

Introduction

This is a book on advanced software testing for test analysts. By that I mean that I address topics that a practitioner who has chosen software testing as a career should know. I focus on those skills and techniques related to test analysis, test design, test execution, and test results evaluation. I assume that you know the basic concepts of test engineering, test design, test tools, testing in the software development lifecycle, and test management. You are ready to mature your level of understanding of these concepts and to apply these mature, advanced concepts to your daily work as a test professional.

This book follows the International Software Testing Qualifications Board's (ISTQB) Advanced Level Syllabus, with a focus on the material and learning objectives for the advanced test analyst. As such, this book can help you prepare for ISTQB Advanced Level Test Analyst exam. You can use this book to self-study for those exams or as part of an e-learning or instructor-lead course on the topics covered in those exams. If you are taking an ISTQB-accredited Advanced Level Test Analyst training course, this book is an ideal companion text for that course.

However, even if you are not interested in the ISTQB exams, you will find this book useful to prepare yourself for advanced work in software testing. If you are a test manager, test director, test analyst, technical test analyst, automated test engineer, manual test engineer, programmer, or in any other field where a sophisticated understanding of software testing is needed, then this book is for you.

This book focuses on test analysis. The book consists of 11 chapters, addressing the following material:

1. Basic aspects of software testing
2. Testing processes
3. Test management

4. Test techniques
5. Testing of software characteristics
6. Reviews
7. Incident (defect) management
8. Standards and test process improvement
9. Test tools and automation
10. People skills (team composition)
11. Preparing for the exam

Since that structure follows the structure of the ISTQB Advanced Syllabus, some of the chapters address the material in great detail, as they are central to the test analyst role. Some of the chapters address the material in less detail, as the test analyst need only be familiar with it. For example, I cover test techniques in detail in this book because that is central to what a test analyst does, while I spend less time on test management.

If you also read the companion volume to this book, which is for test managers, you'll find parallel chapters that address the material in detail but with different emphasis. For example, test analysts need to know quite a bit about incident management. Test analysts spend a lot of time creating incident reports, and you need to know how to do that well. Test managers also need to know a lot about incident management, but they focus on how to keep incidents moving through their reporting and resolution lifecycle and how to gather metrics from such reports.

What should a test analyst be able to do? Or, to ask the question another way, what should you have learned to do—or learned to do better—by the time you finish this book?

- Implement the test strategy with a focus on business domain requirements
- Analyze the system based on user quality expectations and apply that analysis to the testing to be done
- Evaluate the system requirements to determine whether the business objectives can be met by that system
- Prepare and execute adequate testing activities, and report on the progress of these activities
- Provide the necessary evidence and data to support evaluations and findings
- Implement the necessary tools and techniques to achieve the defined goals

In this book, we focus on these main concepts. I suggest that you keep these high-level objectives in mind as we proceed through the material in each of the following chapters.

In writing this book and the companion volume on test management, I've kept foremost in my mind the question of how to make this material useful to you. If you are using this book to prepare for an ISTQB Advanced Level Test Analyst exam, then I recommend that you read chapter 11 first, then read the other 10 chapters in order. If you are using this book to expand your overall understanding of testing to an advanced level but do not intend to take an ISTQB Advanced Level Test Analyst exam, then I recommend that you read chapters 1 through 10 only. If you are using this book as a reference, then feel free to read only those chapters that are of specific interest to you.

Each of the first 10 chapters is divided into sections. For the most part, I have followed the organization of the ISTQB Advanced Syllabus to the point of section divisions, but subsections and sub-subsection divisions in the syllabus might not appear. You'll also notice that each section starts with a text box describing the learning objectives for this section. If you are curious about how to interpret those K2, K3, and K4 tags in front of each learning objective, and how learning objectives work within the ISTQB syllabus, read chapter 11.

Software testing is in many ways similar to playing the piano, cooking a meal, or driving a car. How so? In each case, you can read books about these activities, but until you have practiced, you know very little about how to do it. So I've included practical, real-world exercises for the key concepts. I encourage you to practice these concepts with the exercises in the book. Then, make sure you take these concepts and apply them on your projects. You can become an advanced software testing professional only by doing software testing.

ISTQB Copyright

This book is based on the ISTQB Advanced Syllabus version 2007. It also references the ISTQB Foundation Syllabus version 2007. It uses terminology definitions from the ISTQB Glossary version 2.0. These three documents are copyrighted by the ISTQB and used by permission.

1 Test Basics

“Who am I? Why am I here?”

Admiral James Stockdale, United States vice presidential candidate in 1992, at a vice presidential debate.

The first chapter of the Advanced Syllabus is concerned with contextual and background material that influences the remaining chapters. There are five sections:

1. Introduction
2. Testing in the Software Lifecycle
3. Specific Systems
4. Metrics and Measurement
5. Ethics

Let's look at each section and how it relates to test analysis.

1.1 Introduction

Learning objectives

Recall of content only

This chapter, as the name implies, introduces some basic aspects of software testing. These central testing themes have general relevance for testing professionals.

There are four major areas:

- Lifecycles and their effects on testing
- Special types of systems and their effects on testing

ISTQB Glossary

software lifecycle: The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software lifecycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase. Note that these phases may overlap or be performed iteratively.

- Metrics and measures for testing and quality
- Ethical issues

Many of these concepts are expanded upon in later chapters. While you might suspect this material and the material in the Foundation syllabus to be redundant, based on the name of this chapter, it actually expands on ideas introduced there.

1.2 Testing in the Software Lifecycle

Learning objectives

Recall of content only

Chapter 2 in the Foundation Syllabus discusses integrating testing into the software lifecycle. As with the Foundation Syllabus, in the Advanced Syllabus, you should understand that testing must be integrated into the software lifecycle to succeed. This is true whether the particular lifecycle chosen is sequential, incremental, iterative, or spiral.

Proper alignment between the testing process and other processes in the lifecycle is critical for success. This is especially true at key interfaces and hand-offs between testing and lifecycle activities such as these:

- Requirements engineering and management
- Project management
- Configuration and change management
- Software development and maintenance

- Technical support
- Technical documentation

Let's look at two examples of alignment.

In a sequential lifecycle model, a key assumption is that the project team will define the requirements early in the project and then manage the (hopefully limited) changes to those requirements during the rest of the project. In such a situation, if the team follows a formal requirements process, an independent test team in charge of the system test level can follow an analytical requirements-based test strategy.

Using such a strategy in a sequential model, the test team would start—early in the project—planning and designing tests following an analysis of the requirements specification to identify test conditions. This planning, analysis, and design work might identify defects in the requirements, making testing a preventive activity. Failure detection would start later in the lifecycle, once system test execution began.

However, suppose the project follows an incremental lifecycle model, adhering to one of the agile methodologies like Scrum. The test team won't receive a complete set of requirements early in the project, if ever. Instead, the test team will receive requirements at the beginning of each 30 day “sprint.”

Rather than analyzing requirements at the outset of the project, the best the test team can do is to identify and prioritize key quality risk areas; i.e., they can follow an analytical risk-based test strategy. Specific test designs and implementation will occur immediately before test execution, potentially reducing the preventive role of testing. Failure detection starts very early in the project, at the end of the first sprint, and continues in repetitive, short cycles throughout the project. In such a case, testing activities in the fundamental testing process overlap and are concurrent with each other as well as with major activities in the software lifecycle.

No matter what the lifecycle—and indeed, especially with the more fast-paced agile lifecycles—good change management and configuration management are critical for testing. A lack of proper change management results in an inability for the test team to keep up with what the system is and what it should do. A lack of proper configuration management, as was discussed in the Foundation Syllabus, leads to loss of changes, an inability to say what was tested at

ISTQB Glossary

system of systems: Multiple heterogeneous, distributed systems that are embedded in networks at multiple levels and in multiple domains and are interconnected, addressing large-scale interdisciplinary common problems and purposes.

what point in time, and severe lack of clarity around the meaning of the test results.

The Foundation Syllabus cited four typical test levels:

- Unit or component
- Integration
- System
- Acceptance

The Foundation Syllabus mentioned some reasons for variation in these levels, especially with integration and acceptance.

Integration testing can mean component integration testing—integrating a set of components to form a system, testing the builds throughout that process. Or it can mean system integration testing—integrating a set of systems to form a system of systems, testing the system of systems as it emerges from the conglomeration of systems.

Acceptance test variations, discussed in the Foundation Syllabus, included user acceptance tests and regulatory acceptance tests.

Along with these four levels and their variants, at the Advanced level you need to keep in mind additional test levels that you might need for your projects. These would include the following:

- Hardware-software integration testing
- Feature interaction testing
- Customer product integration testing

You should expect to find most, if not all, of the following for each level:

- Clearly defined test goals and scope
- Traceability to the test basis (if available)

- Entry and exit criteria, as appropriate both for the level and for the system lifecycle
- Test deliverables, including results reporting
- Test techniques that will be applied, as appropriate for the level, for the team, and for the risks inherent in the system
- Measurements and metrics
- Test tools, where applicable and as appropriate for the level
- And, if applicable, compliance with organizational or other standards

When RBCS associates perform assessments of test teams, we often find organizations that use test levels but that perform them in isolation. This is often inefficient and confusing. While these topics are discussed more in the companion volume on advanced test management, test analysts should keep in mind that using documents like test policies and frequent contact between test-related staff can coordinate the test levels to reduce gaps, overlap, and confusion about results.

Let's take a closer look at this concept of alignment. We'll use the V-model shown in figure 1-1 as an example. We'll further assume that we are talking about the system test level.

In the V-model, with a well-aligned test process, test planning occurs concurrently with project planning. In other words, the moment of involvement of testing is at the very start of the project.

Once the test plan is approved, test control begins. Test control continues through to test closure. Analysis, design, implementation, execution, evaluation of exit criteria, and test results reporting are carried out according to the plan. Deviations from the plan are managed.

Test analysis starts immediately after or even concurrently with test planning. Test analysis and test design happen concurrently with requirements, high-level design, and low-level design. Test implementation, including test environment implementation, starts during system design and completes just before test execution begins.

Test execution begins when the test entry criteria are met. More realistically, test execution starts when most entry criteria are met and any outstanding entry criteria are waived. In V-model theory, the entry criteria would include successful completion of both component test and integration test levels. Test

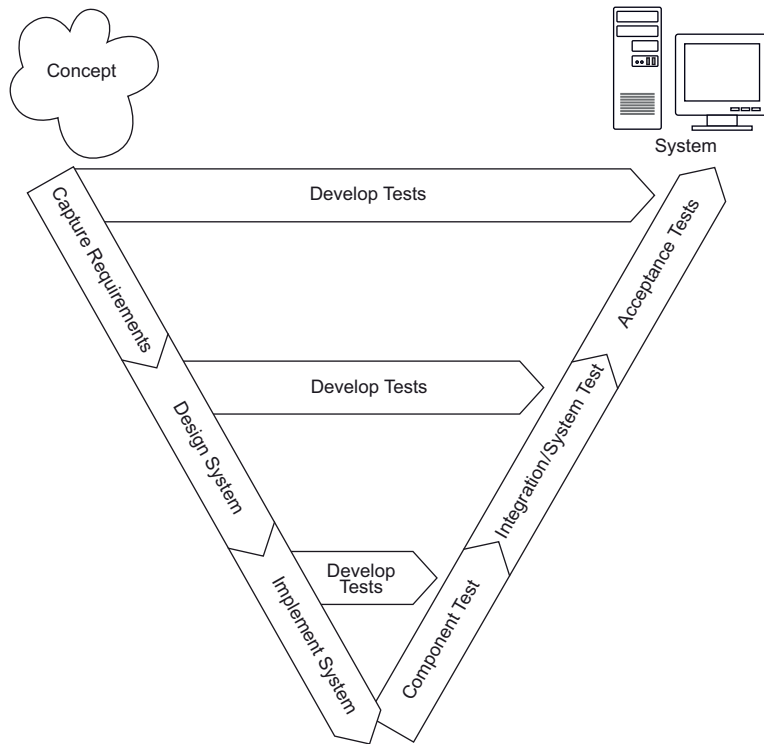


Figure 1-1 V-model

execution continues until the test exit criteria are met, though again some of these will often be waived.

Evaluation of test exit criteria and reporting of test results occur throughout test execution.

Test closure activities occur after test execution is declared complete.

This kind of precise alignment of test activities with each other and with the rest of the system lifecycle absolutely *will not* just happen. Nor can you expect to be able to instill this alignment continuously throughout the process, without any forethought.

Rather, for each test level, no matter what the selected software lifecycle and test process, the test manager must perform this alignment. Not only must this happen during the test and project planning, but test control includes acting to ensure on going alignment.

No matter what test process and software lifecycle are chosen, each project has its own quirks. This is especially true for complex projects such as the systems of systems projects common in the military and among RBCS's larger clients. In such a case, the test manager must plan not only to align test processes, but also to modify them. Off-the-rack process models, whether for testing alone or for the entire software lifecycle, don't fit such complex projects well.

1.3 Specific Systems

Learning objectives

Recall of content only

Systems of systems are independent systems tied together to serve a common purpose. Because they are independent and tied together, they often lack a single, coherent user or operator interface, a unified data model, compatible external interfaces, and so forth.

Such projects include the following characteristics and risks:

- The integration of commercial off-the-shelf (COTS) software along with some amount of custom development, often taking place over a long period.
- Significant technical, lifecycle, and organizational complexity and heterogeneity. This organizational and lifecycle complexity can include issues of confidentiality, company secrets, and regulations.
- Different development lifecycles and other processes among disparate teams, especially—as is frequently the case—when insourcing, outsourcing, and offshoring are involved.
- Serious potential reliability issues due to intersystem coupling, where one inherently weaker system creates ripple-effect failures across the entire system of systems.
- System integration testing, including interoperability testing, is essential. Well-defined interfaces for testing are needed.

At the risk of restating the obvious, systems of systems projects are more complex than single-system projects. The complexity increase applies organizationally, technically, process-wise, and team-wise. Good project management,

formal development lifecycles and process, configuration management, and quality assurance become more important as size and complexity increase.

Let's focus on the lifecycle implications for a moment.

As mentioned before, with systems of systems projects, we are typically going to have multiple levels of integration. First, we will have component integration for each system, and then we'll have system integration as we build the system of systems.

We will also typically have multiple version management and version control systems and processes, unless all the systems happen to be built by the same (presumably large) organization and that organization follows the same approach throughout its software development team. This is not something my associates and I commonly see during assessments of large companies, by the way.

The duration of the project tends to be long. I have seen them be planned for as long as five to seven years. A system of systems project with five or six systems might be considered relatively short and relatively small if it lasted "only" a year and involved "only" 40 or 50 people. Across this project, there are multiple test levels, usually owned by different parties.

Because of the size and complexity of the project, it's easy for handoff and transfers of responsibility to break down. So, we need formal information transfer among project members, especially at milestones.

Even when we're integrating purely off-the-shelf systems, these systems are evolving. That's all the more likely to be true with custom systems. So, we have the management challenge of coordinating development of the individual systems and the test analyst challenge of proper regression tests at the system of systems level when things change.

Especially with off-the-shelf systems, maintenance testing can be triggered—sometimes without much warning—by external entities and events such as obsolescence, bankruptcy, or upgrade of an individual system.

If you think of the fundamental test process in a system of systems project, the progress of levels is not two-dimensional. Instead, imagine a sort of pyramidal structure, as shown in figure 1-2.

At the base, you have component testing. A separate component test level exists for each system.

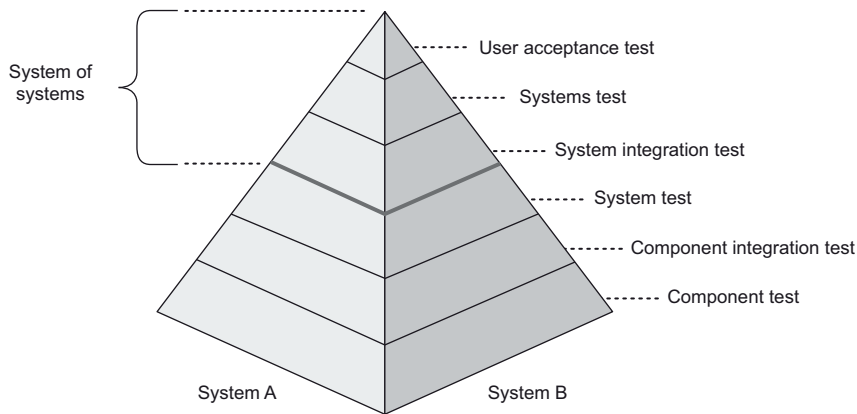


Figure 1-2 Fundamental test process in a system of systems project

Moving up the pyramid, you have component integration testing. A separate component integration test level exists for each system.

Next, you have system testing. A separate system test level exists for each system.

Note that, for each of these test levels, you have separate organizational ownership if the systems come from different vendors. You also probably have separate team ownership, because multiple groups often handle component, integration, and system test.

Continuing to move up the pyramid, you come to system integration testing. Now, finally, we are talking about a single test level across all systems. Next above that is systems testing, focusing on end-to-end tests that span all the systems. Finally, we have user acceptance testing. For each of these test levels, while we have single organizational ownership, we probably have separate team ownership.

Simply put, safety-critical systems are those systems upon which lives depend. Failure of such a system—or even temporary performance or reliability degradation or undesirable side effects as support actions are carried out—can injure or kill people, or, in the case of military systems, fail to injure or kill people at a battle-critical juncture.

Safety-critical systems, like systems of systems, have certain associated characteristics and risks:

ISTQB Glossary

safety-critical system: A system whose failure or malfunction may result in death or serious injury to people, loss or severe damage to equipment, or environmental harm.

- Because defects can cause death, and deaths can cause civil and criminal penalties, proof of adequate testing can be and often is used to reduce liability.
- For obvious reasons, various regulations and standards often apply to safety critical systems. The regulations and standards can constrain the process, the organizational structure, and the product. Unlike the usual constraints on a project, though, these are constructed specifically to increase the level of quality rather than to enable trade-offs to enhance schedule, budget, or feature outcomes at the expense of quality. Overall, there is a focus on quality as a very important project priority.
- There is typically a rigorous approach to both development and testing. Throughout the lifecycle, traceability extends all the way from regulatory requirements to test results. This provides a means of demonstrating compliance. This requires extensive, detailed documentation but provides high levels of audit ability, even by non-test experts.

Audits are common if regulations are imposed. Demonstrating compliance can involve tracing from the regulatory requirement through development to the test results. An outside party typically performs the audits. So, the traceability gathering up front and the audit/compliance activities at the back end affect management, development, testing, and the competent authorities, both from a people and a process point of view.

During the lifecycle—often as early as design—the project team uses safety analysis techniques to identify potential problems. Single points of failure are often resolved through system redundancy.

In some cases, safety-critical systems are complex systems or even systems of systems. In other cases, non-safety-critical components or systems are integrated into safety-critical systems or systems of systems. For example, networking or communication equipment is not inherently a safety-critical system, but

ISTQB Glossary

metric: A measurement scale and the method used for measurement.

measurement scale: A scale that constrains the type of data analysis that can be performed on it.

measurement: The process of assigning a number or category to an entity to describe an attribute of that entity.

measure: The number or category assigned to an attribute of an entity by making a measurement.

if integrated into an emergency dispatch or military system, it becomes part of a safety-critical system.

Formal quality risk management is essential in these situations. Fortunately, a number of such techniques exist, such as failure mode and effect analysis; failure mode, effect, and criticality analysis; hazard analysis; and software common cause failure analysis.

1.4 Metrics and Measurement

Learning objectives

Recall of content only

Throughout this book, we use metrics and measurement to establish expectations and guide testing by those expectations. You can and should apply metrics and measurements throughout the software development lifecycle. This is because well-established metrics and measures, aligned with project goals and objectives, will enable test analysts to track and report test and quality results to management in a consistent and coherent way.

A lack of metrics and measurements leads to purely subjective assessments of quality and testing. This results in disputes over the meaning of test results toward the end of the lifecycle. It also results in a lack of clearly perceived and communicated value, effectiveness, and efficiency for testing.

Not only must we have metrics and measurements, but also we need baselines. What is a “good” result for a given metric? An acceptable result? An unacceptable result? Without defined baselines, successful testing is usually impossible. In fact, when we perform assessments for our clients, we more often than not find ill-defined metrics of test team effectiveness and efficiency, with no baselines and thus bad and unrealistic expectations (which of course aren’t met).

There’s just about no end to what can be subjected to a metric and tracked through measurement. Consider the following:

- Planned schedule and coverage
- Requirements and their schedule, resource, and task implications for testing
- Workload and resource usage
- Milestones and scope of testing
- Planned and actual costs
- Risks, both quality and project risks
- Defects, including total found, total fixed, current backlog, average closure periods, and configuration, subsystem, priority, or severity distribution

During test planning, we establish expectations, which I mentioned as the baselines previously. As part of test control, we can measure actual outcomes and trends against these expectations. As part of test reporting, we can consistently explain to management various important aspects of the process, product, and project, using objective, agreed-upon metrics with realistic, achievable targets.

When thinking about a testing metrics and measurement program, there are three main areas to consider: definition, tracking, and reporting. Let’s start with definition.

In a successful testing metrics program, you define a useful, pertinent, and concise set of quality and test metrics for a project. You avoid too large a set of metrics, as this will prove difficult and perhaps expensive to measure while often confusing rather than enlightening the viewers and stakeholders.

You also want to ensure uniform, agreed-upon interpretations of these metrics to minimize disputes and divergent opinions about the meaning of certain measures of outcomes, analyses, and trends. There’s no point in having a metrics program if everyone has an utterly divergent opinion about what particular measures mean.

Finally, define metrics in terms of objectives and goals for a process or task, for components or systems, and for individuals or teams.

Victor Basili's well-known *Goal Question Metric* technique is one way to evolve meaningful metrics. Using this technique, we proceed from the goals of the effort—in this case, testing—to the questions we would have to answer to know if we were meeting those goals—to, ultimately, the specific metrics.

For example, one typical goal of testing is to build confidence. One natural question that arises in this regard is, “How much of the system has been tested?”. Metrics for coverage include percentage requirements covered by tests, percentage of branches and statements covered by tests, percentage of interfaces covered by tests, percentage of risks covered by tests, and so forth.

Let's move on to tracking.

Because tracking is a recurring activity in a metrics program, the use of automated tool support can reduce the time required to capture, track, analyze, report, and measure the metrics.

Be sure to apply objective and subjective analyses for specific metrics over time, especially when trends emerge that could allow for multiple interpretations of meaning. Try to avoid jumping to conclusions, or delivering metrics that encourage others to do so.

Be aware of and manage the tendency for people's interests to affect the interpretation they place on a particular metric or measure. Everyone likes to think they are objective—and, of course, right as well as fair!—but usually people's interests affect their conclusions.

Finally, let's look at reporting.

Most importantly, reporting of metrics and measures should enlighten management and other stakeholders, not confuse or misdirect them. In part, this is achieved through smart definition of metrics and careful tracking, but it is possible to take perfectly clear and meaningful metrics and confuse people with them through bad presentation. Edward Tufte's series of books, starting with *The Graphical Display of Quantitative Information*, is a treasure trove of ideas about how to develop good charts and graphs for reporting purposes.¹

1. The three books of Tufte's that I have read and can strongly recommend on this topic are *The Graphical Display of Quantitative Information*, *Visual Explanations*, and *Envisioning Information*.

ISTQB Glossary

ethics: No definition provided in the ISTQB Glossary.

Good testing reports based on metrics should be easily understood, not overly complex and certainly not ambiguous. The reports should draw the viewer's attention toward what matters most, not toward trivialities. In that way, good testing reports based on metrics and measures will help management guide the project to success.

Not all types of graphical displays of metrics are equal—or equally useful. A snapshot of data at a moment in time, as shown in a table, might be the right way to present some information, such as the coverage planned and achieved against certain critical quality risk areas. A graph of a trend over time might be a useful way to present other information, such as the total number of defects reported and the total number of defects resolved since the start of testing. An analysis of causes or relationships might be a useful way to present still other information, such as a scatter plot showing the correlation (or lack thereof) between years of tester experience and percentage of bug reports rejected.

1.5 Ethics

Learning objectives

Recall of content only

Many professions have ethical standards. In the context of professionalism, ethics are “rules of conduct recognized in respect to a particular class of human actions or a particular group, culture, etc.”²

Because, as a test analyst, you'll often have access to confidential and privileged information, ethical guidelines can help you to use that information appropriately. In addition, you should use ethical guidelines to choose the best possible behaviors and outcomes for a given situation, given your constraints. Note that “Best possible” means for everyone, not just the tester.

2. Definition from dictionary.com.

Let me give you an example of ethics in action. I am president of three related international software testing consultancies, RBCS, RBCS NZ, and PureTesting. I also serve on the International Software Testing Qualifications Board (ISTQB) and American Software Testing Qualifications Board (ASTQB) boards of directors. As such, I might and do have insight into the direction of the ISTQB program that our competitors in the software testing consultancy business don't have.

In some cases, such as helping to develop syllabi, I have to make those business interests clear to people, but I am allowed to help do so. I helped write both the Foundation and Advanced syllabi.

In other cases, such as developing exam questions, I agreed, along with my colleagues on the ASTQB, that I should not participate. Direct access to the exam questions would make it all too likely that, consciously or unconsciously, I would warp our training materials to "teach the exam."

As you advance in your career as a tester, more and more opportunities to show your ethical nature—or to be betrayed by a lack of it—will come your way. It's never too early to inculcate a strong sense of ethics.

The ISTQB Advanced Syllabus makes it clear that the ISTQB expects certificate-holders to adhere to the following code of ethics.

PUBLIC—Certified software testers shall act consistently with the public interest. For example, if you are working on a safety-critical system and are asked to quietly cancel some defect reports, that's an ethical problem.

CLIENT AND EMPLOYER—Certified software testers shall act in a manner that is in the best interests of their client and employer and consistent with the public interest. For example, if you know that your employer's major project is in trouble and you short-sell the stock and then leak information about the project problems to the Internet, that's a real ethical lapse—and probably a criminal one, too.

PRODUCT—Certified software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible. For example, if you are working as a consultant and you leave out important details from a test plan so that the client has to hire you on the next project, that's an ethical lapse.

JUDGMENT—Certified software testers shall maintain integrity and independence in their professional judgment. For example, if a project manager asks you not to report defects in certain areas due to potential business sponsor reactions, that's a blow to your independence and an ethical failure on your part if you comply.

MANAGEMENT—Certified software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing. For example, favoring one tester over another because you would like to establish a romantic relationship with the favored tester's sister is a serious lapse of managerial ethics.

PROFESSION—Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest. For example, if you have a chance to explain to your child's classmates or your spouse's colleagues what you do, be proud of it and explain the ways software testing benefits society.

COLLEAGUES—Certified software testers shall be fair to and supportive of their colleagues and promote cooperation with software developers. For example, it is unethical to manipulate test results to arrange the firing of a programmer who you detest.

SELF—Certified software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession. For example, attending courses, reading books, and speaking at conferences about what you do help to advance you—and the profession. This is called doing well while doing good, and fortunately, it is very ethical!

1.6 Sample Exam Questions

To end each chapter, you can try one or more sample exam questions to reinforce your knowledge and understanding of the material and to prepare for the ISTQB Advanced Level Test Analyst exam.

-
1. You are working as a test analyst at a bank. At the bank, test analysts work closely with users during user acceptance test. The bank has bought two financial applications as commercial off-the-shelf (COTS) software from large software vendors. Previous history with these vendors has shown that they deliver quality applications that work on their own, but this is the first time the bank will attempt to integrate applications from these two vendors. Which of the following test levels would you expect to be involved in? [Note: There might be more than one right answer.]
 - A Component test
 - B Component integration test
 - C System integration test
 - D Acceptance test

 2. Which of the following is necessarily true of safety critical systems?
 - A They are composed of multiple COTS applications.
 - B They are complex systems of systems.
 - C They are systems upon which lives depend.
 - D They are military or intelligence systems.