

Black Box Software Testing

Fall 2005

Overview for Students

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

Professor of Software Engineering

Florida Institute of Technology

and

James Bach

Principal, Satisfice Inc.

Copyright (c) Cem Kaner & James Bach, 2000-2005

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

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.

My current job titles are Professor of Software Engineering at Florida Institute of Technology, and Research Fellow at Satisfice, Inc. The theme of my career is satisfaction and safety of software customers and workers.

I've worked as a programmer, tester, writer, user interface designer, software salesperson, organization development consultant, teacher, as a manager of user documentation, software testing, and software development, and as an attorney focusing on the law of software quality. These have provided many insights into relationships between computers, software, developers, and customers.

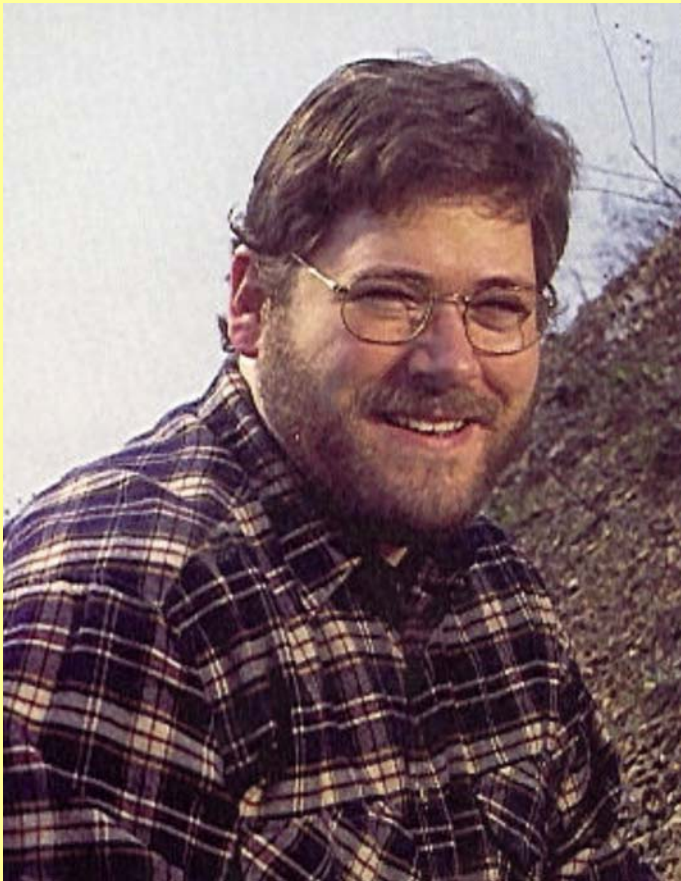
I'm the senior author of three books:

- ***Lessons Learned in Software Testing*** (with James Bach & Bret Pettichord)
- ***Bad Software*** (with David Pels)
- ***Testing Computer Software*** (with Jack Falk & Hung Nguyen).

I studied Experimental Psychology for my Ph.D., with a dissertation on Psychophysics (essentially perceptual measurement). This field nurtured my interest in human factors (and thus the usability of computer systems) and in measurement theory (and thus, the development of valid software metrics.)

About James Bach

www.satisfice.com



I started in this business as a programmer. I like programming. But I find the problems of software quality analysis and improvement more interesting than those of software production. For me, there's something very compelling about the question "How do I know my work is good?" Indeed, how do I know anything is good? What does good mean? That's why I got into SQA, in 1987.

Today, I work with project teams and individual engineers to help them plan SQA, change control, and testing processes that allow them to understand and control the risks of product failure. I also assist in product risk analysis, test design, and in the design and implementation of computer-supported testing. Most of my experience is with market-driven Silicon Valley software companies like Apple Computer and Borland, so the techniques I've gathered and developed are designed for use under conditions of compressed schedules, high rates of change, component-based technology, and poor specification.

Credits & Legal Notices

Many sections of this course were co-developed with Hung Quoc Nguyen or Doug Hoffman. We also thank Jack Falk, Elizabeth Hendrickson, Bob Johnson, Brian Lawrence, Pat McGee, Melora Svoboda, and the participants in the Los Altos Workshops on Software Testing and the Software Test Managers' Roundtables. Additional acknowledgements appear on specific slides.

These notes include some legal information, but you are not my legal client. I do not provide legal advice in the notes or the course. Even if we discuss specific situations in class, you cannot give enough information in a classroom setting for me to respond with a competent legal opinion. If you need legal advice, please consult your own attorney.


The practices discussed in this course are useful for an introduction to testing, but more experienced testers will adopt additional practices. Mission-critical and life-critical software development efforts involve specific and rigorous procedures that are not described in this course.

Center for Software Testing Education & Research

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

Florida Tech's Center for Software Testing Education & Research 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 (Satisfice), Bret Pettichord (Thoughtworks), Douglas Hoffman (Software Quality Methods) and Pat Schroeder, Ph.D. (Milwaukee College of Engineering: Computer Science)



**Donations to the
Center are most
welcome. Please
contact us at
kaner@cs.fit.edu.**

Learning objectives

Testing software involves investigating a product under tight constraints. Our goal is to help you become a better investigator:

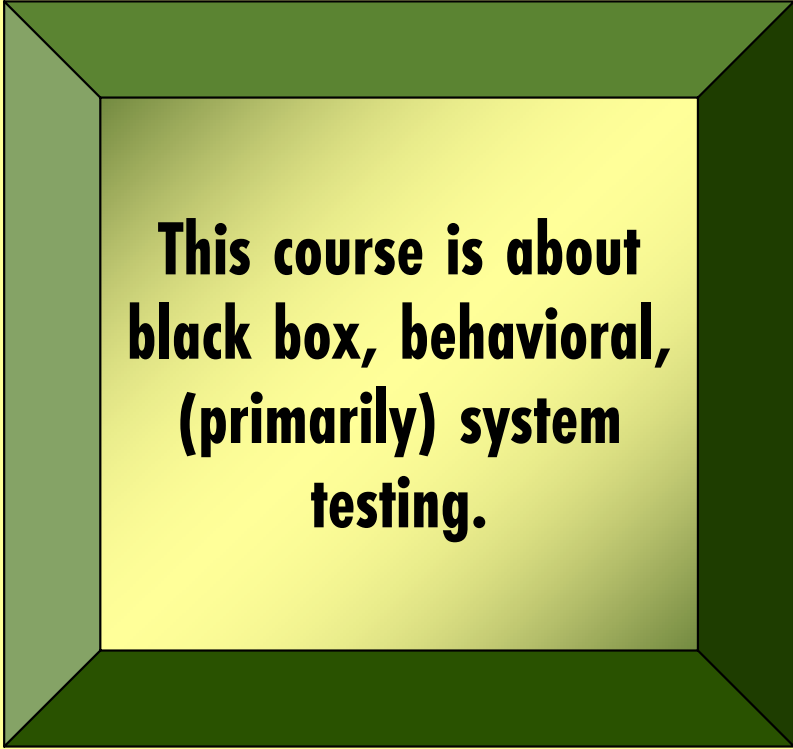
- Develop skills with several techniques
- Choose effective techniques for a given objective under your constraints
- Improve the critical thinking and rapid learning skills that underlie good testing
- Communicate your findings effectively
- Plan investments (in documentation, tools, and process improvement) to meet your actual needs
- Create work products that you can use in job interviews to demonstrate testing skill

Course scope: Key approaches

There are several approaches to testing. We can't cover them all in one course, without making the course too broad / shallow to meet our objectives.

Here are some traditional distinctions:

- Black box versus glass box (white box)
- Behavioral (or functional) versus structural
- Functional versus parafunctional (nonfunctional)
- Unit versus integration versus subsystem versus system
- Verification versus validation
- Acceptance versus independent versus internal black box versus programmer.



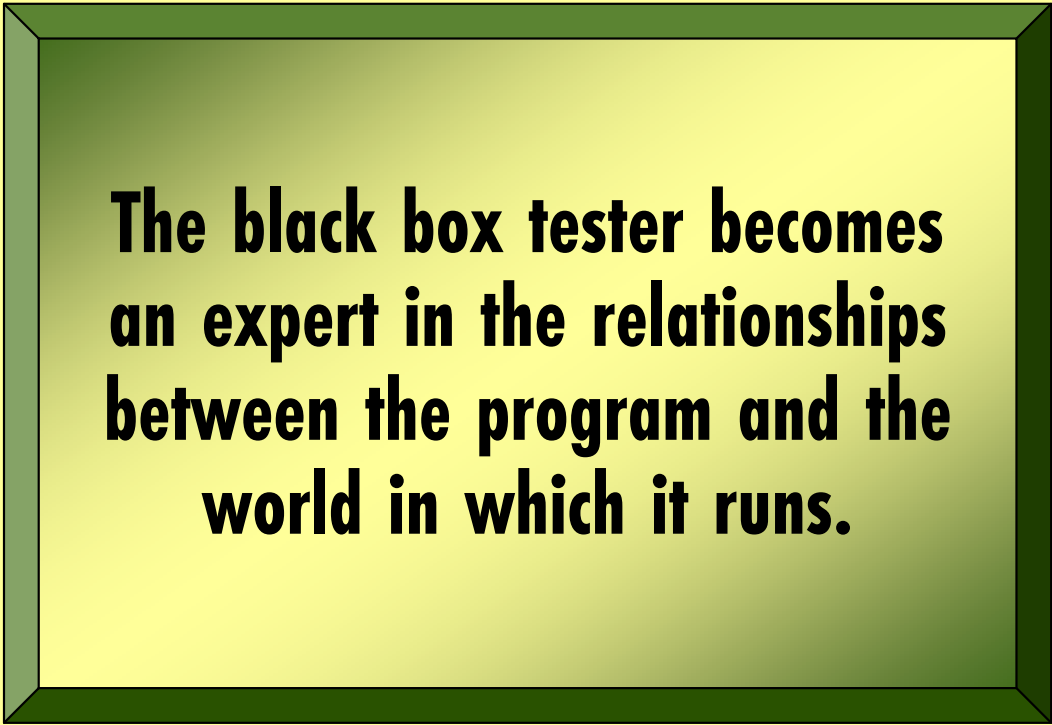
**This course is about
black box, behavioral,
(primarily) system
testing.**

Buzzwords: Black box testing

The **black box tester** works without knowledge of the code.

He design tests from the specification (if there is one) and from his (research-based) knowledge of the product's user needs and characteristics, domain being automated, software / hardware environment, and other risks.

(Some authors narrow this concept to testing exclusively against an authoritative specification.)



The black box tester becomes an expert in the relationships between the program and the world in which it runs.

Buzzwords: Glass box testing

The **glass box tester** works with knowledge of the code.

This is often called “white box” to contrast with “black box” but we can’t see the code through a white box any easier than through black.

The glass box tester is a programmer, who becomes expert in the implementation of the product under test or the implementation of its relationships with the world.

Glass box testers often ask “Does this code do what the programmer expects?” in contrast to the black box, “Does this product fail to do what users (human and software) expect?”

Buzzwords: Structural testing

Structural testing either means the same thing as glass box testing or it narrows the definition slightly to an emphasis on the structure of the program (control flow) and its internal representations and management of data (data structures, data flow).

Buzzwords: Behavioral testing

Behavioral testing is focused on the observable behavior of the product.

It is like black box testing, except that behavioral testers might read the code and design tests on the basis of their knowledge of the code.

Buzzwords: Functional

Functional testing is behavioral testing that looks at the program as a collection of functions (features).

Functional testers may focus too narrowly on capability (*can the program do this*) and reliability, ignoring the **parafunctional** (sometimes called *nonfunctional*) **attributes**, such as usability, scalability, maintainability, speed, security, localizability, supportability, et cetera.

These “**ilities**” are often called the quality attributes or quality characteristics of a system.

Buzzwords: Unit through System

Unit testing is focused on isolated units (modules, classes, functions) of the product. Many people equate unit tests and glass box tests, but you **can do** black box unit tests and glass box integration / system tests.

Integration tests study how two (or more) units work together. You can have low-level (two or three units) and high-level (many units) integration tests. Integration testers often use their knowledge of the code to predict and evaluate how data flows among the units.

System tests are integration tests that look at the product as a system for delivering intended benefits.

Lower-level tests are code-focused.

Higher-level tests are benefit-focused.

Buzzwords: Acceptance testing

Acceptance testing is done by the customer or her agent or by a “surrogate” (someone who plays the role of the buyer or her agent, even though he is neither.)

Some acceptance testers do more verification, others more validation.

Acceptance testing is almost always black box.

Buzzwords: Independent testing

Independent testing is testing done independently of the developers, often an external test lab. In some companies, an in-house test group operates independently.

The essence of independence is that the developers (or development company) doesn't drive the strategy of testing or the testers' evaluations of their results.

Especially in regulated environments, independent testing may include code inspection or review of glass box tests.

Buzzwords: Programmer testing


The **programmer** tests her own code, or code she's going to maintain.

She uses her knowledge of the code to focus and prioritize tests.

Programmers typically use an array of tools to test more easily or more automatically.

The **test-driven programmer** uses tests to guide her design and programming decisions or to improve her understanding of another programmer's code.

Programmer testing is typically (but not necessarily) glass box, verification-oriented, and often includes both structural and behavioral tests.



**At Florida Tech,
the Testing 2
course focuses on
programmer
testing**

Buzzwords: Internal black box

Internal or in-house black box testing is black box testing done by employees of the company developing the product under test. This is what is done by most software testing groups in most companies that develop software.

In a company that does extensive programmer testing, the main value added by in-house black box testers comes from parafunctional analysis and validation, not from functional verification.

The scope of this course

This course focuses on in-house, system-level, black box testing.

A course on acceptance testing would emphasize more specification analysis, contract law (acceptance testers check whether the software offered meets sales promises or its custom-software purchase agreement) and more heavily the extent to which this product will meet the tester's company's needs.

Taking this course at Florida Tech

1. Watch the lecture before class

Sometimes: reading required *before* class.

Other times: readings required *after* a class, to help with an assignment or prep for an exam question.

The website also includes demonstrations / examples for many techniques we offer.

2. Quiz often at the start of class.

3. We use class time for discussions, student presentations, and labs.

4. Assessment based on:

- Quiz scores
- Presentations
- Assignments / Labs
- Midterm and final (see the study guide.)

The study guide

We focus students' work with 100 essay questions in a study guide. We encourage students to develop answers together.

We draw all mid-term and final exam questions from the study guide. We expect well-reasoned, well-presented answers. This is the tradeoff. You have lots of time before the exam to work on the answers. By the time of the exam, you should have good ones.

Some students try to memorize other students' answers instead of working on their own. Don't do that. It's ineffective.

Look at "Assessment in the software testing course" at <http://www.testingeducation.org/a/assessment.pdf>

Tips for self-study

- If possible, work with a friend. Compare notes, evaluate each other's answers to the study guide questions.
- Try the multiple choice questions to check your *basic* understanding of the material. These are superficial tests of knowledge. They help identify what you don't understand at all.
- Try the techniques yourself. We use open source software at Florida Tech. You can join the same projects.
- We have some grading notes for some of the essay questions. We expect to develop more each time we teach the course. Work through these – answer the question yourself first, then critically compare your answer and your reasoning to the notes.

Discussion groups

There are two discussion lists for the BBST course. Both are moderated.

- The broad discussion list is at <http://groups.yahoo.com/group/BBST-discuss>
- To discuss how to *teach* the course or how to improve it for self-study, http://groups.yahoo.com/group/bbst_course