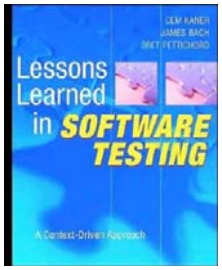
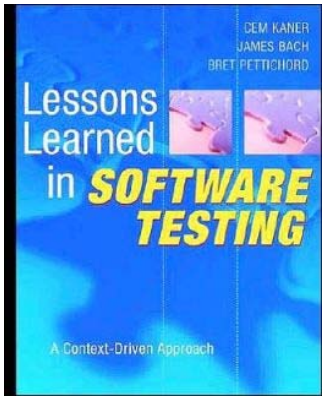


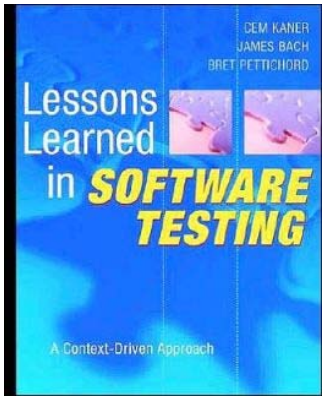
Theme 3:
Testing Techniques

Testing Techniques



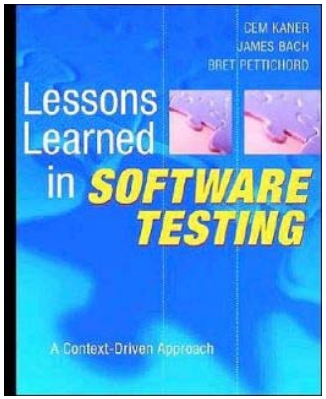
- Lesson 48: "Testing combines techniques that focus on testers, coverage, potential problems, activities, and evaluation"
  - Can be "about":
    - *Who* does the testing (e.g. user testing)
    - *What* gets tested (e.g. function testing)
    - *Why* you're testing (e.g. extreme value testing)
    - *How* you test (e.g. exploratory testing)
    - *How to tell pass/fail* (e.g. comparison to known good result)

Testing Techniques



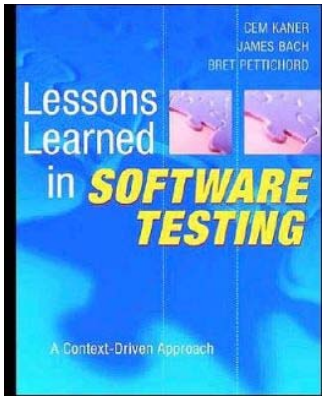
- Lesson 49: "People-based techniques focus on who does the testing"
  - User testing, obviously
  - Subject-expert testing
    - Designing a medical diagnosis system? You probably want some good doctors to evaluate it
  - "Eat your own dogfood"
    - Many companies release tools internally, without "testing" as a goal – just to see if their engineers can find bugs

Testing Techniques



- Lesson 50: "Coverage-based techniques focus on what gets tested"
  - Function testing
    - Cover every function of the program
  - Menu tour
  - Our coverage metrics discussed previously
    - Try covering all lines, branches, logical combinations…

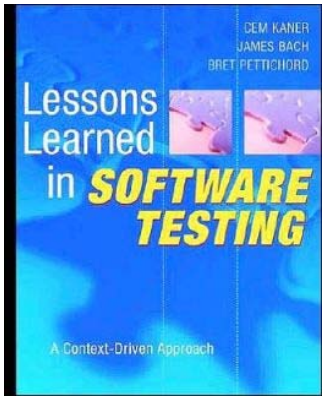
Testing Techniques



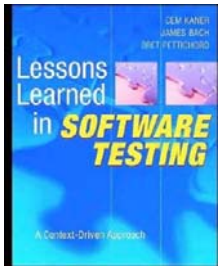
- Lesson 51: "Problems-based techniques focus on why you're testing (the risks you're testing for)"
  - Input constraints
  - Output constraints
  - Computation constraints
  - Storage (or data) constraints

  - Race conditions and timing issues are especially critical to look at here

Testing Techniques

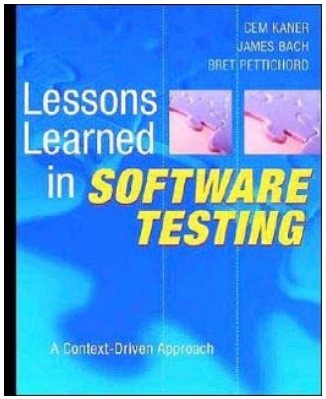


- Lesson 52: "Activity-based techniques focus on how you test"
  - Regression testing
  - Scripted testing
  - Smoke testing
  - Exploratory testing
  - Guerrila testing
  - Installation testing
  - Load testing
  - Performance testing



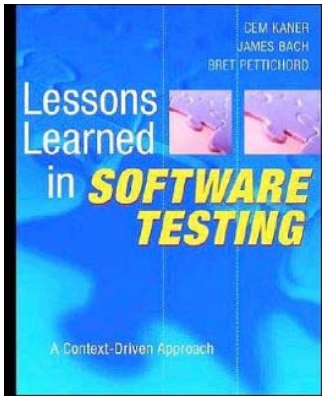
Theme 4: Reporting Bugs and Working with Others

Reporting Bugs and…



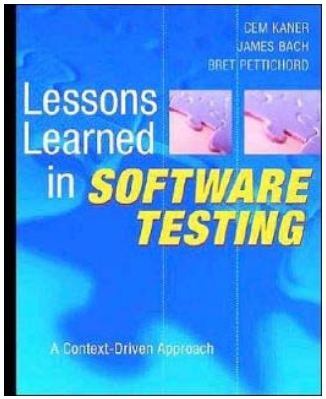
- Lesson 55: "You are what you write"
  - Bug reports are the main "product" of testers
  - **Bug reports::testers** as **source code::developers**
    - In heavily automated testing, your test code may also be a critical product, but it had better contribute to bug reports at some point
  - (Combining points from some other lessons)
    - You need to effectively make the case that *this* bug is worth giving up resources (money, programmer time, other development or bug fixing) to fix; you are the bug's *champion*
    - Be an honest champion!

Reporting Bugs and…



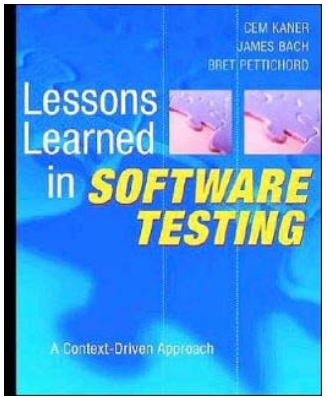
- Contents of a bug report (minimal)
  - Unique ID (name/number)
  - **What is the bug?**
  - How do you make the bug happen (BE SPECIFIC)?
    - If you have code that always produces the bug, include it!
    - If you can minimize (remember delta debugging?) do so
  - What version of the software was this detected on?
  - What is the estimated severity of the bug?
  - What is the estimated priority of the bug?

Reporting Bugs and…



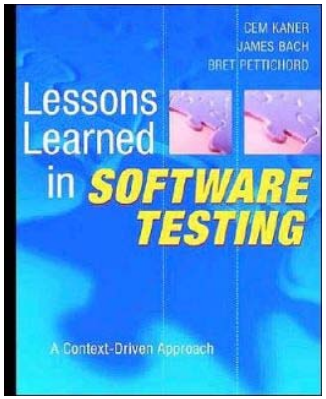
- Lesson 59: "Take the time to make your bug reports valuable"
  - Bug reports are the main "product" of testers
  - **Bug reports::testers** as **source code::developers**
    - In heavily automated testing, your test code may also be a critical product, but it had better contribute to bug reports at some point
  - If your reports aren't understandable and informative, this is like producing bad, buggy code

Reporting Bugs and…

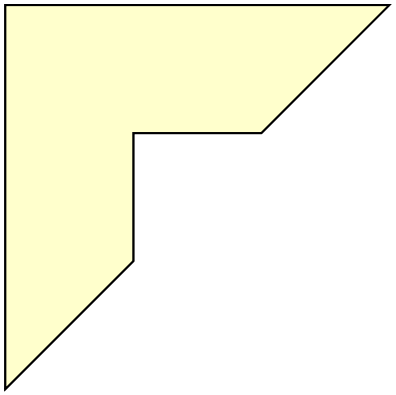


- Lesson 68: "Never assume that an obvious bug has already been filed"
  - Everyone may make this assumption…
    - And the bug will never get filed!

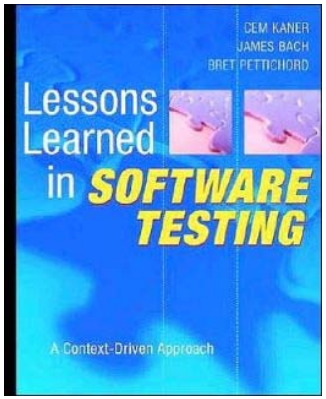
Reporting Bugs and…



- Lesson 71: "Uncorner your corner cases"
  - Programmers can sometimes ignore a test case that relies on particularly "odd" data:
    - You may try corner cases first since they are likely to fail
    - Once you find a bug, make sure you can't reproduce it with a simpler/less weird input
      - If you can, report that version instead!



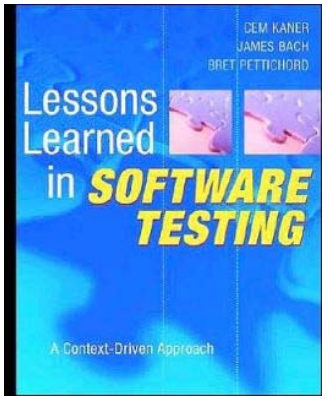
Reporting Bugs and…



- Lesson 73: "Keep clear the difference between severity and priority"
  - *Severity* is about the impact of a bug
    - Severity is about worst-case scenarios, probabilities, risks
    - Examples of high severity bugs: security compromises, incorrect results used in financial calculations, bugs that stop all testing
  - *Priority* is about how soon a bug should be fixed
    - Changes with time and circumstances

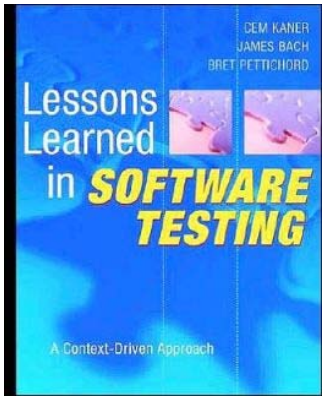
  - High severity isn't always high priority:
    - If a bug corrupts any file saved in July 2010 only, it may not be important to fix
  - High priority isn't always high severity:
    - Misspelling the company's name

Reporting Bugs and…



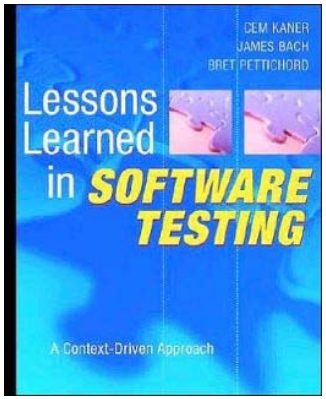
- Lesson 82: "Every bug deserves its own report"
- Lesson 83: "The summary line is the most important line in the bug report"
- Lesson 86: "Be careful of your tone. Every person you criticize will see the report"

Working with Others



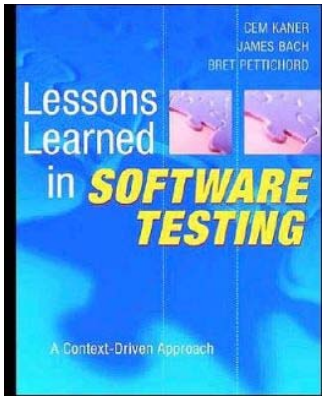
- Lesson 92: "The best approach may be to demonstrate your bugs to the programmers"
  - *Seeing is believing*
    - Don't interrupt!
    - Doesn't remove need for a written report, but can make initial report much better

Working with Others



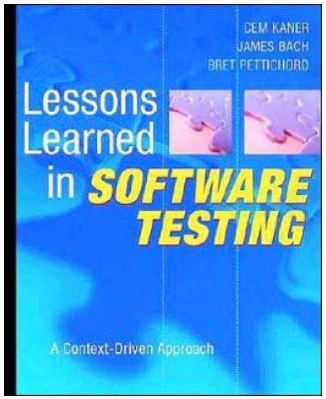
- Lesson 150: "Understand how programmers think"
  - Programmers tend to specialize
    - They often do not know the big picture very well
    - As a tester that may be your job
  - Programmers have a theory of the system
    - Report bugs in terms of programmers own models
  - Programmers often hate routine
    - They may think non-automated tests are "lame" or "wrong"

Working with Others



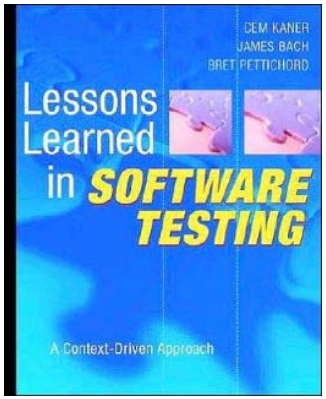
- Lesson 154: "Focus on the work, not the person"
  - Talk about the code and its bugs, not whether John Q. Programmer is a screw-up
    - Maybe he is, but that's not your job
    - Testing is not a management position, usually

Working with Others



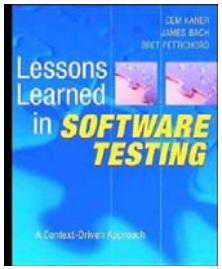
- Lesson 169: "Ask for testability features"
  - Code is not always as easy to test as it could be
    - If you don't ask, programmers won't think much about this aspect of coding
    - If you do ask, the worst that can happen is "no"
    - Programmers are often happy to make your job easier

Working with Others



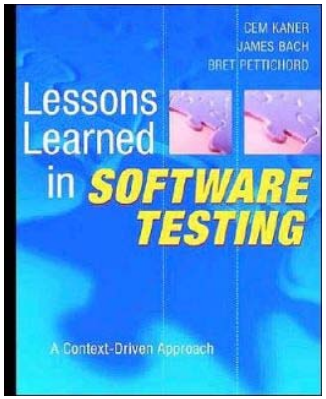
- Lesson 181: "Programmers are like tornadoes"
  - Programmers will do what they will do
  - At some companies that will be great
  - At other places, it may be a problem

  - You cannot solve the testing problem by declaring that programmers "can't act that way"
    - In the Midwest houses have basements because: tornadoes
    - Cannot get away with no basement by declaring tornadoes unreasonable



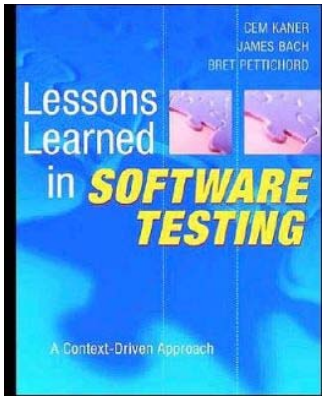
Theme 5:
Planning and Strategy

Planning and Strategy



- Lesson 274: "Three basic questions to ask about test strategy are 'why bother?', 'who cares?', and 'how much?'"
  - Why is this testing being done?
  - Who is the customer for test results?
  - How much is needed?

Planning and Strategy

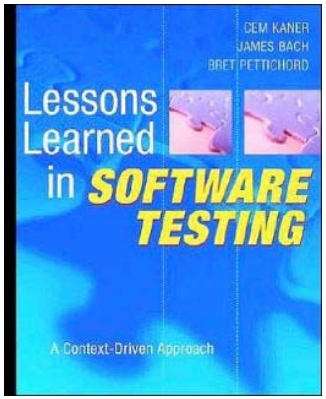


- Lesson 277: "Design your test plan to fit your context"

TEST PLAN OUTLINE (IEEE 829 FORMAT)

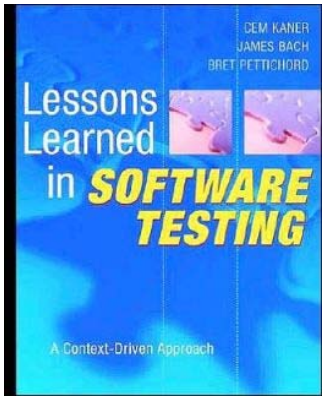
1) Test Plan Identifier
2) References
3) Introduction
4) Test Items
5) Software Risk Issues
6) Features to be Tested
7) Features not to be Tested
8) Approach
9) Item Pass/Fail Criteria
10) Suspension Criteria and Resumption Requirements
11) Test Deliverables
12) Remaining Test Tasks
13) Environmental Needs
14) Staffing and Training Needs
15) Responsibilities
16) Schedule
17) Planning Risks and Contingencies
18) Approvals
19) Glossary

Planning and Strategy



- Lesson 278: "Use the test plan to express choices about strategy, logistics, and work products"
  - The test plan expresses goals
  - It is only valuable in that it helps organize and get testing done
  - Not useful in and of itself

Planning and Strategy



- Lesson 282: "Your test strategy explains your testing"
  - Tests don't exist in a vacuum
  - Need a rationale for "why these tests, not others"
  - A test strategy serves that purpose

What Have We Learned?

- Software engineering is like other engineering disciplines
  - But it is also unlike other engineering disciplines
  - The way we do testing is one key difference
- Testing requires a special kind of thinking
  - Testing is applied epistemology
  - How to find out things about a program
  - Most common way to find out is by having a test case that makes the program fail

What Have We Learned?

- There are many kinds of testing
  - There is no one “right way to test”
  - Manual and automated testing both have a role
  - Random testing is an especially useful automated testing technique
- Coverage metrics help us measure what we have and have not tested
- Debugging is like the scientific method
  - Formulate hypotheses about what is wrong
  - Divide and conquer to narrow down the problem
  - Use evidence (tests and examining executions) to drive your hypothesis making