This or That, V or X?

The X Model purports to fill the gaps left open by the V Model. But does it really handle all the facets of development, such as handoffs, frequent builds and so on? Part 2 of 4. BY ROBIN F. GOLDSMITH

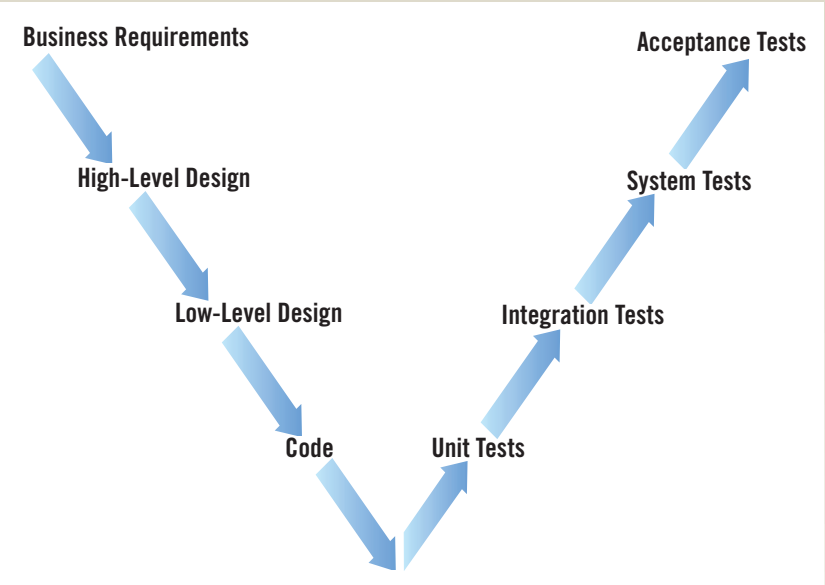
THE V MODEL IS PROBABLY THE BEST-KNOWN testing model, although many practicing testers may not be familiar with it—or any testing model. It has been around a long time and shares certain attributes with the waterfall development model—including a propensity to attract criticism. Since Part 1 of this series, "The Forgotten Phase" (July 2002), dealt in detail with the V Model, I'll give only a brief overview of it here.

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. *Unit testing* is code-based and performed primarily by developers to demonstrate that their smallest pieces of executable code function suitably. *Integration testing* demonstrates that two or more units or other integrations work together properly, and tends to focus on the interfaces specified in low-level design. When all the units and their various integrations have been tested, *system testing* demonstrates that the system works end-to-end in a production-like environment to provide the business functions specified in the high-level design. Finally, when the technical organization has completed these tests, the business or users perform *acceptance testing* to confirm that the system does, in fact, meet their business requirements.

Although many people dismiss the V Model, few have done so with the careful consideration that Brian Marick, author of *The Craft of Software Testing* (Prentice Hall, 1995), has demonstrated. In a debate with Dorothy Graham at the STAR (Software Testing Analysis and Review) 2000 East Conference and on his Web site (www .testing.com), Marick argued against the V Model's relevance.

The X Model

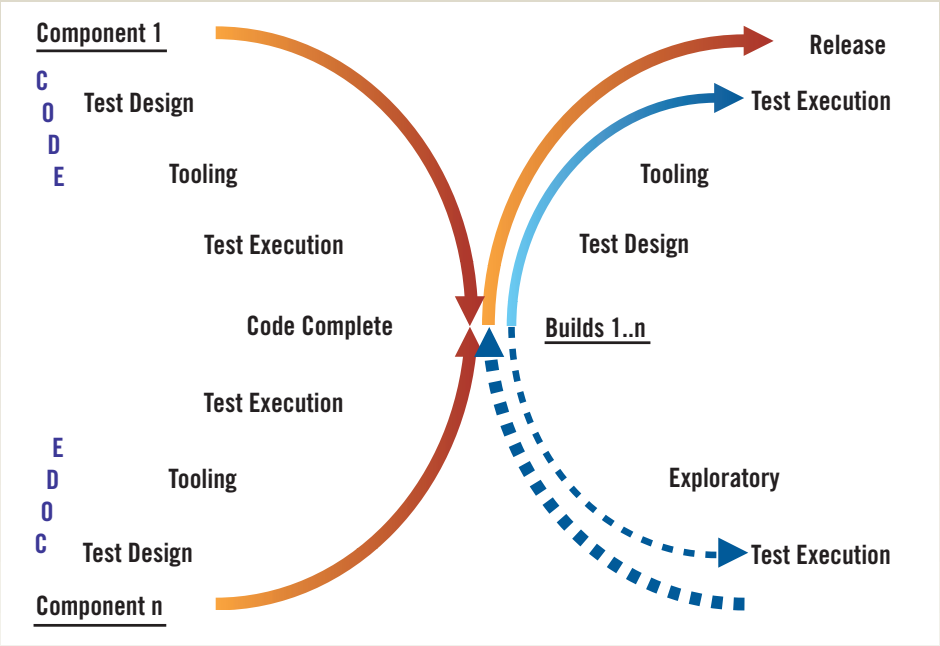
Marick raised issues and concerns, but is the first to acknowledge that he doesn't really propose an alternative model. I've taken the liberty of capturing some of Marick's thinking in what I've chosen to call, for want of a better term, the "X Model." Other than the sheer literary



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.

Robin F. Goldsmith *is the president of Go Pro Management Inc. consultancy in Needham, Mass., which he cofounded in 1982. A frequent conference speaker, he trains business and systems professionals in testing, requirements definition, software acquisition and project and process management. Reach him via www.gopromanagement.com.*



The X Model

The left side of the X Model depicts separate coding and testing of individual components, which then hand off frequently for integration into builds (upper-right quadrant) that also must be tested. Builds that pass their integration tests can be released either to users or for inclusion in larger integration builds.

brilliance of using *X* to contrast with *V*, there are some other reasons for choosing this particular letter: *X* often represents the unknown, and Marick acknowledged that his arguments didn't amount to a positive description of a model, but rather mainly to some concerns that a model should address. Prominent among these concerns is exploratory testing. Furthermore, with advance apologies lest the literary license offends, many of those sharing Marick's concerns undoubtedly could be described as members of Generation X. In addition, with further apologies to Marick, I've actually drawn an X-shaped diagram (see above) that I believe reasonably embodies the points Marick described in a somewhat different format.

Since an X Model has never really been articulated, its content must be inferred primarily from concerns raised about the adequacy of the V Model during the debate and in Marick's paper, "New Models for Test Development" (www.testing .com). My coverage of the X Model is brief because it hasn't been articulated to the extent of the V Model, and because I don't want to repeat some of the applicable general testing concepts discussed in Part 1.

Marick's quibble with the V Model centers largely on its inability to guide the total sequence of project tasks. He feels that a model must handle all the facts of development, such as handoffs, frequent repeated builds, and the lack of good and/or written requirements.

Coping With Handoffs and Frequent Build Cycles

Marick says a model should not prescribe behaviors that are contrary to actual acceptable practices, and I agree. The left side of the X Model depicts separate coding and testing of individual components, which then hand off frequently for integration into builds (upper-right quadrant) that also must be tested. Builds that pass their integration tests can be released either to users or for inclusion in larger integration builds. The lines are curved to indicate that iterations can occur with respect to all parts.

As seen in the graphic above, the X Model also addresses exploratory testing (lower-right quadrant). This is the unplanned "What if I try this?" type of ad hoc testing that often enables experienced testers to find more errors by going beyond planned tests. Marick didn't specifically address how exploratory testing fits in, but I'm sure he'd welcome its inclusion.

However, focusing on such low-level activities also creates concerns. A model isn't the same as an individual project plan. A model can't, and shouldn't be expected to, describe in detail every task in every project. It should, though, guide and support projects with such tasks. Surely, a code handoff is simply a form of integration, and the V Model doesn't limit itself to particular build cycle frequencies.

Marick and Graham agree that tests should be designed prior to execution. Marick advises, "Design when you have the knowledge; test when you have deliverables." The X Model includes test designing steps, as well as activities related to using various types of testing tools, whereas the V Model doesn't. However, Marick's examples suggest that the X Model isn't really a model in this regard; instead, it simply allows test-designing steps whenever, and if, one may choose to have them.

Planning Ahead

Marick questions the suitability of the V Model because it's based on a set of development steps in a particular sequence that may not reflect actual practice.

The V Model starts with requirements, even though many projects lack adequate requirements. The V Model reminds us to test whatever we've got at each development phase, but it doesn't prescribe how much we need to have. It could be argued that without requirements of any kind, how do developers know what to build? I would contend that the need for adequate requirements is no

less an issue for the X or any other model. Though it should work in their absence, an effective model encourages use of good practices. Therefore, one of the V Model's strengths is its explicit acknowledgment of the role of requirements, while the X Model's failure to do so is one of its shortcomings.

Marick also questions the value of distinguishing integration tests from unit tests, because in some situations one might skip unit testing in favor of just running integration tests. Marick fears that people will blindly follow a "pedant's V Model" in which activities are performed as the model dictates them, even though the activities may not work in practice. While it may not be readily apparent in the graphic on page 44, I have endeavored to incorporate Marick's desire for a flexible sense of activities. Thus, the X Model doesn't require that every component first be unit tested (activities on the left side) before being integration tested as part of a build (upper-right quadrant). The X Model also provides no guidelines for determining when skipping unit testing would and would not be suitable.

Out of Phase

A model's major purpose is to describe how to do something well. When the elements the model prescribes are missing or inadequate, a model helps us recognize the deficiency and understand its price. By not only acknowledging, but perhaps advocating that system development can proceed without adequate requirements, the X Model may often promote practices that actually require extra effort. Similarly, one can choose to skip unit tests in favor of integration tests. However, benefits may be illusory. Generally, it takes much more time and effort to fix errors detected by integration tests than those found by unit tests.

A model that is predicated upon poor practices, just because they are common, simply ensures that we'll keep repeating those poor practices without chance for improvement. Moreover, people then assume that the poor practices are not only unavoidable, but necessary; and ultimately end up regarding them as a virtue.

For example, instead of learning how to better define business/user requirements, many developers simply declare it an impossible task because "users don't know what they want" or "requirements are always changing." Then, these developers may further declare that it's not only OK, but preferable, not to know the requirements because they're using a superior development technique, such as prototyping. While such iterative techniques are indeed valuable for confirming comprehension, in practice, they're often merely a way to skip straight to coding. Even iteratively, coding is at best a highly inefficient, ineffective, labor-intensive way to discover real business/user requirements. Iteration is far more valuable when used in conjunction with overall project planning and more direct requirements discovery methods.

Similarly, the X Model and exploratory testing were developed in part as reactions against methods that seem to involve excessive time writing various test documents, with a resulting shortage of time actually executing tests. (Somehow, some testing "gurus" have turned lack of structure and avoidance of written test plans into virtues). I'd be the last person to encourage documentation, or any form of busywork, for its own sake. I've seen too many examples of voluminous test plans that weren't worth the time to write. That doesn't mean that writing test plans is a bad practice, only that writing poor test plans is a bad practice. On the other hand, writing down important information can pay for itself many times over. It makes us more thorough, less apt to forget, and able to share more reliably with others.

In the next two segments of this article, I'll reveal how suitable structure can in fact get software developed quicker, cheaper *and* better, and present what I call the Proactive Testing Model, which my clients, students and I have found useful. I believe it fills the gaps in the V and X Models, and also provides significant additional benefits for both testers and developers.