CENTER FOR SOFTWARE TESTING EDUCATION AND RESEARCH

Cem Kaner, Ph.D., J.D.
Professor
Director

Pat Bond, Ph.D.
Associate Professor

Scott Tilley, Ph.D.
Associate Professor

Mike Andrews, Ph.D.
Assistant Professor

James Whittaker, Ph.D.
Professor

MISSION

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

Teaching the Black Box Testing Course

Cem Kaner & Sabrina Fay

ACM SIGCSE, Norfolk, VA, March 2004 (Faculty Poster)

Software testing has traditionally received little coverage in the Computer Science curriculum, and that is partially because of the way in which it 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 some 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.

Black box testing doesn't have to be taught as an intellectually sterile activity that should be automated or offshored as soon as possible.

The essence of black box testing is **active investigation** of a product by an outsider who is more focused on the acceptability of the product in its usage environment than on the details of construction. Here are some of the factors in the typical investigative context:

- **The product under test is incompletely and inaccurately s**pecified. Even if it is well-specified with respect to some stakeholders, it misses the perspective (and potential objections) of others. Good specs for programmers are often weak descriptions of user intent and value (and vice-versa) for example.
- **Software has a vast array of potential problems; test techniques that are effective at exposing one type of problem can be completely ineffective for exposing another**. For example, *domain testing* is effective for exposing off-by-one errors, but worthless for uncovering memory leaks, wild pointers, race conditions, erroneous onscreen instructions, or clumsy user task sequencing. *High-volume state-transition testing* can expose the leaks, but not the off-by-one bugs or the user interface blunders (for which you might try *user scenario tests)*.
- **Learning the range of possible errors involves ongoing learning** about code and its risks, the application domain and its risks, and the target market and its risks. The tester must synthesize technological, design, and marketing perspectives, and increase her sophistication in them, every day. (The process of simultaneously learning more about the product and designing tests based on the new learning is called **exploratory testing**.)

- **Learning the techniques is like learning the tools available in a crime lab.** There are a lot of them, some are simple, some are very fancy, they all take knowledge and skill to use well, many of them are too expensive or time consuming to use routinely (so tradeoffs are *always* necessary) and experts can figure out how to use one in a slightly new way to solve a problem for which routine analyses don't yet exist.

Students of software testing must learn a lot of techniques, but the techniques are just tools. They aren't enough. We see an analogy to math education--like math students, testers have to learn techniques in terms of the problems they can be used to address, and the application of those techniques to problems worth reasoning about. This is basic, challenging-to-teach, impossible-to-reduce-to-simple-routine, problem-solving.

The materials here support the teaching of a course in Black Box Testing. There are about 120 hours worth of lecture notes, plus various papers, including notes on assessment.

Along with a problem-solving approach to test design and execution, we expect that the learning objectives for many testing courses will include growth in descriptive and persuasive technical writing (bug reports, especially, must be precise, insightful, and motivating), teamwork, measurement theory and practice, and project estimation, scheduling and status tracking. All of these skills are desirable throughout the software engineering and computer science curricula.

Development of this material has been significantly supported by the National Science Foundation, grant EIA-0113539 ITR/SY+PE "Improving the Education of Software Testers" and by Rational/IBM; Texas Instruments; Satisfice, Inc.; and the LogiGear Corporation.

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 or our other sponsors or supporters.

Cem Kaner is Professor of Software Engineering at the Florida Institute of Technology and Director of Florida Tech's Center for Software Testing Education. Prior to joining Florida Tech, Kaner worked as a programmer, tester, test manager, development manager, director, human factors analyst, tech pubs manager, organization development consultant, and attorney (focused on software-related law) in Silicon Valley. He is lead author of **Testing Computer Software**, of **Lessons Learned in Software Testing**, and **Bad Software: What To Do When Software Fails**.

Sabrina Fay is a dual graduate student, studying for an M.Sc. (software engineering) at Florida Institute of Technology (expected graduation May 2004) and an M.Sc. (instructional systems) at Florida State University (expected graduation December 2004). She has extensive experience as an instructional designer at Harris before coming to Florida Tech.

This poster presents a set of course notes. If the disks aren't handy,

you can find the notes at

http://www.testingeducation.org/k04/index.htm

and you can find related papers at

http://www.kaner.com/articles.html and **http://www.testingeducation.org/articles/**