Pragmatic Product Leader Course

Become a Product Manager with Superpowers

LESSON 6F

B.O.O.M Step 2 contd. & Step 3

6.1. Step 2: Discovery contd.

2c. Specify testing

B.O.O.M. Step 2 - Discovery

2c Specify testing

Learn how to design test cases that are most likely to uncover errors— an activity that should begin during the Discovery phase, before the Development phase (design and coding). By specifying these tests up front, you add measurable quality requirements—clear-cut acceptance criteria for the software.

Who does these tests and how does the PM fit in?
PM's responsibility is to support testing.

What is testing?
Testing is any activity aimed at proving that the software system does not do what it is supposed to.

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Black-box testing**

General Guidelines

Ivar Jacobson, a founder of OO, advises that you derive test cases from system use cases as follows:

- Test scenarios that cover the basic flow of each use case
- Tests scenarios that cover the alternate and exception flows of each use case
- Tests of line-item requirements in the PRD, where the requirements are traced to use case(s)
- Tests of features in the user documentation, where the documentation is traced to use case(s)

You'll need more than these suggestions to plan appropriate testing. Fortunately, much groundwork regarding testing has been done prior to OO. This pre-OO approach is called *structured testing*.

Structured testing

Structured testing is the process of using standardized techniques to locate flaws (bugs). Flaws detected by structured testing include those introduced during business analysis, design, and programming.

B.O.O.M. Step 2 - Discovery

2c Specify testing

When is testing performed?

Different activities occur at various phases of project development:

- Structured walkthroughs are performed throughout the project.
- During the Discovery phase, the test cases are designed. On an iterative project, architectural proofs of concept are implemented and tested as well during this phase.
- During the Construction phase, unit testing (tests of the individual software components) is carried out.
- As system use-case scenarios are implemented during the Construction phase, requirements-based (black-box) tests are performed to verify compliance with the requirements.
- Before acceptance of the product, the developers or technical testers perform system tests.
- During the Closeout phase, the user performs and supervises user acceptance testing (UAT).

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Structured testing**

Principles of Structured Testing (Adapted for OO) - REFER

You begin by establishing some principles, adapted from structured testing to OO projects. Why not go directly to the techniques? A sound understanding of basic principles helps avoid the kinds of institutional problems that lead to buggy systems.

Structured Walkthroughs - REFER

Most people think of testing as a process involving the execution of a program by the computer. This is only one type of testing—referred to, appropriately, as computer-based testing. Testing, however, also includes non-computer-based tests. How can you test a system without actually executing it? You walk through some aspect of the system manually with a group of participants. Formally this is the structured walkthrough.

Why are structured walkthroughs an important aspect of testing?

Errors are often thought to be exclusively due to bad programming; in fact they can be introduced at any stage of a project. The beauty of walkthroughs is that, unlike computer- based testing, they can be performed before the software is written. **Early testing means early detection of errors.** Also, unlike computer-based tests, walkthroughs tend to find the cause of a problem, not just its symptom.

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Structured testing**

Testing Techniques

To Do the Following	Use These Tools
Ensure that all of the code has been covered properly during testing	White-box techniques: statement, decision, condition, multiple condition coverage
Test the requirements for completeness and accuracy	Structured walkthrough
Test functional requirements for end-to-end business process	High-level integration tests based on business use cases
Test the system's response to simple conditions	Condition response tables
Test the system's response to a group of input conditions that might occur in any combination	Decision tables, decision trees
Design test data most likely to uncover bugs	Boundary-value analysis
Test how the system handles high volume	Volume test (a type of system test)
Test how the system handles a high level of activity within a short period of time	Stress test (a type of system test)
Test for user-friendliness	Usability test (a type of system test)
Test speed	Performance test (a type of system test)
Ensure processes that should remain unaffected by the release work as before	Regression test (a type of system test)
Submit the system for final acceptance by the user	User acceptance testing (UAT)

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Black-box testing**

Requirements-based (Black-box) Testing

The purpose of requirement-based testing is to find variances between the software and the requirements. The product requirements document (PRD) acts as the reference point for these tests.

Limitations

Time consuming, since for full coverage, you'd have to test every possible set of input values and conditions.

Use-Case scenario testing

One approach to requirements-based testing that tests the various scenarios of each use case.

Deriving Use-case scenario tests

Recall that your processing requirements were grouped around system use cases, each with their own group of scenarios. The flows were chosen so that they would cover all important scenarios:

- Basic flow: The normal path through the use case
- Alternate flows: Rarely occurring flows and other variations from the norm
- Exception flows: Unrecoverable errors and other flows that cause interaction ending without the user meeting the goal of the use case

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Black-box testing**

Requirements-based (Black-box) Testing - Test Template

Test Template

Test #: _____ Project #: _____

System: _____ Test environment: _____

Test type (e.g., regression/ requirements-based, etc.):

Test objective: _____

System use case: _____ Flow: _____

Priority: _____

Next step in case of failure: _____

Planned start date: _____ Planned end date: _____

Actual start date: _____ Actual end date: _____

Times to repeat: _____

Pre-conditions (must be true before test begins): ____

Req #	Action/Data	Expected Result/Response	Actual Result	Pass/ Fail

Tester ID: _____

Pass/fail: _____ Severity of failure: _____

Solution: _____

Comments: _____

Sign-off: _____

(Req # is short for requirement number and corresponds to the number used to identify the requirement in the BRD. Many organizations number their requirements so that they can be traced forward to test cases and other project artifacts. The numbering may be manual, or automatically generated with the use of a tool such as Rational RequisitePro.)

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Black-box testing**

Decision Table for Testing

When the input conditions affecting a system use case are interrelated, it is not enough to test for each input condition separately; you must test all combinations of input conditions. An input condition is any condition that will have an impact on the system response.

What the Decision table does not say about testing

The decision table shows only the net result of each test; it does not show the required sequencing of steps. For this reason, you need to complete the test template for each test scenario, indicating the expected sequence of actions. Use the system use-case description to work out the expected workflow for each case.

		1	2	3	4	5	6	7	8	9	10	11	12
CONDITION	Code of Good Practice Followed (Y/N)	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
	Steps and Procedures Followed? (Y/N)	Y	Y	Y	N	N	N	Y	Y	Y	N	N	N
	# PC Members (0–2,3–5,6–99)	0–2	3–5	6–99	0–2	3–5	6–99	0–2	3–5	6–99	0–2	3–5	6–99
ACTION	Mark as Not Payable	X			X			X	X	X	X	X	X
	Mark as Payable		X	X		X	X						
	Pay ½ Standard Amount					X	X						
	Pay Standard Amount		X										
	Pay Double Amount			X									

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Black-box testing**

Boundary-Value Analysis

Boundary-value analysis is a technique for targeting test data most likely to reveal bugs. The technique is based on the premise that the system is most error-prone at points of change.

Boundary-value analysis can help you pinpoint test data for any requirements-based (black- box) test. If you are working from a decision table, then boundary-value analysis can help you decide which data to use for the test(s) indicated by each column of the table. The technique covers both positive and negative testing:

- A positive test is one that tests the system's response to valid conditions (success scenarios).
- A negative test is one that tests the system's response to invalid conditions (errors).

A summary of boundary-value analysis rules is in speaker notes

B.O.O.M. Step 2 - Discovery

White-box testing

White-box testing is a testing methodology based on knowledge of the internal workings of the IT system.

Who does white-box testing?

Developers perform these tests, since knowledge of programming code is required. But as a PM, you have a supporting role: you may be required to specify the *level* of white-box testing that the software must pass through before it is accepted.

Limitations of white-box testing

To white-box test a program fully, then, you would need to try all possible paths of statement execution. Because the number of tests usually required for full coverage is so high, other approaches are used to winnow the set of tests to a manageable size.

B.O.O.M. Step 2 - Discovery

2c Specify testing > **White-box testing**

White-box Coverage Quality Levels

The following coverage levels are used to specify the degree of thoroughness of white-box testing, listed in order of increasing coverage. Depending on the level of risk, you may specify one of the following coverages:

- Statement coverage
- Decision coverage
- Condition coverage
- Multiple condition coverage

Decision tables can be used in a programming context to derive multiple condition coverage tests, using source- level conditions and actions. Use of decision tables in this way is beyond your role as PM because of the programming knowledge it requires.

B.O.O.M. Step 2 - Discovery

2c Specify testing > **White-box testing**

Sequencing of white-box tests

When software is written, it is developed in modules, or units. In both structured and OO environments, a plan must be put together to sequence the testing of these units and their proper integration within the software. The process of planning and executing these piecemeal tests is called unit testing.

Unit testing and the PM

PM plays a supporting role to the developer in the planning and scheduling of these tests.

Big Bang approach to unit testing

There are a number of approaches for the sequencing of unit tests. In the big bang approach, each unit is first tested individually. Once this is complete, all units are integrated and tested in one “big bang” test.

Incremental Approaches to unit testing

Each unit is added to the system one by one, incrementally and tests are conducted. Two types:

Top-down Testing - From mainline program towards the low-level units that carry out basic functions

Bottom-up Testing - From low-level routines to higher-level routines

B.O.O.M. Step 2 - Discovery

2c Specify testing > **System tests**

System tests

The idea behind system testing is that even if the code has been adequately tested for coverage (white-box testing) and has been shown to do everything expressed in the user requirements, it may still fail because it doesn't do these things well enough. It may not meet other objectives—such as those related to security, speed, and so on. With the exception of usability testing (a type of system test), you will not typically perform these tests, but you may be involved in planning them and in verifying that the tests have been conducted, so you should be aware of the tests in this category. Some popular tests:

- Regression testing
- Volume testing
- Stress testing
- Usability testing
- Security testing
- Performance testing
    - Response time
    - CPU time
    - Throughput
- Storage testing
- Configuration testing
- Compatibility/conversion testing
- Reliability testing
- Recovery testing

B.O.O.M. Step 2 - Discovery

2c Specify testing > **Beyond the System tests**

Beyond the System tests

The PM should plan for a final set of tests to take place after the system tests are complete. These are UAT, beta testing, parallel testing, and installation testing.

More about the tests in speaker notes.

2d. Specify implementation plan

B.O.O.M. Step 2 - Discovery

2d Specify implementation plan

Specify implementation plan

The PRD must include an implementation plan so that steps required when releasing the system can be planned for in advance. The issues addressed typically include the following:

Training:

- Who is to be trained?
- How will training be done?
- What resources (hardware, software, training rooms, trainers, administration, and so on) will be required?

Conversion:

- Identify existing data that will need to be converted (due to new file formats, new database management software, and so on).
- Plan promotion of programs (from the current version to the new one).
- Plan granting of privileges to the users.
- Schedule jobs (for batch systems).
- Advise operations of which jobs to add to the production run: daily, weekly, monthly, quarterly, semi-annually, or annually.
- Ensure that the job is planned to be executed in the right sequence— that is, after certain jobs are run and before others.
- Advise operations of the reports to be printed and the distribution list for reports and files.

Rollout:

- Advise all affected users of the promotion date for the project.

End-user procedures:

- Write up the procedures for the affected departments.
- Distribute an end-user procedures document to affected departments.

6.2. Step 3: Development and onwards

B.O.O.M. Step 3 - Development

As the project moves into the Development phase, the developers (systems analyst, systems architect, database administrator, and so on) start the work of adapting your business model for technical use. On a waterfall project, this is the point at which your active participation stops. On an iterative project, you continue gathering requirements during the Construction phase, completing the requirements needed for the selected use-case scenarios before they are implemented in an iteration. In either case, once design and coding are underway, you need to be available to answer the questions that inevitably arise during the development process.

6.3. Conclusion

Conclusion

By now, most likely you would be overwhelmed with the sheer number of concepts you have to remember. Not much to worry though as you won't be implementing everything described here unless you are working on a very complex product in a large organisation. You don't have to be pedantic either and produce perfect UML diagrams unless of course that's what you have been asked to. Idea is, you need to be aware of the various tools and techniques at your disposal to model various scenarios. As long as you are able to clearly articulate what you want to be built, you are good to go.

Thank you.