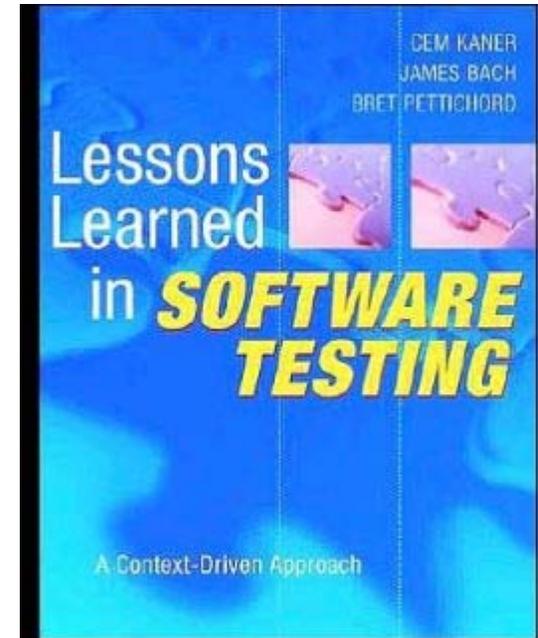


Lessons Learned in Software Testing

- An excellent book covering a range of testing topics
- Practical rather than academic
- In the next few lectures, we'll discuss some of the key “lessons” from this book, and how they apply to all testing efforts
 - Focus on the practicalities of testing, not the technical details: testing is more about a state of mind than a particular “kind of programming”



Testing: What, Not How

- The technical side of testing usually depends on what you are testing
 - To test a file system, you need to understand file systems
 - To test Java code, you probably want to know Java well
 - Test programs aren't "special" programs
 - Often just use a standard scripting language or the language of the program you are testing
- The big difference is the *goal*
 - In typical programming, you want to produce a program that, given input X produces output Y
 - In testing, there is no such simple goal
 - Many radically different solutions
 - You have to THINK *more* than in most coding



How to Test Software

- Five major themes

- **The testing role**

- What does a tester *really* do?

- **Thinking like a tester**

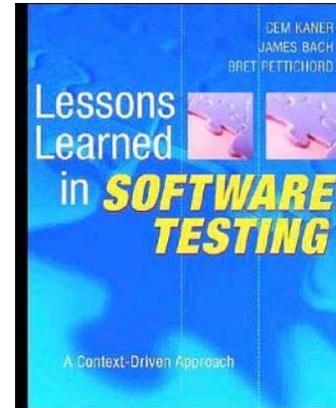
- Are there differences between thinking like a programmer/developer and thinking like a tester?

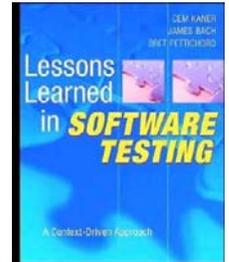
- **Testing techniques**

- **Reporting bugs and working with others**

- If a tree falls in the forest and no one hears it, can the bug possibly be fixed?

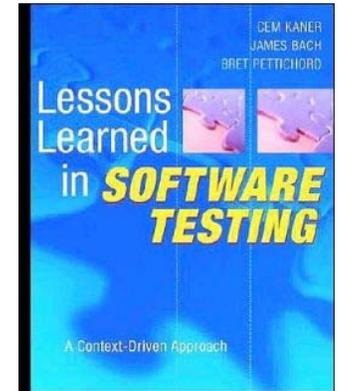
- **Planning and strategy**





Theme 1: The Testing Role

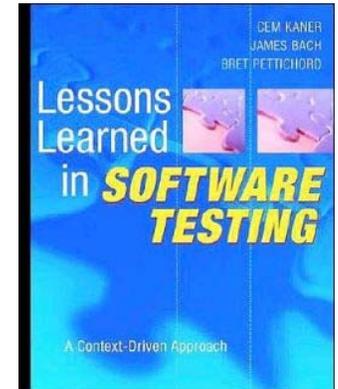
The Testing Role



- Lesson 1: “You are the headlights of the project”
 - A software project is like driving off-road in rugged terrain, at night
 - The tester lights the way!
 - Testing is about *finding information*

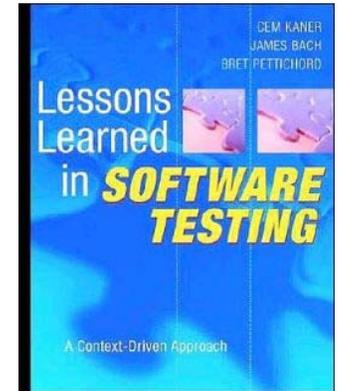


The Testing Role



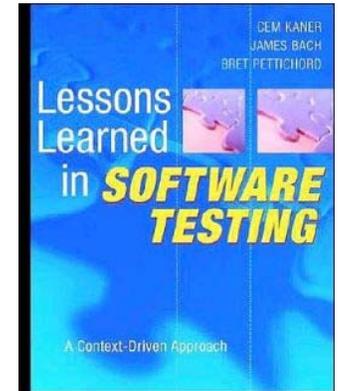
- Lesson 2: “Your mission drives everything you do”
 - Testing depends on the project
 - Goal could be “find every bug, at any cost”
 - Or “satisfy this FAA requirement”
 - Or “knock this into shape for beta release”
 - Or “keep costs minimal without making the initial version too embarrassing”
 - Or “find out if this program we’re considering buying is worth paying for”
 - Or just “Satisfy the client”

The Testing Role



- Lesson 5: “Find important bugs fast”
 - In most cases, finding “killer” bugs is part of the tester’s key mission
 - Test changed code before stable code
 - Test critical functions before rarely used things
 - Test for catastrophic problems before problems users can work around
 - **Test things someone will definitely care about before you test things you aren’t sure anyone will care about at all**

The Testing Role



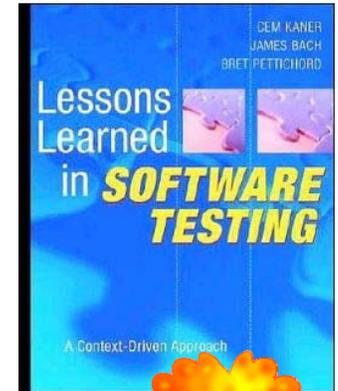
- Lesson 7: “Question everything, but not necessarily out loud”
 - Testing well requires *skepticism* and even a touch of *paranoia*
 - Being skeptical and paranoid all the time can put programmers and managers “on defense”
 - Be helpful, don’t be a pest
 - Use things you keep to yourself to guide testing, though!

“Trust no one”

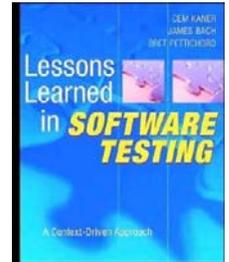
THE **X**-FILES

The Testing Role

- Lesson 14: “Beware of becoming a process improvement group”
 - Tempting to say “I’m tired of finding bugs, let’s make sure these clowns quit introducing so many bugs”
 - It would be nice if programmers worked more carefully, sure
 - But that’s usually not your job
 - Even with management support, testing is seldom a good “home” for a development process criticism society

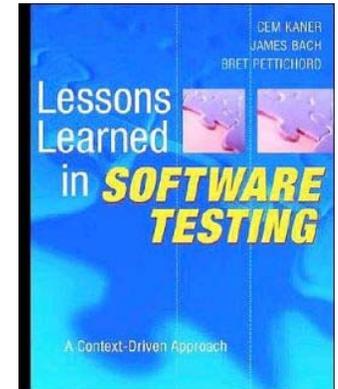


*How testers
see programmers*



Theme 2: Thinking Like a Tester

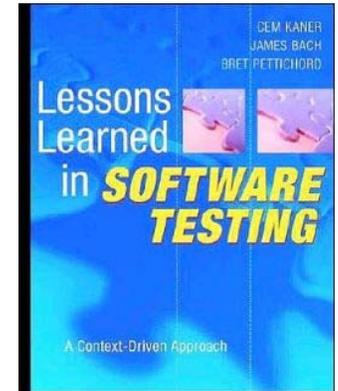
Thinking Like a Tester



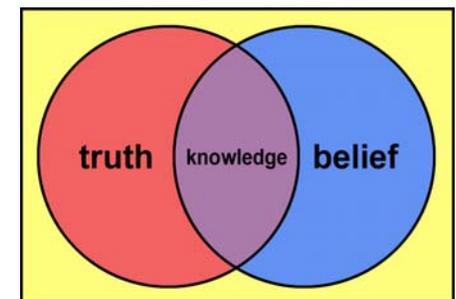
- Lesson 16: “Testing is applied epistemology”
 - What the heck is epistemology?
 - The branch of philosophy that covers *evidence and reasoning*
 - “*How we know what we know*”
 - The key questions of testing:
 - “*How do you know the software is good enough?*”
 - “*How would you know if it wasn’t good enough?*”
 - “*How do you know you’ve tested enough?*”



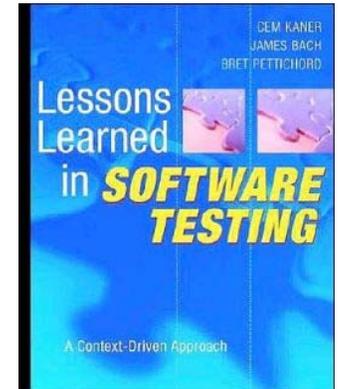
Thinking Like a Tester



- Lesson 17: “Studying epistemology helps you test better”
 - Key topics in epistemology:
 - Gathering/assessing evidence (tests!)
 - Making valid inferences (if this works, that probably also works)
 - Justification of beliefs:
 - How do you know Antarctica is there?
 - How do you know your brakes work?
 - Avoiding fallacies in informal reasoning
 - Using knowledge to make decisions

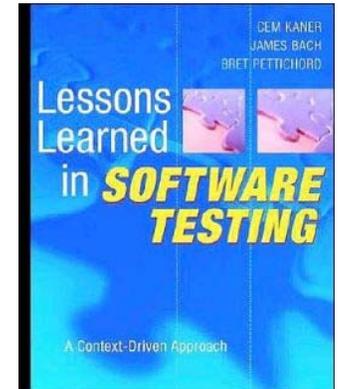


Thinking Like a Tester



- Lesson 20: “Testing requires inference, not just comparison of output to expected results”
 - Must design tests and (usually) infer from a general spec the specific output that should result
 - There is no universal table from inputs->outputs
 - Must infer which other behaviors are “also tested” by each test

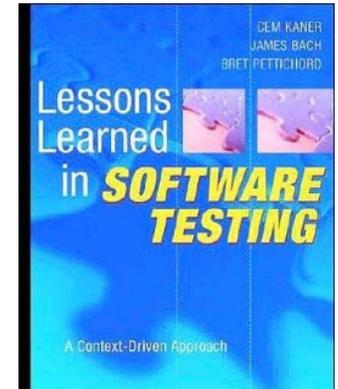
Thinking Like a Tester



- Lesson 22: “Black box testing is not ignorance-based testing”
 - Requires *lots of knowledge*: of requirements, environment, configurations, data accesses, software this program works with, and of users
 - Just avoids building tests based on the source code as the primary source of test ideas
 - After all, to some extent the programmer can do that better, in unit tests



Thinking Like a Tester

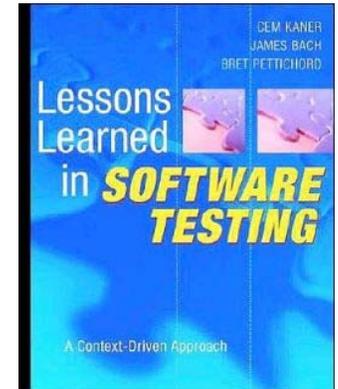


- Lesson 24: “All tests are an attempt to answer some question”



- Epistemology again
- If a test doesn't answer any interesting questions, it probably isn't worth running
 - Caveat: automatic generation may produce a huge number of tests where each in isolation is likely “useless” but the entire set of tests is highly effective
- When manually testing, or scripting a specific test, think about (1) the question and (2) how to interpret the answer (are there ambiguous responses?)

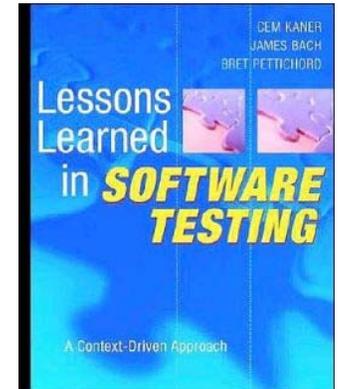
Thinking Like a Tester



- Lesson 25: “All testing is based on models”
 - You have some model of what the software being tested should do
 - You may have an explicit reference model
 - You may model the various components and how they interact
 - You very likely will model the *user* of the software!

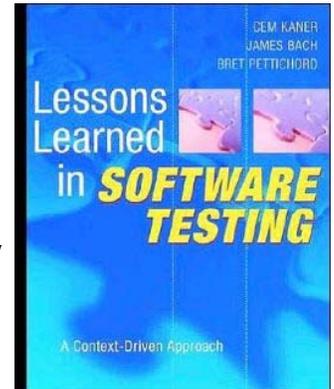


Thinking Like a Tester



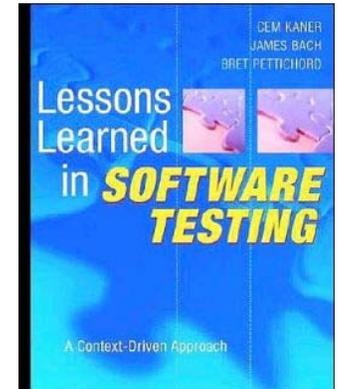
- Lesson 33: “Use implicit as well as explicit specifications”
 - Almost all software systems have (highly) incomplete specifications and requirements
 - You’re going to have to decide what the software “should” do in cases that are not specifically addressed
 - Use common sense, general notions of software safety, security, reliability, and usability
 - Use *domain knowledge* of whatever the software is actually for

Thinking Like a Tester



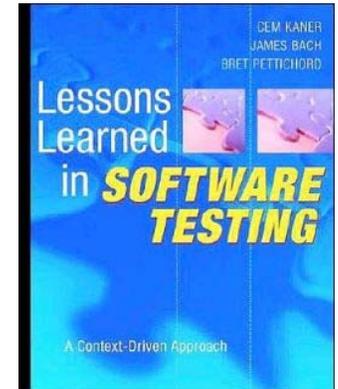
- Lesson 38: “Use heuristics to quickly generate ideas for tests”
 - Test boundaries & corner cases
 - Test error messages/conditions
 - “Rip out the hard drive, unplug the network”
 - Test unusual configurations
 - Run the tests that are annoying to run
 - Avoid redundancy, do not duplicate
 - For some critical conditions, automation’s cheap and you’re not sure it’s totally redundant, try lots of small variations

Thinking Like a Tester



- Lesson 41: “When you miss a bug, check whether the miss is surprising or just the natural outcome of your strategy”
 - Anyone can get unlucky
 - Make sure you aren’t systematically going to “get unlucky” in this particular way

Thinking Like a Tester



- Lesson 45: “... avoid ‘1287’”
 - This lesson I’ve left out part of (if you want the rest, read the book!)
 - Magic numbers are bad in test procedures & test code just as in normal programming
 - Explain the reasoning behind a “test procedure” (whether for a person to follow or a machine to run) like ‘Type 1287 characters into field 1’
 - Why 1287?
 - For a human: “enter a very large (>1024) characters into the field”
 - For a machine: “fieldVal = genString(largeTextSize)”