**Chapter 4: Ensuring Your Requirements Are correct: Requirement Validation Techniques**

**Overview**

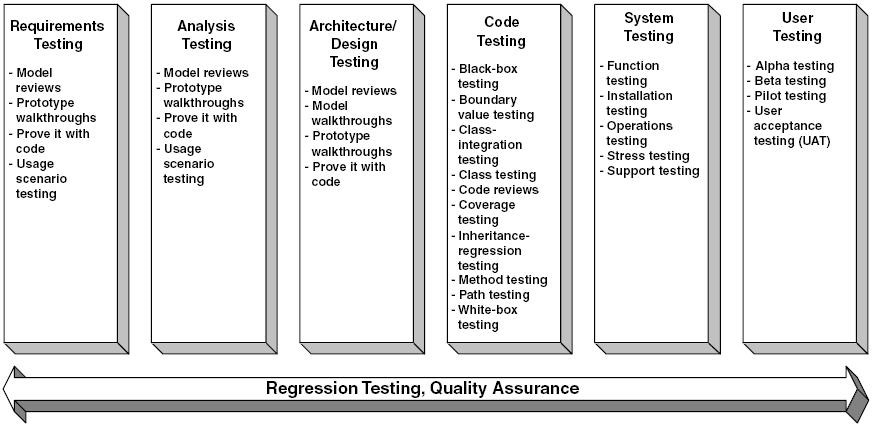
*Anything worth building is worth testing. You build a wide variety of artifacts, including models, documents, and source code.*

Software development is a complex endeavor. You may create a model in order to explore a business rule, a model that may then be used to drive your coding efforts. If the model is wrong then your code will be wrong too. You will also find that many artifacts, such as user manuals and operations manuals, never become code yet still need to be validated. The point is that you will need testing techniques that enable you to validate the wide range of artifacts that you create during software development.

.**4.1 Testing Philosophies**

A few philosophies with regards to testing:

1. **The goal is to find defects.** The primary purpose of testing is to validate the correctness of whatever it is that you are testing. In other words, successful tests find bugs.
2. **You can validate all artifacts.** As you will see in this chapter, you can test all your artifacts, not just your source code. At a minimum you can review models and documents and therefore find and fix defects long before they get into your code.
3. **Test often and early.**The potential for the cost of change to rise exponentially motivates you to test as early as possible.

  
Figure 4.1: The techniques of the full lifecycle object-oriented testing (FLOOT) methodology.

**4.2 Quality Assurance**

Quality assurance (QA) is the act of reviewing and auditing the project deliverables and activities to verify that they comply with the applicable standards, guidelines, and processes adopted by your organization. Fundamentally, quality assurance attempts to answer the following questions: "Are you building the right thing?" and "Are you building it the right way?" Perhaps a more effective question to ask would be "Can we build this a better way?" because it would provide valuable feedback that developers could use to improve the way that they work.

A key concept in quality assurance is that quality is often in the eye of the beholder, indicating many aspects exist to software quality, including the following:

* Does it meet the needs of its users?
* Does it provide value to its stakeholders?
* Does it follow relevant standards?
* Is it easy to use by its intended users?
* Is it reasonably free of defects?
* Is it easy to maintain and to enhance?
* How easy will it integrate into the current technical environment?

Quality assurance is critical to the success of a project and should be an integral part of all project stages, but only when it is done in an effective and efficient manner. For QA professionals to be relevant within an agile world, they need to be able to work in an agile manner. This means that they need to be willing to do the following:

* Work closely with other team members (they must do more than just review the work of others);
* Work in an evolutionary manner, understanding that artifacts change over time and are never "done" until you deliver the working system; and
* Gain a wider range of skills beyond that of QA.

**4.3 Testing Your Models**

You saw that the earlier you detect an error, the less expensive it is to fix. Therefore, it is imperative for you attempt to test your requirements, analysis, and design artifacts as early as you can. Luckily, a collection of techniques exist that you can apply to do exactly that. As you see in [Fig. 4](mk:@MSITStore:C:\Users\Preferred%20Customer\Desktop\OOSAD\the.object.primer.3rd.edition.chm::/0035.html#202).1 these techniques are

* Proving it with code;
* Usage scenario testing;
* Prototype walkthroughs;
* User interface testing; and
* Model reviews.

**4.4 Proving It with Code**

Everything works on a whiteboard, or on the screen of a sophisticated modeling tool, or in presentation slides. But how do you know whether it really works? You don't. The problem is that a model is an abstraction, one that should accurately reflect an aspect of whatever you are building. Until you build it, you really do not know whether it works. So build it and find out. If you have developed a screen sketch you should code it and show your users to get some feedback. If you have developed a UML sequence diagram representing the logic of a complex business rule, write the testing and business code to see whether you have gotten it right. Do a little bit of modeling, a little bit of coding, and a little bit of testing. This shortens the feedback loop and increases the chance that you will find problems as early as possible.

**4.5. Usage Scenario Testing**

Usage scenario testing, formerly called use-case scenario testing, is an integral part of the object-oriented development lifecycle. It is a technique that can be used to test your domain model, which is a representation of the business/domain concepts and their interrelationships, applicable to your system. A domain model helps to establish the vocabulary for your project.

Using a collection of usage scenarios, whereby a usage scenario is a series of steps describing how someone works with your system, you walk through your domain model and validate that it is able to support those scenarios. If it does not, you update your model appropriately. It can and should be performed in parallel with your domain modeling efforts by the same team that created your domain model, and in fact, many people consider usage scenario testing as simply an extension of CRC modeling. Fundamentally, usage scenario testing is a technique that helps to ensure that your domain model accurately reflects your business.

[Figure 4.2](mk:@MSITStore:C:\Users\Preferred%20Customer\Desktop\OOSAD\the.object.primer.3rd.edition.chm::/0040.html#wbp06Chapter3P235)presents an example of a usage scenario for a university information system.

|  |
| --- |
|  |



|  |  |  |
| --- | --- | --- |
| A student successfully enrolls in several seminars and pays partial tuition for them.  **Description**: A student decides to register in three seminars, which the student has the prerequisites for and which still have seats available in them, and pays half the tuition at the time of registration.  **Steps**: The student prepares to register:   * + The student determines the three seminars she wants to enroll in.   + The student looks up the prerequisites for the seminars to verify she is qualified to enroll in them.   + The student verifies spots are available in each seminar.   + The student determines the seminars fit into her schedule.   The student contacts the registrar to enroll in the seminars.   * + The student enrolls in the seminars:   + The student indicates to the registrar she wants to enroll in the seminars.   + For each seminar:     - The registrar verifies a spot is available in it.     - The registrar verifies the student is qualified to take the seminar.     - The registrar registers the student in the seminar.   A total bill for the registration is calculated and added to the student's outstanding balance (there is none).  The outstanding balance is presented to the student.  The student decides to pay half the balance immediately, and does so.  The registrar accepts the payment.  The payment is recorded.  The outstanding balance for the student is calculated and presented to the student. | | |
|  |

Figure 4.2: An example usage scenario.

**4.6 Prototype Reviews/Walkthroughs**

The user interface (UI) of an application is the portion the user directly interacts with: screens, reports, documentation, and your software support staff. A user interface prototype is a user interface that has been "mocked up" using a computer language or prototyping tool, but it does not yet implement the full system functionality.

A prototype walkthrough is a testing process in which your users work through a series of usage scenarios to verify that a user prototype meets their needs. It is basically usage scenario testing applied to a user interface prototype instead of a domain model. The basic idea is that your users pretend the prototype is the real application and try to use it to solve real business problems described by the scenarios. Granted, they need to use their imaginations to fill in the functionality the application is missing (such as reading and writing objects from/to permanent storage), but, for the most part, this is a fairly straightforward process. Your users sit down at the computer and begin to work through the use cases. Your job is to sit there and observe them, looking for places where the system is difficult to use or is missing features. In many ways, prototype walkthroughs are a lot like user-acceptance tests, the only difference being you are working with the prototype instead of the real system.

**4.7 User-Interface Testing**

UI testing is the verification that the UI follows the accepted standards chosen by your organization and the UI meets the requirements defined for it. User- interface testing is often referred to as graphical user interface (GUI) testing. UI testing can be something as simple as verifying that your application "does the right thing" when subjected to a defined set of user-interface events, such as keyboard input, or something as complex as a usability study where human-factors engineers verify that the software is intuitive and easy to use.

**4.8 Model Reviews**

A model review, also called a model walkthrough or a model inspection, is a validation technique in which your modeling efforts are examined critically by a group of your peers. The basic idea is that a group of qualified people, often both technical staff get together in a room to evaluate a model or document. The purpose of this evaluation is to determine whether the models not only fulfill the demands of the user community but also are of sufficient quality to be easy to develop, maintain, and enhance. When model reviews are performed properly, they can have a large payoff because they often identify defects early in the project, reducing the cost of fixing them

There are different "flavors" of model review. A requirements review is a type of model review in which a group of users and/or recognized experts review your requirements artifacts. The purpose of a user requirement review is to ensure your requirements accurately reflect the needs and priorities of your user community and to ensure your understanding is sufficient from which to develop software. Similarly an architecture review focuses on reviewing architectural models and a design review focuses on reviewing design models. As you would expect the reviewers are often technical staff.

*In general if you are going to hold a review, the following pointers should help you to make it effective:*

1. **Get the right people in the review.** You want people, and only those people, who know what they are looking at and can provide valuable feedback. Better yet, include them in your development efforts and avoid the review in the first place.
2. **Review working software, not models.** The traditional, near-serial development approach currently favored within many organizations provides little else for project stakeholders to look at during most of a project. However, because the iterative and incremental approach of agile development techniques tightens the development cycle you will find that user- acceptance testing can replace many model review efforts.
3. **Stay focused.** This is related to maximizing value: you want to keep reviews short and sweet. The purpose of the review should be clear to everyone; for example, if it is a requirements review do not start discussing database design issues. At the same time recognize that it is okay for an informal or impromptu model review to "devolve" into a modeling/working session as long as that effort remains focused on the issue at hand.
4. **Understand that quality comes from more than just reviews.** In application development, quality comes from developers who understand how to build software properly, who have learned from experience, and/or who have gained these skills from training and education. Reviews help you to identify quality deficits, but they will not help you build quality into your application from the outset. Reviews should be only a small portion of your overall testing and quality strategy.
5. **Set expectations ahead of time.** The expectations of the reviewers must be realistic if the review is to run smoothly. Issues that reviewers should be aware of are
   * The more detail a document has, the easier it is to find fault.
   * With an evolutionary approach your models are not complete until the software is ready to ship.
   * Agile developers are likely to be traveling light and therefore their documentation may not be "complete" either.
   * The more clearly defined a position on an issue, the easier it is to find fault.
   * Finding many faults may often imply a good, not a bad, job has been performed.
   * The goal is to find gaps in the work, so they can be addressed appropriately.
6. **Understand you cannot review everything.** You should prioritize your artifacts on a risk basis and review those that present the highest risk to your project if they contain serious defects.
7. **Focus on communication.** Reviews are vehicles for knowledge transfer, that they are opportunities for people to share and discuss ideas. However, working closely with your co-workers and project stakeholders while you are actually modeling is even more effective for this purpose than reviews. This philosophy motivates agile developers to avoid formal reviews, due to their restrictions on how people are allowed to interact, in favor of other model validation techniques.
8. **Put observers to work.** People will often ask to observe a review either to become trained in the review process or to get updated on the project.