Welcome to Foundations of Software Testing! This book is intended to offer exactly what its title implies. It is important that students planning a career in Information Technology take a course in software testing. It is also important that such a course offer students an opportunity to acquire material that will remain useful throughout their careers in a variety of software applications and changing environments. This book is intended to be an introduction to exactly such material and hence ideal as text for a course in software testing. It distills knowledge developed by hundreds of testing researchers and practitioners from all over the world and brings it to its readers in easy to understand form.

Test generation, selection, prioritization, and assessment lie at the foundation of all technical activities involved in software testing. Appropriate deployment of the elements of this strong foundation enables the testing of different types of software applications as well as testing for various properties. Applications include OO systems, web services, Graphical User Interfaces, embedded systems, and others. Properties relate to security, timing, performance, reliability, and others.

The importance of software testing increases as software pervades more and more of our daily lives. Unfortunately, few universities offer full-fledged courses in software testing. Those that do often struggle to identify a suitable text. My hope is that this book will allow academic institutions to create courses in software testing, and those that already offer such courses will not need to hunt for a textbook or rely solely on research publications.

Conversations with testers and managers in commercial software development environments have led me to believe that though software testing is considered an important activity, software testers often complain of not receiving treatment at par with system designers and developers. I believe that raising the level of sophistication in the material covered in courses in software testing will lead to superior testing practices, high quality software, and thus translate into positive impact on the career of software testers. I hope that exposure to even one-half of the material in Volume 1 will establish a student's respect for software testing as a discipline in its own right and at the same level of maturity as subjects such as Compilers, Databases, Algorithms, and Networks.

Target audience: It is natural to ask: What is the target level of this book ? My experience, and that of some instructors who have used earlier drafts, indicates that Volume 1 is best suited for use at senior undergraduate and early graduate levels. Chapters in Volume 2 are appropriate for a course at the graduate level and make it even suitable for a third advanced course in software testing. Some instructors might want to include material from Volume 2, such as security testing, in an undergraduate course. Certainly, no single one-semester course can

hope to cover all material in either of the two volumes without drowning the students! However, a carefully chosen subset of chapters can offer an intellectually rich and useful body of material and prepare the students for a rewarding career in software development and testing.

While the presentation in this book is aimed at a student in a college or university classroom, I believe that both practitioners and researches will find it useful. Practitioners, with patience, may find this book a rich source of techniques they could learn and adapt in their development and test environment. Researchers will likely find this book as a rich reference.

Nature of material covered: Software testing covers a wide spectrum of activities. At a higher level, such activities appear to be similar whereas at a lower level of detail they might differ significantly. For example, most software development environments engage in test execution. However, test execution for an operating system is carried out quite differently than that for a pacemaker; while one is an open-system, the other is embedded and hence the need for different ways to execute tests.

The simultaneous existence of similarities and differences in each software testing activity leads to a dilemma for an author as well as an instructor. Should a book, and a course, focus on specific software development environments and how they carry out various testing activities ? Or, should they focus on specific testing activities without any detailed recourse to specific environments ? Either strategy is subject to criticism and leaves the student in a vacuum regarding the applications of testing activities or about their formal foundations.

I have resolved this dilemma through careful selection and organization of the material presented. Parts I, II, and III of the book focus primarily on the foundations of various testing activities while Part IV focuses on the applications of the foundational material. For example, techniques for generating tests from models of expected program behavior are covered in Part II while the application of these techniques to testing object-oriented programs and for security properties are covered in Part IV. As an exception, Part I does illustrate through examples the differences in software test process as applied in various software development organizations.

Organization: The book is organized into four parts, each distributed over two volumes. Part I covers terminology and preliminary concepts related to software testing. These are divided into three chapters one of which, Chapter 1, appears in Volume 1 while the remaining two on errors and test process are included in Volume 2. Chapter 1 introduces a variety of terms and basic concepts that pervade the field of software testing. Some of the early adopters of this book use Chapter 1 for introductory material covered during the first two or three weeks of an undergraduate course.

Part II covers various test generation techniques. Chapter 2 introduces the most fundamental of all test generation techniques widely applicable in almost any software application one can imagine. These include equivalence partitioning, boundary value analysis, cause-effect graphing, and predicate testing. Chapter 3 introduces powerful and fundamental techniques for automatically generating tests from finite state models. Three techniques have been selected for presentation in this chapter: W, Wp, and UIO methods. Finite state models are used in a variety of applications such as in OO testing, security testing, and GUI testing. State-

charts offer a more powerful modeling formalism than finite state machines. Test generation from statechart models is covered in Volume 2. Modeling the expected timing behavior of a realtime application using timed automata and generating tests from the model is covered in Volume 2. Generation of combinatorial designs and tests therefrom is the topic of Chapter 6. Generation of tests from formal specifications is covered in Volume 2. Random, stress, load, and performance testing testing are covered in Volume 2.

Regression testing forms an integral part of all software development environments where software evolves into newer versions and thus undergoes extensive maintenance. Chapter 7 introduces some fundamental techniques for test selection, prioritization, and minimization of use during regression testing.

Part III is an extensive coverage of an important and widely applicable topic in software testing: test enhancement through measurement of test adequacy. Chapter 9 introduces a variety of control-flow and data-flow based code coverage criteria and explains how these could be used in practice. The most powerful of test adequacy criteria based on program mutation are introduced in Chapter 10. While some form of test adequacy assessment is used in almost every software development organization, material covered in these chapters promises to take adequacy assessment and test enhancement to a new level thereby making a significant positive impact on software reliability.

Practitioners often complain, and are mostly right, that many white-box adequacy criteria are impractical to use during integration and system testing. I have included a discussion on how some of the most powerful adequacy assessment criteria can be, and should be, used even beyond unit testing. Certainly, my suggestions to do so assume the availability of commercial-strength tools for adequacy assessment.

Each chapter ends with a detailed bibliography. I have tried to be as comprehensive as possible in citing works related to the contents of each chapter. I hope that instructors and students will find the bibliography sections rich and helpful in enhancing their knowledge beyond this book. Citations are also a testimony to the rich literature in the field of software testing.

What does this book not cover ?: Software testing consists of a large number of related and intertwined activities. Some of these are technical, some administrative, and some merely routine. Technical activities include test case and oracle design at the unit, subsystem, integration, system, and regression levels. Administrative activities include manpower planning, budgeting, and reporting. Planning activities include test planning, quality assessment and control, and manpower allocation. While some planning activities are best classified as administrative, e.g. manpower allocation, others such as test planning are intertwined with technical activities like test case design.

If you are interested in learning about details of administrative tasks then this is not the right book. While a chapter on test processes in Volume 2 offers insights into administrative and planning tasks such as quality control, details are best covered in many other excellent books cited in sections titled Bibliographic Notes at the end of each chapter.

Several test related activities are product specific. For example, testing of a device driver

often includes tasks such as writing a device simulator. Simulators include heart simulator in testing cardiac pacemakers, a USB port simulator useful in testing I/O drivers, and an airborne drone simulator used in testing control software for airborne drones. While such activities are extremely important for effective testing and test automation, they often require a significant development effort. For example, writing a device simulator and testing it is both a development and testing activity. Test generation and assessment techniques described in this book are applicable to each of the product specific test activity. However, product specific test activities are illustrated in this book only through examples and not described in any detail. My experience has been that it is best for students to learn about such activities through industry sponsored term projects.

Suggestions to instructors:

There is wide variation in the coverage of topics in courses in software testing. This is one of the reasons why this book is divided into two volumes so as to keep the size and weight of each volume at a comfortable level for the reader while providing a comprehensive treatment of the subject matter. I have tried to cover most, if not all, of the important topics in this area. Tables 1 and 2 provide suggested outline of undergraduate and graduate courses, respectively, that could be based entirely on this book.

Table 1: A sample undergraduate course in software testing.

Week	**Topic**	**Chapter**
1	Course objectives and goals, project assignment, testing terminology and concepts	1
2	Test process and management	1
3	Errors, faults, and failures	1
4	Boundary value analysis, equivalence partitioning, decision tables	2
5, 6	Test generation from predicates	2
7	*Interim project presentations Review, Midterm examination*	
8	Test adequacy: control flow	9
9	Test adequacy: data flow	9
10, 11	Test adequacy: program mutation	10
12, 13, 14	Special topics, e.g. OO testing, security testing	Volume 2
15, 16	Review, *Final project presentations*	
17	*Final examination*	

Sample undergraduate course in software testing: We assume a semester long undergraduate course worth 3-credits, that meets twice a week, each meeting lasts 50 minutes, and devotes

Table 2: A sample graduate course in software testing.

Week	Topic	Chapter
1	Course objectives and goals, testing terminology and concepts	1
2	Test process and management	Volume 2
	Errors, faults, and failures	Volume 2
3	Boundary value analysis, equivalence partitioning, decision tables	2
4	Test generation from predicates	2
5,6	Test generation from finite state models	3
7,8	Combinatorial designs	6
	Review, Midterm examination	
9	Test adequacy: control flow	9
10	Test adequacy: data flow	9
11, 12	Test adequacy: program mutation	10
13, 14	Special topics, e.g. realt-time testing, security testing	Volume 2
15, 16	Review, *Research presentations*	
17	*Final examination*	

a total of 17 weeks to lectures, examinations, and project presentations. The course has a 2-hours per week informal laboratory, and requires students to work in small teams of 3 or 4 to complete a term-project. The term project results in a final report and possibly a prototype testing tool. Once every two weeks students are given one laboratory exercise that takes about 4-6 hours to complete.

Table 3 contains a suggested evaluation plan. Carefully designed laboratory exercises form an essential component of this course. Each exercise offers the student an opportunity to use a testing tool to accomplish a task. For example, the objective of a laboratory exercise could be to familiarize the student with JUnit as test runner or JMeter as a tool for the performance measurement of web services. Instructors should be able to design laboratory exercises based on topics covered during the previous weeks. A large number of commercial and open-source testing tools are available for use in a software testing laboratory.

Sample graduate course in software testing: We assume a semester long course worth 3-credits. The students entering this course have not had any prior course in software testing, such as the undergraduate course described above. In addition to the examinations, students will be required to read and present recent research material. Students are exposed to testing tools via unscheduled laboratory exercises.

Testing tools: There is a large set of testing tools available in the commercial, freeware, and open source domains. A small sample of such tools is listed in Table 4.

Table 3: Suggested evaluation components of the undergraduate and graduate courses in software testing.

Level	**Component**	**Weight**	**Duration**
Undergraduate	Midterm examination	15 points	90 minutes
	Final examination	25 points	120 minutes
	Quizzes	10 points	Short duration
	Laboratory assignments	10 points	10 assignments
	Term project	40 points	Semester
Graduate	Midterm examination	20 points	90 minutes
	Final examination	30 points	120 minutes
	Laboratory assignments	10 points	5 assignments
	Research/Term project	30 points	Semester

Evolutionary book: I expect this book to evolve over time. Advances in topics covered in this book, and any new topics that develop, will be included in subsequent editions. Any errors found by me and/or reported by the readers will be corrected. The book's web site listed below contains a link to simplify reporting of any errors found. Readers are encouraged to visit the website for latest information about the book.

`http://www.cs.purdue.edu/homes/apm/foundationsBook/`

In the past I have given cash rewards to students who carefully read the material and reported any kind of error. I plan to retain the cash reward approach as a means for continuous quality improvement.

Aditya P. Mathur
Purdue University
West Lafayette, IN, USA
December 29, 2006

Table 4: A sample set of tools to select from for use in undergraduate and graduate courses in software testing.

Purpose	**Tool**	**Source**
Combinatorial designs	AETG	Telcordia Technologies
Code coverage measurement	TestManager™	IBM Rational
	JUnit	Freeware
	CodeTest	Freescale Semiconductor
	χSuds	Telcordia Technologies
Defect tracking	Bugzilla	Freeware
	FogBugz	Fog Creek Software
GUI testing	WebCorder	Crimson Solutions
	jfcUnit	Freeware
Mutation testing	muJava	Professor Jeff Offut offutt@ise.gmu.edu
	Proteum	Professor Jose Maldonado jcmaldon@icmc.usp.br
Performance testing	Performance Tester	IBM Rational™
	JMeter	Apache, for Java
Regression testing	Eggplant	Redstone Software
	χSuds	Telcordia Technologies
Test management	ClearQuest™	IBM Rational™
	TestManager	IBM Rational™