Proactive Testing

Not V, not X: This combination model that intertwines testing with development provides a happy medium that can expedite your projects. Part 3 of 4.

BY ROBIN F. GOLDSMITH AND DOROTHY GRAHAM

TESTING, WHILE AN INTEGRAL PART OF THE development process, doesn't often get the attention cast on other activities. Consequently, most testing models, despite their importance, are relatively unknown outside the communities that use them. The first two articles in this series, "The Forgotten Phase" (July 2002) and "V or X, This or That" (Aug. 2002) described the strengths and weaknesses of the time-honored V Model and the X Model we defined to describe the approaches currently being emphasized by some testing experts. This month, we introduce the Proactive Testing Model. We, as well as our clients and students, have found it to be useful for understanding and guiding effective software testing. Proactive testing draws value from both V and X Models, while reconciling their weaknesses.

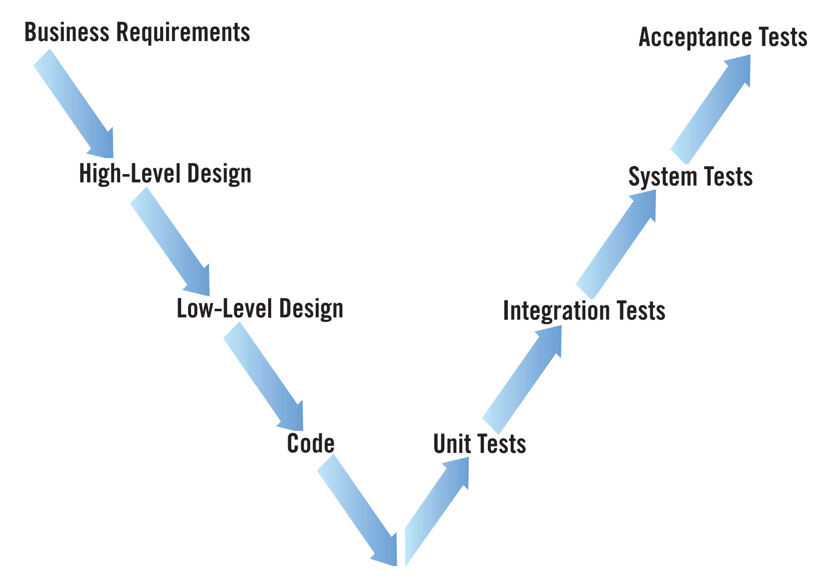
While proactive testing isn't perfect, it can provide significant benefits. Instead of waiting for perfection, we encourage using what you've got to gain as many benefits as you can. Moreover, we continually update the model as we discover issues that need to be addressed.

Because a single diagram isn't sufficient, we use three different graphical representations to represent the full range of concepts covered by the Proactive Testing Model. One of those charts appears on page 44. The other two charts will be published next month in Part 4, the final article in this series.

Development and Testing Combined

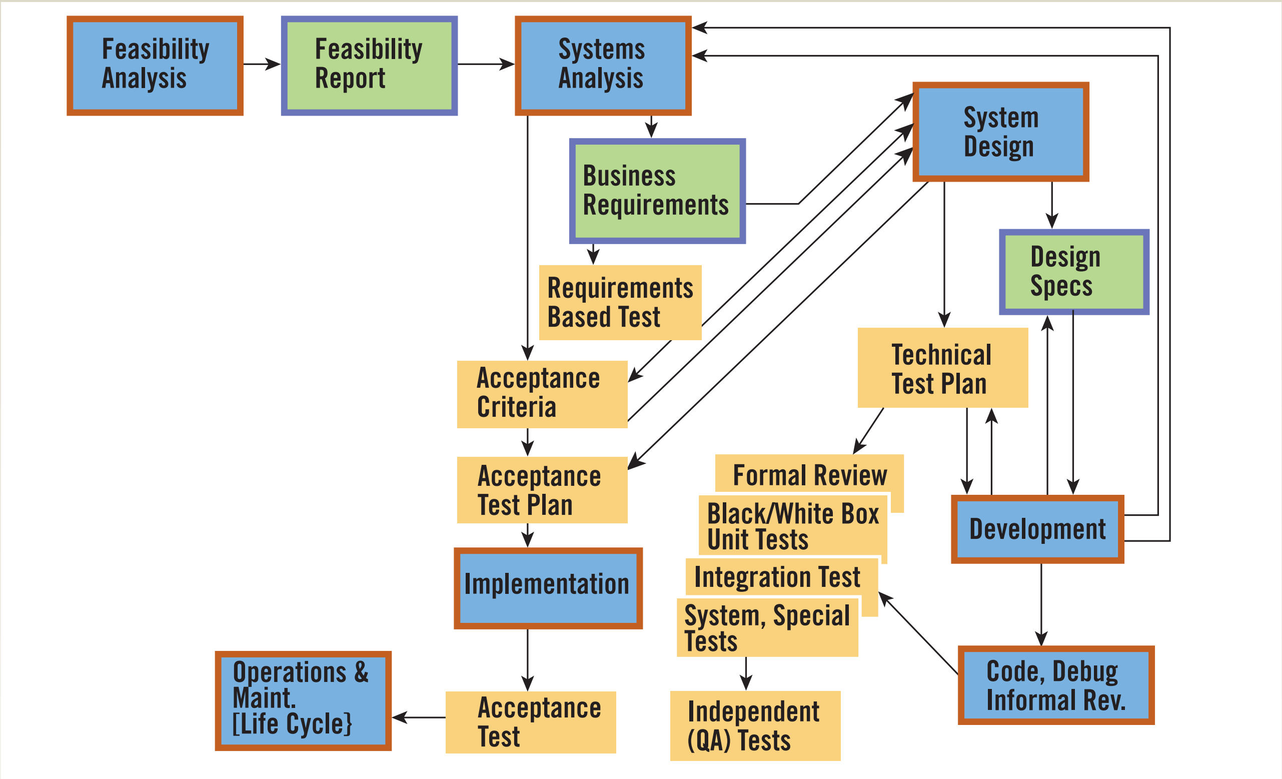
The Proactive Testing Model (page 43)

Robin F. Goldsmith *is president of Go Pro Management Inc., Needham, MA, which he cofounded in 1982. A frequent speaker at leading conferences, he works directly with and trains business and systems professionals in testing, requirements definition, software acquisition, and project and process management. Reach him at robin@gopromanagement.com.* ***Dorothy Graham*** *is the founder of Grove Consultants in the UK, which provides consultancy, training and inspiration in software testing, test automation and Inspection. She is coauthor with Tom Gilb of* Software Inspection *(Addison Wesley, 1993) and coauthor with Mark Fewster of* Software Test Automation *(Addison Wesley, 1999). In 1999, she was awarded the IBM European Excellence Award in Software Testing. She is on the board of* STQE *magazine, testing conferences and the British Computer Society Information Systems Examination Board (ISEB) Software Testing and Business System Development Boards.*



The V Model

The V proceeds from left to right, depicting the basic sequence of development and testing activities. The model is valuable because it highlights the existence of several levels of testing and depicts the way each relates to a different development phase.



The Proactive Testing Model

Proactive testing brings together development and testing lifecycles, identifying key activities from inception to retirement, and delineating the point in the project's lifecycle at which they provide the optimum value.

brings together development and testing lifecycles, identifying key activities from inception to retirement. When the activities are not performed or are performed at inappropriate points, the likelihood of success is compromised.

Thus, for example, systems are developed more effectively when business/user requirements are defined than when they are not. In fact, we'd suggest that it's impossible to develop a meaningful system without knowing the requirements. Moreover, systems are developed more effectively when business/user requirements are defined prior to (as opposed to after) design and development of the affected system components.

The Proactive Testing Model embodies several key concepts:

Test Each Deliverable

Each development deliverable should be tested by a suitable means at the time it is developed. Program code is not the only deliverable that needs to be tested. The shaded box outlines (see diagram above) are meant to show that we also need to test feasibility reports, business requirements and system designs. This is consistent with, but more explicit and expansive than, the level-to-level verification that is sometimes implied on the left (development) side of the V Model (see page 42).

For example, coauthor Goldsmith has identified and presents a popular seminar describing more than 21 techniques for testing the accuracy and completeness of business/user requirements. If they use any at all, most organizations use one or two weak techniques. The Proactive Testing Model includes two test-planning techniques that also happen to be among the more powerful of the 21-plus ways to test requirements.

One of these methods focuses on developing requirements-based test cases. This not only creates an initial set of tests to run against the code when it's delivered later on, but also confirms that the requirements are testable. These tests may be employed by users for acceptance testing and/or by the development organization for technical testing. Many in the testing community consider testability the primary, if not the only, issue with requirements, even to the extent of saying that writing a test case for each requirement is all that is needed. While testing each requirement is necessary, it isn't sufficient. Requirements-based tests are only as good as the requirements. A requirement can be completely wrong, yet still be testable. Moreover, one can't write test cases for requirements that have been overlooked.

The second technique involves defining acceptance criteria, which are what the business/user needs to have demonstrated before they're willing to rely on the delivered system. Acceptance criteria don't just *react* to defined requirements. Instead, they come *proactively* from a testing perspective and can help to reveal both incorrect and overlooked requirements.

Similarly, system designs should be tested to ensure accuracy and completeness before implementing them by writing code. Organizations tend to be better at testing designs than requirements, but still generally use only a few of the more than 15 ways to test designs that Goldsmith presents. One of the most powerful design tests comes from planning how to test the delivered system, which is most effective when performed during the design phase, prior to coding.

Plan and Design Tests During the Design Phase

The design phase is the appropriate time to do test planning and design. Many organizations either don't plan or design their testing or do it immediately prior to test execution, in which case, tests tend to demonstrate only that programs work the way they were written—not the way they *should* have been written.

Effective test planning can increase the number of errors that can be found economically.

There are two major types of tests, each of which needs a plan. As in the V Model, *acceptance tests* are defined earliest to be run latest, demonstrating that the delivered system meets the business/user requirements from the business/user perspective.

Unlike the V Model, the Proactive Testing Model recognizes three components of an acceptance test plan. Two of these components come in conjunction with defining the business/user requirements: defining requirements-based tests and defining acceptance criteria. However, the third component must wait until the system is designed, because the acceptance test plan really is formed by identifying how to use the system as designed to demonstrate that the acceptance criteria have been met and that requirements-based tests have been executed successfully.

Technical tests are tests of the developed code, such as the dynamic unit, integration and system tests identified in the V Model. In addition, proactive testing also reminds us to include static reviews (such as walk-throughs and inspections) and independent QA tests (which typically follow the system test to provide a final check from the technical organization's notion of the user's perspective), as well as special tests. We use the term *special tests* as a catch-all definition for tests such as load, security and usability testing that aren't specifically directed toward the application's business functionality.

Technical tests primarily demonstrate that the code as written conforms to the design. Conformance means that the system does what it should and doesn't do what it shouldn't. Technical tests are planned and designed during the design phase, to be carried out during development, primarily by the technical organization (not by users).

Intertwine Testing with Development

Proactive testing intertwines test execution with development in a code-test, code-test pattern during the development phase. That is, as soon as a piece of code is written, it should be tested. Ordinarily, the first tests would be unit tests, since the developer can carry out these tests to find errors most economically. However, as with the X Model, proactive testing recognizes that a piece of code may also need relevant integration tests and possibly some special tests shortly after the code is written. Thus, with respect to a given piece of code, the sequence of these various tests follows the V Model; but they're intertwined with development of the particular code components, not segregated into separate test phases.

The technical test plans should define this intertwining. The major approach and structure of the testing should be defined during the design phase and supplemented and updated during development. This especially affects code-based tests, which could be testing units or integrations. In either case, the test will be more efficient and effective with a bit of planning and design prior to execution, which of course also facilitates reuse.

Test planning and design are shown on diagrams in next month's article, which addresses additional aspects of this intertwining. Intertwining enables us to detect and eject (fix) more errors sooner after they are injected (or created), reducing error correction time, effort and cost. Note that the test plans themselves, not the Model, specifically determine which tests to run, and in which sequences. Similarly, effective test planning can increase the number of errors that can be found economically, but the plans should leave room for the additional ad hoc exploratory testing that often enables experienced testers to find more defects than their written test scripts would have found alone.

Keep Acceptance and Technical Testing Independent

Acceptance testing should be kept independent of technical testing so that it serves as a double-check to ensure that the design and its implementation in code meet business needs. Acceptance testing is run either as the first step of the implementation phase or the last step of the development phase—the same point in the process, regardless of its name.

The Proactive Testing Model advocates that acceptance and technical testing follow two independent paths. Each defines how to demonstrate that the system works properly from its own perspective. When the independently defined business/user acceptance tests corroborate the design-oriented technical tests, we can be confident that the system is correct.

Develop and Test Iteratively

Development and testing proceed together iteratively. At any point in the process, you can revisit prior phases and steps to correct their deliverables. This could entail fixing errors, eliminating superfluous elements or adding newly discovered ones; the Model does not dictate the size of the system portion involved. This is consistent with informed use of the V Model. However, proactive testing makes the iteration explicit.

Proactively Find Real WIIFMs

Proactive testing consciously seeks to identify and deliver the WIIFM (What's In It For Me) values that make users, developers and managers want to use the testing techniques. The WIIFMs that proactive testing reveals are different from the rationales traditionally given for doing reactive testing.

Instead of merely prioritizing, citing the lower costs of finding errors earlier or emphasizing the fact that testing is important to ensure better quality, proactive testing represents a wholly different attitude toward testing. (Heaven forbid, a paradigm shift!) Throughout the development process, we repeatedly use (and communicate about) testing in ways that enable developers, managers and users to save time and make their jobs easier.

Traditionally, developers often consider (reactive) testing to be an added burden that gets in the way of meeting programming deadlines. However, when we proactively define how to test the program *before* it's written, I've found that the developer can reasonably finish the program in at least 20 percent less time. Although developers are seldom conscious of how they actually spend their time, on reflection they usually realize that a large part of new development time is spent reworking the code they've already written. Conservatively, designing tests before coding can prevent at last half of the rework in the following ways:

- Designing tests is one of the more powerful ways to test a design, and thus help the developer avoid writing code that will need to be changed. Often the tests make design logic flaws apparent. In other instances, writing tests can reveal ambiguities. After all, if you can't figure out how to test the program, the developer probably won't be able to determine how to correctly program it.
- Tests created prior to coding show how the program ought to work, as opposed to tests created after the program has been written, which often just show that the program works the way the developer wrote it. Such tests help the developer catch and correct errors right away.
- When tests are defined prior to coding, the developer can begin testing as soon as the code is written. Moreover, she can be more efficient, executing more tests at a time, because her train of thought won't be interrupted repeatedly to find test data.
- Even the best programmers often interpret seemingly clear design specifications differently from the designer's intentions. By having the test input and expected result along with the specification, the developer can see concretely what is intended, and thereby code the program correctly the first time.

Proactively defining how to test a program before coding is a real WIIFM that developers greatly appreciate once they experience the technique. Not only does it *save* them time, but it reduces the rework they tend to hate the most.

Next month, in the final article of this series, we'll describe how Proactive test planning and design can prevent showstoppers and overruns by letting testing drive development.