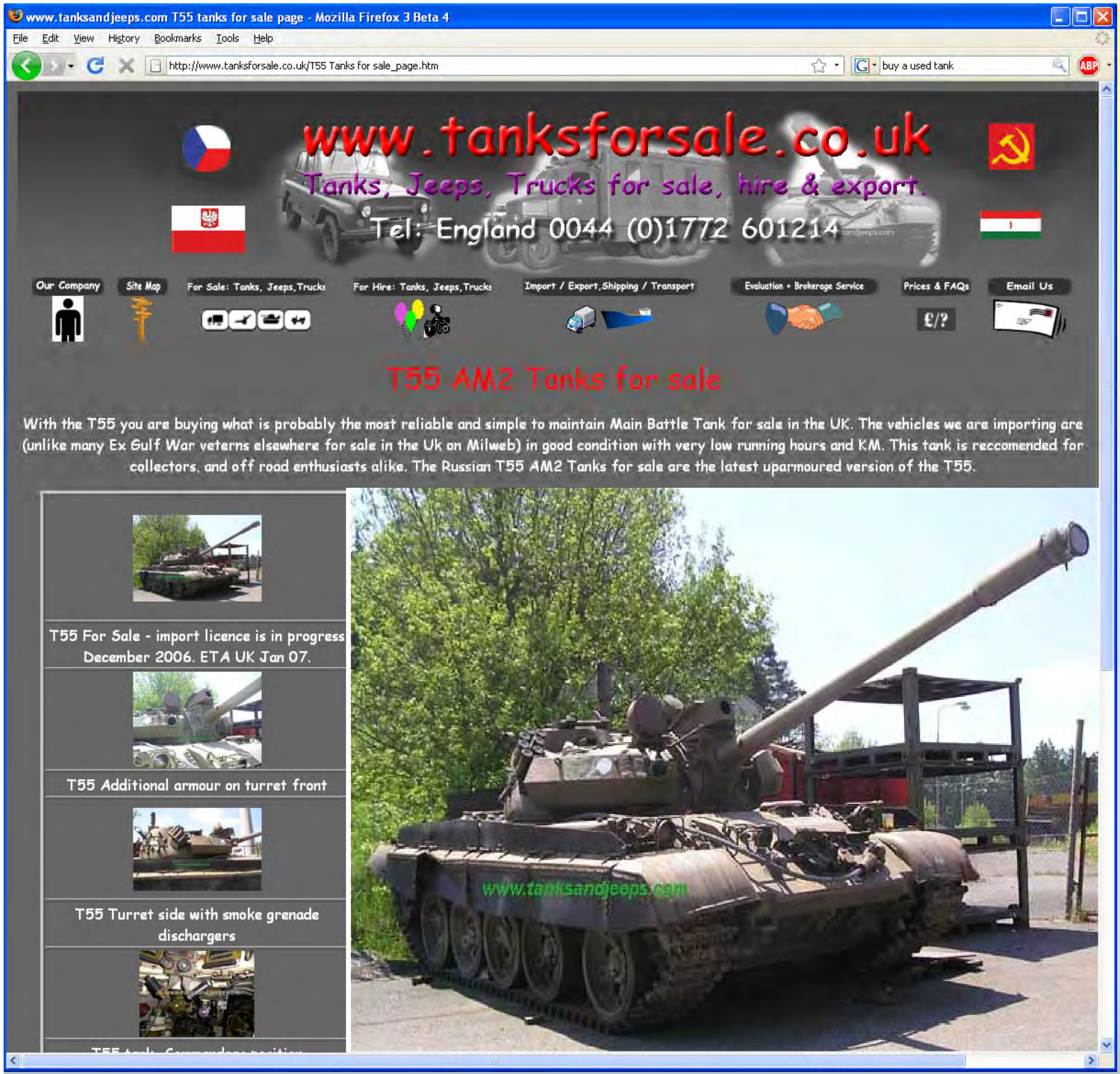
- that reduces its value to a favored stakeholder
- or increases its value to a disfavored stakeholder
- without a sufficiently large countervailing benefit.

An error:

- May or may not be a coding error
- May or may not be a functional error

Any threat to the value of the product **to any stakeholder who matters.**

James Bach


www.tanksandjeeps.com T55 tanks for sale page - Mozilla Firefox 3 Beta 4
File Edit View History Bookmarks Tools Help
http://www.tanksforsale.co.uk/T55 Tanks for sale_page.htm
buy a used tank
www.tanksforsale.co.uk
Tanks, Jeeps, Trucks for sale, hire & export.
Tel: England 0044 (0)1772 601214
Our Company
Site Map
For Sale: Tanks, Jeeps, Trucks
For Hire: Tanks, Jeeps, Trucks
Import / Export, Shipping / Transport
Evaluation + Brokerage Service
Prices & FAQs
Email Us
£/?
www.tanksandjeeps.com

T55 AM2 Tanks for sale

With the T55 you are buying what is probably the most reliable and simple to maintain Main Battle Tank for sale in the UK. The vehicles we are importing are (unlike many Ex Gulf War veterns elsewhere for sale in the Uk on Milweb) in good condition with very low running hours and KM. This tank is reccomended for collectors, and off road enthusiasts alike. The Russian T55 AM2 Tanks for sale are the latest uparmoured version of the T55.

T55 For Sale - import licence is in progress December 2006. ETA UK Jan 07.

T55 Additional armour on turret front

T55 Turret side with smoke grenade dischargers

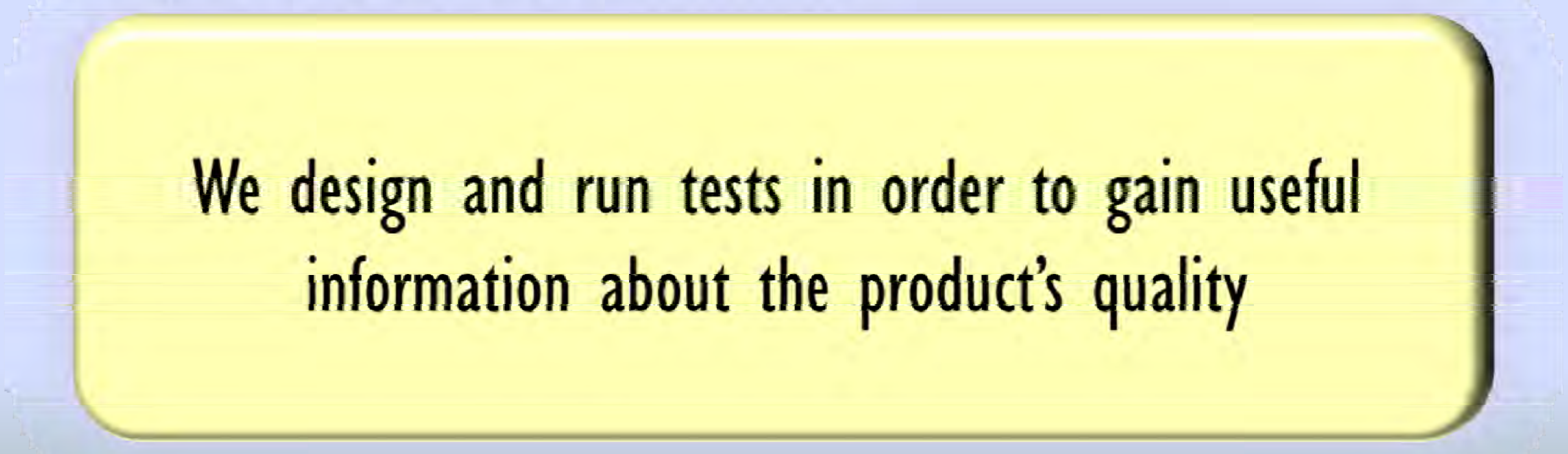
Not every limitation on value is a bug:

Is a car defective because it isn't designed to withstand a 30-mph collision?

We do know how to make traveling machines that are that rugged…

Software testing

- is an empirical
- technical
- investigation
- conducted to provide stakeholders
- with information
- about the quality
- of the product or service under test



We design and run tests in order to gain useful information about the product's quality

Verification

IF you have contracted for delivery of software, and the contract contains a complete and correct specification,

verification-oriented testing can answer the question,

Do we have to pay for this software?

Verification

Verification-oriented testing can answer the question:

Do we have to pay for this software?

But if…

- You're doing in-house development
- With evolving requirements (and therefore an incomplete and non-authoritative specification).

Verification only begins to address the critical question:

Will this software meet our needs?

Verification / Validation

In commercial system testing,

The primary reason we do verification testing is to assist in validation.

Will this software meet our needs?

(obviously, in M&S, this overlaps with accreditation)

System testing (validation)

Designing system tests is like doing a requirements analysis. They rely on similar information but use it differently.

- The requirements analyst tries to foster agreement about the system to be built. The tester exploits disagreements to predict problems with the system.
- The tester doesn't have to reach conclusions or make recommendations about how the product should work. Her task is to expose credible concerns to the stakeholders.
- The tester doesn't have to make the product design tradeoffs. She exposes the consequences of those tradeoffs, especially unanticipated or more serious consequences than expected.
- The tester doesn't have to respect prior agreements. (Caution: testers who belabor the wrong issues lose credibility.)
- The system tester's work cannot be exhaustive, just useful.

Testing is always a search for information

- Find important bugs, to get them fixed
- Assess the quality of the product
- Help managers make release decisions
- Block premature product releases
- Help predict and control product support costs
- Check interoperability with other products
- Find safe scenarios for use of the product
- Assess conformance to specifications
- Certify the product meets a particular standard
- Ensure the testing process meets accountability standards
- Minimize the risk of safety-related lawsuits
- Help clients improve product quality & testability
- Help clients improve their processes
- Evaluate the product for a third party

Different objectives require different testing tools and strategies and will yield different tests, different test documentation and different test results.

Test techniques

A test technique is essentially a recipe, or a model, that guides us in creating specific tests. Examples of common test techniques:

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- All-pairs combination testing
- Data flow testing
- Build verification testing
- State-model based testing
- High volume automated testing
- Printer compatibility testing
- Testing to maximize statement and branch coverage

We pick the technique that provides the best set of attributes, given the information objective and the context.

Techniques differ in how to define a good test

Power. When a problem exists, the test will reveal it

Valid. When the test reveals a problem, it is a genuine problem

Value. Reveals things your clients want to know about the product or project

Credible. Client will believe that people will do the things done in this test

Representative of events most likely to be encountered by the user

Non-redundant. This test represents a larger group that address the same risk

Motivating. Your client will want to fix the problem exposed by this test

Maintainable. Easy to revise in the face of product changes

Repeatable. Easy and inexpensive to reuse the test.

Performable. Can do the test as designed

Refutability: Designed to challenge basic or critical assumptions (e.g. your theory of the user's goals is all wrong)

Coverage. Part of a collection of tests that together address a class of issues

Easy to evaluate.

Supports troubleshooting. Provides useful information for the debugging programmer

Appropriately complex. As a program gets more stable, use more complex tests

Accountable. You can explain, justify, and prove you ran it

Cost. Includes time and effort, as well as direct costs

Opportunity Cost. Developing and performing this test prevents you from doing other work

Differences in emphasis on different test attributes

- **Scenario testing:**
- complex stories that capture how the program will be used in real-life situations
  - Good scenarios focus on validity, complexity, credibility, motivational effect
  - The scenario designer might care less about power, maintainability, coverage, reusability
- **Risk-based testing:**
- Imagine how the program could fail, and try to get it to fail that way
  - Good risk-based tests are powerful, valid, non-redundant, and aim at high-stakes issues (refutability)
  - The risk-based tester might not care as much about credibility, representativeness, performability—we can work on these after (if) a test exposes a bug

What is the referent?

The referent is the best or most appropriate codified body of information available that describes characteristics and behavior of the reality represented in the simulation from the perspective of validation assessment for intended use of the simulation.

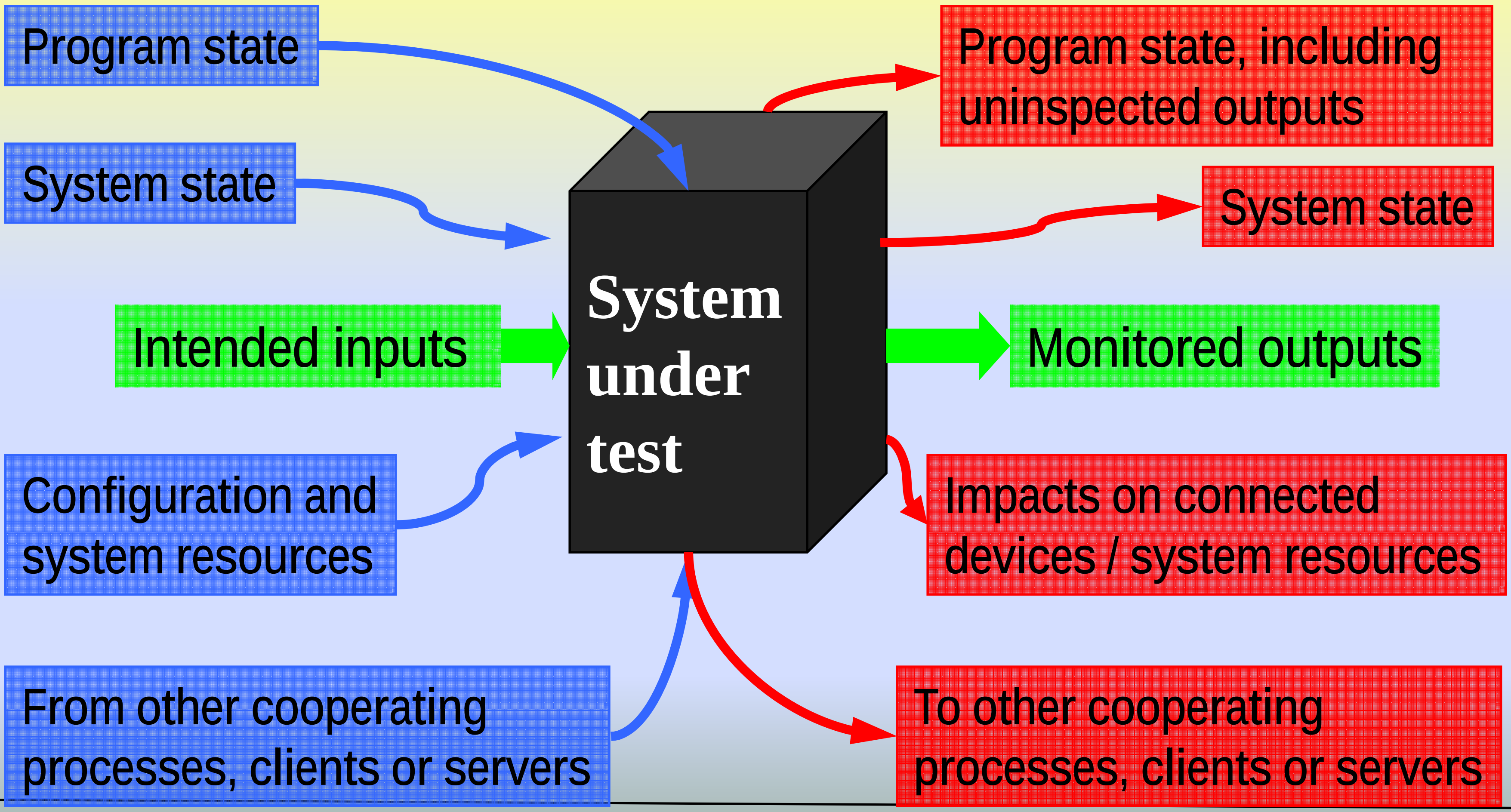
D. K. Pace: The Referent Study Final Report

Pace then suggested heuristics for finding / using referents.

Analogous to commercial oracles…

Even in the simple commercial world, oracles are inadequate

No test fully specifies all dimensions of input and configuration of any test, nor can it list every imaginable outcome (including unacceptable but unexpected side effects) of each test. Example: if you test a program to see if it gets 5 when it adds 2+3, do you test to see whether it oops also erases your hard disk? *Based on notes from Doug Hoffman.*


Program state
System state
Intended inputs
Configuration and system resources
From other cooperating processes, clients or servers
System under test
Program state, including uninspected outputs
System state
Monitored outputs
Impacts on connected devices / system resources
To other cooperating processes, clients or servers

Use oracles to resolve arguments

An oracle is the principle or mechanism by which you recognize a problem.

"..it works"

really means...

"...it appeared to meet some requirement to some degree."

Use oracles to resolve arguments

An oracle is the principle or mechanism by which you recognize a problem.

“..it doesn’t work”

often means…

“...it violates my expectations.”

Some commercial oracle heuristics

Rather than thinking of oracles as deterministic rules, consider them as heuristics that are useful but not always right. Here are typical examples of commonly used heuristics, especially in the face of a non-authoritative or non-complete specification.

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

Supplement these, for M&A

About Cem Kaner

- Professor of Software Engineering, Florida Tech
- Research Fellow at Satisfice, Inc.

I've worked in all areas of product development (programmer, tester, writer, teacher, user interface designer, software salesperson, organization development consultant, as a manager of user documentation, software testing, and software development, and as an attorney focusing on the law of software quality.)

Senior author of three books:

- *Lessons Learned in Software Testing* (with James Bach & Bret Pettichord)
- *Bad Software* (with David Pels)
- *Testing Computer Software* (with Jack Falk & Hung Quoc Nguyen).

My doctoral research on psychophysics (perceptual measurement) nurtured my interests in human factors (usable computer systems) and measurement theory.