

Black Box Software Testing

(Academic Course - Fall 2001)

Cem Kaner, J.D., Ph.D.

Florida Institute of Technology

Section: 15 :

Risk-Based Testing & Risk-Based Test Management

Contact Information:

kaner@kaner.com

www.kaner.com (testing practitioners)

www.badsoftware.com (software law)

www.testingeducation.org (education research)

Copyright (c) Cem Kaner 2001.

I grant permission to make digital or hard copies of this work for personal or classroom use, without fee, provided that (a) copies are not made or distributed for profit or commercial advantage, (b) copies bear this notice and full citation on the first page, and if you distribute the work in portions, the notice and citation must appear on the first page of each portion. Abstracting with credit is permitted. The proper citation for this work is Cem Kaner, *A Course in Black Box Software Testing (Academic Version)*, Fall 2001, www.testing-education.org. To copy otherwise, to republish or post on servers, or to distribute to lists requires prior specific permission and a fee. Request permission to republish from kaner@kaner.com.

Black Box Software Testing

Risk-Based Testing & Risk-Based Test Management

Assigned Reading

- Stale Amland, *Risk Based Testing Risk and Metrics*,
- James Bach, *Risk and Requirements-Based Testing*.
- Carl Popper, *Conjectures & Refutations*

Recommended Reading

- James Bach, various papers at www.satisfice.com

Acknowledgement

These notes are based partially on materials prepared for the Eleventh Los Altos Workshop on Software Testing, Sunnyvale, CA, October 2000. I also appreciate comments from Bob Johnson, Brian Lawrence, James Bach, Stale Amland and Paul Gerrard.

Risk-Based Testing

- **Tag line**
 - “Find big bugs first.”
- **Fundamental question or goal**
 - Define and refine tests in terms of the kind of problem (or risk) that you are trying to manage
 - 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.
 - Failure Mode and Effects Analysis (FMEA)

Equivalence and Risk

- **Our working definition of equivalence:**

Two test cases are equivalent if you expect the same result from each.

- **This is fundamentally subjective. It depends on what you expect. And what you expect depends on what errors you can anticipate:**

Two test cases can only be equivalent by reference to a specifiable risk.

- **Two different testers will have different theories about how programs can fail, and therefore they will come up with different classes.**

- **A boundary case in this system is a “best representative.”**

A best representative of an equivalence class is a test that is at least as likely to expose a fault as every other member of the class.

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.
- 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?

Evaluating Risk

- **Several approaches that call themselves “risk-based testing” ask which tests we should run and which we should skip if we run out of time.**
- **We think this is only half of the risk story. The other half is focuses on test design.**
 - It seems to us that a key purpose of testing is to find defects. So, a key strategy for testing should be defect-based. Every test should be questioned:
 - How will this test find a defect?
 - What kind of defect do you have in mind?
 - What power does this test have against that kind of defect? Is there a more powerful test? A more powerful suite of tests?

Risk-Based Testing

- **Many of us who think about testing in terms of risk, analogize testing of software to the testing of theories:**
 - Karl Popper, in his famous essay *Conjectures and Refutations*, lays out the proposition that a scientific theory gains credibility by being subjected to (and passing) harsh tests that are intended to refute the theory.
 - We can gain confidence in a program by testing it harshly (if it passes the tests).
 - Subjecting a program to easy tests doesn't tell us much about what will happen to the program in the field.
- ***In risk-based testing, we create harsh tests for vulnerable areas of the program.***

Risk-Based Testing

- **Two key dimensions:**
 - **Find errors** (risk-based approach to the technical tasks of testing)
 - **Manage the process of finding errors** (risk-based test management)
- **Let's start with risk-based testing and proceed later to risk-based test management.**

Risk-Based Testing: Definitions

– **Hazard:**

- A dangerous condition (something that could trigger an accident)

– **Risk:**

- Possibility of suffering loss or harm (probability of an accident caused by a given hazard).

– **Accident:**

- A hazard is encountered, resulting in loss or harm.

Risks: Where to look for errors

- **Quality Categories:**

- Accessibility
- Capability
- Compatibility
- Concurrency
- Efficiency
- Localizability
- Maintainability
- Performance
- Portability
- Recoverability
- Installability and uninstallability
- Reliability
- Supportability
- Conformance to standards
- Scalability
- Testability
- Security
- Usability

*Each quality category is a risk category, as in:
“the risk of unreliability.”*

Risk Heuristics: Where to look for errors

- New things: **newer features may fail.**
- New technology: **new concepts lead to new mistakes.**
- Learning Curve: **mistakes due to ignorance.**
- Changed things: **changes may break old code.**
- Late change: **rushed decisions, rushed or demoralized staff lead to mistakes.**
- Rushed work: **some tasks or projects are chronically underfunded and all aspects of work quality suffer.**
- Tired programmers: **long overtime over several weeks or months yields inefficiencies and errors**

» Adapted from James Bach's lecture notes

Risk Heuristics: Where to look for errors

- Other staff issues: **alcoholic, mother died, two programmers who won't talk to each other (neither will their code)...**
- Just slipping it in: **pet feature not on plan may interact badly with other code.**
- N.I.H.: **external components can cause problems.**
- N.I.B.: **(not in budget) Unbudgeted tasks may be done shoddily.**
- Ambiguity: **ambiguous descriptions (in specs or other docs) can lead to incorrect or conflicting implementations.**

» Adapted from James Bach's lecture notes

Risk Heuristics: Where to look for errors

- **Conflicting requirements: ambiguity often hides conflict, result is loss of value for some person.**
- **Unknown requirements: requirements surface throughout development. Failure to meet a legitimate requirement is a failure of quality for that stakeholder.**
- **Evolving requirements: people realize what they want as the product develops. Adhering to a start-of-the-project requirements list may meet contract but fail product. (check out <http://www.agilealliance.org/>)**
- **Complexity: complex code may be buggy.**
- **Bugginess: features with many known bugs may also have many unknown bugs.**

» Adapted from James Bach's lecture notes

Risk Heuristics: Where to look for errors

- Dependencies: **failures may trigger other failures.**
- Untestability: **risk of slow, inefficient testing.**
- Little unit testing: **programmers find and fix most of their own bugs. Shortcutting here is a risk.**
- Little system testing so far: **untested software may fail.**
- Previous reliance on narrow testing strategies: **(e.g. regression, function tests), can yield a backlog of errors surviving across versions.**
- Weak testing tools: **if tools don't exist to help identify / isolate a class of error (e.g. wild pointers), the error is more likely to survive to testing and beyond.**

» Adapted from James Bach's lecture notes

Risk Heuristics: Where to look for errors

- Unfixability: **risk of not being able to fix a bug.**
- Language-typical errors: **such as wild pointers in C. See**
 - **Bruce Webster, *Pitfalls of Object-Oriented Development***
 - **Michael Daconta et al. *Java Pitfalls***
- Criticality: **severity of failure of very important features.**
- Popularity: **likelihood or consequence if much used features fail.**
- Market: **severity of failure of key differentiating features.**
- Bad publicity: **a bug may appear in PC Week.**
- Liability: **being sued.**

» Adapted from James Bach's lecture notes

Bug Patterns as a Source of Risk

- **Testing Computer Software** lays out a set of 480 common defects. You can use these or develop your own list.
 - *Find a defect in the list*
 - *Ask whether the software under test could have this defect*
 - *If it is theoretically possible that the program could have the defect, ask how you could find the bug if it was there.*
 - *Ask how plausible it is that this bug could be in the program and how serious the failure would be if it was there.*
 - *If appropriate, design a test or series of tests for bugs of this type.*

Build Your Own Model of Bug Patterns

- **Too many people start and end with the TCS bug list. It is outdated. It was outdated the day it was published. And it doesn't cover the issues in *your* system. Building a bug list is an ongoing process that constantly pays for itself. Here's an example from Hung Nguyen:**
 - This problem came up in a client/server system. The system sends the client a list of names, to allow verification that a name the client enters is not new.
 - Client 1 and 2 both want to enter a name and client 1 and 2 both use the same new name. Both instances of the name are new relative to their local compare list and therefore, they are accepted, and we now have two instances of the same name.
 - As we see these, we develop a library of issues. The discovery method is exploratory, requires sophistication with the underlying technology.
 - Capture winning themes for testing in charts or in scripts-on-their-way to being automated.

Building Bug Patterns

- There are plenty of sources to check for common failures in the common platforms
 - www.bugnet.com
 - www.cnet.com
 - links from www.winfiles.com
 - various mailing lists

Risk-Based Testing

- **Tasks**

- Identify risk factors (hazards: ways in which the program could go wrong)
- For each risk factor, create tests that have power against it.
- Assess coverage of the testing effort program, given a set of risk-based tests. Find holes in the testing effort.
- Build lists of bug histories, configuration problems, tech support requests and obvious customer confusions.
- Evaluate a series of tests to determine what risk they are testing for and whether more powerful variants can be created.

Risk-Based Testing Exercises

- **Exercises**

- **Given a list of ways that the program *could* fail, for each case:**

- Describe two ways to test for that possible failure
- Explain how to make your tests more powerful against that type of possible failure
- Explain why your test is powerful against that hazard.

- **Given a list of test cases**

- Identify a hazard that the test case might have power against
- Explain why this test is powerful against that hazard.

Risk-Based Testing Exercises

- **Collect or create some test cases for the software under test. Make a variety of tests:**
 - Mainstream tests that use the program in “safe” ways
 - Boundary tests
 - Scenario tests
 - Wandering walks through the program
 - Etc.
 - If possible, use tests the students have suggested previously.
- **For each test, ask:**
 - How will this test find a defect?
 - What kind of defect did the test author probably have in mind?
 - What power does this test have against that kind of defect? Is there a more powerful test? A more powerful suite of tests?

Risk-Based Test Management

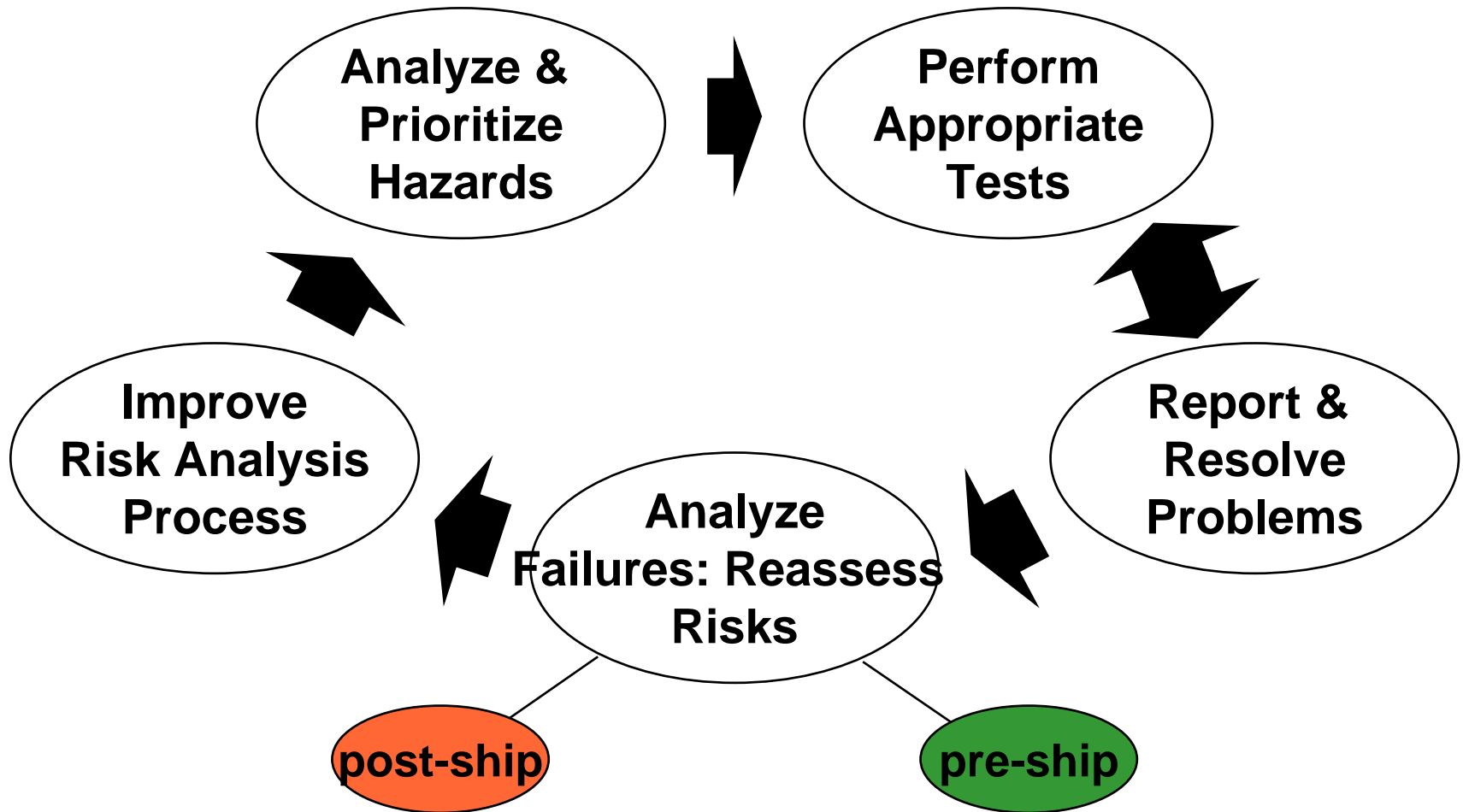
- **Project risk management involves**
 - Identification of the different risks to the project (issues that might cause the project to fail or to fall behind schedule or to cost too much or to dissatisfy customers or other stakeholders)
 - Analysis of the potential costs associated with each risk
 - Development of plans and actions to reduce the likelihood of the risk or the magnitude of the harm
 - Continuous assessment or monitoring of the risks (or the actions taken to manage them)
- **Useful material available free at <http://seir.sei.cmu.edu>**
- **<http://www.coyotevalley.com> (Brian Lawrence)**
- **Good paper by Stale Amland, *Risk Based Testing and Metrics*, 16th International Conference on Testing Computer Software, 1999.**

Risk-Based Test Management

- **Tasks**

- List all areas of the program that could require testing
- On a scale of 1-5, assign a probability-of-failure estimate to each
- On a scale of 1-5, assign a severity-of-failure estimate to each
- For each area, identify the specific ways that the program might fail and assign probability-of-failure and severity-of-failure estimates for those
- Prioritize based on estimated risk
- Develop a stop-loss strategy for testing untested or lightly-tested areas, to check whether there is easy-to-find evidence that the areas estimated as low risk are not actually low risk.

Risk-Driven Testing Cycle





Categories of Risk Sources

- Mission and goals
- Decision drivers
- Organization management
- Customer / end user
- Budget / cost
- Schedule
- Project characteristics
- Development process
- Development environment
- Personnel
- Operational environment
- New technology



Project Consequences

- Cost overruns
- Schedule slips
- Inadequate functionality
- Canceled projects
- Sudden personnel changes
- Customer dissatisfaction
- Loss of company image
- Demoralized staff
- Poor product performance
- Legal proceedings