Paradigms of Black Box Software Testing

Cem Kaner -- kaner@kaner.com

James Bach -- james@satisfice.com

November, 1999

The Puzzle

- **Black box testing groups vary widely in their approach to testing.**
- **Tests that seem essential to one group seem uninteresting or irrelevant to another.**
- **Big differences can appear even when both groups are composed of intelligent, experienced, dedicated people.**
- **Why?**

Paradigms

- **A paradigm provides a structure of thinking about an area.**
  - **Typically, the description of a paradigm includes one or a few paradigmatic cases -- key example. Much of the reasoning within the paradigm is based on analogies to these key cases.**
  - **The paradigm creates a mainstream of thinking--it provides insights and a direction for further research or work. But it implicitly also defines limits on what is relevant, interesting, or possible. Things outside of the paradigm are uninteresting. People who solve the types of puzzles that are characteristic of the paradigm are respected, whereas people who solve other types of puzzles instead are outsiders, and not well respected.**
  - **A testing paradigm would define the types of tests that are (to the person operating under this paradigm) relevant and interesting.**

Paradigms (Kuhn)

- **See Thomas Kuhn, *The Structure of Scientific Revolutions.***
- A paradigm is a model based on a shared experiment that includes law, theory, application and instrumentation together and from which springs particular coherent traditions of scientific research.
- A paradigm includes a concrete scientific achievement as a locus of professional commitment, prior to the various concepts, laws, theories and points abstracted from it.
- ***The pre-paradigm period . . . is regularly marked by frequent and deep debate over legitimate methods, problems, and standards of solution, though these serve rather to define schools than to produce agreement.***

Black Box Testing Paradigms

- **This list reflects our (Kaner's / Bach's) observations in the field and is not exhaustive.**
- **We put one on the list if we've seen credible testers drive their thinking about black box testing in the way we describe. A paradigm for one person might merely be a technique for another.**
  - **We recommend that you try to master the "right" combination of a few approaches. They are not mutually exclusive. The right combination will depend on your situation.**

A List of Paradigms

- **Domain driven**
- **Stress driven**
- **Specification driven**
- **Risk driven**
- **Random / statistical**
- **Function**
- **Regression**
- **Scenario / use case / transaction flow**
- **User testing**
- **Exploratory**

Domain Testing

- **Tag lines**
  - **"Try ranges and options."**
  - **"Subdivide the world into classes."**
- **Fundamental question or goal**
  - **A stratified sampling strategy. Divide large space of possible tests into subsets. Pick best representatives from each set.**
- **Paradigmatic case(s)**
  - **Equivalence analysis of a simple numeric field**
  - **Printer compatibility testing**

Domain Testing

- **Strengths**
  - **Find highest probability errors with a relatively small set of tests.**
  - **Intuitively clear approach, generalizes well**
- **Blind spots**
  - **Errors that are not at boundaries or in obvious special cases.**
  - **Also, the actual domains are often unknowable.**

Stress Testing

- **Tag line**
  - **"Overwhelm the product."**
  - **"Drive it through failure."**
- **Fundamental question or goal**
  - **Learn about the capabilities and weaknesses of the product by driving it through failure and beyond. What does failure at extremes tell us about changes needed in the program's handling of normal cases?**
- **Paradigmatic case(s)**
  - **High volumes of data, device connections, long transaction chains**
  - **Low memory conditions, device failures, viruses, other crises.**
- **Strengths**
  - **Expose weaknesses that will arise in the field. Lots of security holes found this way**
- **Blind spots**
  - **Weaknesses that are not made more visible by stress.**

Specification-Driven Testing

- **Tag line:**
  - **"Verify every claim."**
- **Fundamental question or goal**
  - **Check the product's conformance with every statement in every spec, requirements document, etc.**
- **Paradigmatic case(s)**
  - **Traceability matrix, tracks test cases associated with each specification item.**
- **Strengths**
  - **Critical defense against warranty claims, fraud charges, loss of credibility with customers.**
- **Blind spots**
  - **Any issues not in the specs or treated badly in the specs.**

Risk-Based Testing

- **Tag line**
  - **"Find big bugs first."**
- **Fundamental question or goal**
  - **Prioritize the testing effort in terms of the relative risk of different areas or issues we could test for.**
- **Paradigmatic case(s)**
  - **Equivalence class analysis, reformulated.**
  - **Test in order of frequency of use.**
  - **Stress tests, error handling tests, security tests, tests looking for predicted or feared errors.**
  - **Sample from predicted-bugs list.**

Risk-Based Testing

- **Strengths**
  - **Optimal prioritization (assuming we correctly identify and prioritize the risks)**
  - **High power tests**
- **Blind spots**
  - **Risks that were not identified or that are surprisingly more likely.**

Evaluating Risk

- **My (Kaner's) concern is that risk analysis is an art. Some "risk-driven" testers seem to operate too subjectively. How will I know what level of coverage that I've reached? How do I know that I haven't missed something critical?**
- **Bob Stahl's approach:**
  - **Bring together knowledgeable people from different groups.**
  - **List (perhaps by brainstorming) the various areas of the program and other testing issue (such as config testing).**
  - **Estimate and rank order "risk" of each area in terms of:**
    - **probability that a given failure could occur**
    - **possible consequences (severity) of the failure**
  - **Prioritize testing in terms of "risk"**
- **Another approach: Operational Profile Testing (Musa)**

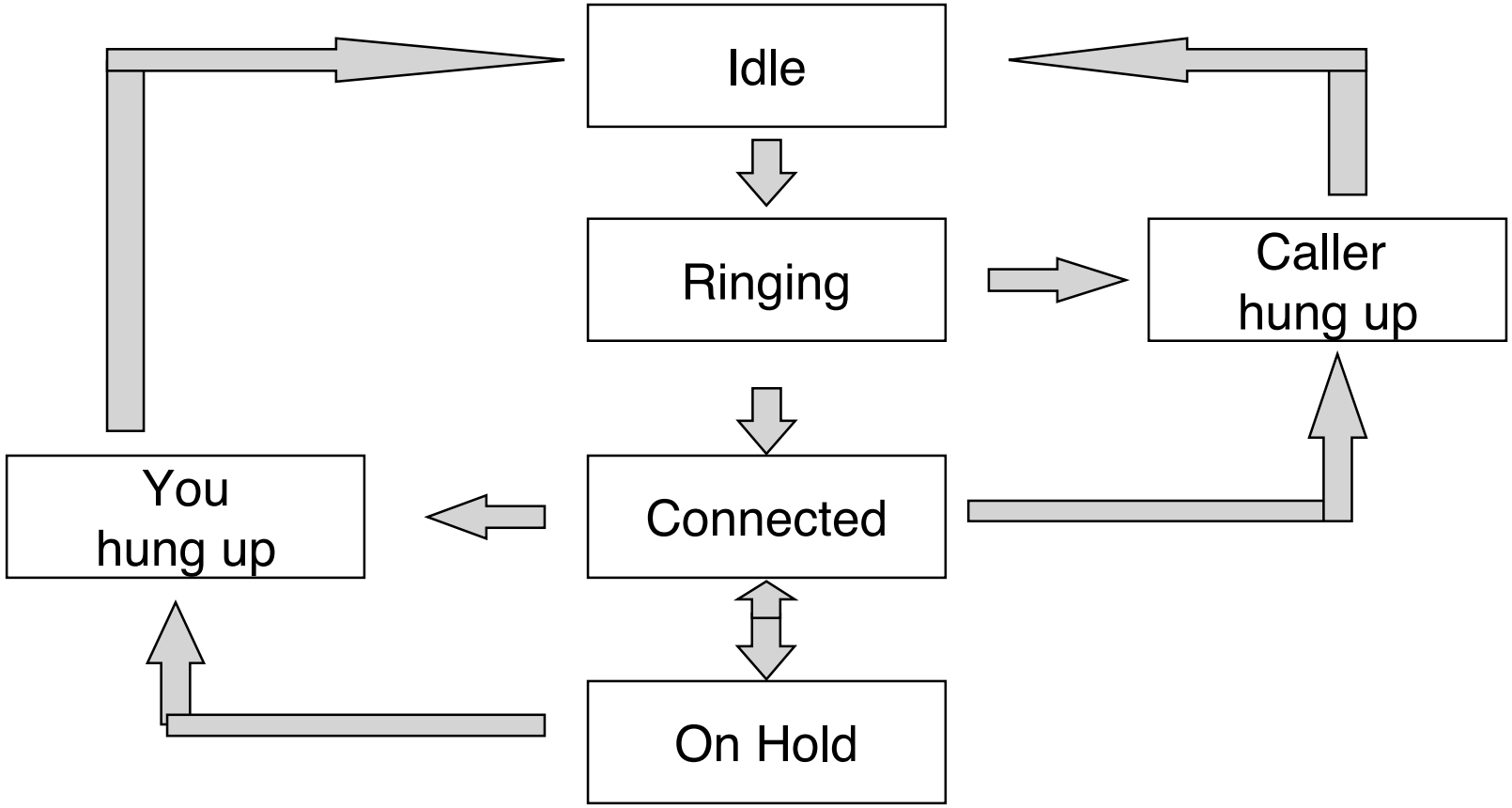
Evaluating Risk

- **Another variation (exploratory/evolutionary):**
  - **List bad things that can happen from using the program.**
  - **List the ways that the bad things can happen.**
  - **List the types of tests needed to check for the bad things.**
  - **Post the list publicly, solicit feedback on prioritization, and extend the list using feedback, updated docs, and testing experience.**
- **And another variation (Tripos style):**
  - **Analyze project factors, product elements, and quality criteria in order to identify sources of risk**
  - **Focus testing in areas of highest potential risk**
  - ***Use test results to refine the risk analysis***
  - **Be careful not to completely neglect low-risk areas--Your risk analysis might be wrong.**

Random / Statistical Testing

- **Tag line**
  - **"High-volume testing with new cases all the time."**
- **Fundamental question or goal**
  - **Have the computer create, execute, and evaluate huge numbers of tests.**
- **Paradigmatic case(s)**
  - **Oracle-driven, validate a function or subsystem (such as function equivalence testing)**
  - **Stochastic (state transition) testing looking for specific failures (assertions, leaks, etc.)**
  - **Statistical reliability estimation**
  - **Partial or heuristic oracle, to find some types of errors without overall validation.**

The Need for Stochastic Testing: An Example



- Refer to Testing Computer Software, pages 20-21

Random / Statistical Testing

- **Strengths**
  - **Regression doesn't depend on same old test every time.**
  - **Partial oracles can find errors in young code quickly and cheaply.**
  - **Less likely to miss internal optimizations that are invisible from outside.**
  - **Can detect failures arising out of long, complex chains that would be hard to create as planned tests.**
- **Blind spots**
  - **Need to be able to distinguish pass from failure. Too many people think "Not crash = not fail."**
  - **Also, these methods will often cover many types of risks, but will obscure the need for other tests that are not amenable to automation.**

Function Testing

- **Tag line**
  - **"Black box unit testing."**
- **Fundamental question or goal**
  - **Test each function thoroughly, one at a time.**
- **Paradigmatic case(s)**
  - **Spreadsheet, test each item in isolation.**
  - **Database, test each report in isolation**
- **Strengths**
  - **Thorough analysis of each item tested**
- **Blind spots**
  - **Misses interactions, misses exploration of the benefits offered by the program.**

Regression Testing

- **Tag line**
  - **"Repeat testing after changes."**
- **Fundamental question or goal**
  - **Manage the risks that (a) a bug fix didn't fix the bug or (b) the fix (or other change) had a side effect.**
- **Paradigmatic case(s)**
  - **Bug regression, old fix regression, general functional regression**
  - **Automated GUI regression suites**
- **Strengths**
  - **Reassuring, confidence building, regulator-friendly**
- **Blind spots**
  - **Anything not covered in the regression series. Also, maintenance of this standard list can be extremely costly.**

Regression Automation

- **Regression tools dominate the automated testing market.**
  - **Why automate tests that the program has already passed?**
  - **Percentage of bugs found with already-passed tests is about 5-20%**
  - **Efficiency of regression testing shows up primarily in the next version or in a port to another platform.**
- **If the goal is to find new bugs quickly and efficiently, we should use a method based on new test cases.**

Scenario Testing

- **Tag lines**
  - **"Do something useful and interesting"**
  - **"Do one thing after another."**
- **Fundamental question or goal**
  - **Challenging cases that reflect real use.**
- **Paradigmatic case(s)**
  - **Appraise product against business rules, customer data, competitors' output**
  - **Life history testing (Hans Buwalda's "soap opera testing.")**
  - **Use cases are a simpler form, often derived from product capabilities and user model rather than from naturalistic observation of systems of this kind.**

Scenario Testing

- **The ideal scenario has several characteristics:**
  - **It is realistic (e.g. it comes from actual customer or competitor situations).**
  - **There is no ambiguity about whether a test passed or failed.**
  - **The test is complex, that is, it uses several features and functions.**
  - **There is a stakeholder who will make a fuss if the program doesn't pass this scenario.**
- **Strengths**
  - **Complex, realistic events. Can handle (help with) situations that are too complex to model.**
  - **Exposes failures that occur (develop) over time**
- **Blind spots**
  - **Single function failures can make this test inefficient.**
  - **Must think carefully to achieve good coverage.**

User Testing

- **Tag line**
  - **Strive for realism**
  - **Let's try this with real humans (for a change).**
- **Fundamental question or goal**
  - **Identify failures that will arise in the hands of a person, i.e. breakdowns in the overall human/machine/software system.**
- **Paradigmatic case(s)**
  - **Beta testing**
  - **In-house experiments using a stratified sample of target market**

User Testing

- **Strengths**
  - **Design issues are more credibly exposed.**
  - **Can demonstrate that some aspects of product are incomprehensible or lead to high error rates in use.**
  - **In-house tests can be monitored with flight recorders (capture/replay, video), debuggers, other tools.**
  - **In-house tests can focus on areas / tasks that you think are (or should be) controversial.**
- **Blind spots**
  - **Coverage is not assured (serious misses from beta test, other user tests)**
  - **Test cases can be poorly designed, trivial, unlikely to detect subtle errors.**
  - **Beta testing is not free.**

Exploratory Testing

- **Tag line**
  - **"Interactive, concurrent exploration, test design and testing."**
- **Fundamental question or goal**
  - **Software comes to tester undocumented. Tester must simultaneously learn about the product and about the test cases / strategies that will reveal the product and its defects.**
- **Paradigmatic case(s)**
  - **Over-the-wall test-it-today testing.**
  - **Third party components.**
  - **Guerrilla testing (lightly but harshly test an area for a finite time)**
- **Strengths**
  - **Thoughtful strategy for obtaining results in the dark.**
  - **Strategy for discovering failure to meet customer expectations.**
- **Blind spots**
  - **The less we know, the more we risk missing.**

About Cem Kaner

The senior author of ***Testing Computer Software***, Cem Kaner has worked with computers since 1976, doing and managing programming, user interface design, testing, and user documentation. Through his consulting firm, KANER.COM, he teaches courses on black box software testing and consults to software publishers on software testing, documentation, and development management. Kaner is also the founder and co-host of the Los Altos Workshop on Software Testing. He is writing a new book, ***Good Enough Testing***, with James Bach and Brian Marick.

An attorney whose practice is focused on the law of software quality, Kaner usually represents customers, individual developers or small consulting firms. He is active in legislative drafting efforts involving software licensing, software quality regulation, and electronic commerce. He has published ***Bad Software: What To Do When Software Fails*** (with David Pels. John Wiley & Sons, 1998). Kaner was recently elected to the American Law Institute, a prestigious organization of legal scholars.

Kaner holds a B.A. in Arts & Sciences (Math, Philosophy), a Ph.D. in Experimental Psychology (Human Perception & Performance: Psychophysics), and a J.D. (law degree). He is Certified in Quality Engineering by the American Society for Quality.

www.kaner.com **kaner@kaner.com** **www.badsoftware.com**

About James Bach

James Bach is the founder and principal consultant of Satisfice, Inc. He writes, speaks, and consults on software quality assurance and testing. In his sixteen years in Silicon Valley-style software companies, including nine years at Apple Computer, Borland International, and Aurigin Systems, he's been a programmer, tester, QA manager, and roving problem-solver. For three years, James was Chief Scientist at ST Labs, an independent software testing company in Seattle that performs substantial testing and tester training for Microsoft.

James is a frequent speaker and writer on the subject of software quality assurance and development processes. James writes and edits the "Software Realities" column in IEEE Computer magazine, is a former section chair of the American Society for Quality and the Silicon Valley Software Quality Association, and was part of the ASQ committee that created the Certified Software Quality Engineer program.

www.satisfice.com **james@satisfice.com**

Acknowledgements

- **Some of this material was developed by James with the support of ST Labs.**
- **We thank Bob Stahl, Brian Marick, Doug Hoffman, Hans Schaefer, and Hans Buwalda for several insights.**