Teaching Software Testing

Tampere, Finland
May 5, 2004

Cem Kaner, J.D., Ph.D.
Professor of Software Engineering
Florida Institute of Technology
kaner@kaner.com
www.kaner.com, www.testingeducation.org

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Teaching Testing?

- Software testing has traditionally received little coverage in the Computer Science curriculum.
  - Partially because of the way testing is often documented and taught.
    - **Test design** is often presented in an oversimplified way as a routine application of one of a small set of briefly-defined basic techniques.
    - Many industrial presentations (and software engineering texts) add extensive descriptions of reams of **test-related paperwork** that "professional" testers or software engineers allegedly generate.

It's not clear that any of this carries a level of intellectual challenge appropriate for university-level instruction.

CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH

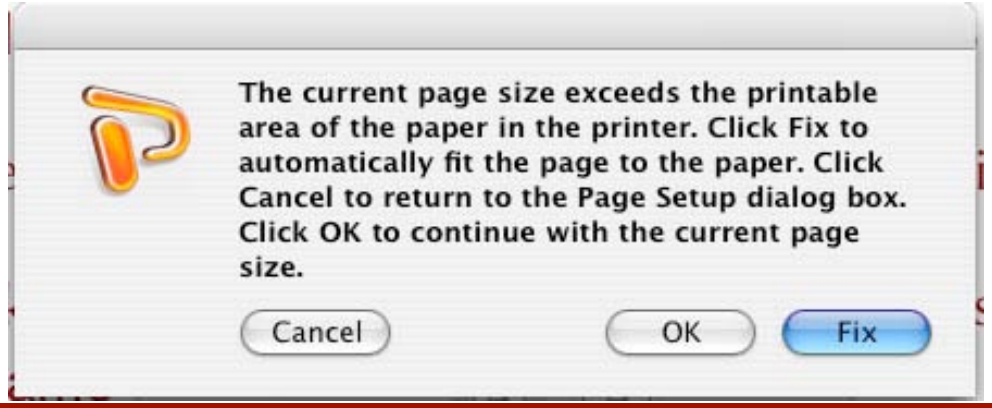
Sowmya Padmanabhan's M.Sc. Thesis

- Domain testing (equivalence class and boundary analysis) is the most often taught testing technique. It is normally presented in a routine way. See Kaner, *Teaching Domain Testing: A Status Report*, at http://www.testingeducation.org/articles/domain_testing_cseet.pdf
- If the descriptions accurately describe competent practice, it **should** be easy to automate the test design or to teach a series of steps for a human to follow.
- For her Master's thesis, Padmanabhan developed a short course (15 hours) on domain testing (equivalence class and boundary analysis, for single and multiple variables).
- The educational style was highly procedural. Students were given detailed examples and checklists and then lots of problems that they were expected to solve, and did solve, in standard ways.
- Finally, students took a performance test . . .

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Sowmya Padmanabhan's M.Sc. Thesis

- The performance test was (should have been) easy. Students were shown the PowerPoint page setup dialog and asked to identify and analyze all input and output variables. Evaluation was by three people with significant experience hiring and supervising testers, against a standard of a job candidate with a year's testing experience who claims to be reasonably good at domain testing.
- All 23 subjects performed extremely well on Padmanabhan's quizzes and assignments.
- All 23 performed remarkably consistently on the performance test.
  - Identified the same variables, analyzed them in the same ways
  - Missed the same issues, made the same mistakes



FLORIDA INSTITUTE OF TECHNOLOGY
1958
CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH

Sowmya Padmanabhan's M.Sc. Thesis

- We were actually surprised by these problems, because we were so accustomed to the routinized descriptions of what domain testers do.
- We should not have been surprised. Decades of work on mathematics education and science education teach us that procedural training produces weak transfer of learning.
- Have you ever noticed?
  - What happens when you drive to a new place
    - from a map, or in some other way, making your own way
    - with a navigator who says, "turn left next light" etc
  - If you try to go there again, do you remember the route?

Proceduralism is the enemy of good testing and of good testing education.

CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH

My approach to testing

- Software testing is a technical investigation of a product, an empirical search for quality-related information of value to a project's stakeholders.
- Testing is an intellectually challenging activity. Examples:
  - technical challenges associated with efficiently testing (such as automation architecture)
  - domain knowledge is vital to effective testing
  - communications challenges of effectively reporting bugs
  - the fundamental optimization problem--the set of worthy testing tasks requires orders of magnitude magnitude more time than we have available
  - the project unfolds over time--we gain information about the product design, risks, and context throughout the life of the project. As a result, all well done testing projects are inherently exploratory.
- Routine solutions are overrated.
  - The value of any practice depends on its context.
  - There are good practices in context, but no best practices.

These views call for a different approach to testing education.

CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH

My Teaching Contexts

- Undergraduate courses
  - Software Testing 1 (black box testing) and Software Testing 2 (programmer testing) are required courses for the B.Sc. in Software Engineering at Florida Tech. Several "special topics" testing courses have also been offered.
- Graduate courses
  - Software Testing 1 (a slightly harder version) is required for every M.Sc. in Software Engineering. Testing 1 and Testing 2 and a third testing course (security-related testing or a special topic) are required for the new Testing track (a specialization of the M.Sc. Software Engineering degree).
- Undergraduate and graduate research
  - projects, paid research assistants, and thesis/dissertation research
- Commercial teaching
  - 3-5 day professional development seminars
- Self-study
  - www.testingeducation.org

CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH

Mission of the Center

Create effective, grounded, timely materials to support the teaching and self-study of software testing, software reliability, and quality-related software metrics.

- The Center for Software Testing Education & Research officially formed in November 2003, as a collaboration among
  - Cem Kaner, Ph.D., J.D. (Professor) (Director)
  - Walter P. Bond, Ph.D. (Associate Professor)
  - Scott Tilley, Ph.D. (Associate Professor)
  - Michael Andrews, Ph.D. (Assistant Professor)
  - James Whittaker, Ph.D. (Professor)
- We also collaborate closely with:
  - James Bach, Bret Pettichord, Douglas Hoffman
  - Steve Condly, Ph.D. (Assistant Professor, Education, U Central Florida)
  - Pat Schroeder, Ph.D. (Assistant Professor, CS, U Wisconsin)
- Funding from United States' National Science Foundation
- Corporate sponsorships/ partnerships in negotiation

FLORIDA INSTITUTE OF TECHNOLOGY
1958
CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH

What We Offer at testingeducation.org

- Course notes:
  - Amland, Bach, Hoffman, Kaner, Pettichord
  - Open (no-royalty) license, for classroom (or equivalent) use in universities, corporations, and self-study groups.
  - Bach's and my black box testing course notes will evolve into an online alternative to a textbook:
    - course slides
    - reference papers for each section
    - videotaped lectures
    - videotaped demonstrations and other short segments (5 to 15 minutes) for lecture support
    - sample exercises, sample exam questions
    - section objectives
- Lab publications, conference materials

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Bloom's taxonomy of educational objectives

- One of the most common ways to describe how well someone has learned a set of material is called "Bloom's Taxonomy." See
  - http://faculty.washington.edu/krumme/guides/bloom.html
  - http://www.kcmetro.cc.mo.us/longview/ctac/blooms.htm
  - http://www.coun.uvic.ca/learn/program/hndouts/bloom.html
- 6 levels (knowledge, comprehension, application, analysis, synthesis, and evaluation).
- Much of what we see in the certification exams tests at level 1, memorized knowledge. This is worthless if your interest lies in
  - ability to transfer the knowledge to a practical situation in a simple and straightforward way (level 3),
  - ability to decide which technique is more applicable to a situation, then modify as needed to be able to apply
  - ability to combine techniques to study a situation from multiple angles

Testing 1 Learning Objectives

- Learning objectives
  - Develop critical analytical skills
  - Learn several black box testing techniques, including
    - ability to apply to complex applications
    - ability to justify details of an application ("why is this a good example of that type of test?")
    - ability to select among candidate techniques
    - ability to use techniques in combination
  - Develop teamwork
    - paired testing, group exercises
  - Improve communication skills
    - Effective bug advocacy
  - Give students an advantage in their job search
  - Build enthusiasm about the technical work of testing

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Course overview

- Five core topic areas, which I prioritize as follows:
  - Paradigms of software testing:
    - 9 dominant styles of black box testing.
  - Bug advocacy:
    - effective replication, analysis, and reporting of bugs.
  - Test documentation:
    - examples of test documentation components;
    - doing requirements analysis to determine what is needed.
  - Additional test design issues: Examples:
    - Design of GUI-level regression tests for maintainability;
    - all pairs combination testing.
  - Process and organizational issues: Examples:
    - structure and missions of typical software testing groups;
    - implications for testing of different lifecycle models on the testing process.

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Assessment-focused approach to teaching

- Student activity more valuable than lecture
  - Group work on assignments and exam preparation are strongly encouraged
- Lecture provides ***context*** for activity
  - Story-based, example-based, hypothetical-based
- Students need to practice what they learn
  - Work with sample application
    - Mozilla, Open Office, many open source applications
- Exams should require answers that show a higher level of competence

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Exam structure

- Definitions
- Short answer: essay or worked problem
- Long answer: essay or worked problem
- See my paper on "Assessment in the Software Testing Course," at www.testingeducation.org, and my presentation on "Carts Before Horses: Using Preparatory Exercises to Motivate Lecture Material"

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Software Testing Methods

Sample Questions

Cem Kaner J.D., Ph.D., ASQ-CQE

Florida Institute of Technology

Spring 2004

These notes are partially based on research that was supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." 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.

Notes on studying and answering tests (1)

- Your tests and exams will be based on these questions and a later, smaller, update to this set of questions.
- You may not use any reference materials during the test (closed book test).
- I recommend that you study with one or more partners. 3-5 people is a good sized group. 8 is too many.
- I strongly recommend that you read the questions carefully before the test, and that you attempt an answer to each one. *AFTER* you have tried your own answers, compare notes with your friends.
- Working with others will help you discover and work through ambiguities *before* you take the test. ***If a question is unclear, send me a note before the test. If you tell me early enough, I can fix it. If a question takes too long to answer, send me a note about that too.***
- When you write the test, keep in mind that I am reading your answer with the goal of finding reasons to give you points:
  - If the question contains multiple parts, be sure to give me a separate answer for each part.
  - If a question asks about "some", that means at least two. I normally expect three items in response to a "some". Similarly if the question asks for a list, I am expecting a list of at least three.
  - Be aware that different words in questions have different meanings. For example, each of the following words calls for a different answer: identify, list, define, describe, explain, compare, contrast. If I ask you to list and describe some things, give me a brief identification (such as a name) of each and then a description for each one.
  - If you find it hard to define or describe something, try writing your answer around an example.
  - If you are asked to describe the relationship among things, you might find it easiest to work from a chart or a picture. You are not required to use a diagram or chart (unless I ask for one), but feel free to use one if it will help you get across your answer.
  - If I ask you to describe or define something that is primarily visual (such as a table or a graph), your answer will probably be easier to write and understand if you draw an example of what you are defining or describing.

Notes on studying and answering tests (2)

- If I ask you for the result of a calculation, such as the number of paths through a loop, show your calculations or explain them. Let me understand how you arrived at the answer.
- If I ask you to analyze something according to the method described in a particular paper or by a particular person, I expect you to do it their way. If I ask you to describe their way, do so. If I ask you to apply their way, you don't have to describe it in detail, but you have to do the things they would do in the order they would do them, and you have to use their vocabulary to describe what you are doing.
- The test is time-limited—75 minutes. Plan to spend no more than 4 minutes on any definition, no more than 10 minutes on any short answer, and no more than 15 minutes on any long answer. Spend less on most answers. Suppose the test has 4 definitions (20 points), 2 short answers (20 points), and 3 long answers (60 points). You should plan to spend, on average, about 3 minutes per definition, about 8 minutes per short answer, and about 12 minutes per long answer (total = 64 minutes). Use the remaining 11 minutes to check your work.
- Pick the order of your answers. If you spend too long on definitions, start writing your long answers first. If you are nervous, start with the questions you find easiest to answer.
- Be aware of some factors that, in general, bias markers. These are generalizations, based on research results. I try, of course, to be unbiased, but it's a good idea to keep these in mind with ANY grader for ANY exam:
  - ***Exams that are hard to read tend to get lower grades.*** Suggestions: Write in high contrast ink (such as black, medium). Write in fairly large letters. Skip every second line. Don't write on the back of the page. If your writing is illegible, print.
  - ***Answers that are well-organized tend to be more credible.*** Suggestions: If the question has multiple parts, start each part on a new line, and identify each part at its start. In a list, start each list item on a new line—maybe bullet the list.
  - ***Don't answer what has not been asked.*** For example, if I ask you to define one thing, don't define that and then give me the definition of something related to it. If you do, (a) I won't give you extra credit, (b) I'll think that you don't know the difference between the two things, and (c) if you make a mistake, I'll take off points.
  - ***Start a new question on a new page.***

Notes on studying and answering tests (3)

- ***If I can't read something you wrote, I reserve the right to ignore it.***
- Break a question down into component answers. For example, consider the question: "What is the difference between black box and white box testing? Describe the advantages and disadvantages of each." This should have five headings:
  - Difference between black and white
  - Advantages of black box
  - Disadvantages of black box
  - Advantages of white box
  - Disadvantages of white box
- Beware of simply memorizing some points off a slide. If I see these as a memorized list w/o understanding, I will ruthlessly mark you down for memorization errors. In general, if you are repeating a set of bullet points, write enough detail for them that I can tell that you understand them.
- Use a good pen. Lawyers and others who do lots of handwriting buy expensive fountain pens for a reason. The pen glides across the page, requiring minimal pressure to leave ink. If you use a fountain pen, I suggest a medium point (write large) to avoid clogging. Also try gel pens or rollerballs. Get one that you can write with easily, to avoid writer's cramp. Basic ballpoints are very hard on you, look at how tightly you hold it and how hard you press on the page.

Definitions

1. **Domain testing**
2. **Equivalence class**
3. **Boundary condition**
4. **Best representative**
5. **Fault vs. failure vs. defect**
6. **Function testing**
7. **Regression testing**
8. **Specification-based testing**
9. **Power of a test**
10. **Public bugs vs private bugs**
11. **Prevention costs**
12. **Appraisal costs**
13. **Internal vs. external failure costs**
14. **Oracle**
15. **Exploratory testing**
16. **Waterfall lifecycle**
17. **Lifecycle model**
18. **Evolutionary development**
19. **Line (or statement) coverage**
20. **Boundary chart**
21. **Software quality**
22. **Black box testing**
23. **Glass box / white box testing**
24. **Risk-based testing**
25. **Corner case**

Definitions

26. **Finite state machine**
27. **Stochastic testing**
28. **Dumb monkey**
29. **State**
30. **Combination testing**
31. **All-pairs combination testing**
32. **Input constraints**
33. **Storage constraints**
34. **Smart Monkey**
35. **State variable**
36. **Value of a state variable**
37. **Testing project plan**
38. **Test matrix**
39. **Manual test script**
40. **Attribute to be measured**
41. **Surrogate measure**
42. **Defect arrival rate**
43. **Defect arrival rate curve (Weibull distribution)**
44. **Stress testing**
45. **Computation constraints**
46. **Output constraints**

Short Answers

1. Give two examples of defects you are likely to discover and five examples of defects that you are unlikely to discover if you focus your testing on line-and-branch coverage.
2. Give three different definitions of "software error." Which do you prefer? Why?
3. Ostrand & Balcer described the category-partition method for designing tests. Their first three steps are:
   (a) Analyze
   (b) Partition, and
   (c) Determine constraints

   Describe and explain these steps.

Short Answers

4. A program asks you to enter a password, and then asks you to enter it again. The program compares the two entries and either accepts the password (if they match) or rejects it (if they don't). You can enter letters or digits. How many *valid* entries could you test? (Please show and/or explain your calculations.)

5. A program is structured as follows:

   - It starts with a loop, the index variable can run from 0 to 20. The program can exit the loop normally at any value of the index.
   - Coming out of the loop, there is a case statement that will branch to one of 10 places depending on the value of X. X is a positive, non-zero integer. It can have any value from 1 to MaxInt.
   - In 9 of the 10 cases, the program executes X statements and then goes into another loop. If X is even, the program can exit the loop normally at any value of its index, from 1 to X. If X is odd, the program goes through the loop 666 times and then exits. In the 10^{th} case, the program exits.

     Ignore the possibility of invalid values of the index variable or X. How many paths are there through this program? Please show and/or explain your calculations.

   Note: a test question might use different constants but would be identical to this question in all other respects.

Short Answers

6. Consider a program with two loops, controlled by index variables. The first variable increments (by 1 each iteration) from -3 to 20. The second variable increments (by 2 each iteration) from 10 to 20. The program can exit from either loop normally at any value of the loop index. (Ignore the possibility of invalid values of the loop index.)
   - If these were the only control structures in the program, how many paths are there through the program?
     - If the loops are nested
     - If the loops are in series, one after the other
   - If you could control the values of the index variables, what test cases would you run if you were using a domain testing approach?
   - Please explain your answers with enough detail that I can understand how you arrived at the numbers.
   - ***Note: a test question might use different constants but would be identical to this question in all other respects.***

Short Answers

7. List and briefly explain three strengths of the waterfall lifecycle.
8. List and briefly explain three strengths of the evolutionary lifecycle.
9. Describe the characteristics of a good scenario test.
10. List and explain four claimed strengths of manual scripted tests and four claimed weaknesses.
11. List (and briefly describe) four different missions for a test group. How would your testing strategy differ across the four missions?
12. What is the Defect Arrival Rate? Some authors model the defect arrival rate using a Weibull probability distribution. Describe this curve and briefly explain three of the claimed strengths and three of the claimed weaknesses or risks of using this curve.

Short Answers

13. Distinguish between using code coverage to highlight what has not been tested from using code coverage to measure what has been tested. Describe some benefits and some risks of each type of use. (In total, across the two uses, describe three benefits and three risks.)
14. In lecture, I used a minefield analogy to argue that variable tests are better than repeated tests. Provide five counter-examples, contexts in which we are at least as well off reusing the same old tests.
15. List and describe five different dimensions (different "goodnesses") of "goodness of tests".

Long Answers

1. Imagine testing a date field. The field is of the form MM/DD/YYYY (two digit month, two digit day, 4 digit year). Do an equivalence class analysis and identify the boundary tests that you would run in order to test the field. (Don't bother with non-numeric values for these fields.)
2. I, J, and K are signed integers. The program calculates K = I * J. For this question, consider only cases in which you enter integer values into I and J. Do an equivalence class analysis from the point of view of the effects of I and J (jointly) on the variable K. Identify the boundary tests that you would run (the values you would enter into I and J).
   Note: In the exam, I might use K = I / J or K = I + J or K = IntegerPartOf (SquareRoot (I*J))

Long Answers

3. Ostrand & Balcer described the category-partition method for designing tests. Their first three steps are:
    1. Analyze
    2. Partition, and
    3. Determine constraints

    Apply their method to this function:

    I, J, and K are unsigned integers. The program calculates K = I *J. For this question, consider only cases in which you enter integer values into I and J. Do an equivalence class analysis from the point of view of the effects of I and J (jointly) on the variable K.

    Note: In the exam, I might use K = I / J or K = I + J or K = IntegerPartOf (SquareRoot (I*J))

Long Answers

4. The Spring and Fall changes between Standard and Daylight Savings time creates an interesting problem for telephone bills. *Focus your thinking on the complications arising from the daylight savings time transitions.* Create a table that shows risks, equivalence classes, boundary cases, and expected results for a long distance telephone service that bills calls at a flat rate of $0.05 per minute. Assume that the chargeable time of a call begins when the called party answers, and ends when the calling party disconnects.

Long Answers

5. **Describe a traceability matrix.**
   - How would you build a traceability matrix for the display rules in Mozilla Firebird?
   - What is the traceability matrix used for?
   - What are the advantages and risks associated with driving your testing using a traceability matrix?
   - Give examples of advantages and risks that you would expect to deal with if you used a traceability matrix for the display rules. Answer this in terms of two of the display rules that you can change in Mozilla Firebird. You can choose which two features.

Long Answers

6 What is regression testing? What are some benefits and some risks associated with regression testing? Under what circumstances would you use regression tests?

7. Why is it important to design maintainability into automated regression tests? Describe some design (of the test code) choices that will usually make automated regression tests more maintainable.

Long Answers

8. Suppose that you find a reproducible failure that doesn't look very serious.
   - Describe three tactics for testing whether the defect is more serious than it first appeared.
   - As a particular example, suppose that the display got a little corrupted (stray dots on the screen, an unexpected font change, that kind of stuff) in the Mozilla Firebird browser when you drag the scroll bar up and down. Describe three follow-up tests that you would run, one for each of the tactics that you listed above.

Long Answers

9. Compare the evolutionary and waterfall lifecycle models. Consider the four factors that project managers have to trade off against each other, and any additional issues (1 to 3 of them) that you think are important.
10. Consider the spiral lifecycle. Describe the tradeoffs in following this, compared to evolution or waterfall in terms of the four factors that project managers have to trade off against each other.
11. Imagine testing a file name field. For example, go to an Open File dialog, you can enter something into the field. Do a domain testing analysis: List a risk, equivalence classes appropriate to that risk, and best representatives of the equivalence classes. For each test case (use a best representative), briefly explain why this is a best representative. Keep doing this until you have listed 12 best-representative test cases.

Long Answers

12. Consider testing an HTML form (displayed in Mozilla Firebird) that has you enter data into a table.
    - How would you develop a list of risks for this capability? (If you are talking to people, who would you ask and what would you ask them?) (If you are consulting books or records or databases, what are you consulting and what information are you looking for in it?)
    - Why is this a good approach for building a list of risks?
    - List 10 risks associated with this function.
    - For each risk, briefly (very briefly) describe a test that could determine whether there was an actual defect.

Long Answers

13 Imagine that you were testing how Mozilla Firebird's Password manager saves login passwords.

- Explain how you would develop a set of scenario tests that test this feature.
- Describe a scenario test that you would use to test this feature.
- Explain why this is a particularly good scenario test.

14 Imagine that you were testing how Mozilla Firebird's Password manager saves login passwords.

- Explain how you would develop a set of soap operas that test this feature.
- Describe one test that might qualify as a soap opera.
- Explain why this is a good soap opera test.

Long Answers

15. Define a scenario test and describe the characteristics of a good scenario test. Imagine developing a set of scenario tests for handling security certificates in Mozilla Firebird. What research would you do in order to develop a series of scenario tests? Describe two scenario tests that you would use and explain how these would relate to your research and why each is a good test.

Long Answers

16. (The following statement is not true, but pretend it is for exam purposes.) Mozilla.org has just announced that they will include digital rights management support in Release 1.0 of the Firebird product, which they plan to ship in March, 2005. They announce that their implementation will have the same features as Microsoft's Windows Media DRM, described at http://www.microsoft.com/windows/windowsmedia/drm.aspx.
    - List and briefly explain 5 risk factors that you would expect to find associated with the spreadsheet interface to the email database. (By "risk factor", I mean a project-level source of risk. You might might it helpful to refer to Amland's paper for discussion of risk factors.)
    - For each risk factor, predict 2 defects that could arise in the spreadsheet interface part of the 2.0 project. By "predict", I mean name and describe the potential defect, and explain why that particular risk factor makes this defect more likely.

Long Answers

17. Suppose that you had access to the Mozilla Firebird source code and the time / opportunity to revise it. Suppose that you decided to use a *diagnostics-based* high volume automated test strategy to test Firebird's treatment of links to different types of files.
    - What diagnostics would you add to the code, and why?
    - Describe 3 potential defects, *defects that you could reasonably imagine would be in the software that handles downloading or display of linked files*, that would be easier to find using a diagnostics-based strategy than by using a lower-volume strategy such as exploratory testing, spec-based testing, or domain testing.

Long Answers

18 We are going to do some configuration testing on the Mozilla Firebird Office browser. We want to test it on

- Windows 98, 2000, XP home, and XP Pro (the latest service pack level of each)
- Printing to an HP inkjet, a LexMark inkjet, an HP laser printer and a Xerox laser printer
- Connected to the web with a dial-up modem (28k), a DSL modem, a cable modem, and a wireless card (802.11b)
- With a 640x480 display, an 800 x 600 display, a 1024x768 display and a an 1152 x 720 display
  - How many combinations are there of these variables?
  - Explain what an all-pairs combinations table is
  - Create an all-pairs combinations table. (Show at least some of your work.)
  - Explain why you think this table is correct.

Note: In the exam, I might change the number of operating systems, printers, modem types, or display

Long Answers

19. Imagine that you are an external test lab, and Mozilla.org comes to you with Firebird. They want you to test the product. How will you decide what test documentation to give them? (Suppose that when you ask them what test documentation they want, they say that they want something appropriate but they are relying on your expertise.) To decide what to give them, what questions would you ask (up to 7 questions) and for each answer, how would the answer to that question guide you?

Long Answers

20 The *oracle problem* is the problem of finding a method that lets you determine whether a program passed or failed a test.

Suppose that you were doing automated testing of page layout (such as the display of pages that contained frames or tables) in the Firebird browser.

Describe three different oracles that you could use or create to determine whether layout-related feature were working. For each of these oracles,

- identify a bug that would be easy to detect using the oracle. Why would this bug be easy to detect with this oracle? and
- identify another bug that your oracle would be more likely to miss. Why would this bug be harder to detect with this oracle?

Long Answers

21. You are using a high-volume random testing strategy for the Mozilla firebird program. You will evaluate results by using an oracle.
    - Consider entering a field into a form, using the entry to look up a record in a database, and then displaying the record's data in the form. How would you create an oracle (or group of oracles)? What would the oracle(s) do?
    - Now consider displaying a table whose column widths depend on the width of the column headers. Test this across different fonts (vary typeface and size). How would you create an oracle (or group of oracles) for this? What would the oracle(s) do?
    - Which oracle would be more challenging to create or use, and why?

Long Answers

22. Imagine that you were testing the "Find in this page" feature of the Mozilla Firebird browser. Describe four examples of each of the following types of attacks that you could make on this feature, and for each one, explain why your example is a good attack of that kind.
    - Input constraint attacks
    - Output constraint attacks
    - Storage constraint attacks
    - Computation constraint attacks.

    (Notes for you while you study. Refer to Jorgensen / Whittaker's paper on how to break software. Don't give me two examples of what is essentially the same attack. In the exam, I will not ask for all 16 examples, but I might ask for 4 examples of one type or two examples of two types, etc.)

Long Answers

23. Imagine that you were testing the Mozilla "Manage bookmarks" feature.

    Describe four examples of each of the following types of attacks that you could make on this feature, and for each one, explain why your example is a good attack of that kind.

    - Input constraint attacks
    - Output constraint attacks
    - Storage constraint attacks
    - Computation constraint attacks.

    (Notes for you while you study. Refer to Jorgensen / Whittaker's paper on how to break software. Don't give me two examples of what is essentially the same attack. In the exam, I will not ask for all 16 examples, but I might ask for 4 examples of one type or two examples of two types, etc.)

Testing 2

- Programmer Testing
  - What programmers can/should do to test their code or code of peers
- Test-first programming in Java
  - JUnit, Eclipse
- Application of test-first programming skills to a legacy product
  - For example, Marick's "multi" program
  - We prefer working on maintenance of test tools
  - students work through the code, propose improvements, must create JUnit tests for everything they touch, and write test-first modifications
- Integration testing below the user interface
  - Ward Cunningham's FIT program
- System testing below the user interface
  - Using Ruby (Perl, Python, whatever) scripts to drive program through the COM (or similar) API
  - Final exam: build a test tool, test-first
    - Use Excel as an oracle for Open Office spreadsheet
    - Support high-volume function equivalence testing of single functions and function combinations.

CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Back to the Vision

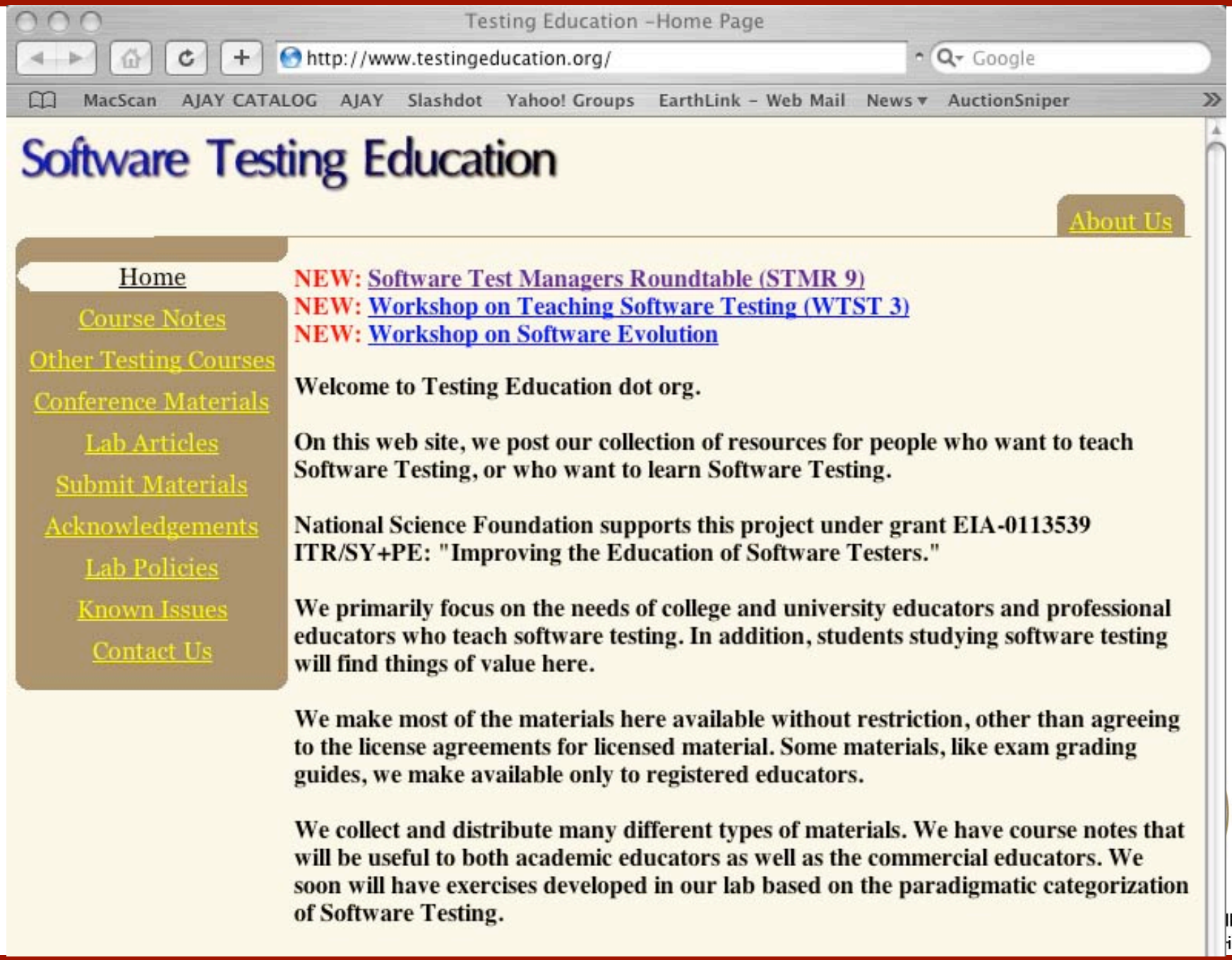
- We are training investigators, not technicians
- Analogy to forensic investigators
  - Many expensive tools
  - Many fixed procedures
  - But the key challenges are what to study, how to decide what is relevant to collect as data, which tests to run, how to interpret the data.
- Critical challenges that the technician-level approaches don't prepare students to cope with
  - Impossibility of complete testing
    - testers (teachers, students) must live and breathe tradeoffs
    - multiple potential missions, imply different tradeoffs
  - Oracle problem
  - Technological solutions must solve worthwhile problems
  - Multiple schools of software testing, and little discussion of the genuine controversies that exist
  - The project, its risks, and our knowledge evolve
  - Software engineering metrics not rooted in measurement theory

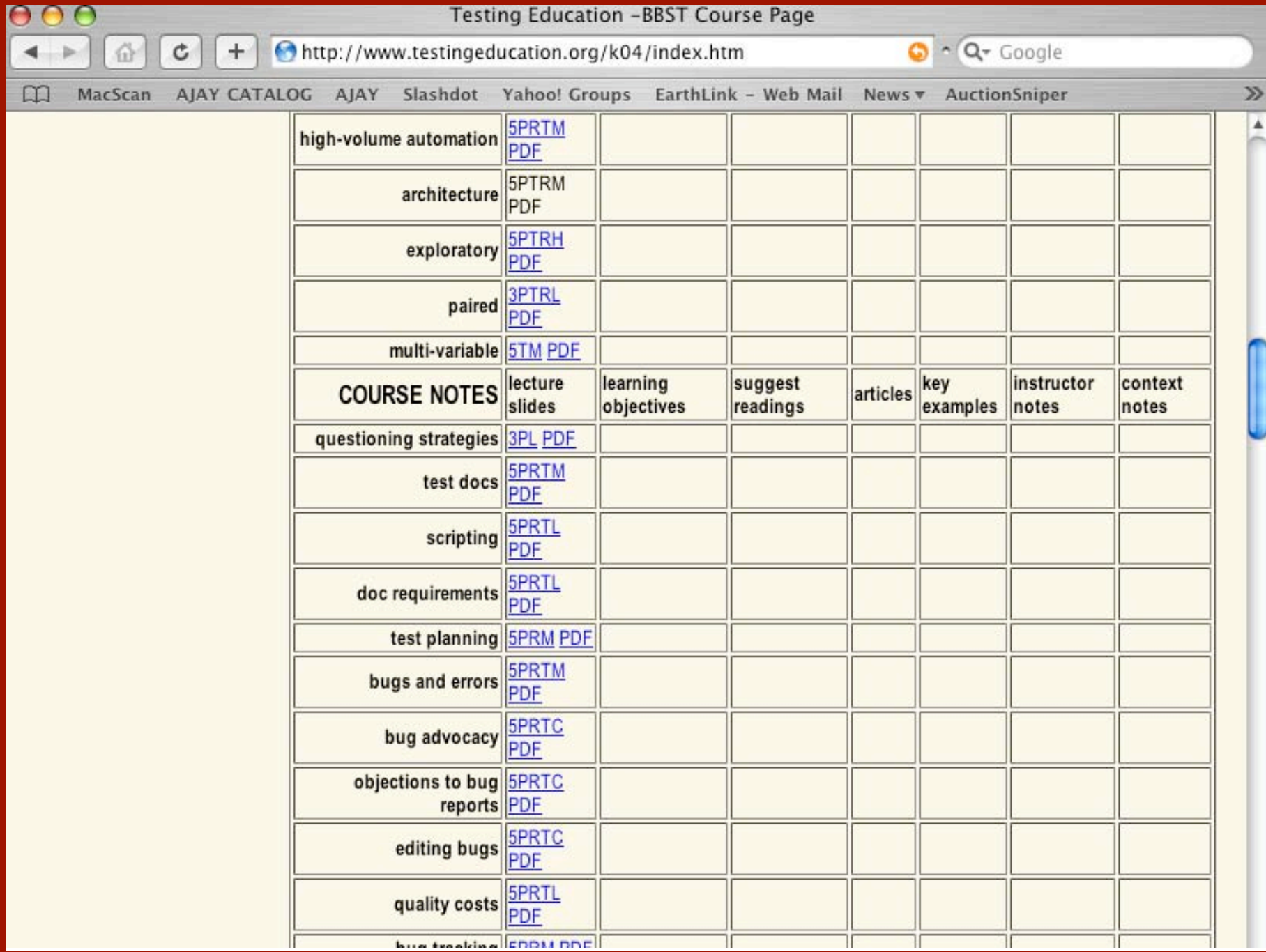
CENTER FOR SOFTWARE TESTING
EDUCATION AND RESEARCH

Opportunities for Collaboration

- testingeducation.org is a clearinghouse
  - Generally, we are looking for
    - course notes
    - course support materials
    - research on effectiveness of course materials or teaching approaches
- Association for Software Testing has just formed
  - I edit the new Journal, which will be online, free, open to the public
    - literature reviews, empirical reports, thorough case studies --> essentially, materials that will advance the field or the teaching of the field
    - need papers, editorial board
  - Collaborative conferences, extended LAWSTs (structure might be like IFIP Working Group meetings)

CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH


Testing Education -Home Page
http://www.testingeducation.org/
Google
MacScan
AJAY CATALOG
AJAY
Slashdot
Yahoo! Groups
EarthLink - Web Mail
News
AuctionSniper
Software Testing Education
About Us
Home
Course Notes
Other Testing Courses
Conference Materials
Lab Articles
Submit Materials
Acknowledgements
Lab Policies
Known Issues
Contact Us
NEW: Software Test Managers Roundtable (STMR 9)
NEW: Workshop on Teaching Software Testing (WTST 3)
NEW: Workshop on Software Evolution
Welcome to Testing Education dot org.
On this web site, we post our collection of resources for people who want to teach Software Testing, or who want to learn Software Testing.
National Science Foundation supports this project under grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers."
We primarily focus on the needs of college and university educators and professional educators who teach software testing. In addition, students studying software testing will find things of value here.
We make most of the materials here available without restriction, other than agreeing to the license agreements for licensed material. Some materials, like exam grading guides, we make available only to registered educators.
We collect and distribute many different types of materials. We have course notes that will be useful to both academic educators as well as the commercial educators. We soon will have exercises developed in our lab based on the paradigmatic categorization of Software Testing.



high-volume automation	5PRTM PDF						
architecture	5PTRM PDF						
exploratory	5PTRH PDF						
paired	3PTRL PDF						
multi-variable	5TM PDF						
COURSE NOTES	lecture slides	learning objectives	suggest readings	articles	key examples	instructor notes	context notes
questioning strategies	3PL PDF						
test docs	5PRTM PDF						
scripting	5PRTL PDF						
doc requirements	5PRTL PDF						
test planning	5PRM PDF						
bugs and errors	5PRTM PDF						
bug advocacy	5PRTC PDF						
objections to bug reports	5PRTC PDF						
editing bugs	5PRTC PDF						
quality costs	5PRTL PDF						
bug tracking	5PRM PDF						