Best Practices for the Formal Software Testing Process

A Menu of Testing Tasks

Rodger D. Drabick

foreword by William E. Perry

BEST PRACTICES FOR THE FORMAL SOFTWARE TESTING PROCESS

Best Practices for the Formal Software Testing Process

A Menu of Testing Tasks

Rodger D. Drabick

DH

Dorset House Publishing
353 West 12th Street
New York, NY 10014

Library of Congress Cataloging-in-Publication Data

Drabick, Rodger.
Best practices for the formal software testing process : a menu of testing tasks / Rodger Drabick.
p. cm.
Includes bibliographical references and index.
ISBN 0-932633-58-7 (softcover)
1. Computer software--Testing. I. Title.

QA76.76.T48D73 2003
005.1'4--dc22

2003062472

Cover Design: Nuno Andrade
Cover Graphic: Detail from Figure 1-5
Cover Author Photograph: Courtesy of Lockheed Martin

Distributed in the English language in Singapore, the Philippines, and Southeast Asia by Alkem Company (S) Pte. Ltd., Singapore; in the English language in India, Bangladesh, Sri Lanka, Nepal, and Mauritius by Prism Books Pvt., Ltd., Bangalore, India; and in the English language in Japan by Toppan Co., Ltd., Tokyo, Japan.

Printed in the United States of America

Library of Congress Catalog Number: 2003062472

ISBN: 0-932633-58-7 12 11 10 9 8 7 6 5 4 3 2 1

Digital release by Pearson Education, Inc., June, 2013

Dedication

To my wife, Karen, without whose love and support this book and the other things I do would not be possible; and to my daughters, Alyson and Liz—I love you all.

This page intentionally left blank

Acknowledgments

Many people—including coworkers, managers, SQA and testing associates, family, and friends—have earned a special Thank You for helping me bring my ideas from a concept to a published book.

I'd like first to express my thanks—in memoriam—to Peter Natale, formerly of Eastman Kodak Company, who provided me with my first opportunity in software testing. Without Pete's support and mentorship, this book would not have been possible. Thanks also to Frank Mondziel, who provided guidance in our first efforts in software QA and testing at Kodak, and to Ed Cattron, a long-time friend, coworker, and manager at Kodak, who supported my initial efforts to develop and formalize this process model on a large and complex Internal Revenue Service program in the early 1990's.

I also owe a great debt of gratitude to Bill Perry of the Quality Assurance Institute and Dr. Edward Miller of Software Research, Inc., both of whom gave me the opportunity to present portions of this model at their testing conferences. I made my first presentation on the topic as a keynote address at a Software Quality Week, hosted by Ed and his company. Independent consultant Denis Meredith gave me the opportunity to present the model end-to-end at one of the Testing Computer Software Conferences he coordinated. Denis remains my "go to guy" for questions about requirements and risk management.

Thank you very much, gentlemen! It's been a pleasure knowing you, and learning from you.

To so many at Software Quality Engineering who have been helpful to me along the way, I give special thanks: Bill Hetzel, former president of SQE, and Dave Gelperin and Rick Craig, associates at SQE—all of you have provided invaluable technical and moral support throughout my testing career.

Lisa Crispin, coauthor of *Testing Extreme Programming,* introduced me to special techniques for approaching testing of that agile method. For that knowledge and for reviewing a number of draft chapters, Lisa, I am deeply grateful.

It's difficult to adequately express the gratitude I owe to several editors. First, my friend of many years, Debbie Lafferty, formerly Acquisitions Editor at Addison-Wesley, started me working on this book. Debbie introduced me on-line to Wendy Eakin at Dorset House Publishing, and that introduction has resulted in the final product you hold in your hands. To Wendy and the rest of the Dorset House staff, including Nuno Andrade, Vincent Au, and David McClintock, many thanks for helping me turn a concept into a real book that I hope will provide valuable information to the testing community! Thanks also to Dorset House author and software systems consultant Tim Lister, for his recommendation that this book be published.

And finally, a lifetime of gratitude and thanks to Karen, my wife and friend of many years. Without your help and encouragement, this book would never have been started, much less completed.

Contents

Foreword

Rodger and I first met in 1985 at the Testing Computer Software Conference, which I coordinated. This was Rodger's first software testing conference, and like many folks new to software testing, he was determined to learn as much as he could about the fascinating field of software testing in the limited time available. At the time, Rodger had been working as a team lead for system testing on a large software program for a government customer.

Over the years, I watched Rodger's career develop, and witnessed as his focus changed from learning and using testing techniques, to testing management, and finally to concentration on the software testing process. In the mid-1990's, Rodger presented aspects of his process model at the Quality Assurance Institute's International Conference on Software Testing, the same process model that this book describes in detail.

Best Practices for the Formal Software Testing Process: A Menu of Testing Tasks is the product of the knowledge that Rodger has gained through personal experience as well as from seminars and conferences put on by the QAI and other well-respected testing organizations.

Primarily intended to help new test engineers and test engineering managers understand the steps involved in testing software systems and hardware-software systems, this book will be useful to any testing organization interested in documenting and improving its testing process. It begins by showing you how to develop a test plan based on the software and system requirements, which identifies exactly what will be tested, and goes on to detail the tasks and processes involved in executing the test and documenting the results of the testing.

While this book focuses primarily on testing associated with large development programs—using either a spiral life cycle or the Unified Software Development Process, with its iterative phases—the information Rodger offers can be put to the most effective use through tailoring the process to fit your specific environment.

Readers familiar with the testing documentation and testing life cycle contained in IEEE-Standard-829, the Standard for Software Test Documentation, will readily see the links to the process model detailed in this book. Nevertheless, those working with other life cycles will find guidance in tailoring this testing process to fit those life cycles.

Best Practices for the Formal Software Testing Process serves as a fine example of What to Do to Test. As Rodger writes, this is the book he wished he'd had when he started his software testing career. This book should be in the hands of every member of every software testing organization. If you buy this book, plan to use it; don't just set it on your bookshelf.

September 2003 William E. Perry
Orlando, Florida

Preface

What is "the formal software testing process"?

One of the definitions the Institute of Electrical and Electronics Engineers (IEEE) Software Standards collection provides for *process* is "a course of action to be taken to perform a given task" or "a written description of a course of actions, for example, a documented test procedure." Various editions of *Webster's Dictionary* offer a series of definitions for process, including "a particular method of doing something, generally involving a number of steps or operations."

A simpler definition of a process is "the way we do things."

Thus, we could say that the "testing process" is "a course of action to be taken to perform formal testing," or "the way we do testing here."

But, how does "the way we do testing here" stack up against industry standards for best testing practices, and why did I go to all the effort to define a Formal Testing Process Model? Let me share a short personal retrospective.

I have spent twenty-eight years of a mostly exciting technical career testing and managing testing programs. These programs started out as hardware programs, and evolved into hardware and software systems programs in the mid-1970's. I was lucky in some respects to work for a large company in the Department of Defense (DoD) area; thus, we were always in an environment that forced us to be rigorous and methodical, both in test execution and in test documentation. Note that being methodical didn't always lead to delivering successful systems.

When I first began working in programs that were primarily software, I was looking for sources of material on "best practices" relative to the software testing process. Being in the DoD environment, our customers supplied Contract Deliverable Requirements Lists (CDRLs) that identified the documents we had to write. Thus, we had a framework for our programs. Later, I discovered a number of military standards (for example, DoD-Standards 2167A and 2168), and the IEEE Software Engineering Standards, which I continue to use to this day. Following the series of documentation contained in IEEE-Standard-829, the Standard for Software Test Documentation, suggested a standard process to utilize for software and systems test documentation.

In 1992, while working in the Commercial and Government Systems Division of the Eastman Kodak Company, my job was to develop a testing estimate for a program we were bidding for the Internal Revenue Service. Having built estimates previously for large, complex, testing programs, I knew that I needed a logical, structured list of tasks to form the basis of my estimate. I could then convert this task list into a work breakdown structure (WBS), and estimate the time to perform each task.

About this same time, I was reading Watts Humphrey's fascinating book *Managing the Software Process* (Humphrey, 1989). Watts showed how to use Input-Process-Output (IPO) diagrams to document a process; I applied that technique to the testing process. That was the first formal documentation of the top levels of the model you will see described in this book. I published an article describing how aspects of this model could be used in estimating, in 1993 in the June issue of Ed Yourdon's *American Programmer* newsletter (Yourdon, 1993).[1] That's one use of such a model. Over the years, I continued to refine the model until it became the fairly detailed model presented in this book.

Incidentally, our prime contractor accepted my estimate, and both it and the Internal Revenue Service accepted the overall estimate.

The first light bulb that lit as I began to learn about software testing is that testing has a life cycle that is closely linked to the software development life cycle. What seems like a logical truth in 2003 was a real revelation to many in the mid-1970's. I still think that the "V-diagram" is a thing of beauty. The V-diagram is a means of illustrating the relationship between the concurrent software development and testing life cycles (see Figure 1-2 for one example of the V-diagram). Unfortunately, there are still a large number of people and corpora-

[1]Rodger D. Drabick, "A Process Model Tool for Estimating Software Costs," *American Programmer*, Vol. 6, No. 6 (1993), pp. 20–27.

tions developing software who haven't recognized this basic truth. We write a test plan while requirements are being developed. We then do test design and write test cases as the software design is being developed, and write test procedures during the coding phase.[2] Execution of formal testing occurs following unit testing and integration testing.

The second light bulb that lit pertained to the critical importance of test planning, and the consequent need for "testable requirements." Many good books have been written on these subjects (Gause and Weinberg, 1989; Wiegers, 1999), so I will not belabor the point here.

The third light bulb that was turned on was realizing that to properly estimate testing programs, we need a structured, internally consistent list of tasks that test engineers will perform on a specific program. Once such a list of tasks is in hand, we can then estimate each task individually to get a bottoms-up estimate for a testing program. And if we perform the same set of tasks (or a very similar set) on each program, we can begin to establish accurate metrics for how much our testing costs, and how long it can be expected to take.

In brief, what I am providing in this book is a soup-to-nuts list of tasks and processes for a program of formal software or system testing. If you are just starting to implement a testing process, you will not want to try to implement all parts of this model in one fell swoop; it would be too overwhelming and too costly. In later chapters, I suggest ways to implement this process in a prioritized, piecewise fashion.

Objectives

There is no "one true way" to test software, so the goal of this book is to provide you with information that you can use to develop a formal testing process that fits the needs of you, your customers, and your organization.

Glenford Myers woke up the testing community in 1979 with his book *The Art of Software Testing;* he stated that the purpose of testing is to find errors, rather than to prove that a system works. Regardless of what your opinion of his thesis is, we test to see if we can find any problems with developed systems. A best-practices test process is performed throughout the duration of the development process. The test process puts appropriate emphasis on requirements testability, documentation prior to test execution, review of the documentation, systematic interface with the development community, and test execution according to the approved documentation. A series of tasks to accomplish this testing process is presented in this book.

[2]See the Glossary at the end of this book for a definition of "test case" as there is much confusion in the industry regarding this term.

In addition to defining the sequence of tasks in the testing process, I address interfaces between the formal testing group and the other organizations involved in the software development life cycle, including development engineers, managers, software quality engineers, and software configuration management personnel. All of these people have significant roles that must be performed in a quality manner to make the software development process and the software testing process work. One of the primary purposes of test documentation is communication with the other groups on the program, so we as test engineers must be sure to make these intergroup interfaces work. All of us should be aware that there's a Level 3 key process area (KPA) in the Software Engineering Institute's Capability Maturity Model (SEI–CMM) for Intergroup Coordination. Early in the development life cycle, test engineers and managers review program-level plans, including the Program Management Plan (PMP), the Configuration Management Plan (CMP), the Software Quality Assurance Plan (SQAP), and the Software Development Plan (SDP), or whatever you call these documents in your environment. During the requirements phase of the software development life cycle, test engineers review the requirements. Later in the life cycle, the groups that authored these plans and the requirements get to return the favor by reviewing the test plan. Test engineers review software design (and sometimes code) for input to their test designs, test cases, and test procedures. Once again, development engineers, QA staff members, and CM personnel should review the test designs, test cases, and test procedures. As has been shown in other reference materials (Wiegers, 2002), peer reviews minimize defects and cost-of-quality by finding defects as early as possible in the life cycle.

Intended Audience

I have written this book for four specific audiences, who have much in common:

- new test engineers, who want to learn more about the framework of tasks performed as part of a good testing process
- newly assigned test managers and team leaders, who are not experienced testers (but may have come from a development, quality assurance, or systems engineering background) and who need to learn more about testing quickly

- more experienced test engineers, who are looking for ways to improve their testing process
- process improvement leaders, such as members of software engineering process groups and quality assurance staff

All of these folks realize that testing is necessary to verify the quality of the delivered product(s). They should also be aware that a coordinated program of peer reviews and testing can support a good software development process, but if the developers fall short of building a quality product, it's not possible to "test quality in."

Organizations that are pursuing Level 3 of the Capability Maturity Model for Software (for additional detail on the Capability Maturity Model from the Software Engineering Institute at Carnegie Mellon University, see Appendix A and Website www.sei.cmu.edu) will also find this book valuable, since testing is a significant component of the Software Product Engineering key process area for CMM Level 3. The newer Capability Maturity Model Integration (CMMI), with components of Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing, has process areas for Verification and for Validation, both of which involve aspects of testing. This book will assist them in documenting their testing process, which will support them in the Level 3 key process area Organization Process Definition.

There's actually a fifth audience as well, those software development team leads, supervisors, and managers who are interested in learning more about what test engineers and the testing group are doing, and why.

Prerequisites

What should you know to read this book? You should be aware that testing is not a "phase," where software developers "throw the software over the wall" to test engineers when the developers have finished coding. To be cost effective, testing needs a life cycle that runs concurrently with the software development life cycle. The testing life cycle begins during the requirements elucidation phase of the software development life cycle, and concludes when the product is deemed ready to install or ship following a successful system test (or whatever you call the final test in your culture).

Ideally, you should be aware of the importance of test documentation, especially a formal test plan, in your life cycle. You should be aware that there are good standards for this documentation in the software industry; I strongly recommend the IEEE Software Engineering Standards collection in general, and IEEE-Standard-829, the Standard for Software Test Documentation, in particular.

Finally, you should have the desire to improve your knowledge of the testing process, and a commitment to make improvements.

How to Read This Book

This book is laid out in the following order:

1. **Chapter 1** provides a general overview of the software development life cycle, and an overview of the concurrent testing life cycle. It also shows the need for a testing process model, and illustrates the Level 0 and Level 1 IPO diagrams for this model.
2. **Chapters 2 through 7** show decomposition of the Level 1 IPO diagram into a series of Level 2 and Level 3 diagrams, to show the tasks involved in the review of program plans, the creation of a test plan, the creation of test design (and other test documentation), the performance of formal test, and finally, the updating of test documentation. This set of tasks can be used as the basis for an estimate of a formal testing program on a particular development program.
3. **Chapter 8** provides some thoughts on how to use and customize this process model, which is a soup-to-nuts model. You may not be in a position to use such a model initially, but on a large program, putting this process in place should be your goal. Customizing the model should be your goal on a medium- or small-size project. Thus, this model provides a template for a testing process improvement program. To end the text itself, **Chapter 9** provides a brief summary of the model in its entirety.
4. To conclude the book, I've added four practical Appendices, a Glossary, a Bibliography, and, of course, an Index. Although most of these final sections are standard fare, the Appendices bear describing. **Appendix A** provides a brief description of the Capability Maturity Model for Software (CMM-SW) from the Software Engineering Institute at Carnegie Mellon University, as well

as a brief description of the SEI's more recent Capability Maturity Model Integration (CMMI). **Appendix B** provides templates for five preferred practices that are significant to a software development program, including a Program Management Plan (PMP), a Software Development Plan (SDP), a Software Quality Assurance Plan (SQAP), a Configuration Management Plan (CMP), and a Test Plan (TP). **Appendix C** provides a questionnaire you can use to evaluate the state of your testing process. Let this guide your use of the Formal Testing Process Model as a mechanism to help you improve your testing process. **Appendix D** contains guidelines for test execution.

To gain a detailed understanding of the tasks, inputs, and outputs involved in formal testing, you can simply read the book from front to back. For readers desiring to explore specific aspects of the testing process (for example, test execution), I would recommend skimming Chapter 1 (Introduction), reviewing Chapter 2 (The Formal Testing Process Model: Level 1 IPO Diagrams) to identify which process you want to explore in more detail, and then proceeding directly to that chapter (for one example, test execution, see Chapter 6, "Perform Formal Test," and Appendix D). Though there are sections for each subprocess of the major processes (for example, subprocess Execute Test, major process Perform Formal Test), I would suggest there are enough interfaces between the various subprocesses that the entire section should at least be skimmed.

I don't spend much time discussing the topic of testing tools and test automation, because these are really software development programs, not test processes. However, there are two subprocesses defined for Acquire Test Software in Chapter 5, which should be reviewed. One subprocess deals with building your own test software tools; the second subprocess addresses the activities involved in buying the tools you need.

I hope that this book will start you thinking about the following questions:

- Does our current testing process have all the tasks we need to identify defects in the products under test? If not, which tasks should we add first to improve our testing process?
- Have we identified the next step in our continuous process improvement plan?

- Do we have the necessary reviews in our testing process (for example, Test Plan inspection)?
- Are we writing all the test documentation we need to provide high-quality repeatable tests?
- Is testing and test coverage adequate for our program (for example, enough time, enough test points)?

If you can answer yes to all these questions, then you probably don't need to read this book in its entirety. But are you *sure* that your test process is as good as it can be?

There is one additional caveat to keep in mind as you read this book: You will not find the term "bugs" used. I once heard a testing guru state at a conference that he didn't like the term "bugs," since the term implies that bugs have a life of their own. Those of you with a memory for computer trivia will remember that the term originated when workers on an early Navy computer found that a large moth had become trapped between the contacts of a relay, causing the computer to malfunction.[3] Well, folks, in software, defects don't jump in by themselves. Human beings, as software engineers, put defects in code. Now, the defects may not all be our fault, since we may just be doing what we were told to do when the customer gave us some incorrect requirements, but nonetheless, people put defects in; defects don't just appear spontaneously. So, in this book, you'll see the terms "defects," "incidents," and "problems," but you won't see the term "bugs."

Scope of Coverage

This book will present a description of a process and a set of tasks that can be used to implement or improve a formal testing program. Formal testing means test activities performed by a group of trained professional test engineers whose chain of command is independent of development management. Examples of formal testing are system testing and acceptance testing. By contrast, informal testing consists of test activities performed by the development group. Examples of informal testing are unit testing and component testing. Integration testing may also be informal testing, if members of the software development group perform that testing.

In addition, this book will identify the testing tasks, the inputs and outputs associated with each task, presenting them by means of classical Input-Process-Output diagrams. Tasks that span the testing life

[3]As a Lieutenant in the United States Navy, the late Retired Rear Admiral Grace Hopper and her team discovered a moth trapped between the contacts of a relay, preventing the relay from passing current. As a result of this episode, Hopper is credited with coining the terms "bug" and "debug," to describe computer errors and how to fix them. **Source:** People and Discoveries, www.pds.org/wgbh/aso/databank/entries/btmurr.html.

cycle, from test planning through test execution and beyond, will also be addressed.

Features

This book addresses testing activities and tasks that span the entire testing life cycle—from review of program plans, to developing the formal test plan, through developing the remainder of the test documentation (test design, test cases, test procedures) as well as acquiring any needed automated testing tools, through test execution, to final updating of the test documentation after test execution is complete.

The test activities and tasks will be represented using the technique of classical Input-Process-Output (IPO) diagrams, as suggested by Watts Humphrey (Humphrey, 1989). The test activities, tasks, and processes are in accordance with those documented in IEEE-Standard-829, the Standard for Software Test Documentation, and are techniques suggested by a number of testing specialists, including personnel at the Quality Assurance Institute (Orlando, Florida) and Software Quality Engineering (Jacksonville, Florida). These are techniques I have used for many years on a variety of hardware and software programs, of large and small size. Hopefully, use of this method and adoption of some or all of the processes will help you as much as they have helped me.

Readers should keep one additional point in mind: The full-blown model described in this book details a full-featured formal testing process that is applicable to large programs and that would fully support programs deliverable to state and federal governments, or on programs delivering safety-critical systems or having significant impact on corporate profits. For work on smaller systems, or in organizations where a formal testing process is just getting organized, this process model must be significantly tailored and reduced in scope. Guidance in tailoring this process model and using it on projects and programs of various sizes is provided in Chapter 8. As with any process improvement opportunity, don't try to do everything at once. Prioritize your process improvements and proceed slowly and methodically. Plan your work on small projects so as to have lots of little victories. In today's sound-bite management environment, it is very important to show lots of accomplishments. It is essential to make sure you can publicize your and your group's accomplishments if you are going to survive in this new world of short-term planning and thinking.

Benefits to the Reader

Why should you, the reader, care about a book that provides a detailed definition of a software testing process? A short answer is, because you need to learn more about how to do testing; this means knowing what the testing process should be. Another answer is, you believe the testing process you are now using could be improved and you need some guidance.

This book, and the model detailed in it, is designed to fill both needs.

If you are a new test engineer or test engineering manager who has not spent ten-to-twenty years in the testing arena, such a model can serve as a confirmation of the "good things you are doing" and a useful guide to "things you should be doing." In *Critical Testing Processes*, testing expert Rex Black goes into a beautifully detailed set of steps for developing a testing estimate (Black, 2003). Use of a process model such as the one I define in this book can identify a list of tasks that form the foundation for such an estimate.

At the risk of repeating myself, I'd again like to note: My goal is to address testing activities and tasks that span the entire testing life cycle, from reviewing program plans, to developing the formal test plan, through developing the remainder of the test documentation (test design, test cases, test procedures) as well as acquiring any needed automated testing tools, through test execution, to final updating of the test documentation after test execution is complete. To do all this, you can use the technique of IPO diagrams shown in the following chapters to model your current testing process. This approach will provide you with a model of a software system that is an analog to the current logical model defined in the many seminars and books on the structured methods put out during the 1970's and 1980's by staff and consultants at Yourdon, Inc., and published by Yourdon Press.[4] You could then take your current logical model and compare it with the model defined in this book, and decide whether any elements are missing from your process. Finally, you could start a process improvement activity to implement some of those missing elements, so that your process will be improved. An improved testing process should result in your identifying more defects prior to delivering your products, either by finding those defects prior to coding (through reviews associated with developing test documentation), or by finding defects during test execution because your improved procedures result in better testing.

[4]Although I could cite many seminars and books, the seminal works on structured methods seem to me to be ones published by Yourdon Press (Yourdon and Constantine, 1975; DeMarco, 1978; Page-Jones, 1980; McMenamin and Palmer, 1984).

Before you jump into the book, let me offer another advisory note: Read the Glossary. Currently, there is ambiguity in the testing industry regarding some of the terms we use on a daily basis, such as "test plan," "test cases," and "test procedures." This book adheres to the terminology from IEEE-Standard-610.12, the Standard Glossary of Software Engineering Terminology. Many companies use the term "test plan" to encompass all test documentation. In such an instance, the IEEE-type test plan is the first chapter, and test cases and test procedures are contained in appendices. That isn't what is meant by "test plan" in this book. Similarly for the term "test cases"—in some companies, test cases are considered to be the step-by-step instructions for executing tests. This book uses the term "test procedures" to cover those step-by-step instructions; in IEEE terminology, "test cases" contain the input data and expected results when input data are input to the system under test. So, keep this book's Glossary handy to avoid confusion.

Now that you have finished this Preface, turn to Chapter 1, "Introduction," and begin to read the book itself. Hopefully, you'll enjoy reading it as much as I have enjoyed developing the model and writing the book.

August 2003 R.D.D.
Columbia, Maryland

Best Practices for the Formal Software Testing Process

This page intentionally left blank

Chapter 3
Extract Test Information from Program Plans: Levels 2 and 3 IPO Diagrams

3.1: Overview

In Chapter 2, we reviewed the five Level 1 IPO diagrams for the formal testing process, as listed below:

1. Extract Test Information From Program Plans, process 1.1
2. Create Test Plan, process 1.2
3. Create Test Design, Test Cases, Test Software, And Test Procedures, process 1.3
4. Perform Formal Test, process 1.4
5. Update Test Documentation, process 1.5

In this chapter, I expand the detail of the Level 1 IPO diagram presented in Chapter 2 for process 1.1, Extract Test Information From Program Plans. The goal of this process is to extract any requirements for testing from the higher-level plans, so that the scope of the test effort can be identified.

There can be a significant variation in the specific program plans associated with different organizations and the various projects worked on by those organizations. Requirements for program plans may vary by customer, with some programs requiring formal Program Management Plans, Configuration Management Plans, Quality Assurance Plans, and others. Other programs may require various formal and informal forms of documentation plans, build plans, marketing plans, and sales plans. Sample plans are listed below, and are typical of plans required on a large program in a highly regulated environment. You can customize your process model to show the specific plans your program requires.

In the Level 2 IPO diagram (see Figure 3-2), four subprocesses are included within one possible instance of Extract Test Information From Program Plans, process 1.1:

1. Review Program Management Plan
2. Review Quality Assurance Plan
3. Review Software Development Plan
4. Review Configuration Management Plan

Following analysis of these four subprocesses, we'll look at the Level 3 IPO diagrams for each of the four.

Preferred practices, including a template and table of contents (TOC) for each of these documents, are shown in Appendix B.

In this chapter, we change the format somewhat from the IPO diagrams that were presented in Chapter 2, in which an individual IPO diagram was used to illustrate each process.

In this and subsequent chapters, "combined" IPO diagrams provide supporting detail to the Level 1 IPO diagrams that appeared in Chapter 2. These combined IPO diagrams are similar in format to Figure 1-7, which showed a group of IPO diagrams, connected in series, that provided additional detail to the Level 0 IPO diagram of Figure 1-6.

3.2: Extract Test Information from Program Plans—Level 2 IPO Diagram

The IPO diagram for process 1.1, Extract Test Information From Program Plans, is shown in Figure 3-1. The accompanying Level 2 IPO diagram is shown in Figure 3-2. As named in the preceding section, there are four distinct subprocesses associated with these IPO diagrams, which in this listing have been shortened to first-initial abbreviations.

1. Review PMP
2. Review QAP
3. Review SDP
4. Review CMP

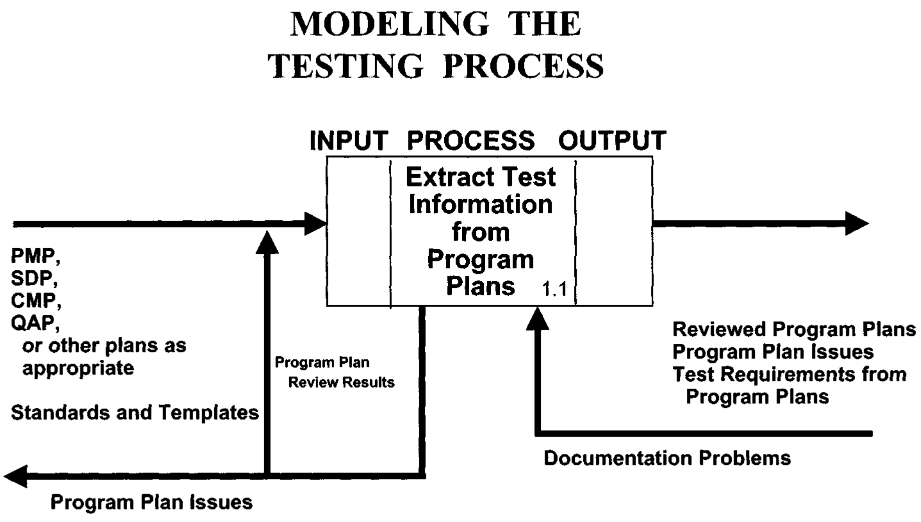

Figure 3-1: *The goal of process 1.1, Extract Test Information From Program Plans, is to reveal what sorts of requirements are mandated by higher-level plans.*

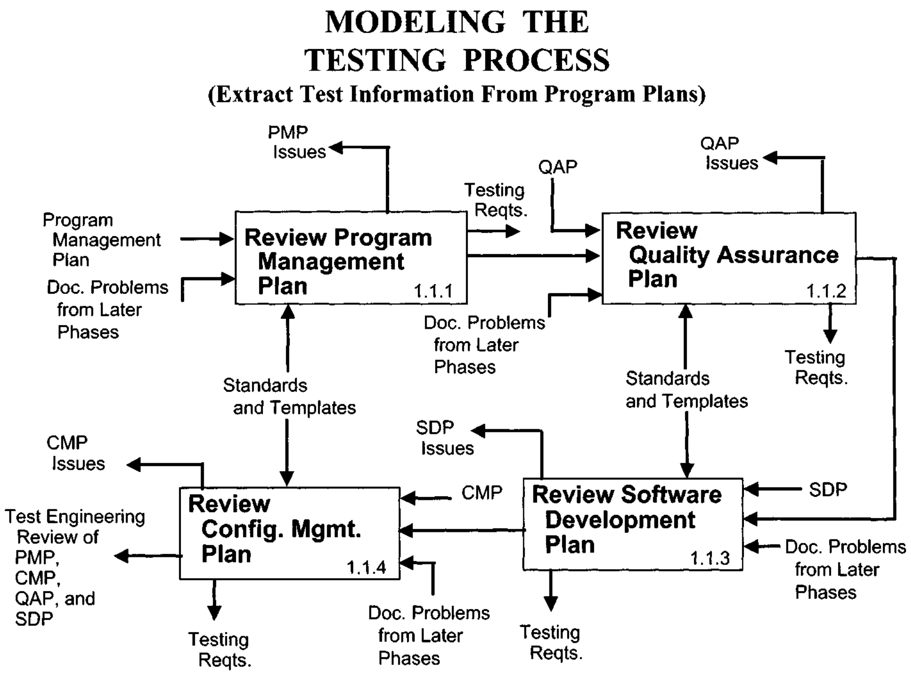
MODELING THE
TESTING PROCESS
(Extract Test Information From Program Plans)
Program Management Plan
Doc. Problems from Later Phases
PMP Issues
Review Program Management Plan 1.1.1
Testing Reqts.
QAP
QAP Issues
Review Quality Assurance Plan 1.1.2
Doc. Problems from Later Phases
Standards and Templates
Standards and Templates
Testing Reqts.
CMP Issues
Test Engineering Review of PMP, CMP, QAP, and SDP
Review Config. Mgmt. Plan 1.1.4
CMP
SDP Issues
Review Software Development Plan 1.1.3
SDP
Doc. Problems from Later Phases
Doc. Problems from Later Phases
Testing Reqts.
Testing Reqts.

Figure 3-2: *Process 1.1 is expanded here to become a Level 2 IPO diagram.*

3.2.1: Inputs

The four program plans—Program Management Plan (PMP), Quality Assurance Plan (QAP), Software Development Plan (SDP), and Configuration Management Plan (CMP)—as well as the standards and templates that provide the formats for these documents, are the principal inputs to this IPO diagram.

If your program uses a different set of program plans, just insert the names of your program plans in the appropriate process blocks, and implement the tasks shown in the Level 3 IPO diagrams for those plans.

3.2.2: Feedback

Results of the review of the four program plans are fed back into this process. When updates to the plans are made and the plans are submitted for a subsequent review, these review results are factored into that review.

3.2.3: Feedback to Earlier Process Steps

Issues identified in each of the four plans are fed back to their respective authors.

3.2.4: Process Flow, Level 2 IPO Diagram

This testing process is generally concerned with reviewing other program- or project-level documents, to see what requirements they contain that will impact or influence the formal testing staff and the system test plan. Usually, most of these documents are developed during the planning portion of the life cycle (see Figure 1-1) that precedes the requirements phase. As far as the testing process is concerned, the order in which the documents cited are reviewed is immaterial. However, since the Program Management Plan (see IEEE-Standard-1058 for suggested content and format) is the document to which all other program plans must be linked and synchronized, the PMP should be reviewed first.[1]

Note that the review of program plans to extract test information from those program plans links to CMMI process area Integrated Project Management, and to the practice Manage Dependencies within the specific goal Coordinate with Relevant Stakeholders.

---

[1]For additional information, see CMMI process area Project Planning, discussed in Ahern et al., 2001, or the CMMI section of the SEI's Website, www.sei.cmu.edu.

Quality assurance personnel or software quality assurance staff members are responsible for performing functions and tasks that contribute to the quality of the processes and products on the project.[2] (See IEEE-Standard-730 for suggested content and format, and Drabick, 2000, for more details regarding software quality assurance functions.) These functions and tasks are normally documented in a formal Quality Assurance Plan. Since QA has many interfaces with test engineering (TE), the QA functions can impact the test plan.

The Software Development Plan is normally only written on large programs or projects, and identifies the tasks that software engineers are assigned to perform. Since there are many interfaces between the software engineering and test engineering divisions in a company, this plan, if developed, is of critical importance to development of the system test plan. This plan should identify the informal and semiformal testing tasks that software engineering performs (for example, unit testing, component structural testing, and integration testing).

Configuration management is normally responsible for maintaining the versions of documentation and software, and is often responsible for software builds. These functions and tasks are normally documented in a formal Configuration Management Plan (see IEEE-Standard-828 for suggested content and format). Since configuration management maintains the integrity of the code that test engineering is testing, the interfaces between CM and the test team are very important, and the CM functions can impact the test plan.

Most organizations will have available a set of standards and templates to use in development of documentation and computer software. IEEE and a variety of military standards provide such templates. A CDRL can also include data item descriptors that specify the organization of deliverable documents. As in other instances, use of such standards and templates can prevent an organization from having to reinvent the wheel when creating documentation. While test engineering staff members review the program plans, they and QA personnel should be sure that the plans are in conformance with the applicable standards and templates used on the program.

3.2.5: Outputs

The reviewed program plans, with included comments, are one major output of this subprocess. The other major output of this subprocess is the testing requirements extracted from each of the program plans.

[2]Remember that I use the abbreviations "QA" and "SQA," as well as the terms they stand for, synonymously in this book, although there can be significant differences between the two on a program or project that involves both hardware and software products.

One example of a testing requirement is the schedule for formal testing contained in the PMP. Another example is the description of the defect- or incident-handling procedure normally documented in the QAP.

The comments in the reviewed program plans will identify any issues that test engineering staff members have with those documents. The issues may be simple inconsistencies between the various documents (for example, the program plan specifies one set of test levels, and the QAP specifies a different set), or they may indicate philosophical problems between the various groups. Again, early resolution of these problems will simplify matters later in the program.

3.2.6: Feedback from Later Process Steps

It is quite possible that not all the inconsistencies or problems with the various program plans will be identified in the review. As other problems are identified during later phases in the program life cycle (most commonly during the detailed design phase), it may be necessary to update one or more of the program plans to resolve those problems. As changes are made to the program plans to incorporate those fixes, some additional review of the plans by test engineering staff (and also by quality assurance personnel) will be required.

3.3: Review Program Management Plan—Level 3 IPO Diagram

The Level 3 IPO diagram for subprocess 1.1.1, Review Program Management Plan, is shown in Figure 3-3. Significant inputs for this subprocess are straightforward:

1. Program Management Plan
2. standards and templates

The outputs from this process include four significant products:

1. Program Management Plan following review by the personnel in the test engineering (TE) area
2. review comments (from TE)
3. PMP issues (including format and testing issues)
4. testing requirements (including testing life cycle requirements)

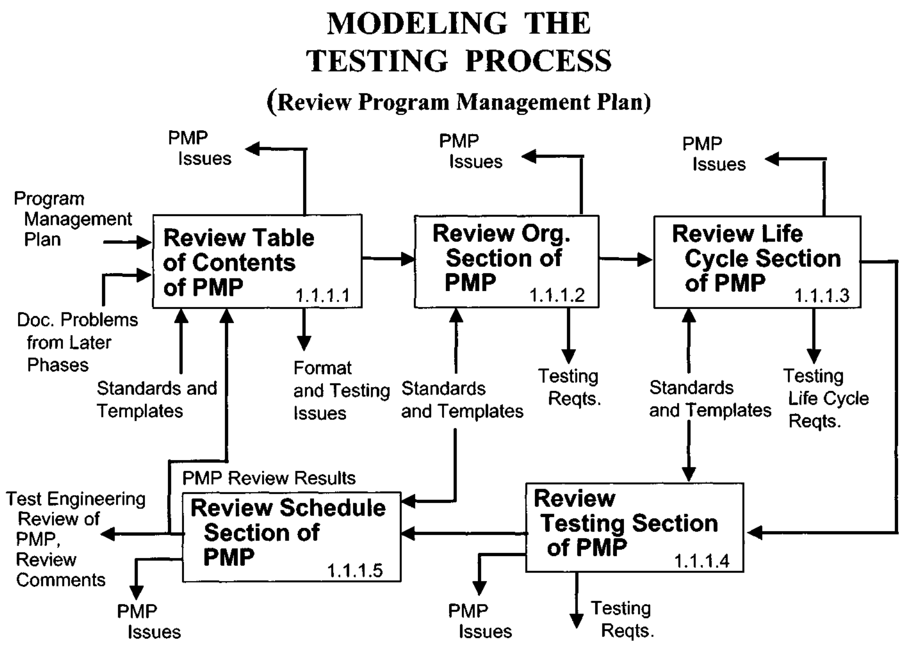
MODELING THE TESTING PROCESS
(Review Program Management Plan)
Program Management Plan
Doc. Problems from Later Phases
Review Table of Contents of PMP 1.1.1.1
PMP Issues
Standards and Templates
Format and Testing Issues
Review Org. Section of PMP 1.1.1.2
Standards and Templates
Testing Reqts.
Review Life Cycle Section of PMP 1.1.1.3
Standards and Templates
Testing Life Cycle Reqts.
PMP Review Results
Test Engineering Review of PMP, Review Comments
Review Schedule Section of PMP 1.1.1.5
Review Testing Section of PMP 1.1.1.4
Testing Reqts.

Figure 3-3: *The Level 3 IPO diagram shows greater detail with expanded PMP inputs and outputs.*

Five distinct subprocesses are identified in this Level 3 IPO diagram:

1. Review Table Of Contents Of PMP
2. Review Organization Section Of PMP
3. Review Life Cycle Section Of PMP
4. Review Testing Section Of PMP
5. Review Schedule Section Of PMP

3.3.1: Inputs

The primary input for this subprocess is the Program Management Plan, which is ready for review.

3.3.2: Internal Feedback

Since review of any document is an iterative process, review comments may flow to the authors on multiple occasions, depending on how many times the Program Management Plan is revised before approval takes place. Thus, updated versions of the PMP (under configuration management version control), based on comments from the review, may reenter this process asynchronously, triggering a repeat review.

3.3.3: Feedback to Earlier Process Steps

When issues are found in the PMP, those issues should be documented and presented to the program manager, so that they can be resolved in a timely manner. Timely resolution of these issues will assist test engineers as they put together a written, high-quality test plan, as well as the other program-level plans (for example, the Configuration Management Plan, the Quality Assurance Plan, and the Software Development Plan).

3.3.4: Process Flow, Level 3 IPO Diagram

The most important sections of a Program Management Plan that should be thoroughly read by test engineering staff members are the table of contents, the organization section, the life cycle section, the testing section, and the schedule section. Though there are other sections of the document that will be of interest, the cited sections deserve to be scrutinized.

Test engineers and quality assurance experts should review the table of contents and compare it with the standards and templates that apply to a Program Management Plan. Why is the table of contents so important? One reason is that comparing it with the PMP standards and templates will immediately alert personnel if any important sections have been omitted. For example, if you record documentation problems in your defect-tracking system (something I highly recommend) and the table of contents shows no comparable steps, you know you will have a tracking problem. Such omissions should be documented as a potential defect, and entered into the defect-tracking system to alert program management to the potential problem area, and then a report of the omissions should be sent to the report's author as soon as possible, so that the issues can be resolved.

Next, test engineering staff should review the organization section of the PMP. If IEEE-Standard-1058 is followed for the PMP, the organization section should identify each organization on the program, their tasks, and the interfaces between the organizations. Although this list may not be fully complete, test engineering staff should review the interfaces with each of the other groups on the program since these interfaces will influence the testing tasks, configuration control, responsibilities, and staffing sections of the test plan.

The section of the PMP that defines the development life cycle also should be carefully reviewed because the program life cycle will impact the testing life cycle. As an example, if the program schedule is based on an iterative life cycle and the testing staff has assumed it is based on a one-shot waterfall life cycle, the test plan will not synchronize with the PMP, leading to innumerable problems as the program progresses.

Test engineering staff members probably wrote the testing section of the PMP, and therefore will not need to read it again. If they did not, they need to be sure that they really are planning to do the tasks that this section of the PMP has allocated to them. If a manager who is not familiar with testing techniques, processes, and life cycles wrote this section, there will be synchronization problems as the program proceeds.

Test engineering staff members should review the schedule section of the PMP to ensure that there are no surprises in the schedule. Often, this section of a PMP references a Microsoft Project™ schedule; if that is the case, that schedule should be reviewed. The TE staff should ensure that all the proper work breakdown structure tasks appropriate to the program are identified on the schedule, and are in the proper order. Although it is not likely that anyone would expect formal testing to begin prior to the start of unit testing or integration testing, you never can take this for granted. The testing staff also must ensure that the proper reviews, inspections, and walkthroughs identified in the schedule, including review of test documentation by other staff members, have been completed.

A PMP written according to the format of IEEE-Standard-1058 will have additional sections that will be of some interest to test engineering, including the Risk Management and Change Reporting sections. However, the five subprocesses discussed here are the most important ones to be considered by personnel in test engineering, and should draw the most attention.

3.3.5: Outputs

Outputs from these subprocesses include six products:

1. PMP issues
2. format and testing issues (in context of the PMP)
3. testing life cycle requirements (for use in the test plan)

4. testing requirements (from both the testing and schedule sections)
5. Program Management Plan (reviewed by test engineering staff members)
6. test engineering's review of the PMP

3.3.6: *Feedback from Later Process Steps*

The subject of feedback has been discussed in Section 3.2. Note, however, that it is quite possible that not all the inconsistencies or problems with the Program Management Plan will be identified in the review. As other problems are identified during later phases in the program life cycle (for example, detailed design), it may be necessary to update the PMP to resolve those problems and hold another review.

3.4: Review Quality Assurance Plan—Level 3 IPO Diagram

The Level 3 IPO diagram for subprocess 1.1.2, Review Quality Assurance Plan, is shown in Figure 3-4. Significant inputs are again straightforward:

1. Quality Assurance Plan
2. standards and templates

Outputs from this process follow:

1. Quality Assurance Plan (reviewed by test engineering)
2. review comments (from test engineering)
3. QAP issues (including format and testing issues)
4. testing requirements

Five distinct subprocesses are identified in this Level 3 IPO diagram:

1. Review Table Of Contents Of QAP
2. Review Standards Section Of QAP (including practices and conventions)
3. Review "Reviews" Section Of QAP
4. Review CM Section Of QAP
5. Review Incident-Reporting Section Of QAP

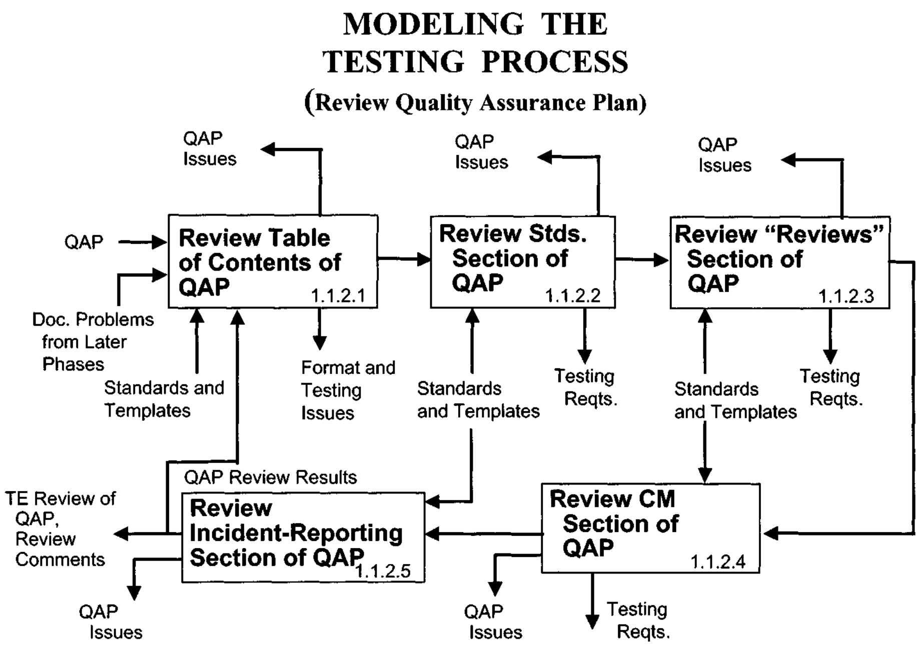
MODELING THE TESTING PROCESS
(Review Quality Assurance Plan)
QAP
Doc. Problems from Later Phases
Review Table of Contents of QAP 1.1.2.1
QAP Issues
Standards and Templates
Format and Testing Issues
Review Stds. Section of QAP 1.1.2.2
QAP Issues
Standards and Templates
Testing Reqts.
Review "Reviews" Section of QAP 1.1.2.3
QAP Issues
Standards and Templates
Testing Reqts.
QAP Review Results
TE Review of QAP, Review Comments
Review Incident-Reporting Section of QAP 1.1.2.5
QAP Issues
Review CM Section of QAP 1.1.2.4
QAP Issues
Testing Reqts.

Figure 3-4: *A Level 3 IPO diagram for the Review Quality Assurance Plan subprocess shows added detail.*

3.4.1: Inputs

The primary input for this subprocess is the Quality Assurance Plan, which is ready for review.

Test engineering needs to be aware of the standards used on a program, including standards for documentation—and especially for testing documentation. This information on standards used in an organization should be documented in the QAP. The testing staff should review this section of the QAP, to be sure that quality assurance staff members are using the same standards for testing documentation as are being used by test engineering personnel, and also to be aware of what standards, templates, practices, and conventions will be used for requirements, design, coding, and testing documentation.

Test engineering staff members should participate in reviews of developer documentation (requirements, design, code, unit test plans, and so on), to be aware of what reviews and audits QA is planning. This section of the QAP will identify the reviews and audits that department staff members will conduct in the testing areas. Individuals on the testing staff need to be aware of these reviews so that they can prepare for them and participate in them.

QA requirements regarding configuration management activities are also important to test engineering, since CM will most probably prepare builds for test. Test engineering staff should thoroughly review this section of the QAP to make sure that QA activities in this area are synchronized with test engineering activities.

Finally, test engineering personnel need to be familiar with the policies, process, and tools used for documenting and tracking incidents and defects recorded on the program, since test engineering staff members will have to conform to these policies and processes when documenting and tracking incidents found during formal test. If no Quality Assurance Plan is written (possibly because the project is too small to need a formal plan), then information regarding incident-tracking must be documented in the test plan.

3.4.2: Internal Feedback

Since review of any document is an iterative process, review comments may flow to the authors on multiple occasions, depending on how many times the QAP is revised before approval takes place. Thus, updated versions (under configuration management version control, of course) of the Quality Assurance Plan, based on comments from the review, may reenter this process asynchronously, setting off a repeat review.

3.4.3: Feedback to Earlier Process Steps

When issues are found in the Quality Assurance Plan, those issues should be documented and presented to the QA manager, so that the issues can be resolved. Timely resolution of these issues will help assure a high-quality test plan, as well as the quality of the other program-level plans.

3.4.4: Process Flow, Level 3 IPO Diagram

Principal sections of the Quality Assurance Plan that should be reviewed carefully by test engineering personnel are the table of contents; the standards, practices, and conventions section; the reviews and audits section; the configuration management section; and the incident-reporting section. Other sections of this document may be of interest to TE staff, of course, but the cited sections should be reviewed most thoroughly.

Test engineering staff members first should quickly review the contents listings in order to compare contents with the standards and templates that apply to a QAP. This will immediately alert TE staff members whether any sections deemed significant by them have been omitted. If you record documentation problems in your defect-tracking system, as I recommend, such omissions should be entered into the defect-tracking system to alert management to the potential problem area. If this is not an option, at the least, send a report detailing omissions to management as soon as possible, so that the issues can be resolved.

The next section test engineering staff members review is the standards, practices, and conventions section of the QAP. This section of the plan should identify material to be used by all functions on the program. As well as specifying standards for documentation produced by test engineering, this section will also identify policies and procedures for other program activities, which can give test engineers valuable insight into tasks other groups are performing that may impact the way test engineers perform their job.

Reviews and audits are defined in another section of the QAP. Because members of the testing staff should attend some of these reviews (for instance, requirements inspection), and because some of the reviews (for example, inspection of the test plan) will be targeted at deliverables produced by test engineering, it is important for TE personnel to pay close attention. The reviews and audits specified in the QAP can provide requirements for tasks in the test plan and in the testing schedule.

The configuration management section of the QAP will provide information about CM activities that can affect test engineering. These activities will be documented in more detail in the Configuration Management Plan, which will also be reviewed by test engineering, but the QAP contains detail test engineers need to know.

Test engineering staff also should review the incident-reporting section of the QAP to understand the policies, processes, and tools involved in tracking defects and incidents on the program. Test engineering staff will use this system during formal test execution, so it is essential that testing staff be adept in its use, and that staff members are knowledgeable about the policies and processes involved.

A Quality Assurance Plan written according to the format proposed by IEEE-Standard-730 will have additional sections that will be of interest to test engineers (see especially the Tools, Techniques, and

Methodologies section and the Records Retention section). However, the five sections discussed here are the most valuable for test engineering.

3.4.5: Outputs

Five outputs are produced:

1. QAP issues
2. format and testing issues
3. testing requirements (for use in the test plan)
4. the reviewed (and updated) QAP
5. QAP review comments from test engineering

3.4.6: Feedback from Later Process Steps

Not all the inconsistencies in or problems with the Quality Assurance Plan will be identified in the review. As other problems are identified during later phases in the program life cycle, it may be necessary to update the QAP to resolve those problems.

3.5: Review Software Development Plan—Level 3 IPO Diagram

On a large program, the software engineering staff may consist of so many members and have so much work to do that it will need its own written, formal Software Development Plan to support the Program Management Plan. If a Software Development Plan is written, test engineers should review this plan. A Software Development Plan normally will have the following sections:

1. introduction
2. software life cycle
3. software quality factors
4. software components
5. development schedule
6. software engineering specification
7. software quality assurance and testing
8. software configuration management

The Level 3 IPO diagram for subprocess 1.1.3, Review Software Development Plan, is shown in Figure 3-5, and has two inputs:

1. Software Development Plan
2. standards and templates

Four outputs result from this process:

1. Software Development Plan after review by test engineering
2. review comments from test engineering
3. Software Development Plan issues (including format and testing issues)
4. testing requirements

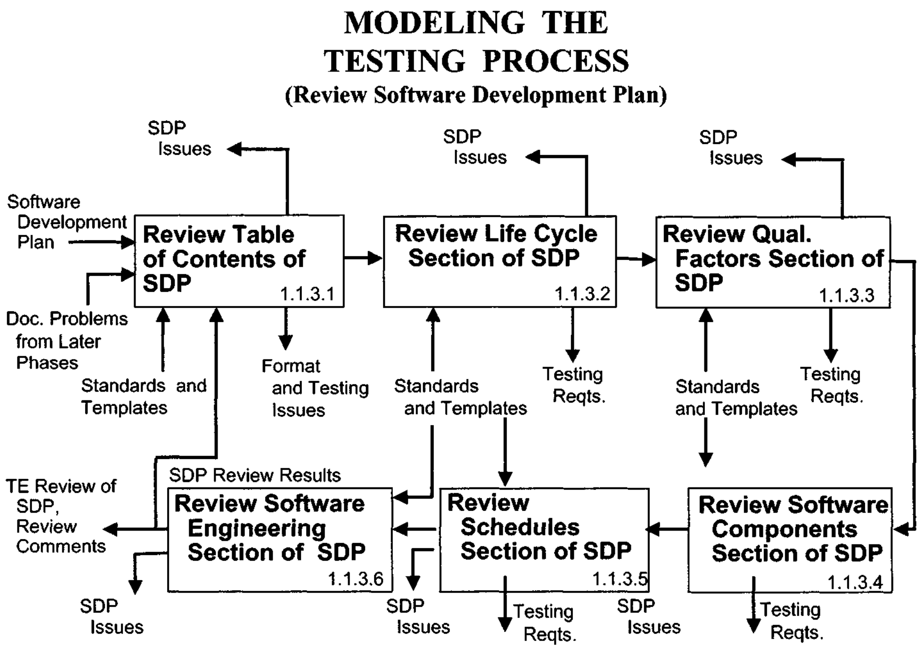
MODELING THE TESTING PROCESS
(Review Software Development Plan)
SDP Issues
SDP Issues
SDP Issues
Software Development Plan
Review Table of Contents of SDP 1.1.3.1
Review Life Cycle Section of SDP 1.1.3.2
Review Qual. Factors Section of SDP 1.1.3.3
Doc. Problems from Later Phases
Standards and Templates
Format and Testing Issues
Standards and Templates
Testing Reqts.
Standards and Templates
Testing Reqts.
TE Review of SDP, Review Comments
SDP Review Results
Review Software Engineering Section of SDP 1.1.3.6
Review Schedules Section of SDP 1.1.3.5
Review Software Components Section of SDP 1.1.3.4
SDP Issues
SDP Issues
Testing Reqts.
SDP Issues
Testing Reqts.

Figure 3-5: *The Level 3 IPO diagram contains six SDP subprocesses to be completed by test engineering staff members.*

Six distinct subprocesses are identified in this Level 3 IPO diagram:

1. Review Table Of Contents Of SDP
2. Review Life Cycle Section Of SDP

3. Review Quality Factors Section Of SDP
4. Review Software Components Section Of SDP
5. Review Schedules Section Of SDP
6. Review Software Engineering Section Of SDP

3.5.1: Inputs

The primary input for this subprocess is the current version of the Software Development Plan, which is ready for review.

Test engineering staff members need to be aware of the standards used to develop the Software Development Plan, so that they can compare the plan to the appropriate standards or templates.

3.5.2: Internal Feedback

Since review of any document is an iterative process, review comments may flow to the authors on multiple occasions, depending on how many times the Software Development Plan is revised before final approval takes place. Thus, updated versions of the Software Development Plan (under configuration management version control, of course), based on comments from the review, may reenter this process asynchronously, setting off a repeat review.

3.5.3: Feedback to Earlier Process Steps

When issues are found in the Software Development Plan, those issues should be documented and presented to the software development manager, so that they can be resolved in a timely manner. Resolution of these issues will assist test engineers to prepare a high-quality test plan, as well as the other program-level plans (for example, the Configuration Management Plan, the Program Management Plan, or the Quality Assurance Plan).

3.5.4: Process Flow, Level 3 IPO Diagram

The sections of a Software Development Plan that should be reviewed by test engineers are the table of contents, the life cycle section, the quality factors section, the software components section, the schedule section, and the software engineering section.

Test engineers should review the table of contents and compare it with the standards and templates that apply to a Software Development Plan. Such comparison will alert test engineering personnel to any omitted sections. Omissions should be documented as a potential defect, and entered into a defect-tracking system, if available, to alert management to the potential problem area. Otherwise, a report of the omissions should be sent to department managers as soon as possible, so that the issues can be resolved.

Next, test engineering staff members review the life cycle section of the Software Development Plan, comparing it against the life cycle section of the Program Management Plan. The two life cycles should be identical. As with the PMP, the program life cycle defined in the Software Development Plan will impact the testing life cycle. For example, if the software development schedule is based on a spiral life cycle and the testing staff has assumed that a one-shot waterfall life cycle is being used, the test plan will not synchronize with the software development life cycle (which in turn may be out of sync with the rapid application development life cycle identified in the PMP), leading to no end of problems as the program progresses.

The Software Development Plan should identify quality factors that are considered significant during development. Some examples of quality factors include maintainability and reliability. Knowing which quality factors are deemed important on the program will provide insight to test engineers, which may help them to develop a testing strategy to verify that those quality factors are present in the deliverable software products.

The software components section of the Software Development Plan defines components of the software deliverables that are being developed. This section should be reviewed by the test engineering staff, since the software components are highly likely to become test items in the test plan, test design, and test procedures. This sort of domain knowledge is extremely important to test engineering staff members as they lay out their test strategy. Later, during test execution, this information may assist the test engineering staff in documenting incidents when software problems are detected.

Test engineering staff members should review the schedule section of the Software Development Plan to ensure that there are no surprises in the schedule. Test engineering and quality assurance personnel also should compare the schedule section in the document with that contained in the PMP, and determine whether the two schedules synchronize. Often, this section of a Software Development Plan references a Microsoft Project™ schedule; if that is the case, that schedule should be reviewed. The testing staff should ensure that all the proper work breakdown structure tasks appropriate to software engineering are identified on the schedule, and are in the proper order—for example, formal testing should not start prior to the start of unit testing or integration testing. Testing staff also must confirm that the proper reviews, inspections, and walkthroughs, as identified in the schedule but including development review of test documentation, have been identified.

Since the software engineering section of the Software Development Plan will contain information regarding the software engineering environment as well as software development techniques and methodologies, test engineers should review this section. Test engineering personnel should be able to obtain some information regarding components of the test environment from this section. In addition, the development techniques and methodologies may provide information on development tools that can be used to assist in formal testing.

3.5.5: Outputs

Outputs from these subprocesses are listed below:

1. Software Development Plan issues
2. format and testing issues
3. testing requirements
4. reviewed Software Development Plan (after review by test engineering)
5. test engineering's comments on the Software Development Plan

3.5.6: Feedback from Later Process Steps

Test engineering personnel may not be able to identify all inconsistencies, omissions, and problems with the Software Development Plan during this review, prior to writing the test plan. As other problems are identified during the later phases in the program life cycle (for example, during detailed design) and in comparable phases in the testing life cycle (for example, during test design), it may be necessary to make updates to the Software Development Plan to resolve problems as they are discovered.

3.6: Review Configuration Management Plan—Level 3 IPO Diagram

The Level 3 IPO diagram for subprocess 1.1.4, Review Configuration Management Plan, is shown in Figure 3-6. There are two inputs for this subprocess:

1. Configuration Management Plan
2. standards and templates

There are four outputs from this process:

1. Configuration Management Plan reviewed by test engineering
2. review comments from test engineering
3. CMP issues, including format and testing issues
4. testing requirements

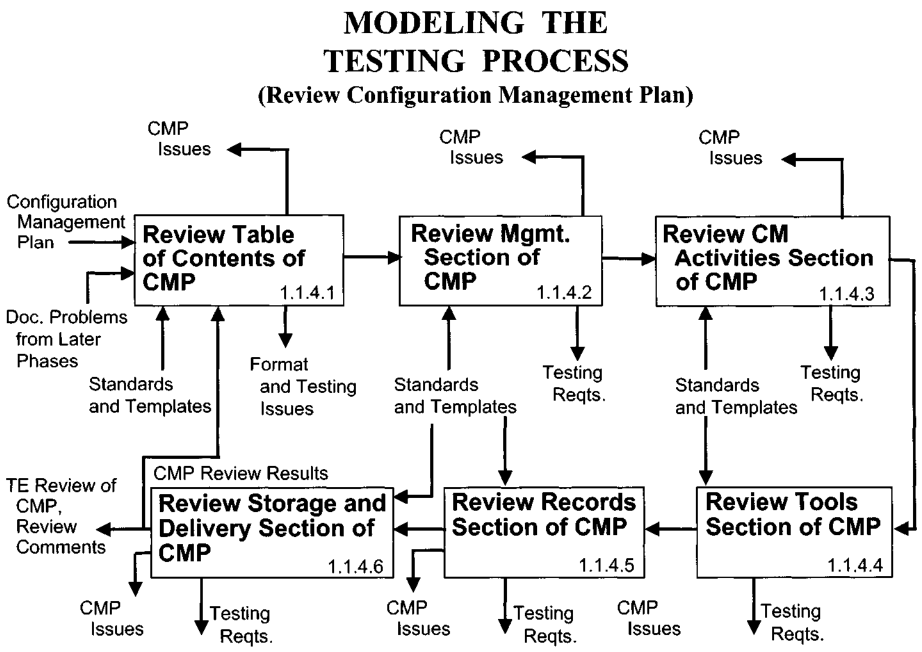
MODELING THE TESTING PROCESS
(Review Configuration Management Plan)
Configuration Management Plan
Doc. Problems from Later Phases
Review Table of Contents of CMP 1.1.4.1
Review Mgmt. Section of CMP 1.1.4.2
Review CM Activities Section of CMP 1.1.4.3
Review Tools Section of CMP 1.1.4.4
Review Records Section of CMP 1.1.4.5
Review Storage and Delivery Section of CMP 1.1.4.6
CMP Issues
Standards and Templates
Format and Testing Issues
Testing Reqts.
CMP Review Results
TE Review of CMP, Review Comments

Figure 3-6: The Level 3 IPO diagram contains six CMP subprocesses for test engineers to review.

Six distinct subprocesses are identified in this Level 3 IPO diagram:

1. Review Table Of Contents Of CMP
2. Review Management Section Of CMP
3. Review CM Activities Section Of CMP
4. Review Tools Section Of CMP
5. Review Records Section Of CMP
6. Review Storage and Delivery Section Of CMP

3.6.1: Inputs

The primary input for this subprocess is the Configuration Management Plan, which is ready for review. The testing staff needs to be aware of the standards used to develop the Configuration Management Plan, so that the plan can be compared against the appropriate standards or templates. IEEE-Standard-828, supported by IEEE-Standard 1042, can be used.

3.6.2: Internal Feedback

Since review of any document is an iterative process, review comments may flow back to the authors on multiple occasions, depending on how many times the CMP is revised before approval takes place. Thus, updated versions (under configuration management version control) of the CMP may reenter this process asynchronously, setting off a repeat review by test engineering (as well as by other interested groups or personnel).

3.6.3: Feedback to Earlier Process Steps

When issues are found in the CMP, those issues should be documented and presented to the configuration manager, so that those issues can be resolved promptly. Development of a high-quality test plan, as well as the other program-level plans (the Software Development Plan or the Program Management Plan, for example), will be assisted by timely resolution of such issues.

3.6.4: Process Flow, Level 3 IPO Diagram

The most significant sections of a Configuration Management Plan that should be reviewed by test engineering are the table of contents; the management section; the configuration management activities section;

the tools, techniques, and methodologies section; the records collection and retention section; and the storage and delivery section of the program media section. Though there are other sections of this document that may be of interest, the sections listed here should be reviewed most thoroughly.

Test engineering should review the table of contents and compare it with the standards (for example, IEEE-Standard-828) and templates that apply to a CMP. The test engineering staff should immediately alert its managers to any discrepancy or omission. Such omissions should be documented as a potential defect, and entered into the defect-tracking system to alert management to the potential problem area. Otherwise, a report of the omissions should be sent to the configuration manager as soon as possible so that the issues can be resolved.

Next, test engineering personnel should review the management section of the CMP, which should document CM staff responsibilities as well as the top-level policies that the CM group will administer on this specific program. Since these responsibilities and policies will impact test engineering, its staff needs to be aware of and comfortable with the responsibilities and policies.

The CM activities section of the CMP should document configuration identification methods, configuration status accounting, configuration verification, CM audits and reviews, and the structure of the CM libraries. Again, test engineering staff members need to be aware of and comfortable with the activities and techniques used by CM, since there will be frequent interfaces between CM and the testing staff.

The tools, techniques, and methodologies section of the CMP should be reviewed by test engineering because each tool or technique or methodology can impact the interfaces between the CM group and test engineering. Use of the tools and methodologies by CM may well affect how long it will take test engineering personnel to get an updated build from CM. Also, if CM is planning to inject new builds on a daily basis and test engineering is planning to receive a new build only weekly, the interface will not be synchronized and will affect the program schedule.

Test engineering staff should review the records section of the CMP so that the engineers will know where and how to find information regarding previous builds and change requests.

Test engineering staff members also should review the storage and delivery section of the Configuration Management Plan so that they will know where and how to obtain copies of updated builds, updated program documentation, and previous and current issues of the test documentation.

3.6.5: Outputs

Outputs from these subprocesses include the following:

1. Configuration Management Plan issues
2. format and testing issues
3. testing requirements (for use in the test plan)
4. the CMP review results (from test engineering)
5. comments on the CMP (from test engineering)

3.7: What's Next?

The following chapter presents details in the Level 2 and Level 3 IPO diagrams for process 1.2, Create Test Plan.

Index