# Requirements Specification and Verification of Production Tooling

by

Roland J. Ayala

Bachelor, Mechanical Engineering
Georgia Institute of Technology, 1989

Submitted to the Department of Mechanical Engineering and
the Sloan School of Management
in Partial Fulfillment of the Requirements for the Degrees of

Master of Science in Mechanical Engineering
and
Master of Science in Management

at the
Massachusetts Institute of Technology
June 1995



Signature of Author _____
Department of Mechanical Engineering
MIT Sloan School of Management
May 23, 1995

Certified by _____
John G. Kassakian
Professor of Electrical Engineering and Computer Science

Certified by _____
Anantaram Balakrishnan
Associate Professor of Management

Read by _____
Kevin N. Otto
Assistant Professor of Mechanical Engineering

Accepted by _____
Ain Sonin
Chairman, Department Committee of Graduate Studies

Accepted by _____
Jeffrey A. Barks
Associate Dean, Master's and Bachelor's Programs

**Requirements Specification and Verification of Production Tooling**

Roland J. Ayala

Submitted to the Department of Mechanical Engineering and
the MIT Sloan School of Mechanical Engineering
on May 23, 1995, in partial fulfillment of the requirements for the Degrees of
Master of Science in Mechanical Engineering and
Master of Science in Management

## Abstract

This thesis provides the motivation and development of a process for rigorously specifying and verifying production tooling requirements. The process is most important in industries where product lifecycles are short and production ramps must be fast. The hard disk drive industry is one such industry. Given that hard disk drive market lives are approaching one year in length, the success or failure of a production ramp can greatly influence the overall profitability of a product line. In an effort to understand the factors impairing a successful production ramp, the management at Hewlett-Packard's Disk Memory Division has identified problematic production tooling as a leading contributor to production ramp delays. The management at Disk Memory Division feel many of these problems result from poor requirements specification and verification methodologies.

In developing the process for requirements specification and verification, this thesis explores some of the methodologies used by the requirements engineering profession to specify and verify the requirements of software and systems. We will apply these methods to the requirements specification and verification process that this thesis proposes. Two of these methods, object-oriented analysis and quality factors analysis, are used to support rapid specification development and organizational learning. Moreover, we will show how the Standard Generalized Markup Language (SGML) can be used to create a system that supports the proposed requirements specification and verification process.

The author's experiences at Hewlett-Packard's Disk Memory Division are used throughout this thesis to motivate the need for the requirements specification and verification of production tooling and are key factors influencing the design of this process.

**Thesis Advisors:**

Dr. Anantaram, Balakrishnan, Associate Professor of Management
Dr. John G. Kassakian, Professor of Electrical Engineering and Computer Science

## Acknowledgments

I would like to thank the people at Hewlett-Packard's DMD for their time and constant willingness to share information with me. Without their help, this thesis would not have been possible. Thanks also to Anantaram Balakrishnan and John Kassakian, my thesis advisors, for their patience and guidance. Their ideas and questions where extremely helpful to me during my internship and while writing this thesis. Finally, a special thanks to my family and friends for their support and encouragement while writing this thesis.

# Table of Contents

LIST OF FIGURES ..... 11

1 INTRODUCTION ..... 13
1.1 Motivation ..... 13
1.2 Production Tooling and its Role in the Manufacturing Process ..... 14
1.3 The Purpose of Requirements Specification and Verification ..... 15
1.4 The Importance of Requirements Specification and Verification ..... 16
1.5 Thesis Overview ..... 20

2 REQUIREMENTS ENGINEERING PRINCIPLES ..... 23
2.1 Introduction ..... 23
2.2 The System Development Lifecycle Model ..... 25
2.3 Problem Analysis ..... 27
2.3.1 Methodology: Hierarchy Definition, Allocation, and Flowdown ..... 28
2.4 Requirements Documentation ..... 30
2.4.1 Modifiable ..... 31
2.4.2 Verifiable ..... 31
2.4.3 Traceable ..... 32
2.4.4 Annotated ..... 32
2.5 Verification and Validation ..... 33
2.5.1 Verification ..... 34
2.5.2 Validation ..... 34
2.6 Summary ..... 35

3 THE REQUIREMENTS SPECIFICATION AND VERIFICATION PROCESS AT HEWLETT-PACKARD'S DMD ..... 39
3.1 Introduction ..... 39
3.2 Case Study: DMD's Semi-Automated Head-Merge Workcell ..... 41
3.3 DMD's Tooling Development Cycle ..... 50
3.3.1 Requirements Phase ..... 50
3.3.2 Design Phase ..... 52
3.3.3 Build Phase ..... 54
3.3.4 Verification Phase ..... 54

3.3.5 Installation Phase ..... 55
3.3.6 Summary ..... 55
3.4 Problem Analysis ..... 56
3.4.1 Methodology ..... 56
3.5 Requirements Documentation ..... 58
3.5.1 Methodology ..... 58
3.6 DMD's Requirements Verification and Validation Process ..... 63
3.6.1 Verification ..... 63
3.6.2 Validation ..... 65
3.7 Conclusions ..... 66
3.8 Summary ..... 67

4 FRAMEWORK AND TOOLS FOR A LEARNING ORGANIZATION ..... 71
4.1 Introduction ..... 71
4.2 An Object-Oriented Approach to Requirements Specification ..... 71
4.2.1 Object Orientation ..... 72
4.2.2 Object-Oriented Analysis (OOA) ..... 73
4.3 Creating a Knowledge Base Using Quality Factors ..... 80
4.3.1 The Quality Factors Matrix (QFM) ..... 80
4.3.2 The Quality Factor Recipe (QFR) ..... 83
4.4 Using the OO Framework as a Knowledge Index ..... 85
4.5 Summary ..... 86

5 SPECIFYING AND VERIFYING PRODUCTION TOOLING REQUIREMENTS: A NEW PROCESS ..... 91
5.1 Introduction ..... 91
5.2 Process Overview ..... 92
5.3 The New Process: Step by Step. ..... 95
5.3.1 Allocate Requirements ..... 95
5.3.2 Preliminary Concept Review ..... 97
5.3.3 Transformation and Documentation of Requirements ..... 101
5.3.4 Develop Concept ..... 107
5.3.5 Specify Requirements (Final) ..... 108
5.3.6 Design Tool and Verify ..... 108
5.3.7 Build Tool and Verify ..... 109
5.3.8 Install Tool and Validate ..... 109
5.4 The Value of Organizational Learning ..... 110

5.5 Summary ..... 111

6 USING SGML TO SPECIFY PRODUCTION TOOLING REQUIREMENTS ..... 115

6.1 Introduction ..... 115

6.2 SGML Basics ..... 116

6.3 Using HyperText to Support Organizational Learning ..... 117

6.4 Using SGML to Support the Requirements Specification Process ..... 119

6.4.1 RML Document Type Definition (DTD) ..... 119

6.5 An SGML Based Requirements System ..... 121

6.5.1 The SGML Database ..... 121

6.5.2 The SGML Parser ..... 122

6.5.3 Application Library ..... 122

6.6 Conclusion ..... 123

7 CONCLUSION ..... 125

REFERENCES ..... 125

APPENDIX A – EXAMPLE CLASSIFICATION HIERARCHIES FOR TWO DIFFERENT INDUSTRIES ..... 127

APPENDIX B – EXPANDED CLASSIFICATION HIERARCHY FOR A HARD DISK DRIVE MANUFACTURER ..... 131

APPENDIX C – DOCUMENT TREE AND STRUCTURE FOR THE RML ..... 135

APPENDIX D – DOCUMENT TYPE DEFINITION ..... 137

APPENDIX E – EXPLANATION OF RML RULE SET ..... 139

APPENDIX F – EXAMPLE RML DOCUMENT SHOWING MARKUP AND DATA ..... 141

# List of Figures

Figure 1.1 – The Cumulative Effects of Error (CEE) model applied to a production tooling specification. ....19
Figure 2.1 – Production tooling as a sub-system of the manufacturing process ....24
Figure 2.2 – The activities of the Requirements Phase used for system development. ....25
Figure 2.3 – The five phase Standard Waterfall model used for software and systems development. ....26
Figure 2.4 – The five phase Incremental Development model used for software and systems development (Dorfman, 1990, p.5). ....27
Figure 2.5 – System hierarchy (breakdown) of a manufacturing process. ....29
Figure 2.6 – Iteration in partitioning, allocation, and flowdown of requirements. ....30
Figure 2.7 –The verification and validation (V&V) cycle for a production tool (adapted from Thayer and Royce, 1990, p. 94). ....35
Figure 3.1 – A cutaway view of a typical hard disk drive (Goodman, 1993, p. 36). ....40
Figure 3.2 – A typical view of the spindle, media, and head-stack assembly (Goodman, 1993, p. 110). ....42
Figure 3.3 – A view of an arm-stack assembly showing the arms that position the heads over the surface of the disks. ....42
Figure 3.4 – A view of the HSA with the heads positioned over the landing zone (Goodman, 1993). ....43
Figure 3.5 – A simplified, partial process flow of DMD's production line. ....44
Figure 3.6 – A rough view of DMD's head-merge workcell. ....45
Figure 3.7 – A view of the hard disk drive after the HSA has been inserted into the baseplate assembly (left) and after the HSA has been rotated into position just prior to merge (right). ....46
Figure 3.8 – Rod assemblies used to support and position the mirrors for the head-merge workcell. ....48
Figure 3.9 – A view of a rod assembly with the connectors that provide various degrees of freedom depending on the number of connectors used. ....49
Figure 3.10 – The five phases of the tooling development cycle at Hewlett-Packard's DMD. ....50
Figure 3.11 –The Requirements Phase in DMD's tooling development cycle. ....50
Figure 3.12 –The Design Phase in DMD's tooling development cycle. ....53
Figure 3.13 – DMD's Requirements Phase with and without feedback. ....64
Figure 4.1 – Definition of the specification class production_tool. ....75
Figure 4.2 – The production_tool class and three instances of it (a, b, and c). ....75
Figure 4.3 – Specification classes created for four different types of a production tool. ....76
Figure 4.4 – Class hierarchy developed from the base class, production_tool. ....77
Figure 4.5 – New class definition developed using multiple inheritance. ....79
Figure 4.6 – Quality Factors Matrix (QFM) for production tooling showing the needs of the manufacturing process (Qualify Factors) versus a tool's technical attributes (Quality Sub-Factors). ....82
Figure 4.7 – The quality factor 'maintainability' broken down into its five sub-factors. ....83
Figure 4.8 – Example QFR for software maintainability. ....83
Figure 4.9 - Using the specification classes as an index to the knowledge of the organization. ....86

Figure 5.1 – The process flow for a new requirements specification and verification process. ....93
Figure 5.2 – The process flow for DMD's current requirements specification and verification process. ....94
Figure 5.3 – The five phases of DMD's tooling development cycle. ....94
Figure 5.4 – The five phase Incremental Development model for production tooling. ....95
Figure 5.5 - Requirements of the head-merge process. ....96
Figure 5.6 – Allocating the requirements of head-merge process among its sub-processes. ....97
Figure 5.7 – Example tooling request form following the allocation of requirements from process to tool. ...97
Figure 5.8 – Inputs and outputs to the Preliminary Concept Review. ....98
Figure 5.9 – The requirements specification package for a production tool. ....99
Figure 5.10 – Requirements allocation among automated (workcell) and non-automated tasks. ....100
Figure 5.11 –Tooling requirements specification based on the second round of requirements allocation. .100
Figure 5.12 – The process flow for "Specify Requirements." ....102
Figure 5.13 – Plugging existing specifications into the new head-merge specification. ....104
Figure 5.14 – Head-merge workcell class (fully expanded). ....105
Figure 5.15 – Head-merge workcell class showing the vision system class "plugged in." ....108
Figure 6.1 – An SGML document showing how tags can be used to format text. ....117
Figure 6.2 – An SGML document as viewed from an application that interprets and displays SGML documents. ....117
Figure 6.3 – The four basic subsystems of an SGML system ....122

# 1
# Introduction

## 1.1 Motivation

Today's US manufacturers are in the midst of a manufacturing crisis. Forced to compete in the global marketplace, emphasis has shifted from product to process in an effort to become more competitive in the areas of cost, quality, and delivery. Furthermore, where technology driven markets are concerned, the rate of technological innovation and new product introduction have also become critical success factors. Shapiro (1991), after having studied Hewlett-Packard's New Product Introduction (NPI) process, states that "The ability to quickly respond to customer demands with new product introductions is critical to [HP's] success. 'There is evidence that without dramatically improving the ability to reduce time to market, the US will fall hopelessly behind foreign competition in the next decade' (Dertouzos et al. 1989, Skinner 1986, and Thurow 1987)." Hewlett-Packard's Disk Memory Division (DMD), a manufacturer of hard disk drives, understands the importance of NPI and has mounted a large scale effort to improve its NPI process. Part of this effort involves improving the process used to specify and verify production tooling requirements. The reason: problematic production tools are adversely affecting DMD's production ramp times. DMD's management feels many of these problems are the result of poor requirements specification and verification methods.

DMD is not the first organization to understand the importance of requirements specification and verification. Almost thirty years ago, in the wake of the software crisis, the software industry launched investigations to determine why so many software projects were failing. Merlin Dorfman (1990) reports that:

> "These investigations determined that requirements deficiencies were among the most important contributors to the problem: in nearly every software project that fails to meet performance and cost goals, requirements inadequacies play a major and expensive role in project failure, and development of the requirements specification in many cases seems trivial, but is probably the part of the process which leads to more failures than any other. (Dorfman, 1990, p. 4)"

These findings led to the birth of the requirements engineering profession. In Chapter 2, we will introduce some of the methods requirements engineers use to specify and verify system requirements, and apply these methods to the requirements specification and verification process that develops for production tooling in Chapter 5. The remainder of this chapter discusses the role of production tooling in the manufacturing process, the purpose of requirements specification and verification, and the importance of requirements specification and verification.

## 1.2 Production Tooling and its Role in the Manufacturing Process

In this thesis, the term *production tool* shall be used to mean any hardware, or hardware-software combination that supports a manufacturing process (e.g., assembly fixture, automated workcell, milling machine). More abstract than hardware is the concept of software as a production tool. At Hewlett-Packard's DMD, software modules are used to add value to a hard disk drive assembly–hardware is present only as an interconnect mechanism between the hard disk drive assembly and the functionality of the software. One example of a software tool is an electronic configuration module that writes configuration data to the non-volatile, read-only memory portion of a hard disk drive's printed circuit assembly. The electronic configuration data varies according to the

customer of each hard disk drive being processed. The software module communicates with the hard disk drive to determine its identification number, cross-references this with a customer, downloads the customer configuration data, writes this data to the read-only memory, and then verifies the configuration. In this case, it is the software configuration module that acts in the capacity of a production tool because it is adding the value, not the interconnect hardware.

DMD makes a point of distinguishing between a manufacturing process and a production tool. A production tool is only part of the system defined by a manufacturing process (i.e., a production tool is a subsystem of the manufacturing process). We will elaborate on this important distinction in Section 2.1.

## 1.3 The Purpose of Requirements Specification and Verification

A requirements specification is a precise definition of the specific attributes a system must possess (Keller and Kahn, 1990). Its purpose is to:

1. Communicate the requirements of the system among its customers, users, analysts[1], and designers.
2. Support the verification and validation of the system.
3. Control the evolution of the system.

In Chapter 2, we will introduce the branch of science known as requirements engineering and discuss some of the methods requirements engineers[2] use to create requirements specifications for software and systems. The output of the requirements specification process is a requirements specification document and it is this document that enables the communication, support, and control listed above.

---

[1] A requirements analyst is the person or one of the persons responsible for determining the needs of a system and transforming these needs into a systems requirements specification.

[2] The terms requirements engineer and requirements analyst are interchangeable. *Analyst* is often used to mean either because it is syntactically less cumbersome.

The complement of requirements specification is requirements verification. Verification occurs throughout the development cycle of a system and its purpose is to determine whether the products of a given phase[3] fulfill the requirements established in the previous phase (Thayer and Royce, 1990). Verification differs from validation in that validation is concerned about whether the completed system satisfactorily meets the *needs* of its customers and users[4]. The combination of verification and validation is commonly referred to as the verification and validation (V&V) cycle and it will be discussed in Section 2.5.

## 1.4 The Importance of Requirements Specification and Verification

"One of the most common reasons systems fail is because the definition of system requirements is bad." (Scharer, 1981). If this is true, it follows that having a good system requirements specification is important. In Chapter 3, we will show how the failure to specify a good set of requirements for a production tool can have a negative impact on the manufacturing process using this tool. This will be shown using an actual case study of a head-merge workcell procured by DMD (Section 3.2). A production tool that fails to meet the needs of the manufacturing process, and ultimately the production system, can be very costly. Let us assume that, as a manufacturer, we have production tools that fail to meet their requirements for throughput, quality, and safety. The costs associated with these failures goes beyond the costs involved in replacing or repairing them; we will call these costs the *real cost* of a production tool that fails to meet its requirements. Included in the real cost must also be the costs of lost production, decreased quality, and injury. Perhaps the greatest contributor to the real cost of a production tool are those costs that are not easily quantifiable. These costs can include the amount of customer goodwill lost from missed delivery targets and poor product quality, and lowered work force morale stemming from unsafe and difficult to use production tools. Given the real cost of a

---

[3] The development phase concept is introduced in Section 2.2.

[4] The needs of a system are not necessarily reflected in the requirements of a system.

production tool that fails to meet its requirements, the author feels that it is important to have a good requirements specification because it will empower the tool's purchaser to determine objectively whether the tool satisfactorily meets the manufacturing process. It is the real cost of a production tool that fails to meet its needs that has sparked DMD's interest in the requirements specification and verification process. Davis (1990) further motivates the importance of requirements specification. He states:

1. The later in the development lifecycle that a [system] error is detected, the more expensive it will be to repair.
2. Many errors remain latent and are not detected until well after the stage at which they are made.
3. There are requirements errors being made.
4. Errors made in requirements specifications are typically incorrect facts, omissions, inconsistencies, and ambiguities.
5. Requirements errors can be detected.

To support his first claim, Davis compiles the results of three independent studies performed by GTE, TRW, and IBM. Even though these companies were completely unaware of each other's activities regarding the studies on the importance of requirements, they all reached roughly the same result. Davis tabulates the results of these investigations in Table 1.1 where he arbitrarily assigns a cost to the effort required to repair an error during the coding stage and expresses the repair costs in the other stages in terms of the coding repair effort. During the author's interviews with DMD's engineers and managers, most agreed that a similar profile exists for their production tools while the others felt that this profile is conservative (i.e., the cost of repairing a tooling error in the maintenance phase exceeds 20 times the cost of repairing it when it is in the build phase).

The conclusions drawn from Table 1.1 are only concerned with the cost of the repair itself–it does not reflect the real cost associated with the error. Davis explains the apparently dramatic cost of repair increase shown in Table 1.1 using the Cumulative

Effects of Error (CEE) model (Mizuno, 1983) which is shown in Figure 1.1. Davis explains the model:

> "[Assume] that we begin with a real problem [and] then write a requirements specification. Some part of that specification will be correct and the remainder erroneous. Then we move on to design. During the design stage, design based on the erroneous specification will certainly be incorrect; meanwhile design based on correct requirements specification will result in part in correct design and in part in erroneous design. Then we move on to implementation. During implementation, [tools] based on design originating from erroneous requirements specification will certainly be incorrect; [tools] based on erroneous design will certainly be incorrect. Meanwhile, [tools] based on correct design will result in [parts of the tool being correct and the other parts being incorrect]. Then we move on to testing. During testing, the part of the [tool] that is correct will hopefully be demonstrated to work correctly. Some errors will be detected and corrected, some will be detected and left uncorrected, and some will not be detected at all (Davis, 1990)."[5]

| Stage | Relative Cost of Repair |
|---|---|
| Requirements | 0.1-0.2 |
| Design | 0.5 |
| Coding | 1 |
| Unit Test | 2 |
| Acceptance Test | 5 |
| Maintenance/Ops | 20 |

Table 1.1 – Cost (effort) to repair software as a function of the time of detection in its development cycle.

The hidden errors shown in Figure 1.1 are the worst type of errors because the purchaser of the tool accepts its delivery thinking the tool meets its needs. It is not until the tool is being installed or in operation that the errors are actually realized. From Figure 1.1, it is

[5] Davis' explanation has been reworded in the context of a production tool.

apparent that the only path to hidden errors is through errors in the requirements specification[6].

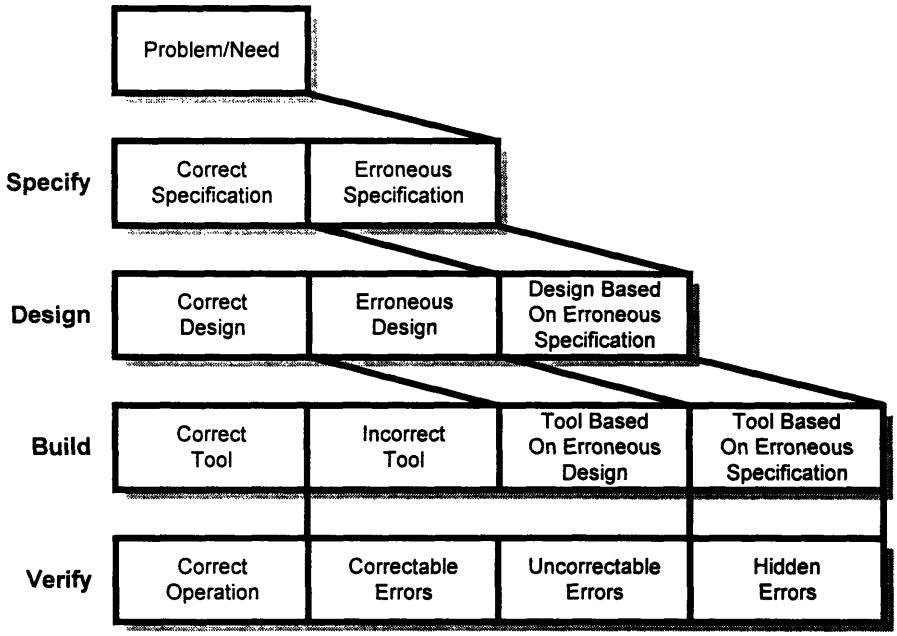



Figure 1.1 – The Cumulative Effects of Error (CEE) model applied to a production tooling specification.

Given the CEE model, the author feels that most important reason for having a good requirements specification is to eliminate the presence of hidden errors. Only when a production tool has been determined to be error free is it accepted by its purchaser. Those errors which are hidden, however, will escape detection and will be present in the tool upon its acceptance. It will not be until the tool is being installed or in operation that these errors will become apparent. If an error is sufficiently severe, the purchaser may have to delay the integration of the tool with the manufacturing process in the case where the error becomes apparent at installation, or the purchaser may have to interrupt production in the case where the error becomes apparent during the tool's operation.

[6] An error in the requirements specification does not apply only to those instances where a requirement is explicitly stated incorrectly in the requirements specification document (e.g., the weight of a tool is specified as 15 pounds maximum when it was supposed to be 10 pounds maximum). An error in a requirements specification can also apply to those instances where a requirement should have been specified but was not.

Given either of these two scenarios, it is clear how a production tool that is accepted in the presence of hidden errors can have a negative impact on the ramp time of the production line using this tool.

## 1.5 Thesis Overview

The goal of this thesis is to demonstrate a need, create a process, and develop a set of tools and methods for specifying and verifying production tooling requirements. Chapter 2 introduces the requirements engineering profession and some of the methodologies they use to specify and verify system requirements. Requirements engineering is the science and discipline concerned with analyzing, documenting, and verifying system requirements. We will show how a manufacturing process is a system and motivate the use of the proven requirements engineering methods in a process for specifying and verifying production tooling requirements.

Chapter 3 describes the process being used by Hewlett-Packard's Disk Memory Division (DMD) to specify and verify production tooling requirements. The author spent six months at DMD as part of a internship made available to the Fellows of the MIT Leaders for Manufacturing Program (LFM). The charter of this internship was to understand the process used by DMD to specify and verify the requirements of its production tooling and to develop a new process based on those areas with the greatest opportunities for improvement. The description of DMD's requirements specification and verification process is based on the author's internship experience with DMD. A description of the new process is given in Chapter 5, but only after the set of tools and methods that will be used in this new process are complete.

Chapter 4 develops a set of tools and methods that can be used to help with the process of specifying and verifying production tooling requirements. Included in these tools and methods are Object-Oriented Analysis (OOA) and the development of a Quality Factors Matrix (QFM). The purpose of OOA and the QFM is to address the need for organizational learning in the process of specifying and verifying production tooling

requirements. This need was determined in the analysis of DMD's process in Chapter 3. As we will see, however, the usefulness of OOA extends well beyond that of an aid to organizational learning.

Chapter 5 develops a new process for specifying and verifying production tooling requirements. In developing this process, we draw from what has been learned and developed in Chapters 2, 3, and 4. The new process emphasizes organizational learning as the means to developing good requirements specifications.

Finally, Chapter 6 shows how the Standard Generalized Markup Language (SGML) can be used to create a requirements specification and verification system that supports the process developed in Chapter 5.

# 2

# Requirements Engineering Principles

This chapter introduces the requirements engineering profession and some of the methodologies it uses to specify and verify system requirements.

## 2.1 Introduction

Requirements engineering is "the science and discipline concerned with analyzing and documenting requirements, including needs analysis, requirements analysis, and requirements specification (Thayer and Dorfman, 1990, p. 1)." There are two branches of requirements engineering, systems and software, but they share a common methodology in their approach to requirements specification and verification. This chapter briefly explains those methodologies that we will draw upon in the chapters that follow[7]. We shall apply the methodologies that are discussed in this chapter to the requirements specification and verification process that develops for production tooling in Chapter 5. This is possible because a manufacturing process is a system and a production tool, if one is used, is a sub-system of it. Thayer and Dorfman define a system as:

> "... a collection of hardware, software, people, facilities, and procedures organized to accomplish some common objectives (Thayer and Dorfman, 1990, p. 662)."

[7] For a more thorough explanation, see Dorfman (1990), and Davis (1990).

Given this definition, a manufacturing process is clearly a system. It organizes hardware, software, people, facilities, and procedures for the purpose of manufacturing a product. In Section 1.2 we discussed the role of production tooling in the manufacturing process and how it can be a hardware component, or a combination of hardware and software. Figure 2.1 shows the manufacturing process and how production tooling is a sub-system of it. Given this relationship, it makes sense that we should want to apply the methodologies of the requirements engineering profession to the process of specifying and verifying production tooling requirements. The requirements engineering profession has spent almost thirty years learning how to develop good requirements specifications and we want to leverage the methods that they have developed.

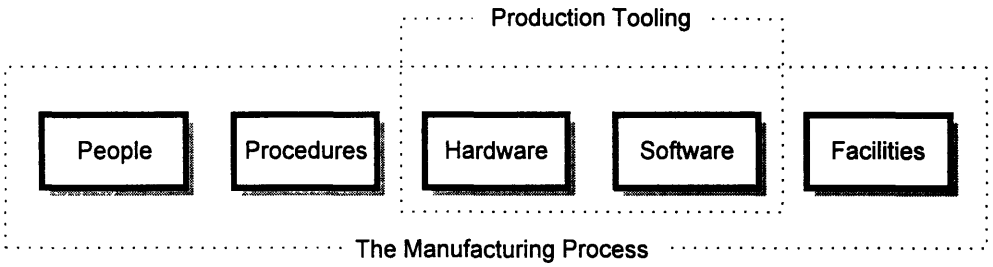



Figure 2.1 – Production tooling as a sub-system of the manufacturing process

Thayer and Dorfman describe the role of systems requirements engineering:

> "Systems requirements engineering is the science and discipline concerned with analyzing and documenting system requirements. It involves transforming an operational need into a system description, system performance parameters, and a system configuration through the use of an iterative process of analysis, trade-off studies, and prototyping (Thayer and Dorfman, 1990, p. 1)."

In this chapter we will discuss the three major activities that take place in the transformation of an operational need into the evolved system. These activities are: problem analysis; requirements documentation; and verification and validation. Before we can discuss these activities however, the concept of the system development lifecycle

model must be introduced. Analysts use the system development lifecycle model to analyze and manage the development of a system.

## 2.2 The System Development Lifecycle Model

As a system develops, it passes through a series of phases in its transformation from need to finished product. These phases characterize the major activities of a system's development at any point in time, but do not exclude the possibility that some of the activities of a previous phase may carry over into the next phase or vice versa. Given that the subject of this thesis is requirements specification and verification, we will dedicate a significant portion of it to the activities that take place in the Requirements Phase. The two major activities that take place in the Requirements Phase are problem analysis and requirements documentation. These activities are shown in Figure 2.2 and each is discussed separately in Sections 2.3 and 2.4. We shall call the activities that take place during the Requirements Phase *requirements specification.*

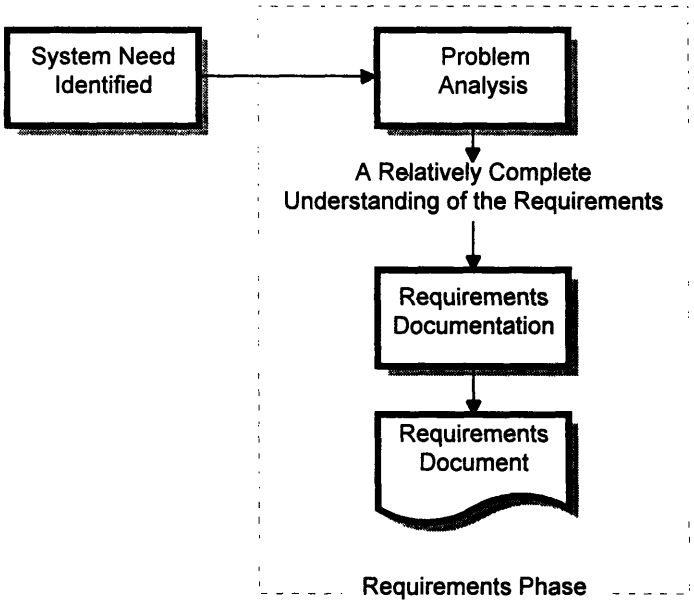



Figure 2.2 – The activities of the Requirements Phase used for system development.

Incorporating the Requirements Phase are several lifecycle models which have been developed by the requirements engineering profession; two of the better known models

are the Standard Waterfall model and the Incremental Development model (Dorfman, 1990, pp. 4-6). Analysts use these models to help analyze and manage the development of the system. The Standard Waterfall model shown in Figure 2.3 breaks a development effort into five distinct phases: requirements, design, construction, test, and integration. Ideally, requirements activities are confined to the Requirements Phase. It is normal, however, for a system's requirements to be enhanced, changed, or deleted after the project has progressed beyond the Requirements Phase. As a system's design begins to unfold, it is likely that changes to the requirements specification will be necessary. This is the reason for the iterative loop shown between the Requirements Phase and the Design Phase. This scenario can be extended to each of the development phases. Unlike the requirements specification, verification activities take place at each phase in the development lifecycle. Section 2.5 discusses the role of verification and validation in the development lifecycle. In Chapter 3, we will show how the lifecycle model used for the tooling procured by DMD is very similar to the Standard Waterfall Model.

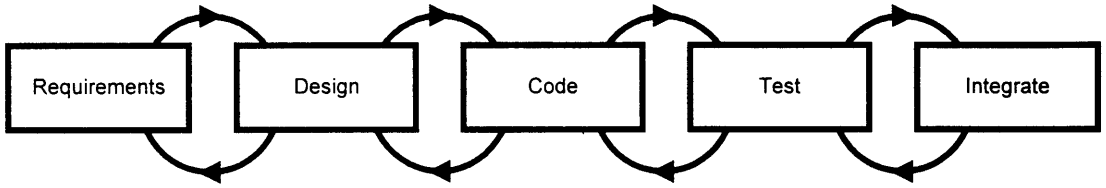



Figure 2.3 – The five phase Standard Waterfall model used for software and systems development.

Very similar to the Standard Waterfall model is the Incremental Development model[8] shown in Figure 2.4. Like the Standard Waterfall model, the Incremental Development model has five phases: requirements, design, construction, test, and integration. The difference between these two models is that the Incremental Development model uses feedback from the customers and users of operational systems to affect the outcome of a

[8] The Incremental Model, like the Standard Waterfall model, is iterative. The iteration loops have not been shown in Figure 2.4 for the purpose of clarity only.

development effort. The requirements specification and verification process described in Chapter 5 is based on the Incremental Development model.

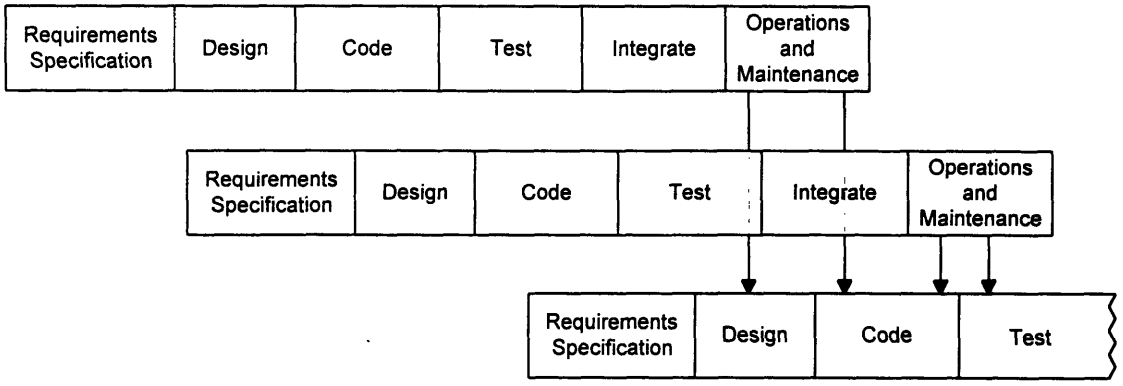



Figure 2.4 – The five phase Incremental Development model used for software and systems development (Dorfman, 1990, p.5).

## 2.3 Problem Analysis

The two major activities that take place during requirements specification are problem analysis and requirements documentation. This section addresses problem analysis– requirements documentation is left to Section 2.4. Davis (1990) briefly describes the purpose of problem analysis:

> "Problem analysis is the activity that encompasses learning about the problem to be solved, understanding the needs of the potential users, trying to find out who the user really is, and understanding all the constraints on the [system] (Davis, 1990, p. 41)."

In Chapter 3, we will show that Hewlett-Packard's DMD performs an activity similar to problem analysis in what it calls a *Preliminary Concept Review.*

If the system being developed is intended to function as part of a larger system (the parent system), problem analysis also involves understanding what is required for the system (e.g., production tool) to work within the constraints of the parent system (e.g.,

manufacturing process). Moreover, if the system being developed is sufficiently complex, it may be necessary to decompose it into a subset of smaller, more manageable subsystems. Requirements engineers employ three methods to ensure that the requirements of a subsystem meet the needs (i.e., requirements) of the parent system. These methods are commonly referred to as hierarchy definition, allocation, and flowdown (Dorfman, 1990).

### 2.3.1 Methodology: Hierarchy Definition, Allocation, and Flowdown

In the process of developing the requirements specification for a system, it is important to think about the needs of the entire system. A concern voiced by DMD's managers during the author's interviews was that they do not feel system needs (manufacturing process and production system) are being reflected in the designs and performance of production tooling. Requirements engineers have a well defined methodology to deal with this type of problem. They use a process that defines a system hierarchy, allocates the requirements of the system to each of the elements of in the hierarchy, and writes a set of requirements for each element in response to the allocation. This process ensures that the requirements of the system are being supported in its subsystems[9]. The process starts by defining a hierarchy which decomposes a system into smaller, more manageable subsystems. Dorfman describes the hierarchy creation process:

> "Early in the system development process, as the system-level requirements are being generated (in itself an iterative process), requirements engineers and others begin to consider what elements should be defined in the hierarchy. By the time the system requirements are complete in draft form, a tentative definition of at least one and possibly two levels should be available. This definition will include names and general functions and elements. Definition of the system hierarchy is often referred to as 'partitioning' (Dorfman, 1990)."

---

[9] In the case of a manufacturing process, the subsystems may include production tooling.

An example system hierarchy for a manufacturing production line is shown by Figure 2.5.

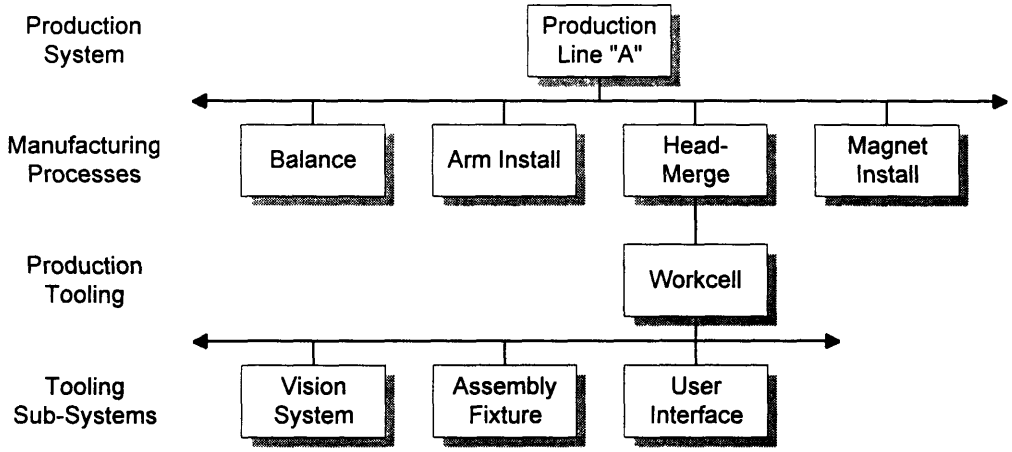



Figure 2.5 – System hierarchy (breakdown) of a manufacturing process.

The next step in the process is called requirements *allocation*. Dorfman describes the allocation process:

> "Each system-level requirement is usually allocated to one or more elements at the next level (i.e., it is determined which elements will participate in meeting the requirement). In performing the allocation, it will become apparent that (a) the system requirements need to be changed (additions, deletions, and corrections) and (b) the definitions of the elements are not correct. The allocation process therefore is iterative, leading eventually to a complete allocation of the system requirements (Dorfman, 1990, p. 8)."

Once all of the requirements have been allocated to the elements of the system (i.e., the subsystems), a *flowdown* process occurs. Requirements flowdown involves writing a requirements specification for each of the subsystems in response to those requirements which have been allocated to it from above. Figure 2.6 shows the processes just described with a series of verification steps inserted. The purpose of the verification steps is to catch errors in the allocation and flowdown processes.

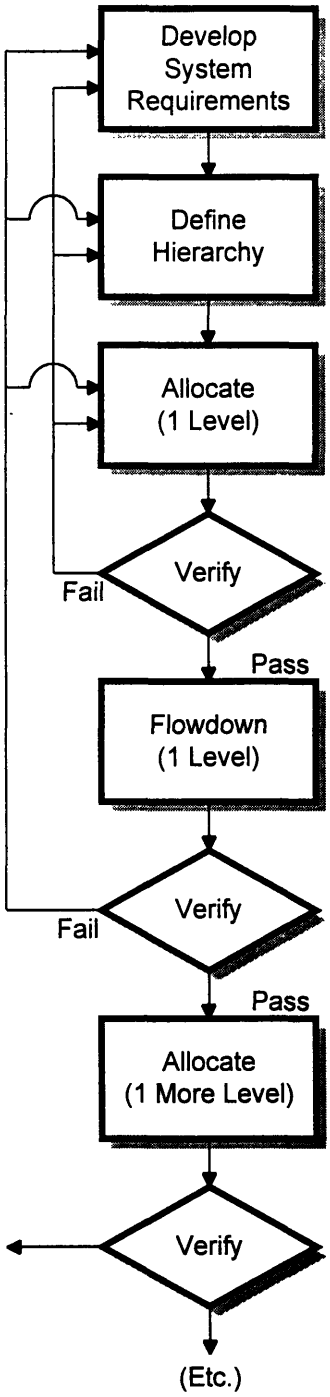



Figure 2.6 – Iteration in partitioning, allocation, and flowdown of requirements.

## 2.4 Requirements Documentation

Requirements documentation is the second step in the requirements specification process. It contains a complete description of the performance and behavior a system must possess

as well as the constraints being placed upon it. There are three reasons why a requirements specification document is necessary:

1. It communicates the requirements of the system among the customers, users, analysts, and designers.
2. It supports verification and validation of the system.
3. It controls the evolution of the system.

To be effective, Davis (1990) states nine attributes that a requirements specification document must possess. It must be:

| | | |
|---|---|---|
| 1. Correct | 2. Nonambiguous | 3. Complete |
| 4. Consistent | 5. Understandable | 6. Modifiable |
| 7. Verifiable | 8. Traceable | 9. Annotated |

The first five attributes listed are relatively straightforward and we will briefly discuss only the last four attributes. For an in-depth explanation on each of the nine attributes, the author refers the reader to Davis (1990).

### 2.4.1 Modifiable

A requirements specification document is modifiable only if its structure and style are such that any necessary changes to the requirements can be made easily, completely, and consistently (IEEE, 1984).

### 2.4.2 Verifiable

One of the reasons for having a requirements specification document is to support system verification and validation activities. Upon completion of a system, it will be verified against the requirements stated in its requirements specification document. To objectively determine whether an attribute of a system conforms to its specified requirement, the requirement itself must be verifiable. A requirement is verifiable if and

only if there exists a cost effective and time effective method of determining whether it is satisfied by the system (Davis,1990; Nelson, 1990). Davis states:

> " ... the statement 'The product shall have an easy-to-use human interface' is ambiguous, that is, has multiple interpretations because opinions of what is easy to use varies greatly from individual to individual and thus cannot be verified as an attribute of the final product (Davis, 1990, p. 190)."

The requirements specified in the requirements specification document, then, must be specific and quantifiable (e.g., "all locating points shall use carbide inserts", "all wires shall be marked with a unique identification number", "the operator shall not have to apply more than two pounds force to any component of the tool").

### 2.4.3 Traceable

Requirements traceability is concerned with why a particular requirement has been specified and, in doing so, it facilitates the decision making process during the verification and evolution of a system. A requirements specification document is traceable only if the reason for each of its requirements is clear and if it facilitates the referencing of each requirement in the future development or enhancement of documentation (IEEE, 1984).

### 2.4.4 Annotated

Annotating requirements assigns a level of importance to each of the requirements specified for a system. A system will almost certainly have some requirements that supersede others in terms of importance: some requirements will be critical to the system's correct operation while others might be superficial. By annotating the requirements in the requirements specification document, it is made clear to the designer what requirements are the most important and, thus, should be given the most attention. Requirements annotations are also useful during verification activities when decisions

must be made about whether a tool can be allowed to progress to the next phase of development even though it may not have met a particular requirement.

## 2.5 Verification and Validation

Thayer and Royce define verification and validation[10] as follows:

> "*Verification* is the process of determining whether or not the products of a given phase of the system development cycle fulfill the requirements established in the previous phase. *Validation* is the process of ensuring that what intended to be built corresponds to what is actually required; it is concerned with the completeness, consistency, and correctness of the requirements (Thayer and Royce, 1990, p. 93)."

Based on Thayer and Royce's definition, verification is not an activity that happens only at the end of a development cycle as part of a system's test activities. At each stage of the development process, the products of a given phase should be checked for errors before proceeding to the next phase. The CEE model (p. 17) makes clear the need for verification at each stage of the development process. Since the total amount of error in a system is the accumulation of error at each of its development phases, a verification step designed to catch these errors at the end of each phase can significantly reduce the total amount of error in the system.

Thayer and Royce explain the need for verification and validation:

> "Verification and validation (popularly called "V&V") is a group term for a set of system tools that is used to continually monitor the processes and products of a [tooling] development project. V&V ensures that the [tool] performs its intended functions correctly, that it will perform no unintended functions, that it will work within the total system, and that it will meet its intended performance, external

---

[10] We are rewording their definition slightly to put it in the context of production tooling.

interfaces, design constraints, and quality attributes (Thayer and Royce, 1990, p. 93)."

Figure 2.7 shows the verification and validation (V&V) cycle for a production tool (adapted from Thayer and Royce, (1990)). It makes clear the difference between verification and validation of a production tool.

### 2.5.1 Verification

Verification makes sure the products of a given phase fulfill the requirements of the previous phase. For example, before a requirements specification document should be delivered to the tool designer, its accuracy should be verified first. This process verifies that the requirements stated in the tool's requirements specification document accurately reflect the needs of manufacturing process. An analogous process exists for the design phase. Once the tool's design is complete, it must be verified that the design is consistent with the requirements stated in the tool's requirements specification document. And finally, once the tool's vendor feels the tool is ready for delivery, it must be verified against its design and the requirements specified in the requirements specification document.

### 2.5.2 Validation

There always exist the possibility that a requirements specification document will contain errors. In this case, even if a production tool is verified to meet all of the requirements specified by its requirements specification document, it will not satisfy all the needs of the manufacturing process. This is the purpose of requirements validation. It asks the question "Is the tool doing what it is supposed to be doing?" As we will see in Chapter 5, the validation process presents the manufacturing organization with an opportunity for learning in the area of how to develop good requirements specifications. Since validation is concerned with the completeness, consistency, and correctness of a requirements specification, the manufacturing organization can compare what was actually built to

what was actually needed. If there is no disparity between the two, it is safe to say that the organization knows how to develop a requirements specification. If there is a disparity, however, the manufacturing organization must ask why and take the necessary steps to correct the problem.

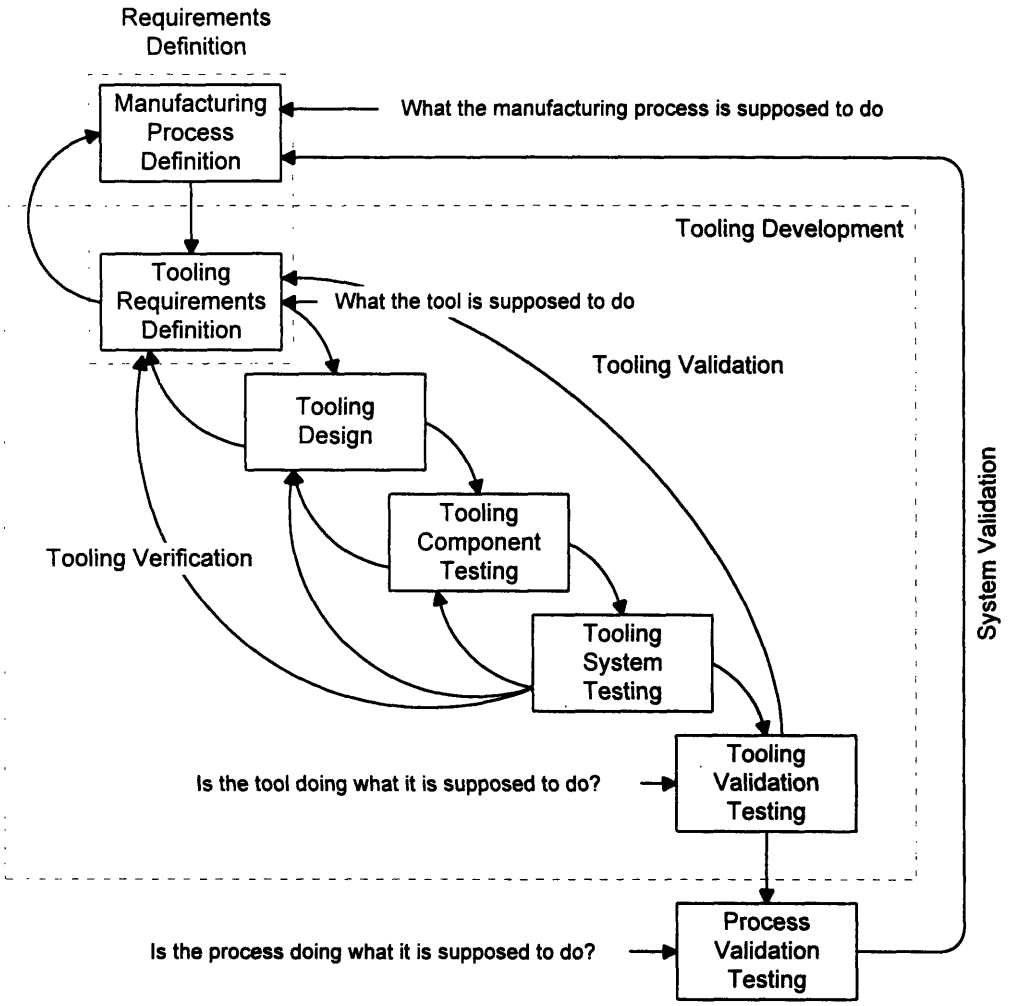



Figure 2.7 –The verification and validation (V&V) cycle for a production tool (adapted from Thayer and Royce, 1990, p. 94).

## 2.6 Summary

This chapter introduced requirements engineering methodology that we will be drawing from throughout the remainder of this thesis. Specifically, we examined problem

analysis, requirements documentation, and verification and validation. We showed how a manufacturing process is a system and how a production tool is a subsystem of it. In analyzing the requirements of a production tool, it is important to understand the needs of the manufacturing process first. These needs are "driven down" into the subsystems of the manufacturing process using a requirements specification process that defines a system hierarchy, allocates requirements to each of the subsystems, and specifies additional requirements in response to each allocated requirement (flowdown).

Once the needs of a system are relatively well understood, a requirements specification document must be created. In Section 2.4 we identified nine attributes a requirements specification document must possess. If any of these attributes are not present, there will be a negative effect on the ability of the requirements specification document to communicate requirements, support verification and validation activities, and control the evolution of a system.

Finally, we showed how requirements verification is an ongoing process. Its purpose is to determine whether the products of a given phase in the system development cycle fulfill the requirements established in the previous phase. Verification differs from validation in that validation is concerned with whether a system meets its needs. It is concerned with the completeness, consistency, and correctness of the requirements specification document.

# 3

# The Requirements Specification and Verification Process at Hewlett-Packard's DMD

This chapter describes the process currently being used by Hewlett-Packard's Disk Memory Division (DMD) to specify and verify the requirements of its production tooling.

## 3.1 Introduction

Hewlett-Packard's DMD has been designing and manufacturing hard disk drives since 1971. It has earned a reputation for performance and quality that allows it to compete successfully in the high-end disk drive market[11]. Two of DMD's success factors have been (1) its ability to develop new products that keep pace with the rate of technological change and (2) its ability to deliver these products to market on time. Today, however, the challenge of success in the hard disk drive market is higher than ever. Technological advances coupled with market demand are driving new product development cycles to just over a year, making the need for fast production ramps essential. Like so many US manufacturers, DMD has reevaluated the role of the manufacturing process in their operations and has reacted by placing increased emphasis on it. The creation of DMD's

[11] These markets include large servers and mainframes where the emphasis is on reliability and performance as opposed to cost.

Process Development Lab[12] (Process Lab) is evidence of DMD's increased emphasis on and commitment to the manufacturing process. In an effort to decrease the amount of time it takes to develop and launch a new product, members of DMD's Process Lab and Product Engineering organization work together on a New Product Introduction (NPI) team which is assembled early in the lifecycle of each new product being developed. Part of the team's mission is to make sure that once a new product is released to manufacturing, the production ramp proceeds smoothly and quickly. Often standing in the way of success, however, are production tools that fail to work properly. These failures have sparked a new concern at DMD regarding its ability to specify and verify the requirements of its production tooling. DMD's management is committed to understanding the requirements specification and verification process.

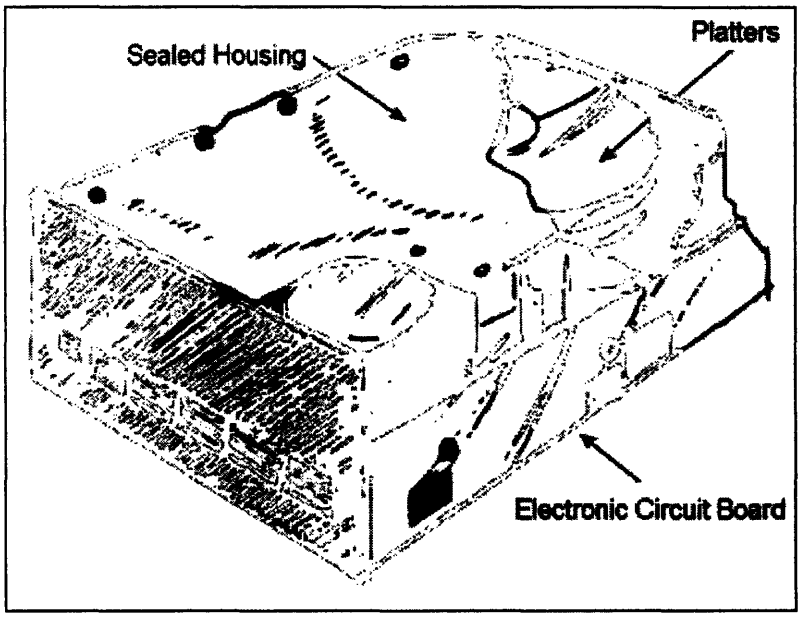



Figure 3.1 – A cutaway view of a typical hard disk drive (Goodman, 1993, p. 36).

This chapter starts by examining the case of a semi-automated head-merge workcell–a production tool procured by DMD to work with a head-merge process on one of its

[12] DMD's Process Lab is an engineering organization dedicated to developing DMD's manufacturing processes.

production lines. The case study exposes some of the problems with DMD's current requirements specification and verification process and sets the tone for the remainder of the chapter. Following the case study, the development lifecycle for the tools DMD procures is examined.[13] This chapter concludes with Sections 3.4, 3.5, and 3.6 which map to Sections 2.3, 2.4, and 2.5 with a discussion on problem analysis, requirements documentation, and verification and validation. The conclusions drawn regarding DMD's process are based on a comparison of each of the paired sections.

## 3.2 Case Study: DMD's Semi-Automated Head-Merge Workcell

The typical hard disk drive is made up of four major sub-components: the disk-stack assembly, the head-stack assembly, a sealed housing, and a printed circuit assembly (see Figure 3.1 and Figure 3.2). The disk-stack assembly is a series of aluminum disks (platters) coated with a thin layer of magnetizable material mounted on a spindle. A single read/write head is dedicated to each of the two sides of a disk. These read/write heads are attached to the end of an arm that positions them over the area of the disk where data must be read or written during a read/write operation. Since most of today's hard disk drives have more than one disk, these arms are stacked together to form an arm-stack assembly (Figure 3.3). Once the heads and electronic subassemblies have been attached to the arm-stack assembly, the assembly is called a head-stack assembly (HSA). During a hard disk drive's operation, its spindle rotates at a constant, high speed during which time the heads are never allowed to come in contact with the surfaces of the disks. To keep the heads off the surface a the disk while the drive is in operation, each head is aerodynamically designed to generate lift from the airflow caused by the rotation of the disks[14]. The only place a read/write head is allowed to touch the surface of a disk is in a

---

[13] This applies only to those tools that must be custom designed and built to meet DMD's specific manufacturing needs. It certainly does not apply to simple, off-the-self tooling (e.g., torque drivers, soldering irons, micrometer).

[14] If a head touches a read/write surface of a hard disk drive while it is in operation, it will almost certainly cause damage to the surface of the disk, and could lead to total drive failure (this is where saying "hard drive crash" comes from).

dedicated area of the disk called the landing zone (Figure 3.4). When a hard disk drive is shut down, it goes through a series of procedures which include positioning the heads over the landing zone, cutting the power to the spindle motor (this cases the heads to land because there is no more airflow to create lift), and locking the head-stack assembly into place (this is called parking the heads). The purpose of this discussion has been to stress the importance of a read/write head never touching surface of a disk (except in the landing zone). This restriction also applies during the manufacture of a hard disk drive.

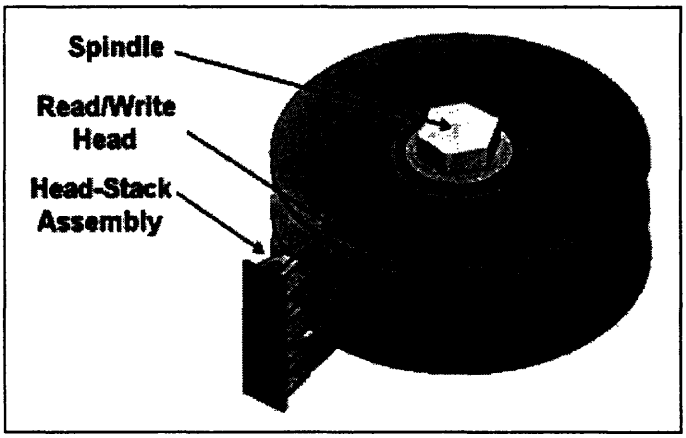



Figure 3.2 – A typical view of the spindle, media, and head-stack assembly (Goodman, 1993, p. 110).

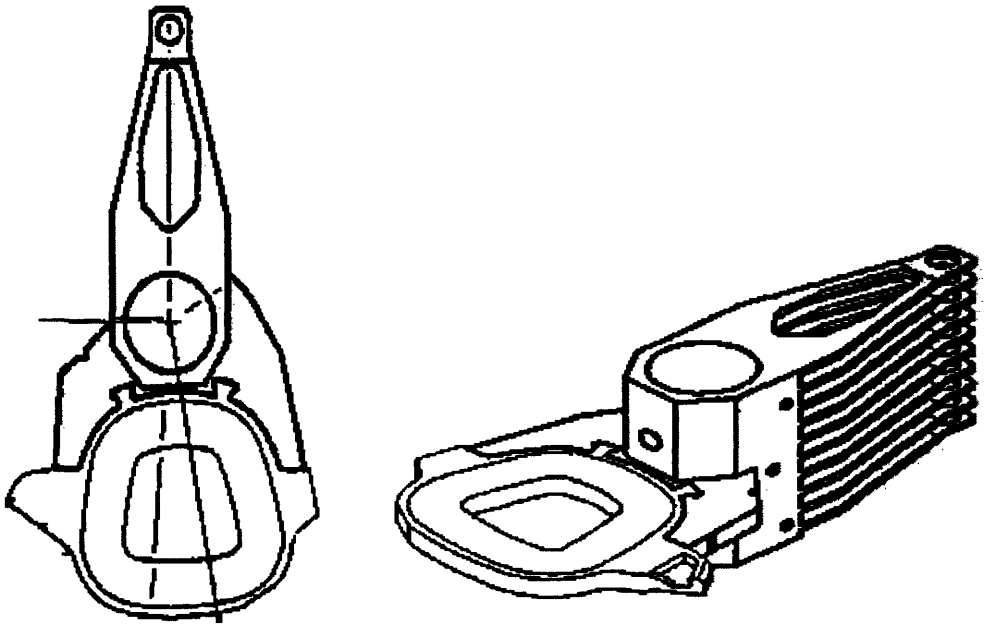


Figure 3.3 – A view of an arm-stack assembly showing the arms that position the heads over the surface of the disks.

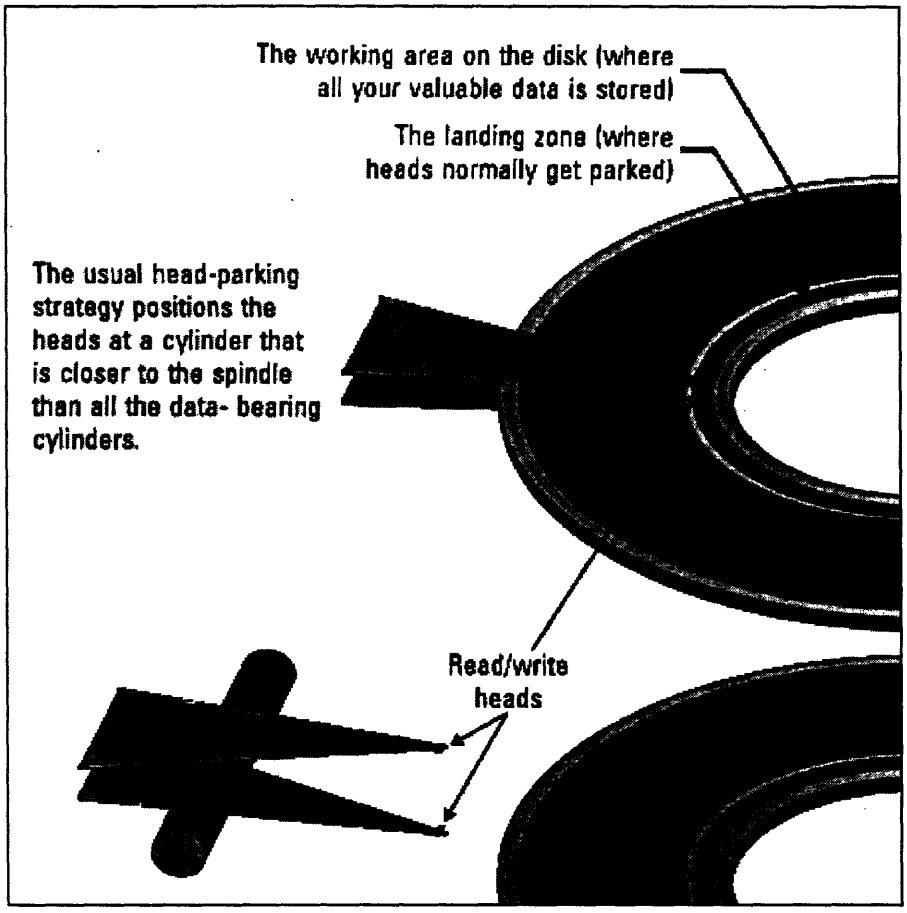



Figure 3.4 – A view of the HSA with the heads positioned over the landing zone (Goodman, 1993).

There comes a time in the manufacture of a hard disk drive when the heads must be merged with the disks. The merge process involves rotating the HSA into a position that locates the heads over their respective landing zones. While being rotated into position, the heads must be mechanically held off the surface of the disks to avoid causing damage[15]. Once the heads are in position, they are lowered onto each of their respective

[15] The arms also act as a spring which applies a force to the head in the direction of the disk. This force is not so great that it cannot be overcome by the force applied to a head from the lift generated while the disks are spinning. The presence of the spring force is necessary but it also complicates the manufacturing process. Since the disks are not spinning, the heads must be mechanically held off the surfaces of the disks while the HSA is being rotated to a position that places the heads over their respective landing zones.

surfaces and the tool that was used to rotate the heads into position and keep them off the surfaces of each disk is retracted. The process just described is called the head-merge process and it is shown in relation to its surrounding processes in Figure 3.5. Prior to procuring a semi-automated workcell, DMD used a head-merge process that incorporated a manual head-merge tool. However, the level of attention required by the process operator was high and therefore quick to cause stress and fatigue. This prompted DMD to design a more automated process which included the procurement of the semi-automated workcell which is the subject of this case study (Figure 3.6).

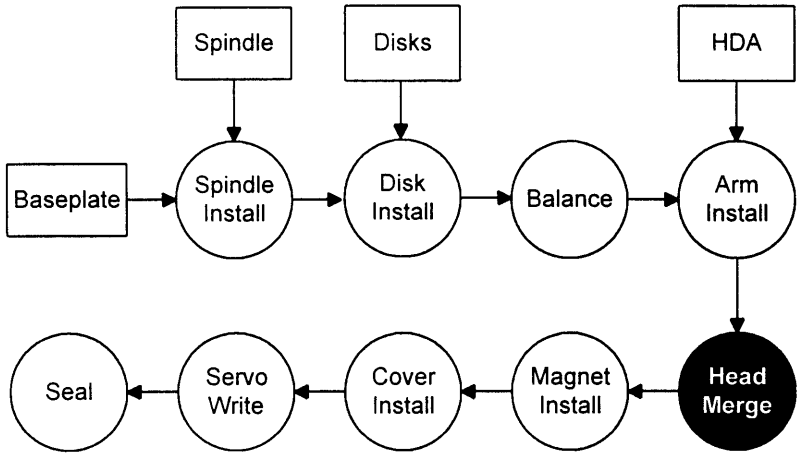



Figure 3.5 – A simplified, partial process flow of DMD's production line.

During the course of procuring the new workcell, DMD identified a tooling vendor who had developed similar systems for other hard disk drive manufacturers. DMD made the decision to use this tooling vendor. A full description of the head-merge process was given to this vendor along with all of the relevant engineering drawings for the hard disk drive assembly. There was, however, no requirements specification document for the workcell; its requirements were communicated verbally and by written correspondence. Also included with these requirements were two boilerplate requirements specification documents that had been developed for other production tools procured by DMD but intended to be generic enough that they could be applied to other production tools as well. One of these boilerplates was a workcell specification and the other was a graphical

user interface (GUI) specification. Upon completion of the tool's build process, a verification team was assembled by DMD for the purpose of verifying that the workcell met its specified requirements. There were, however, no specific set of requirements to verify against (there was no requirements specification document), but the workcell appeared to be in good working order. The verification team spent the day performing head-merge operations on a set of sub-assemblies they had brought along with them. The heads were merged without notable incident and the verification team accepted delivery of the tool.

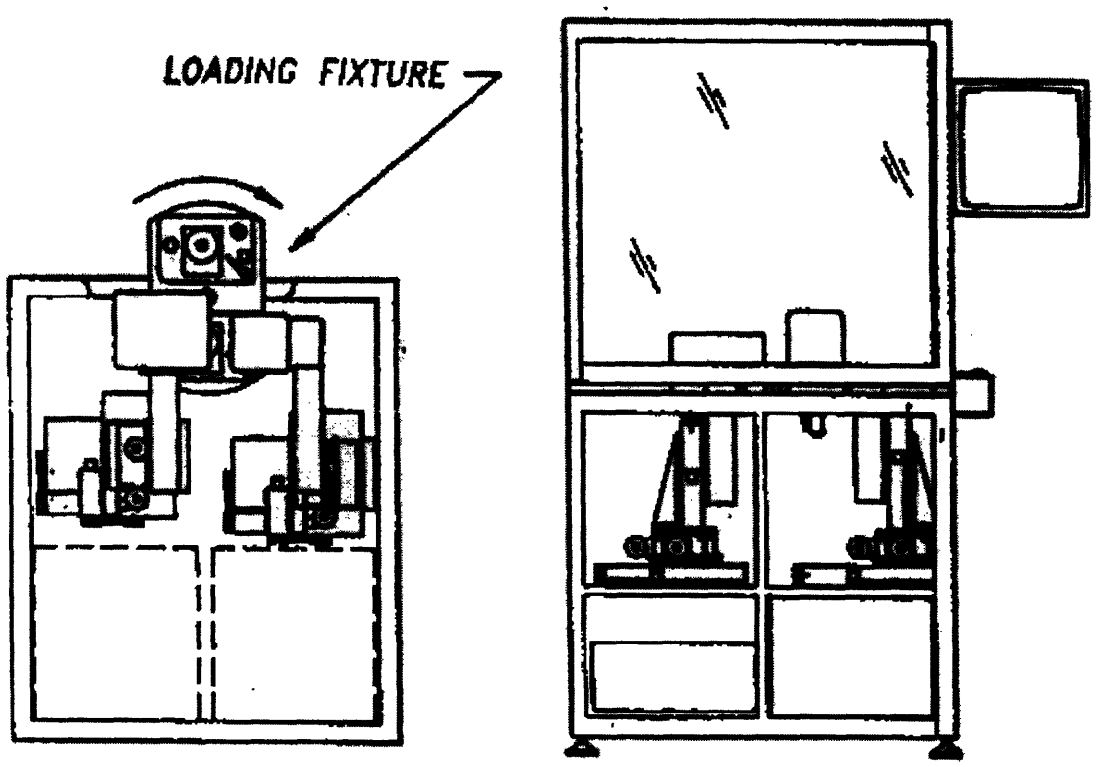



Figure 3.6 – A rough view of DMD's head-merge workcell.

Upon delivery, the workcell was integrated with the head-merge process, but there were problems from the start. One of the problems was with the vision system[16]. This was

[16] The workcell uses a vision system to verify, prior to merge (Figure 3.7), that the heads will not come in contact with the surface of the disks while the HSA rotated into position.

odd because there were no apparent problems with this system while the workcell was being verified at vendor's facility. The cause of the problem was determined to be the high intensity lighting DMD uses on the production floor. The vision system uses a backdrop light to create the level of contrast required by the vision system in the area of measure. The ambient lighting on DMD's production floor, however, is sufficiently intense to render this backdrop light ineffective[17]. The ambient lighting at the vendor's facility was lower in intensity DMD's and had no apparent effect on the performance of the vision system. Modifications were eventually made to the vision system but it has never worked as well as originally expected. Lighting however, was not the only problem with the vision system.

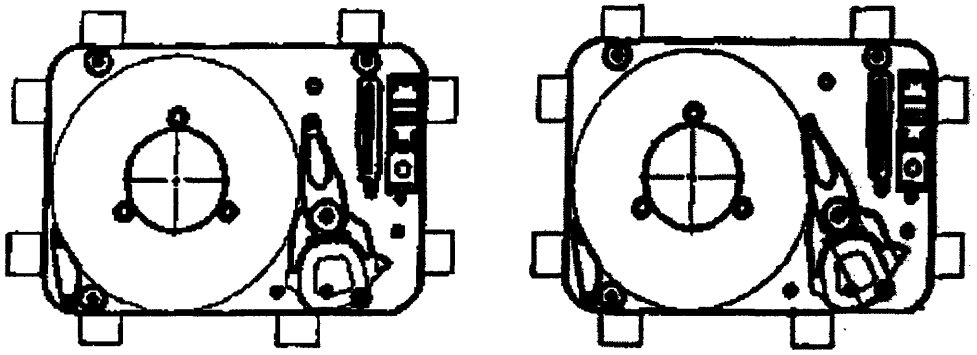


Figure 3.7 – A view of the hard disk drive after the HSA has been inserted into the baseplate assembly (left) and after the HSA has been rotated into position just prior to merge (right).

The vision system uses an array of mirrors to create a line of reflected light that is focused on the region of measure. Mirrors are located at various points inside the workcell and are attached to connectors that are in turn mounted on rods (Figure 3.8). A connector is selected based on the diameter of the rod and degrees of freedom required for the alignment of the mirror (Figure 3.9). Once aligned, a mirror is "locked" into place by tightening the thumb-screws on the rod assembly and the connector(s) supporting it.

[17] If a person wants to measure their height by casting their own shadow against a wall, he or she would enter a sufficiently dim room with a light source located directly behind his or her head. If somebody walks in, however, and floods the room with light there will be no shadow and there will be no measurement. This is effectively the problem the head-merge workcell had with DMD's bright lighting.

The locking process, however, causes a mirror to "pull back" and it is no longer properly aligned. As a result, the alignment process is very iterative and time consuming even for persons experienced with the alignment process. This might have been tolerable if the alignment process was a one-time setup process, but this was not the case. During routine cleaning operations performed by the process operator and during regularly scheduled and unscheduled maintenance operations performed by a maintenance technician, it is common for the mirrors to be jarred by the person performing the operations[18]. Once jarred, the time consuming process of aligning the mirrors be repeated.

An even bigger problem than the vision system for the workcell was software related. The operator interface was cluttered with unnecessary information and the maintenance interface did not provide all of the required functionality. Moreover, the software code used to program the controller which, in turn, controls the operation of the workcell was difficult to understand and maintain. Program constants that should have been defined globally were defined locally in each of the software modules supporting the system (e.g., a constant used for a dimensional offset was located in multiple software modules). This required that changes be made in all of the software modules each time a change to a constant was made. Not only was this time consuming, it also significantly increased the chances of an error being made either because a change was incorrectly entered or because a constant in one of the software modules was overlooked. To make matters worse, these constants where often given different names in each of the software modules (e.g., X1, offset, and A were used to refer to the same dimensional offset). A two week delay in the in the operational status of the workcell was attributed to the problematic software[19]. Interestingly enough, all of the workcell's software was produced in the last two weeks of the Build Phase.

---

[18] This head-merge tool continues to be used by DMD.

[19] Production tooling software has gained a certain level of notoriety at DMD–the author never found a single engineer at DMD who was satisfied with the quality of software used for production tooling and they all had their own set of "horror" stories. Intel has currently launched an effort to improve the quality of software on their production equipment because they have similar problems.

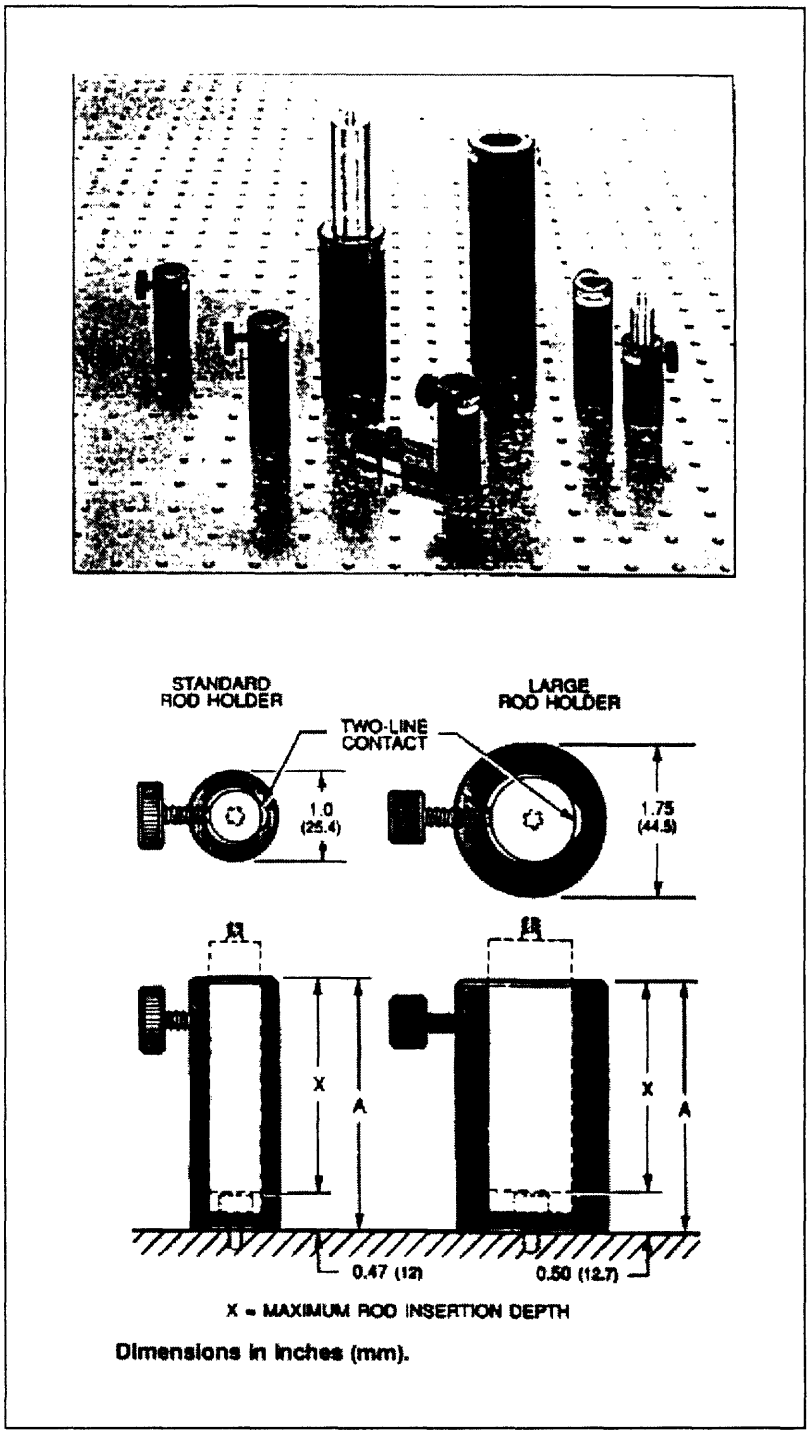



Figure 3.8 – Rod assemblies used to support and position the mirrors for the head-merge workcell.

DMD has since ordered another workcell from the same vendor. The same engineer who was involved with the procurement of the first workcell was also involved with the

procurement of the second workcell. This time, there was a requirements specification document created for the workcell and all of the issues regarding the previous workcell were addressed in the new specification. The new workcell was verified against the requirements specified in the requirements specification document and delivered to DMD where it was subjected to two weeks of rigorous testing. In that time the only incident with the workcell was a loose vacuum tube. Upon completion of testing, the workcell met or exceeded all of its performance expectations. It is the author's understanding, after having left DMD, that the workcell has continued to meet or exceed all of its expectations.

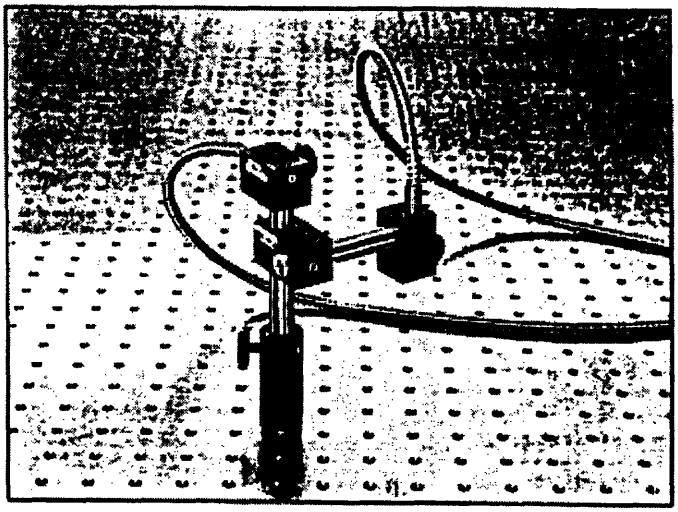



Figure 3.9 – A view of a rod assembly with the connectors that provide various degrees of freedom depending on the number of connectors used.

It is doubtful that the new workcell is perfect in every respect. What is certain, however, it that there was a significant improvement in the ability of the second workcell to satisfy DMD's needs when compared to the first workcell. This improvement is attributed to the improvement in the requirements specification process (not the verification process) that

took place as a result of the learning by the engineer who performed the requirements specification.

## 3.3 DMD's Tooling Development Cycle

We can model the development cycle for the production tooling DMD procures using the Standard Waterfall model (Section 2.2)–its five phases are: requirements, design, build, verify, and install (Figure 3.10). This section examines each of these five phases. It includes a statement regarding the purpose of each phase, what the major activities are, who is involved, and what is required for advancing to the next phase. A more in-depth discussion of DMD's requirements specification and verification process follows.

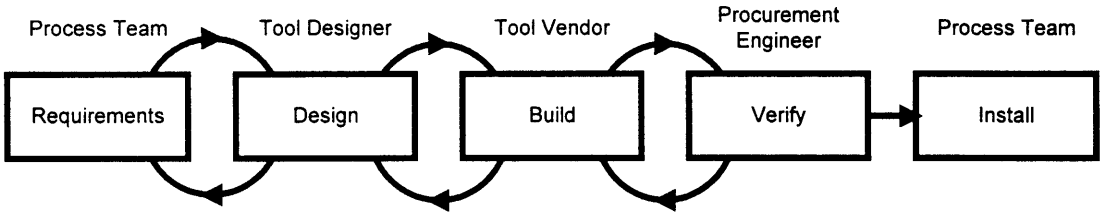



Figure 3.10 – The five phases of the tooling development cycle at Hewlett-Packard's DMD.

### 3.3.1 Requirements Phase

The purpose of the Requirements Phase is to deliver to the tool designer the document that specifies a tool's requirements. Figure 3.11 shows the Requirements Phase, expanded to show its input, output, and major activities. We will discuss each of these starting with the tool request.

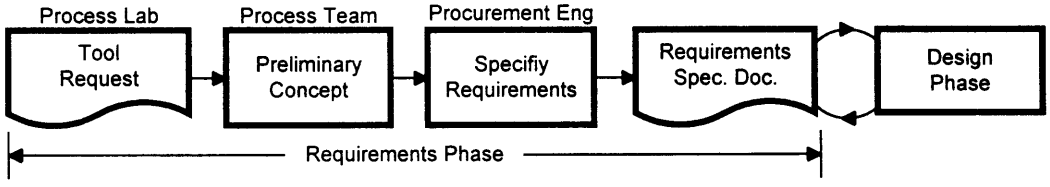



Figure 3.11 –The Requirements Phase in DMD's tooling development cycle.

The Requirements Phase officially starts when a tool request form is issued to a tooling procurement engineer by the Process Lab. The tool request form provides the tooling procurement engineer with the "black box" specification for the tool, and attached, is a complete description of the process that will use this tool. The next step in the Requirements Phase involves developing the tool's preliminary concept during a preliminary concept review–this meeting moves the tool specification out of its "black box" stage. Present at the preliminary concept review is a process team whose members include at least one process engineer, production engineer, maintenance technician, process operator, line supervisor, and the tooling procurement engineer[20]. The types of decisions that are made in the preliminary concept review include whether to automate, whether the tool should self-verify its work, and whether the tool should handle exceptions.

**Example: Preliminary Concept Review**

The Process Lab needs a tool that will center the disks (platters) around the spindle and then lock the disks into place by installing and securing the disk clamp. The tool must complete the operation in 60 seconds, be available 98 percent of the time, and have a process capability index (Cpk) of at least 1.33 on all critical dimensions and torques. This is the black box specification (partial). The Process Lab fills out a tool request form and sends it to Factory Engineering which assigns the task of procuring the tool to a tooling procurement engineer. A Preliminary Concept Review meeting is held in the presence of the process team. After considering many alternatives, the process team decides this tool should be a semi-automated workcell. A process operator will load and unload parts to and from the workcell and handle any exceptions that arise. The workcell will not inspect its own work.

Once a tool's preliminary concept has been developed, the process team communicates to the procurement engineer its needs. Each of the groups represented by the process team

---

[20] If the tool's vendor has already been selected, one of its engineers may also be part of the process team.

has a specific set of requirements that are most important to them, and all of these needs must be satisfied in the tool if it is to be successful. The tooling procurement engineer's role on the team is to understand these needs so that he or she can develop a requirements specification for the tool that ensures these needs are satisfied[21].

The last major activity in the Requirements Phase is *specify requirements*. Once the procurement engineer understands what the tool must do and what its need are, he or she can start developing the requirements specification for it. This involves much more than simply transcribing the needs the process team developed and calling it a requirements specification. Many of these needs are too general or qualitative to be meaningful in a requirements specification. Maintainability, for example, is a need, not a requirement specification. The procurement engineer specifies a set of requirements that will help ensure maintainability based on his or her own experience regarding the elements of maintainability. Upon completion of the tool's requirements specification the procurement engineer drafts the tool's requirements specification document. The requirements specification document is used for bidding purposes in vendor selection and, upon selection, communicates the requirements of the tool to its designer. The requirements phase is complete following the completion of the requirements specification document. The requirements specification document, however, is not a static document; we will see in the Design Phase that requirements can be enhanced, removed, or changed.

### 3.3.2 Design Phase

The purpose of the Design Phase (Figure 3.12) is to design a tool that conforms to the requirements specified by the requirements specification document. In most cases, the tool designer works for the vendor selected to build the tool. DMD does, however,

---

[21] The procurement engineer also plays a valuable role while the team is developing the tool's concept because of the expert knowledge he or she brings to the table.

employ a small staff of tool designers that are occasionally used for small design jobs. The first step in the Design Phase is to finish developing a tool's concept. The concept development process started back in the Requirements Phase during in the Preliminary Concept Review meeting. Since the tool designer has expert knowledge with respect to the particular aspects of the tool being procured, he or she is better qualified than DMD's process team to develop the finer details of a tool's concept. Once a concept has been developed, a concept review meeting is held between the process team and the tool designer. If the concept is unsatisfactory, it will require modifications. If the concept is satisfactory, additional requirements may be specified to reflect new details of the tool's design.

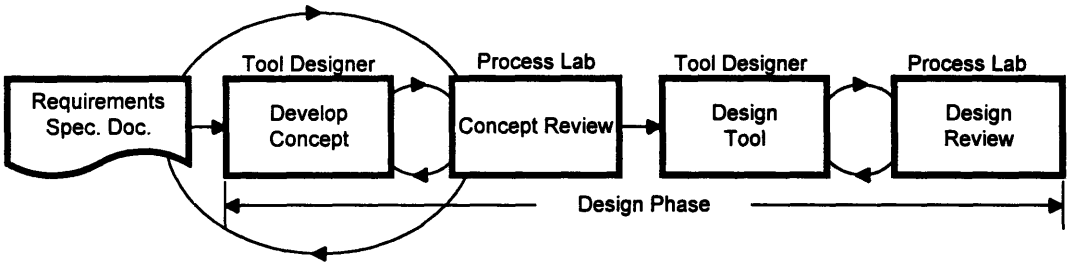



Figure 3.12 –The Design Phase in DMD's tooling development cycle.

**Example: Preliminary Concept Review**

In developing the concept of the disk center and clamp workcell, the tool designer decides to use a vision system to verify the concentricity of the disks to the media. During a concept review, the process team rejects the vision system concept because of the problems experienced with vision systems in the past. The tool designer reworks the workcell concept to use a laser measurement system. The process team approves the revised concept even though the cost is higher. Before the actual design process commences, an additional set of requirements are specified by the process team in response to the addition of a laser measurement system.

Once a tool's concept has been agreed upon and the requirements specification updated, the design process commences. During the process of design, the tooling procurement

engineer follows the progress of the tool's design and is available to help solve problems if they arise. When the tool designer feels a tool's design is complete, a design review meeting is held in the presence of the process team and the tool designer. A design review does not necessarily take place in a single meeting. In the case of the head-merge workcell, there were three design reviews in a one month period. In these design reviews there were 58 additional requirements specified and 35 action items generated[22]. These action items often involved gathering additional information regarding the requirements of the workcell (e.g., "prepare list of utility requirements", "send information on light tower standards"). Once a design has been approved by the process team, the development of a tool advances to the Build Phase.

### 3.3.3 Build Phase

The purpose of the build phase is to build a tool that is consistent with its design. During the Build Phase, the procurement engineer maintains a level of contact with the tool vendor that includes regular telephone conversations and at least two visits to the vendor's facility even if no problems are perceived or detected[23]. The purpose of these visits is to verify that the tool's construction is on schedule and to detect any potential problems. Unforeseen problems requiring design changes can sometimes arise. When the tool vendor notifies DMD that the tool is ready for verification, the development of a tool advances to the Verification Phase.

### 3.3.4 Verification Phase

The purpose of the verification phase is to ensure that the tool satisfactorily meets the requirements stated in the tool's requirement specification document. A verification team

---

[22] Another workcell, disk-stack, is included in these statistics because it was being built at the same time that the head-merge workcell was being built; the design reviews for both were held at the same time.

[23] This level of contact between procurement engineer and tooling vendor excludes simple production tools.

consisting of the procurement engineer, process engineer, and a maintenance technician visit the vendor's facility to carry out the verification activities. During the verification process, there are usually problems which must be resolved before a tool can be accepted for delivery. If a problem is sufficiently severe, the verification team may have to arrange another visit in order to give the vendor time to correct the problem. Once it has been determined by the verification team that a tool satisfactorily conforms to all of its requirements, the tool is accepted for delivery and it advances to the Installation Phase.

### 3.3.5 Installation Phase

The installation phase is the final phase in a tool's development cycle. The purpose of this phase is to integrate the tool into the manufacturing process and it requires the involvement of the entire process team. If there are problems with the tool, it is normal for the tool's vendor to assist DMD in correcting these problems; however, now that the tool is at DMD, it also consumes DMD's resources. After a tool has been installed and is determined to be in good working order, it becomes operational and exits the development lifecycle. Upon lifecycle exit, a process team will occasionally create a *lessons learned* document summarizing what it has learned throughout the development cycle of a tool.

### 3.3.6 Summary

This section described the events that take place in the development cycle of a production tool procured by DMD. The description of this process, however, is the ideal model. Not all of these events always take place nor are all of the members of the process team always present. This chapter continues with a closer look at DMD's requirements specification and verification process in terms of the activities defined by the requirements engineering profession: problems analysis, requirements documentation, and verification and validation.

## 3.4 Problem Analysis

Recall from Section 2.3 that problem analysis is the activity in the Requirements Phase concerned with learning about the problem a system is to solve, understanding the needs it must fulfill, and identifying all of the constraints acting on the system. DMD performs an activity similar to this in its Preliminary Concept Review. And while the goals may be the same as those discussed in Chapter 2, the methodologies are not. How this disparity in methodology impacts the effectiveness of DMD's requirements specification and verification process is the subject of this section.

### 3.4.1 Methodology

Currently, DMD does not have a formal method for the problem analysis part of the requirements specification process. In Section 2.3, we showed how requirements engineers use a process of hierarchical definition, allocation, and flowdown (1) to make sure the system under development will work with the larger system, and (2) to divide the system under development into smaller, more manageable sub-systems. Rather than attempt to describe DMD's more spontaneous approach to problem analysis, we will simply describe its effects.

#### 3.4.1.1 Systems Approach

Earlier, we expressed the concern DMD's management has for production tools that do not work well with the manufacturing process. The following two examples support their claims and emphasize the need for a systems approach to requirements specification:

1. A production tool was designed, built, and verified to meet the needs of a manufacturing process. When it was being installed on the production line, however, it was discovered that another part of the manufacturing process was physically interfering with the correct operation of the tool.

2. A production tool was designed, built, and verified to have a given cycle time. This specified requirement, however, did not take into account the time required for the other procedures in the manufacturing process using this tool. The time specified for the tool was equal to the time allotted to the entire manufacturing process. Although the tool met its specified cycle time requirement, it did not meet the needs of the manufacturing process.

#### 3.4.1.2 Organizational Learning

The opportunity to improve the requirements specifications of production tooling through organizational learning is great. Organizational learning is not the subject of most requirements engineering texts and articles because, typically, the requirements analyst is not the user or customer of the system. Therefore, emphasis is placed on using interviews to learn about the needs of the users and customers of a system. Interviews are necessary because, as an outsider, the analyst does not have an inherent understanding of what the system needs are (i.e., its requirements). In the case of the manufacturing organization, however, the analyst does belong to the same organization as the users and customers of the tool–in the case of DMD, the analyst is the tooling procurement engineer. Moreover, more often that not, production tools tend to evolve into better designs as opposed to being developed from radically new, and unproven designs. Given the preceding statements, we can argue that a manufacturer has a great deal to gain from having a process in place that supports organizational learning with respect to specifying and verifying the requirements of its production tooling. To show why, we return to our study of the head-merge case.

In the head-merge case we saw how the learning process enabled an engineer to improve the requirements specification for the second workcell. What the case did not mention, however, was that a similar workcell was procured by a different engineer for a head-merge process on another production line. The tool was procured from the same tooling vendor and in the time frame between the two workcells mentioned in the case.

Following delivery of this workcell, it had the same problems with its vision system and software as the first workcell mentioned in the case had. This suggests that while there is individual learning taking place at DMD, the degree to which organizational learning is taking place is questionable. In this case, if the engineer had known about the problems with the first workcell mentioned in the case, it is likely the same problems with the second workcell could have been avoided.

## 3.5 Requirements Documentation

From Section 2.4, the purpose of a requirements specification document is to communicate the requirements of a system, support verification and validation activities, and control the evolution of a system. Like the analyst who creates a requirements specification document for a system, DMD's procurement engineer creates a requirements specification document for each of the production tools DMD procures (some of the time). The difference lies in that DMD's requirements specification document serves only to communicate the requirements of the production tool to the tool designer (i.e., it is single purpose). Moreover, the documentation DMD produces for the requirements of each of its production tools does not always conform to the nine attributes listed in Section 2.4. How these disparities impact the effectiveness of DMD's requirements specification and verification process is the subject of this section.

### 3.5.1 Methodology

Like problem analysis, DMD does not currently have a formal methodology for creating requirements specification documents for its production tooling. The role of DMD's procurement engineer is to transform the requirements developed in the preliminary concept review into a set of verifiable requirements that will ensure the tool meets the needs of the production system. To do this, the procurement engineer relies heavily on his or her own internal knowledge of production tooling to create a set of verifiable requirements. The procurement engineer also uses his or her own "personal system" to document the requirements of a production tool. The result of these individually applied

learnings and methodologies is that the quality of documentation at DMD varies in terms of the nine attributes a requirements specification document must possess. For production tools of similar complexity and scope, the author identified requirement specification documents ranging from 4 to over 50 pages in length[24]. In those cases where requirements specification documents were weak, additional requirements were often supplied via telephone conversations and written correspondence. The problem with requirements that are generated this way is that (1) it increases the probability that inconsistencies will exist, (2) it makes it difficult, if not impossible, to create traceability linkages, and (3) it results in a loosely organized set of requirements which are difficult to follow. The remainder if this section will analyze the effectiveness of DMD's requirements documentation process in terms of the attributes from Section 2.4. [25]

#### 3.5.1.1 Correctness

As part of an experiment, the author took a requirements specification document for a production tool and elicited feedback from three members of its process team. This feedback process resulted in over 60 problems and questions being brought to the attention of the tooling procurement engineer. Of the problems, there were:

1. Six instances where the requirements specification would lead to certain design features known to have serious problems in the past at DMD.
2. Three instances where it was felt changes to the specification could improve the performance of the tool.
3. Three instances where the requirements specification would or could lead to a tool that jeopardizes the quality of a hard disk drive processed by it.
4. Two instances where the requirements specification failed to reflect the near future strategies of the Process Lab.

---

[24] This is only meaningful to the extent that it is an indicator of completeness and rigor.

[25] Not all nine attributes are analyzed; only those attributes where the greatest opportunities for improvement exist.

The requirements specification document used in this experiment was in its final stages of preparation just prior to being sent to the tool vendor. As a result of the experiment, the procurement schedule was allowed to slip so that the requirements specification could be reevaluated.

#### 3.5.1.2 Nonambiguity

The following examples are representative of how ambiguous the requirements DMD specifies for its production tooling can be:

1. Easy to operate.
2. Easy to understand every day error conditions.
3. Requires little operator intervention/input.
4. Easy to access.
5. Software shall be maintainable.
6. Easy to recover from errors.

These requirements come from actual requirements specification documents and by no means exist in isolation. The problem with these requirements is that the process used to verify them is extremely subjective (i.e., subject to multiple interpretations). For example, what is "easy to operate" for one person is not necessarily easy to operate for another person. These requirements cannot be objectively verified because they are too ambiguous.

#### 3.5.1.3 Verifiable

The following examples are representative of how unverifiable the requirements DMD specifies on its production tooling can be[26]:

1. The Mean Time Between Failures (MTBF) shall by no less than 100 hours.

---

[26] The actual data used in these examples do not accurately reflect DMD's specification.

2. The Mean Time To Repair (MTTR) shall be no greater than 10 minutes.

The rule for verifiability is that there must exist a cost effective and time effective method for determining whether a system satisfactorily meets the requirements stated in the requirements specification document. What is cost effective and time effective depend on the system being verified. DMD specifies a requirement for the Mean Time To Failure[27] (MTTF) for the disk drives it manufactures. In this case, time effective and cost effective mean time frames on the order of weeks, months, and years depending on the test, and costs that constitute a significant portion of the total development cost of a hard disk drive.[28] Through this extensive testing, DMD can statistically determine with an acceptable degree of confidence whether the hard disk drive meets its MTTF requirement. On average, the time frame for verifying the requirements of a production tool at DMD is between two and three days.[29] Given this time frame, the requirements specified for MTTR and MTBF cannot be satisfactorily verified at any cost.

#### 3.5.1.4 Traceable

DMD does not create traceability linkages for any of the requirements of its production tools. The evolutionary nature of production tooling at DMD, however, highlights the need for requirements traceability. As a tool continues to evolve, it inherits the requirements of its predecessors. Not knowing why a particular requirement exists for the preceding tool, it is often automatically transferred to the requirement specification of the new tool.[30] This requirement may be inappropriate or incorrect for the new tool. In

---

[27] MTTF for DMD's hard disk drives are on the order of 500,000 hours (over 50 years). Drives must be tested under normal and accelerated conditions for days, weeks, and years to verify this level of reliability.

[28] A hard disk drives differs from many products in that product testing continues even after it has been released to market; sometimes, well into its mature production phase.

[29] A single production tool at DMD can cost on the order of $250,000 and take months to design and build. Because of the complexity and uniqueness of these tools, it is very common for a development schedule to slip. This slippage "eats away" at the narrow window DMD has to install a tool and train personnel. This often places a great deal of pressure on personnel to expedite the verification process.

[30] If an engineer does not know why a requirement was specified, he or she is not likely to change it.

asking engineers why a particular requirement was specified for a production tool, it was common to hear "I do not know." Organizational change at DMD is fast-paced with engineers often changing positions every couple of years. It is too difficult to "chase down" every engineer who has ever specified a requirement for a production tool to ask them why they specified it.

Requirements traceability is also valuable during verification activities. If it is determined that a tool cannot meet a requirement, knowing why the requirement was specified will help determine the criticality of the nonconformance.

Finally, requirements traceability is valuable because changes to a high-level requirement will identify those lower level requirements which are affected by the change. A good example of an instance where lack of requirements traceability had an impact on a requirements specification at DMD follows:

> A requirements specification document was created for a new workcell based on a boilerplate requirements specification targeted for all of DMD's workcells. Included in the boilerplate specification was a requirement for the maximum length, width, and height of a workcell. No reason for these requirements where given (i.e., they were not traced back to another requirement). As a result, when the Process Lab made the decision to reduce factory floor space, the changes to the requirements for maximum length, width, and height were never reflected in the boilerplate document. It was latter determined that these changes needed to be made.

#### 3.5.1.5 Annotated

DMD does not annotate its requirements specifications. One of the engineers the author spoke with commented on how a tooling vendor had spent a great deal of effort trying to satisfy a requirement that was not critical to the needs and functionality of the tool. It was also commented that it would have been preferred that this effort was applied to another element of the tool's design. This type of incident is precisely the purpose for

annotating requirements.

## 3.6 DMD's Requirements Verification and Validation Process

This section examines DMD's verification and validation process. Recall from Section 2.5 that requirements *verification* is the process of determining whether the products of a given phase of the tooling development cycle fulfill the requirements established in the previous phase, and that *validation* is the process of ensuring that what was intended to be built corresponds to what is actually required. We start by examining the process DMD uses to verify the requirements of its production tooling.

### 3.6.1 Verification

Verifications activities for the production tools DMD procures are limited to the testing that occurs following the completion of a tool's build process.[31] This contrasts sharply with the process requirements analysts use to verify the requirements of a system. Recall from Section 2.5 that requirements verification is an ongoing activity that should occur at the end of each lifecycle phase (see Figure 1.3). DMD's lack of a rigorous verification cycle is most apparent in the Requirements Phase. The last step in the Requirements Phase is to create a requirements specification document that results in a tool being designed and built that meets the needs communicated to the tooling procurement engineer by the process team. However, there always exists the possibility that the procurement engineer can misinterpret or overlook one of these needs when drafting the requirements specification document. This is reason requirements analysts always verify the integrity of the requirements specification document prior to releasing it to design.

---

[31] DMD does not allow the development of a tool to proceed completely unchecked. Concept and design reviews require that the process team approve a tool's concept and design before further development can continue. These reviews, however, are not strictly verification activities because they lack the rigor that verifies the requirements of a previous phase have been met.

Section 3.5.1.1 showed how over 60 problems and questions were brought to the attention of the tooling procurement engineer when a verification step was introduced.

Errors in the requirements specification document are the worst types of errors because they are hidden errors. The effects of these errors will not be realized until the after the production tool has been delivered to the production floor. The best line of defense against these hidden errors is the presence of a requirements verification step just prior to releasing the requirements specification document to the tool designer. The verification step requires that the process team verify that its needs have been accurately communicated and are complete in the requirements specification document. In other words, there must be a feedback loop present in the requirements specification process (Figure 3.13). The purpose of this feedback is to reduce the amount of error in the requirements specification document.

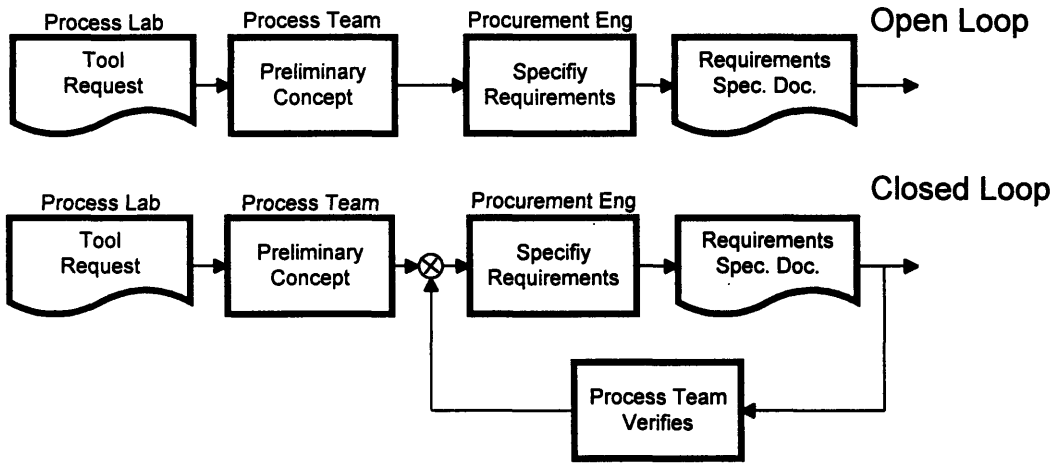



Figure 3.13 – DMD's Requirements Phase with and without feedback.

Another contrast between the methods of requirements analysts and DMD is the role the requirements specification document. Requirements analysts use the requirements specification document when verifying the requirements of a tool. Each requirement specified in the requirements specification document is methodically checked against the actual system. If all of the requirements are verifiable, the process should be relatively

straightforward and mundane. DMD, on the other hand, creates an independent checklist of requirements to verify. The requirements listed on this checklist do not necessarily map to all of the requirements communicated to the tool designer. When verifying the requirements of the tool, DMD's engineers systematically go through each of the requirements contained in the verification checklist. As with all of Hewlett-Packard's engineers, they excel technically and understand how to verify quantifiable requirements (e.g., the vision system shall be capable of measuring Dimension A with a precision of 0.001, the process capability index for the tool shall exceed 1.33). It is with the general, qualitative requirements that problems exist (e.g., the tool shall be maintainable). If maintainability is an important requirement, the verification team will include a maintenance technician. However, if there are no specific set of requirements to verify, all the maintenance technician can do is "check out the tool."

### 3.6.2 Validation

DMD does not currently have a validation process in place for its production tooling. The closest thing that they have to a validation process is the creation of a *lessons learned* document. This document is created following the installation of a new production tool and it contains the lessons that the process team has learned throughout the tool's development cycle. By having these lessons learned documents available for future reference, DMD can leverage from its successes and avoid repeating mistakes. The lessons learned documentation process, however, is not a required process step in the procurement of a new production tool and more often that not, it is not generated. Moreover, when lesson learned documents are created, they stay with the process team that procured the tool (i.e., there is no system in place that organizes these lessons learned and makes them available to the entire manufacturing organization).

DMD's lessons learned process, however, is not a validation process. The validation process for a production tool is a two step process. It ensures that the tool conforms to what is actually required and is concerned with the completeness, consistency, and

correctness of the requirements specification. After a tool has been delivered to DMD for integration into the manufacturing process, it quickly becomes apparent whether the tool built corresponds to what was actually required (i.e., the needs of the manufacturing process). Entries into the lessons learned document are made whenever the tool fails to satisfy its actual requirements (e.g., maintainability, usability, reliability). What the lessons learned process fails to do, however, is go back to the tool's actual requirements specification document and explain why it failed to deliver a tool to the production floor that met all of its actual requirements (i.e., the lessons learned process is not concerned with the completeness, consistency, and correctness of the requirements specification document).

## 3.7 Conclusions

In the case study presented early in this chapter, we saw how an engineer was able to improve the quality of a requirements specification, and thus the tool itself, based on what was learned from previous experience. We also learned that while individual learning took place on the part of one engineer, DMD failed to learn form this experience as an organization. If requirements specification is a knowledge based discipline, and the author believes it is, then there is a great deal to be gained from creating a requirements specification based on the knowledge of the organization rather than on the knowledge of the individual. These gains not only include better requirements specification, but also help overcome the effects of organizational change. The author feels DMD can greatly improve its requirements specification process by putting a process in place that supports organizational learning.

In this chapter, we also saw the importance of procuring a production tool that meets the needs of the manufacturing process (Section 3.4.1.1). The needs of the manufacturing process are unarguably the most important requirements that will be imposed upon a tool since these needs define a tool's highest level requirements. With no process for allocating the needs of a manufacturing process among its subsystems, DMD has run into problems in the past where production tools appear to function properly outside the

constraints of the production system but fail to work within the system. The author feels DMD can greatly improve its requirements specification process by putting a process in place that supports the allocation of system requirements to production tooling.

Finally, we saw how DMD limits the verification activities of its production tooling to the test activities following the build process. There are two problems with this approach. First, it is reactive. By waiting until the tool is already built to start verifying that it meets its requirements, DMD loses the opportunity to catch errors early in the development cycle when errors are relatively inexpensive and easy to repair. The second problem with DMD's approach are is that it obscures reality. It is easy for DMD's management to say "we will not allow a production tool on the manufacturing floor until all of its requirements have been met." However, when reality creeps into the equation and DMD must decide whether to use a production tool that works, but does not meet all of it requirements, or whether to postpone production until the tool meets its requirements, the latter decision inevitably is made. With product market lives lasting only a little over a year, DMD cannot afford to spend time fixing all the errors in a production tool. Therefore, the author feels DMD can greatly benefit from adopting a more proactive approach to requirements specification that is consistent with the methods of the requirements engineering profession. In this way, DMD will not have to wait until the final hour to discover all of the errors a production tool contains.

## 3.8 Summary

In this chapter, we described a process used by Hewlett-Packard's DMD to specify and verify the requirements of its production tooling. Inadequacies in this process often lead to problematic production tools can have a negative effect on the manufacturing process. In the case of the head-merge workcell, tooling errors accounted for over two weeks of delay before it could be made operational. The head-merge case helps to confirm the belief held by DMD's management that problematic production tools are causing production ramp delays for new product lines. It was concluded the opportunities for improvement in DMD's process are greatest if the following steps are taken: (1) use

organizational learning to improve the quality of requirements specifications, (2) understand the needs of the manufacturing process prior to specifying the requirements of the production tool, and (3) make requirements verification an integral part of the tooling development cycle as opposed to something that happens only at the end. In the chapters that follow, we will develop requirements specification and verification process that addresses these issues.

# 4

# Framework and Tools for a Learning Organization

This chapter develops a set of tools and methods that will be used in the requirements specification and verification process proposed in Chapter 5.

## 4.1 Introduction

In Chapter 5, a new process for specifying and verifying production tooling will be described. Before this description is possible, however, the set of tools and methods used by this new process must be developed. They are aimed at creating a framework for the requirements specification process, and at creating a knowledge base that will provide the procurement engineer with the knowledge required to specify verifiable requirements. A branch of classification theory called object-oriented analysis (OOA) will create the framework, and a tool called the Quality Factors Matrix (QFM) will be one of two indices accessing knowledge base. Both OOA and the QFM support organizational learning. We begin our discussion with the OOA method.

## 4.2 An Object-Oriented Approach to Requirements Specification

In our analysis of DMD's requirements specification process we saw the need for organizational learning. The new requirements specification process, therefore, will

emphasize organizational learning as a means to developing good requirements specifications. Hammond (1993) gives eight principles for effective learning. One of these principles has to do with the effects of prior knowledge.

> "An expert seems to be able to pick up new knowledge quite effortlessly. Superior remembering by experts is attributed to the fact that they develop highly sophisticated frameworks or *schemas* which enable new facts to be slotted into the existing structure and immediately elaborated and enriched; the framework and the elaborations of the newly acquired knowledge together constitute a highly effective means for retrieving information at a latter date (Hammond, 1993)."

Developing a good requirements specification requires knowledge of the system being specified. This knowledge enables the procurement engineer to describe precisely the attributes a system must possess. We want to create an organization that, through learning, has expert knowledge. Hammond states, that to do this, there must exist a framework that supports learning. Object orientation is a branch of classification theory that we will use to develop this framework.

### 4.2.1 Object Orientation

Object orientation (OO) is a term often associated with software development. The Simula programming language made object-oriented programming (OOP) practical in 1967 and today's popularity of C++ has helped make *object-oriented* the IT buzz-word of the 1990's. OO, however, is not the exclusive domain of the IT world. OO is a philosophy used for system development that attempts to model the way humans think. In fact, James Martin and James Odell (1995) call object orientation "an index for knowledge." In fact, it will be the OO framework that develops in this chapter that serves as the second index to the requirements knowledge base (the QFM is the other index). Jacobson *et al* (1994) define OO as follows:

> "Object orientation is a special approach to the construction of models of complex systems, in which a complex system, consisting of a large number of occurrences,

is regarded as a set of objects. The relations between these objects are seen as associations between objects; their properties are attributes of these objects. In addition the occurrences can have static as well as dynamic characteristics (Jacobson *et al*, 1995, p. 45)."

### 4.2.2 Object-Oriented Analysis (OOA)

This section describes the principles of OOA in the context of production tooling requirements. The advantages of using this approach will become clear as we proceed. The overriding advantage of using OOA is the framework for building the knowledge base that we propose installing to support organizational learning. OOA will allow us to define and communicate the requirements of a production tool within the three basic elements of human organization: object and attributes, classification structure, and assembly structure (Coad and Yourdon, 1989). The notation and methods used in OOA are built upon these three basic elements.

#### 4.2.2.1 Object and Attributes

One mechanism humans use to collect and store information is classification. OOA uses the *class* to mimic this human process. On the production floor, for example, we might classify all of the objects around us into one of three class structures: person, part, and tool. Loy (1990) defines *object*:

"An object is an entity defined by a set of common attributes, and the services or operations associated with it. Whatever form of system requirements that is provided by the user or client can be used as a starting point for this task. Objects can be found among devices that the system interacts with (e.g., sensors), other systems that interface with the system under study, organizational units (departments, divisions, etc.), and things that must be remembered over time (e.g., details about events occurring in the system's environment) (Loy, 1990, p. 295)."

An easy way to think of the class-object relationship is to define a class called person. The definition used here calls for two attributes to be defined for each instance of the

class: species and sex. The first attribute, species, we always know: *homo sapien* (i.e., it is a static attribute for person). The second attribute sex, must be determined for each instance of the class person. An instance of a class is called an object and it always has its own unique identifier (e.g., Peter, Paul, and Mary). Having sufficiently defined *class* and *object*, we continue by casting these concepts into the context of a specifying production tooling requirements.

In the manufacturing environment, there will exist different classifications of production tooling (e.g., workcell, grinding machine). For each class of tool, there will exist a set of requirements that are common to all the tools in that class (e.g., safety, uptime). Within a class, the tools are distinguished through their differences. The manufacturer who must specify production tooling requirements can use this class concept to its advantage. The advantages are shown in the following example:

**Example: Creating A Production Tool Class**

A manufacturer has learned that all of its production tools have three requirements in common with each other: they are all required to conform to the company's standards for (1) safety and (2) ergonomic design, and (3) a maximum weight must be specified. These requirements apply to all of the manufacturer's production tools regardless of whether a tool is a torque driver, workcell, or an assembly fixture. Rather than draft an entirely new requirements specification document for tool that is procured, the manufacturer decides to create a requirements specification class called production_tool;[32] The class definition for production_tool given by Figure 4.1.

---

[32] This definition uses the keyword **constructor** to tag requirements which are used to initialize the class.

```
class production_tool:
        corporate safety specification
        corporate ergonomic specification
constructor:
        weight      pounds
```

Figure 4.1 – Definition of the specification class production_tool.

In the preceding example, all of the attributes of the class production_tool are tooling requirements.[33] Two of these attributes, safety and ergonomics, are the same for all instances of production_tool. For these requirements, a manager would be correct in stating "All of our production tools are required to conform to our company standards for safety and ergonomics." The last attribute, weight, is the only same for all production tools in terms of the requirement type. The weight requirement differs from the first two requirements in that it must be assigned a value for each instance of the class. In this case, a manager would be correct in stating "We require that a maximum weight be specified for all of our production tools." In this example, the weight of a production tool is specified for all instances of production_tool. Figure 4.2 shows the production_tool class and three instances of it: tool "a", tool "b", and tool "c."

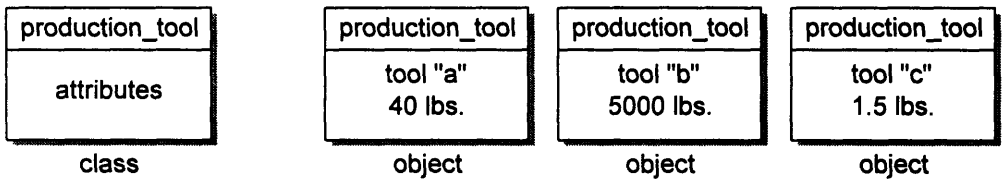



Figure 4.2 – The production_tool class and three instances of it (a, b, and c).

### 4.2.2.2 Classification Structure

We have just shown how a class can be created to categorize production tools of a similar type. In our last example we created a class whose members included all production tools

[33] Since the subject of this thesis is requirements specification, this also holds true for all of the classification structures developed hereinafter.

and whose attributes were requirements. This classification, however, is too general to be of any value by itself. This time we will be more specific and define a class named fixture instead–all tools which can be classified as a fixture are members of it. We also define the classifications hand_tool, press, and workcell (see Figure 4.3). There are now four specialized production tooling classifications instead of a single, generic classification. This is exactly what Hewlett-Packard's DMD has done with their boilerplate workcell specification (Section 3.2).

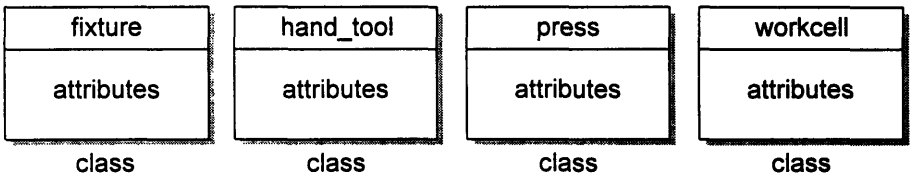



Figure 4.3 – Specification classes created for four different types of a production tool.

There are two advantages to developing specialized requirements specifications for the various classifications of production tools an organization regularly procures; they are:

1. Savings in time for the requirements specification process. Once a specification for a particular type of tool has been created, it can be reused the next time a tool of a similar type is having its requirements specified.

2. Reduction in the amount of error and inconsistency in a specification over time. As a specification is used time and time again, errors and inconsistencies will be detected and corrected.

DMD has developed some very specific requirements for the workcells they procure in its boilerplate specification. Many of the requirements contained in the workcell boilerplate, however, are very different and inappropriate for the requirements of a hand tool (e.g., the requirements pertaining to controller, electrical system, and user interface). Using this approach, DMD must develop a complete, yet generic requirements specification for each specialized class of production tool it plans to use. This is not practical. The time and effort required to develop and maintain a requirements specification for every type of tool DMD procures could easily consume the time of its entire procurement staff. We are in

caught in a dilemma, then, between the benefits and costs of creating specialized requirements specifications. We will now continue with our discussion of OOA and show how it can resolve this dilemma.

The classifications fixture, hand_tool, press, and workcell all have something in common: they are all production tools. OOA takes advantage of this commonality through a principle called *inheritance*. Using the *child-parent* relationship, we can say that fixture, hand_tool, press, and workcell are all children of production_tool (the parent). These children inherit the attributes of their parent and also have their own unique set of requirements that distinguishes them from each other. Using inheritance we can create very specialized classes that are built upon more generic classes. Figure 4.4 shows an example class hierarchy based on production_tool.

Appendix A shows some illustrative examples of the complete hierarchies that a hard disk drive manufacturer (Figure A.1) and an aircraft engine manufacturer (Figure A.2) might use.

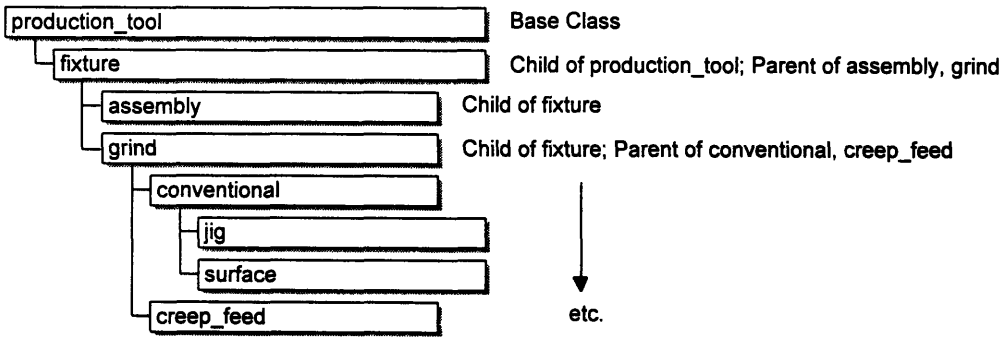



Figure 4.4 – Class hierarchy developed from the base class, production_tool.

Inheritance allows us to retain the advantages of having specialized requirements specifications while at the same time it overcomes its drawbacks in terms of the time and effort required to develop and maintain a requirements specification for every type of production tool that a manufacturer uses. To see why we turn our attention back to Figure 4.4. We want to create requirements specifications for a jig grinding

fixture and one for a surface grinding fixture. There are two courses of action available to us: (1) we can create two complete requirements specifications for each fixture type that contain all of the respective requirements or (2) we can inherit the requirements specification of a conventional grinding fixture. By inheriting the conventional grinding fixture specification we greatly reduce the amount of effort necessary to create a new specification. We need only specify those requirements that are specific to the differences between a conventional grinding fixture and a jig grind fixture or a conventional grinding fixture and a surface grinding fixture. This principle extends all the way back to the base class of a classification hierarchy (e.g., jig inherits from conventional, conventional inherits from grind, grind inherits from fixture, and fixture inherits from production_tool). In addition to the benefits we have just mentioned, there are two other benefits of inheritance–one of which prompted the use of OOA in the first place.

1. It creates the framework Hammond referred to at the beginning of this chapter. This framework will enable new facts (requirements) to be slotted into an existing structure as well as a highly effective means for retrieving this information at a latter date. We will use this framework in Chapter 5 to help create a learning organization. This is possible because the structure of this framework maps the way people think about tools, making it a convenient index for storing and retrieving information.

2. It eases the maintenance of specifications. If a requirement is added to, changed, or removed from the class production_tool, it propagates through all the tool classifications. Using a non-object based approach would require updating all of the specialized specifications individually (assuming they existed which is unlikely because of the amount of effort required to create them).

#### 4.2.2.3 Assembling a New Specification Class Using Multiple Inheritance

By itself, inheritance can be quite restrictive. To truly benefit from inheritance, there must exist a parent class that provides us with the majority of the specification for the new specification we are trying to create. We can get around this restriction with an OO feature known as multiple inheritance. In OOA, a class can inherit requirements from more than just its parent. It can inherit requirements from any class. Figure 4.5 shows the requirements specification for a head-merge workcell which has inherited its requirements from six different requirements specification classes. Four of these (user_interface, electrical_system, pneumatic_system, and vision_system) have no lineage back to production_tool whatsoever (see Appendix A).[34] Using multiple inheritance, we are able to construct a new specification by simply "plugging" in the specification components that we need. As a result, multiple inheritance gives us a great deal of flexibility in the type of tool for which a requirements specification must be created. Even if a particular type of tool has never been procured before (i.e., it is a radically new design), multiple inheritance makes it possible to piece together a good part of its requirements specification.

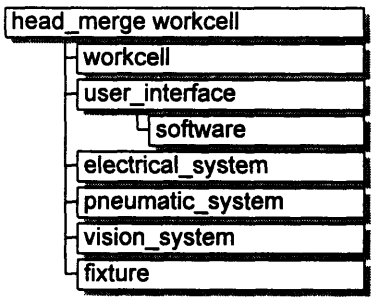



Figure 4.5 – New class definition developed using multiple inheritance.

[34] Since an electrical system, pneumatic system, fixturing, and user interface are common to all workcells, these class definitions would have normally been *hidden* in the workcell class definition; they were shown here just for illustrative purposes.

#### 4.2.2.4 Summary of OOA

To enter into a discussion about all of the methodologies and benefits of OOA is beyond the scope of this thesis.[35] We have only scratched the surface of OOA by presenting only those concepts that will enable us to show how OOA can be used to specify production tooling requirements. We will show how to apply these concepts in Chapter 5 where we create a new requirements specification and verification process. This will include additional discussions on the benefits of OOA when we present the examples that make them most apparent.

## 4.3 Creating a Knowledge Base Using Quality Factors

In the last section we showed how an OO requirements specification hierarchy can be used to speed the development of a production tool's requirements specification. Another benefit of this specification hierarchy is that it creates a framework for the storage and retrieval of knowledge. We will use this framework in conjunction with the Quality Factors Matrix (QFM) that is developed in this chapter to create an efficient requirements specification knowledge base that can be used for production tooling.

### 4.3.1 The Quality Factors Matrix (QFM)

In their paper Specifying Software Quality Requirements with Metrics, Steven Keller and Laurence Kahn (1990) introduce the concept of a Quality Factors Matrix (QFM) to specify requirements. Their QFM shows the relationship between consumer needs and technical attributes of a software product. Consumer needs are called "quality factors" and the technical attributes are called *criteria* or *quality sub-factors*. We shall use the term quality sub-factors. A QFM can be used to specify tooling requirements if we think of a tool's quality factors as the needs of the manufacturing organization and the quality

---

[35] To learn more about object-oriented methods and their benefits, the author highly recommends Martin and Odell (1995).

sub-factors as the technical attributes of the tool. Keller and Kahn motivate our use of the QFM as a requirements specification tool:

> ". . . customers often express their needs in general, qualitative terms, such as reliability, maintainability, portability, efficiency, etc. To affect the outcome of a development effort or to determine objectively whether a given product is satisfactory, they need to define precisely the specific attributes they want the [system] to possess (Keller and Kahn, 1990, p. 145)."

This is precisely the situation DMD finds itself with respect to verifying the requirements of its production tooling. Requirements specifications can be so general or qualitative, it becomes difficult or impossible to determine objectively whether a tool satisfactorily meets the needs of a manufacturing process.

Keller and Kahn's QFM shows the relationship between customer needs and a software's technical attributes. Similarly, we create a QFM that shows the relationship between a manufacturer's tooling needs and a tool's technical attributes. Figure 4.6 shows a QFM created by the author for production tooling. It is a two-dimensional matrix with quality factors in the columns and quality sub-factors in the rows. In the case of DMD, using a QFM will enable the tooling procurement engineer to take the high-level requirements of the manufacturing process and break them down into elements that can be more easily quantified. To see how a QFM works, we turn our attention to maintainability. DMD's managers and engineers have a difficult time quantifying it. A QFM can help. Again, turning our attention to Figure 4.6, the author has identified five quality sub-factors of maintainability: consistency, modularity, self-descriptiveness, simplicity, and visibility (see Figure 4.7). These quality sub-factors of maintainability are more quantifiable than the quality factor, maintainability. A QFM is an excellent vehicle for capturing knowledge because it breaks needs down into not too specific terms. Ask an organization to state specifically what makes a tool maintainable and they have a difficult time answering.[36] Ask the same organization, instead, what the elements of maintainability

---

[36] This excludes maintenance; they have answers but they are loosely organized. The QFM helps organize this information which will be used in the quality factor recipes discussed in the next section.

are, and they can answer this question.[37] A QFM also increases the visibility of quality sub-factors that might otherwise be overlooked. A good example is simplicity of design. Some of the managers, engineers, and maintenance personnel the author interviewed identified "simplicity" of design as an important element in delivering a production tool that is reliable, usable, and maintainable[38]. Simplicity of design, however, is not addressed in any of the requirements specification documents reviewed by the author. If DMD had a QFM similar to the one shown in Figure 4.6, it would be evident that simplicity of design is a tooling attribute that needs to be addressed in the requirements specifications it develops for its production tools.

| | Quality Factors | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Quality Sub-Factors** | Quality | Reliability | Throughput | Usability | Maintainability | Verifiability | Expandability | Flexibility |
| Accuracy | ● | ● | | | | | | |
| Consistent Design | | | | ● | ● | | | |
| Cycle Time | | | ● | | | | | |
| Ergonomic Design | | | | | | | | |
| Exception Handling | | ● | ● | | | | | |
| Generality | | | | | | | ● | ● |
| Labor Content | | | | | | | | |
| Modularity | | | | | ● | ● | ● | ● |
| Precision | ● | ● | | | | | | |
| Process Capability | ● | | | | | | | |
| Quality Components | | ● | | | | | | |
| Self-Descriptiveness | | | | ● | ● | ● | ● | ● |
| Setup Time | | | ● | | | | | |
| Simplicity | | ● | | ● | ● | ● | ● | ● |
| Training | | | | ● | | | | |
| Visibility | | | | | ● | ● | | |

Figure 4.6 – Quality Factors Matrix (QFM) for production tooling showing the needs of the manufacturing process (Qualify Factors) versus a tool's technical attributes (Quality Sub-Factors).

[37] This statement is based on the author's interviews with engineers and managers at DMD. It is also supported by the findings of Keller and Kahn (1990).

[38] The author also feels simplicity of design is an important attribute leading to the acquisition concerns: verifiability, expandability, and flexibility.

A QFM should be designed to reflect each organization's needs. It must not, however, be cast in stone once it has been created. Figure 4.6 is an example of a QFM the author developed. Upon further consideration, it is clear that another quality factor should be present: safety. At DMD, there is a new interest in creating flexible production lines capable of manufacturing multiple generations of a given product line. This is a change in strategy for DMD. In the past, DMD has created dedicated production lines for each new product line (i.e., DMD coupled product and process development). This change in strategy creates a new set of quality factors for DMD's production tools (e.g., flexibility, expandability) that were not present only a couple of years ago.

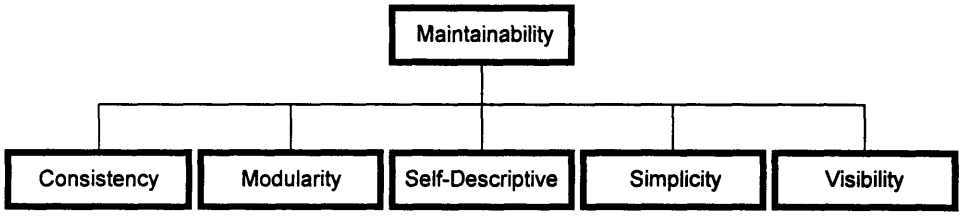



Figure 4.7 – The quality factor 'maintainability' broken down into its five sub-factors.

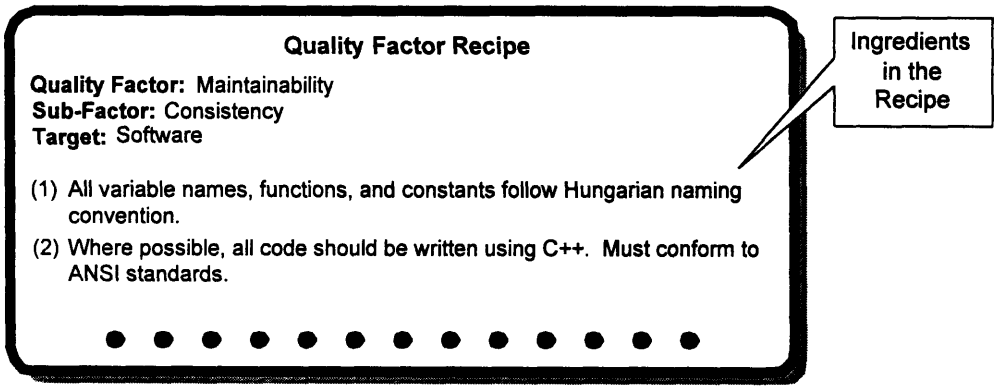



Figure 4.8 – Example QFR for software maintainability.

### 4.3.2 The Quality Factor Recipe (QFR)

This section introduces the concept of a Quality Factor Recipe (QFR). It picks up where the QFM left off in terms of creating a set of verifiable requirements that support a high-level quality factor. As an engineer gains experience specifying tooling requirements, the

elements of knowledge that have been learned through this experience can be funneled into an appropriate QFR. Figure 4.8 shows an example of an QFR that specifies requirements targeting software maintainability through consistency of code.

In the case of the QFR shown in Figure 4.8, maintainability is the quality factor, and consistency is the quality sub-factor. These two factors cross-reference the element of an QFM that points to the QFR shown in Figure 4.8. The requirements specified by this QFR are verifiable.

The QFR in Figure 4.8 has two ingredients in it. It did not start out this way and there is no reason it should continue to stay this way. It should continue to grow and change as the engineer continues to gain experience and the organization develops standards[39]. If the engineer maintains a file of these recipe cards, he or she will have effectively created his or her own personal knowledge base. This process can be taken one step further. If the QFRs are developed and maintained at an organizational level instead of a personal level, the organization will have effectively created its own organizational knowledge base. The following are three benefits the manufacturing organization will realize from this organizational learning[40]:

1. It overcomes the effects of organizational change. Instead of engineers taking their knowledge away with them when they leave the organization, they leave a "copy" of it behind in the knowledge base.
2. It creates shared vision. Because the entire organization is working from the same knowledge base, it is much easier to create a shared vision about what the organization's tooling needs are and how to acquire them.

---

[39] The first ingredient is the result of experience gained from the frustration of trying to maintain a system that did not impose naming conventions whereas the second ingredient is an internal standard of the manufacturing organization.

[40] We will see some additional benefits in Chapter 5 when we show how to use SGML as a requirements specification tool.

3. It eases the process of developing a verifiable requirements specification document. Because the procurement engineer has access to a wealth of verifiable requirements, he or she does not have to rely only on his or her own personal knowledge to create a verifiable requirements specification document.

For this QFM and QFR concept to work, there must be a requirements specification and verification process in place that supports its use. It is straightforward to see that if the QFRs are not updated and maintained, they are of no value. One of the process steps in the requirements specification and verification process that develops in the next chapter will be to add information to the QFRs based on lessons learned.

## 4.4 Using the OO Framework as a Knowledge Index

If we dismiss all of the benefits of using OOA to specify requirements that we have discussed so far, we can still motivate its use because of the framework it creates for indexing knowledge. In the last section, we showed how a QFM can be used to cross-reference and point to the knowledge of the organization that is contained in a QFR. The QFR shown in Figure 4.8, however, is of no value to an engineer who wants to know about the elements of maintainability for an electrical system. One solution to this problem is to have one, all encompassing, QFR that specifies the elements of maintainability for every type of system imaginable. This solution is unmanageable and equally unusable. Users of the system would simply get lost in the plethora of requirements specifications. The other solution is the use an OO specification hierarchy as an index for retrieving knowledge. We will choose this solution. Using the classification hierarchy, QFMs and QFRs can be created for each of the class definitions in the hierarchy. Once this is done, an engineer can very quickly focus on his or her area of interest. Figure 4.9 shows how each of the class specifications inherited by a head-merge workcell has its own QFM. If the procurements engineer wants to specify a set of requirements that are specific to the workcell, he or she has access to a knowledge base

that maps itself to the workcell. The result is a very efficient information retrieval system.

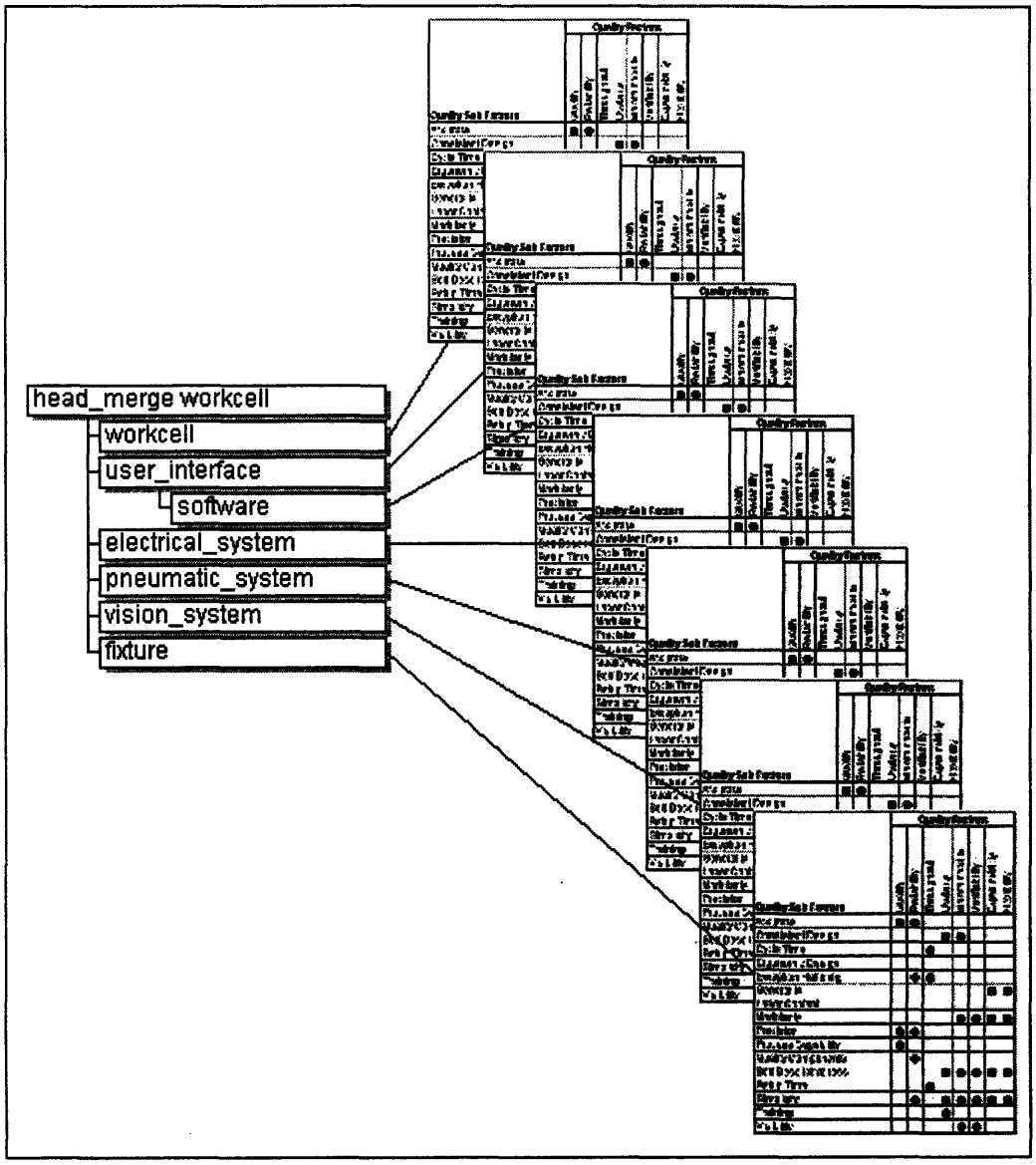



Figure 4.9 - Using the specification classes as an index to the knowledge of the organization.

## 4.5 Summary

This chapter developed the set of tools and methods that are used by the requirements specification and verification process in the next chapter. These tools and methods included OOA, the QFM, and the QFR, and they all work together to create a highly

effective means of storing and retrieving information pertinent to specifying verifiable requirements based on the learning of the organization. The actual information is contained in the QFR and is indexed by the QFM and the OO specification framework. Organizational learning, however, was only one of the benefits of OOA. By creating a requirements classification hierarchy, we also create a method by which a tooling procurement engineer can quickly assemble a requirements specification for a production tool.

# 5

# Specifying and Verifying Production Tooling Requirements: A New Process

This chapter proposes a new process for specifying and verifying production tooling requirements.

## 5.1 Introduction

This chapter describes a new process for specifying and verifying production tooling requirements. We have only one goal for this new process: deliver to the manufacturing floor, production tools that meet the needs of the manufacturing process. In Chapters 2 through 4, we laid the groundwork for this new process. Before getting started, however, a brief recap of the discussions up until this point is in order.

In Chapter 2, we drew attention to the methodologies used by the requirements engineering profession that are also used by the new process. Specifically, requirements allocation and requirements flowdown are used to help ensure that a production tool works with its owning manufacturing process. We also discussed the purpose of a requirements specification document and the attributes that it must possess. Finally, requirements engineers use a Verification and Validation (V&V) cycle that helps to ensure the correctness of the system as it advances through its lifecycle.

In Chapter 3, we described a process used by Hewlett-Packard's Disk Memory Division (DMD) to specify and verify the requirements of its production tooling. Based on this discussion, we want a requirements specification and verification process that (1) supports organizational learning, (2) ensures the needs of the manufacturing process are present in the production tool, and (3) is proactive in its approach to requirements verification.

Finally, Chapter 4 developed a set of tools and methodologies that will support the new process. These tools and methodologies include the Quality Factors Matrix (QFM) and Object-Oriented Analysis (OOA).

## 5.2 Process Overview

The process flow for the requirements specification and verification process described by this chapter is shown in Figure 5.1; it is based on the process flow DMD currently uses for its requirements specification and verification process which is shown in Figure 5.2. By using DMD's process as a basis for the design of the new process, we are leveraging a process that is already known to work[41]. There are, however, some noteworthy differences between these two processes. The new process places increased emphasis on organizational learning (knowledge building), verification and validation, and systems analysis. These are the areas where the author feels DMD's process is weakest (Section 3.7). The emphasis on organizational learning has its effect on the development lifecycle models used to describe each process. Recall that DMD's process is best modeled using the Standard Waterfall model which is show in the context of a production tool in Figure 5.3. The new process, however, is best described using the Incremental Development model (Figure 5.4) which uses feedback from the users of operational tooling to affect the outcome of a tooling development effort. This feedback is communicated by the QFRs.

---

[41] DMD does, after all, procure production tools for a production system that allows it to be a successful manufacturer of hard disk drives.

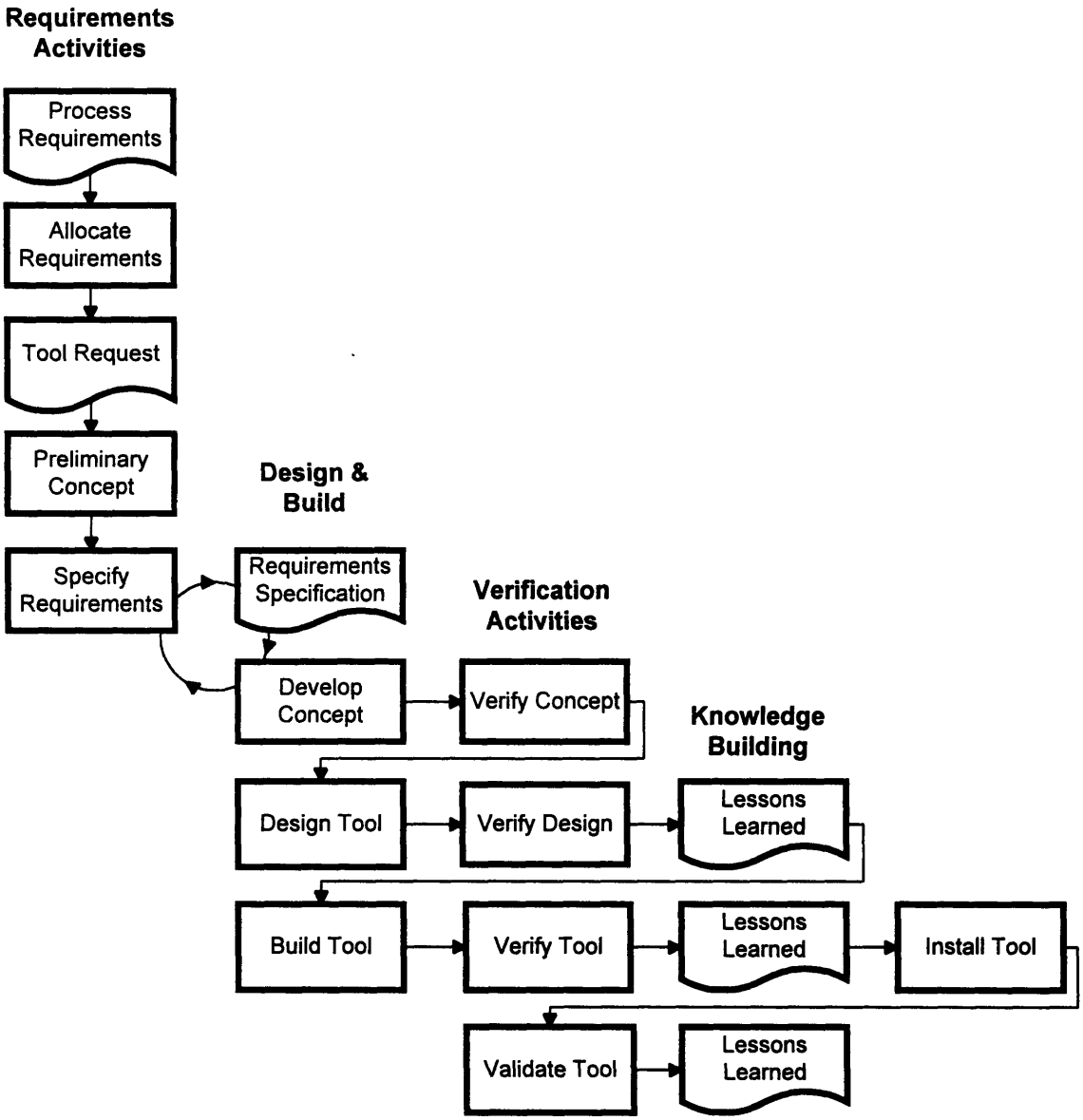



Figure 5.1 – The process flow for a new requirements specification and verification process.

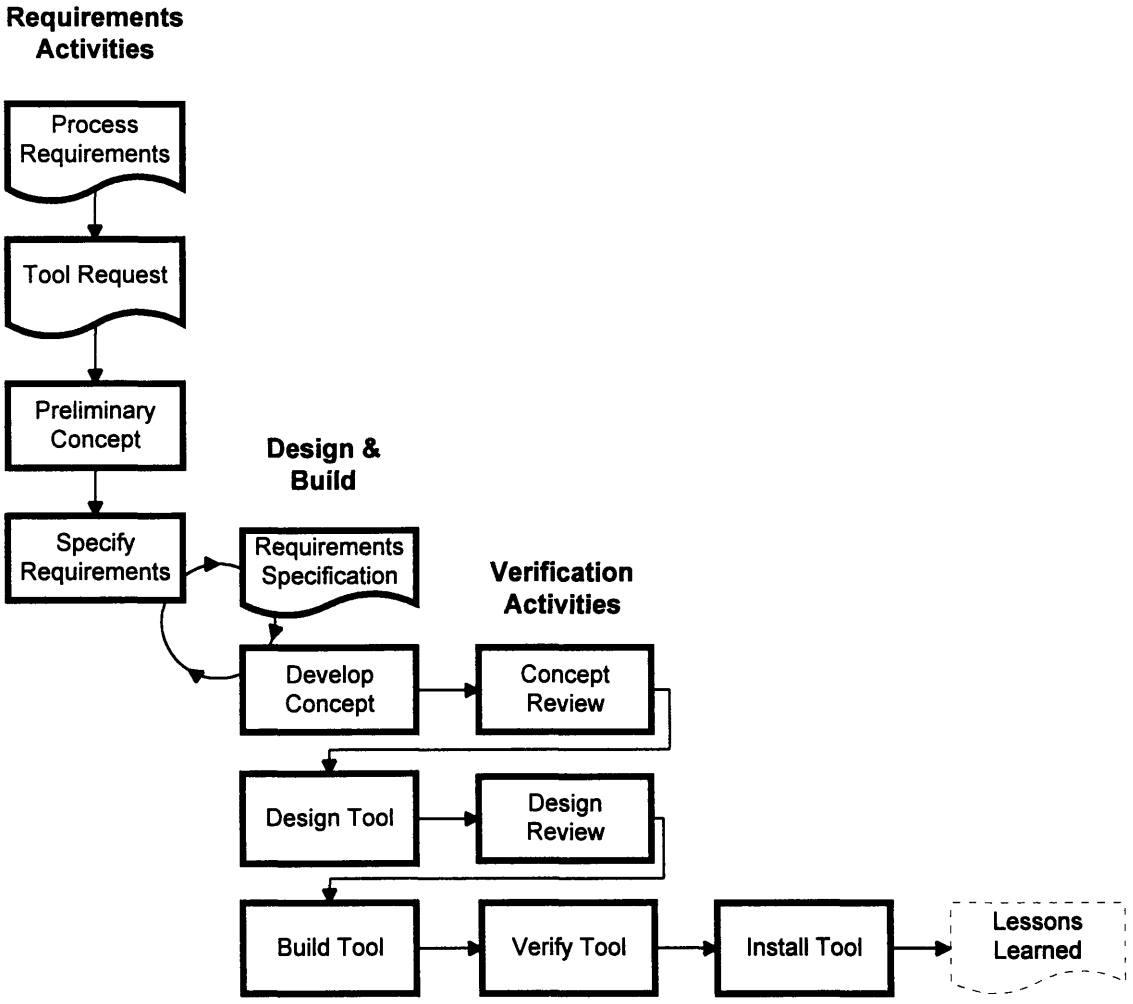



Figure 5.2 – The process flow for DMD's current requirements specification and verification process.

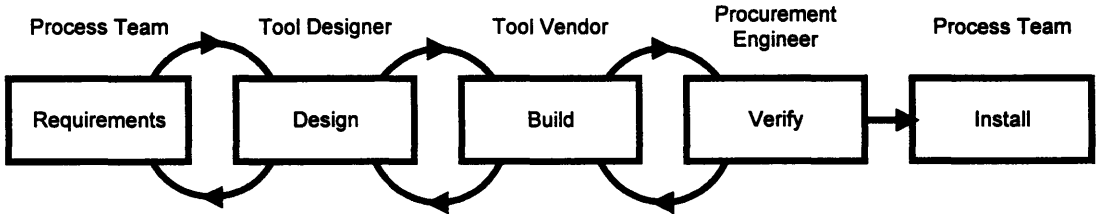



Figure 5.3 – The five phases of DMD's tooling development cycle.

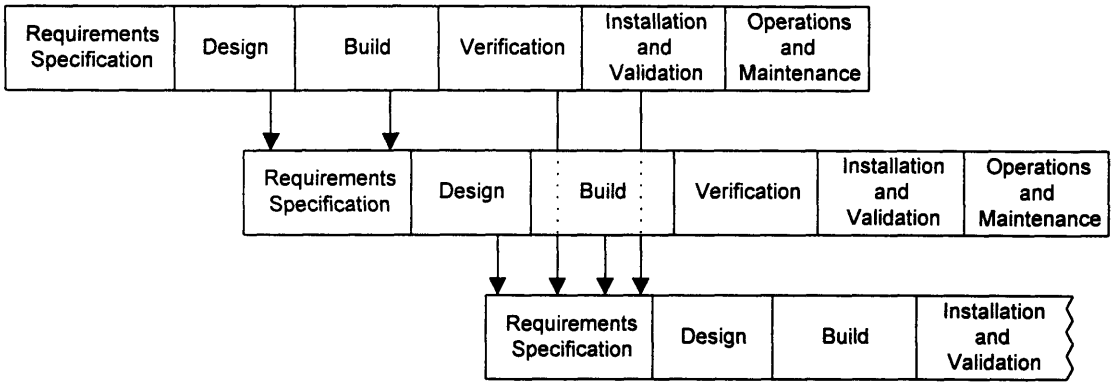



Figure 5.4 – The five phase Incremental Development model for production tooling.

## 5.3 The New Process: Step by Step.

This section describes a process for specifying and verifying production tooling requirements. During the description of this process, we will use the same functional organizations and job titles as were used in the description of DMD's process in Chapter 3. This is for the purpose of consistency only. Also, we will use examples where appropriate to clarify each step of the process. These examples lead us through the development of a requirements specification for a head-merge workcell similar to the one in the case study[42] (Section 3.2). The author encourages the reader to use Figure 5.1 as a roadmap during the description of the new process. From this roadmap, we see that "Allocate Requirements" is the first step in the new process. This is where we shall start.

### 5.3.1 Allocate Requirements

The requirements allocation process was described in Section 2.3.1. The first step in the requirements specification process involves allocating the requirements of a manufacturing process among each of its subsystems.[43] For each of the production tools

---

[42] These examples do not accurately reflect any of the requirements specifications for DMD's head-merge workcells.

[43] The specification process assumes that the requirements of the manufacturing process are complete and correct.

required by the manufacturing process,[44] the requirements allocation process defines the highest level requirements. The Process Lab performs the requirements allocation process, and the output is a tooling request form for each tool that is procured.

**Example: Allocating the requirements of the head-merge process down to a head-merge tool.**

A manufacturing process is required to perform an assembly operation on a hard disk drive. This process is called head-merge and it is similar to the one described in the head-merge case (Section 3.2). The requirements of the head-merge process are shown by Figure 5.5.

1. The process shall not occupy more than 60 $ft^2$ of floor space.
2. The process cycle time shall not exceed 60 seconds.
3. The process shall be capable of merging 400 assemblies every 8 hours.
4. The process shall be available 98.0 percent of the time.
5. The process yield shall exceed 99.0 percent.
6. The process shall not require more than one operator.

Figure 5.5 - Requirements of the head-merge process.

The Process Lab designs a process consisting of two parts: (1) load HSA into base-plate assembly, and (2) merge heads. The first part of the process, load HSA, will be performed by a process operator and requires no tooling. The second part of the process, merge heads, is more complex process and will require a sophisticated production tool.

The Process Lab allocates the requirements of the head-merge process between the two sub-processes (Load HSA and Merge Heads). Figure 5.6 shows the allocated requirements. Following the allocation process, a tool request form is issued to Factory Engineering stating the requirements of the head-merge tool (Figure 5.7). The tool request form is then assigned to a tooling procurement engineer who is responsible for understanding the needs the head-merge tool must fulfill and

[44] We are assuming the manufacturing process will use a production tool although this is not a necessary condition.

transforming these needs into a set of verifiable requirements. The next step in the development cycle of the head-merge tool is the Preliminary Concept Review meeting. It is during this meeting that the tooling procurement engineer will learn about the needs that the head-merge tool must fulfill.

| | Load HSA | Merge Heads | Head-Merge Process |
|---|---|---|---|
| Floor Space (sq-ft) | 20 | 40 | 60 |
| Cycle Time (sec) | 20 | 40 | 60 |
| Availability (percent) | 100 | 98 | 98 |
| Yield (percent) | 99.5 | 99.5 | 99.0 |
| Labor Content (sec) | 20 | 40 | 60 |

Figure 5.6 – Allocating the requirements of head-merge process among its sub-processes.

**Tool Request Form**

**Tool Name:** Head-Merge Tool

**Tool Classification:** Assembly Tool

**Requirements:**

1. The assembly tool shall occupy no more than 40 $ft^2$ of floor space.
2. The assembly tool's cycle time shall not exceed 40 seconds.
3. The assembly tool shall be available 98 percent of the time.
4. The assembly tool's yield shall exceed 99.5 percent.
5. The assembly tool shall require no more that 40 seconds of labor content.

Figure 5.7 – Example tooling request form following the allocation of requirements from process to tool.

### 5.3.2 Preliminary Concept Review

The purpose of the Preliminary Concept Review meeting is to (1) decide upon the tooling class from the classification hierarchy and (2) agree upon the needs the tool must fulfill. This agreement must come from the process team that is present during the meeting. The

process team is similar to DMD's process team with the exception that it includes a representative from product engineering.[45]

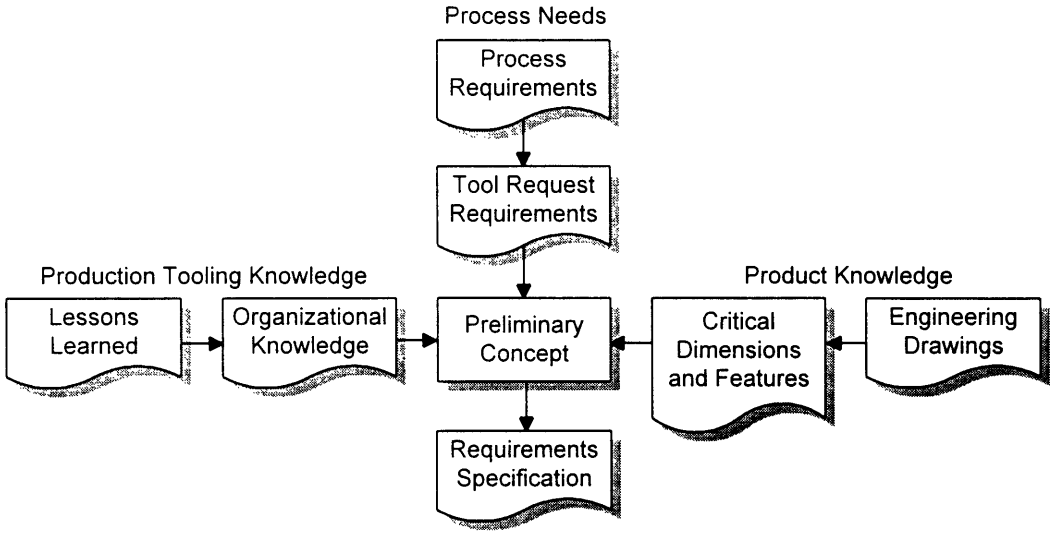



Figure 5.8 – Inputs and outputs to the Preliminary Concept Review.

The first order of business in the Preliminary Concept Review meeting involves deciding upon a classification for the tool. In doing so, each member of the process team draws from his or her own personal experience about what has and has not worked well in the past. While at DMD, the author met with a group of process operators and production engineers who were opposed a tooling concept that would result in a manual process. They cited reasons, based on their experience, why this tool would not meet DMD's needs and therefore needed to be automated. This is the value of having a Preliminary Concept Review meeting–it brings important issues to the table early in a tool's development cycle when the cost of making changes is relatively small. The experiences communicated by the process team are a knowledge input to the Preliminary Concept Review process (Figure 5.8). Rather than rely only on the internal knowledge of each of

[45] The purpose of having a product engineer on the team is to make known all those features of the product design that will be or could be affected by the tool. This includes identification of all critical dimensions that the tool is responsible for holding. The product engineer brings to the team an inherent understanding of what is important to the performance and quality of the product. This goes beyond what engineering drawings can tell.

the process team members, the team should also identify the tools that have fulfilled a similar need in the past and strongly consider those concepts which have proven themselves successful–DMD calls this leveraging. Also during the Preliminary Concept Review, time should be spent identifying which elements of the product's design will or might be affected by the production tool. At a minimum, all of a product's critical dimensions that the tool is responsible for holding must be identified.

The output of the Preliminary Concept Review is a requirements specification for the tool. During this review, the process team should focus on specifying only those requirements that are unique to the tool (i.e., those requirements that are not already specified by the specification classes). In this way, the process team is focusing its thoughts and energies on what is unknown. For example, spending time specifying requirements like "all oil reservoirs on the tool shall have fill lines" is not as value added as trying to make some fundamental decisions regarding the tool's design. The lower level requirements just described should be hidden away in the specification classes.

Prior to exiting the Preliminary Concept Review, there should be a complete description of the process using the tool, and a complete set of engineering drawings for all of the relevant components of the product being manufactured. These will be part of the requirements specification package sent to the tooling vendor (Figure 5.9). The next step in the requirements specification process involves the tooling procurement engineer's transformation of the current requirements specification into the complete set of requirements specified by the requirements specification document.

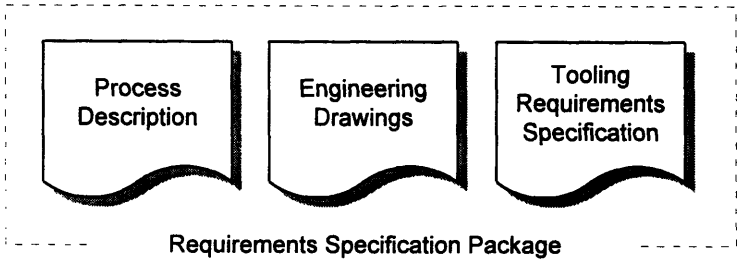



Figure 5.9 – The requirements specification package for a production tool.

| | Load Workcell | Workcell | Unload Workcell | Merge Heads |
|---|---|---|---|---|
| Floor Space (sq-ft) | 0 | 40 | 0 | 40 |
| Cycle Time (sec) | 7 | 28 | 5 | 40 |
| Availability (percent) | 100 | 98 | 100 | 98 |
| Yield (percent) | 100 | 99.5 | 100 | 99.5 |
| Labor Content (sec) | 7 | 0 | 5 | 12 |

Figure 5.10 – Requirements allocation among automated (workcell) and non-automated tasks.

**Tooling Requirements Specification**

**Tool Name:** Head-Merge Workcell
**Tool Classification:** Semi-Automated Assembly Workcell

**Requirements:**

1. Shall occupy no more than 40 $ft^2$ of floor space.
2. Cycle time shall not exceed 33 seconds.
3. Shall be available 98 percent of the time.
4. Yield shall exceed 99.5 percent.
   - 4.1 Process Capability Index for Dimension "A" shall exceed 1.33.
   - 4.2 Process Capability Index for Dimension "B" shall exceed 1.00.
   - 4.3 Read/Write heads shall never come in contact with the disks except for the landing zone.
5. Shall require no operator interaction while processing.
6. Shall have separate maintenance and operator graphical user Interfaces (GUI).
7. The Workcell shall conform to standards for class 100 clean room.
8. The Workcell shall conform to standards for ESD control.

Notes:

1. The operator will load the drive assembly into the workcell and initiate the start sequence. This shall take no more than 7 seconds.
2. The operator will unload workcell. This shall take no more than 5 seconds.

Figure 5.11 –Tooling requirements specification based on the second round of requirements allocation.

**Example: Preliminary Concept Review**

A process team has convened to discuss the preliminary concept for the head-merge tool. They have decided to use a semi-automated workcell (this is the specification class). Following this decision, the team focuses on what the needs are of each of the groups represented. Also, since this a semi-automated workcell, the team allocates the requirements of the tool among its automated (workcell) and non-automated tasks (load and unload workcell). The results of the allocation process are shown in Figure 5.10. Based on this most recent allocation of requirements and the original tool request, the requirements specification for the tool has been created and is shown by Figure 5.11.[46]

### 5.3.3 Transformation and Documentation of Requirements

This section describes the actual process the tooling procurement engineer will use in specifying the requirements of a production tool. This is the main thrust of this thesis so it will be the most detailed subsection in Section 5.3. We start with an overview of the requirements specification process.

#### 5.3.3.1 Overview

The purpose of expanding the specification developed in the preliminary concept review is to create a set requirements that will help the verification team satisfactorily determine whether the tool meets its real requirements. The tooling procurement engineer plays the primary role in this part of the requirement specification process and must have, in hand, the agreed upon set of requirements determined by the process team prior to proceeding. Figure 5.12 shows the process flow for the requirements specification process. The input to the process is the set of requirements established by the process team (this is shown by the "Tool Requirements" document in Figure 5.12. The output of this process is the tool's requirements specification document. The requirements specified in this document must be verifiable (Section 2.4.2).

---

[46] This requirements specification is not intended to be complete; it is merely intended to illustrate a point.

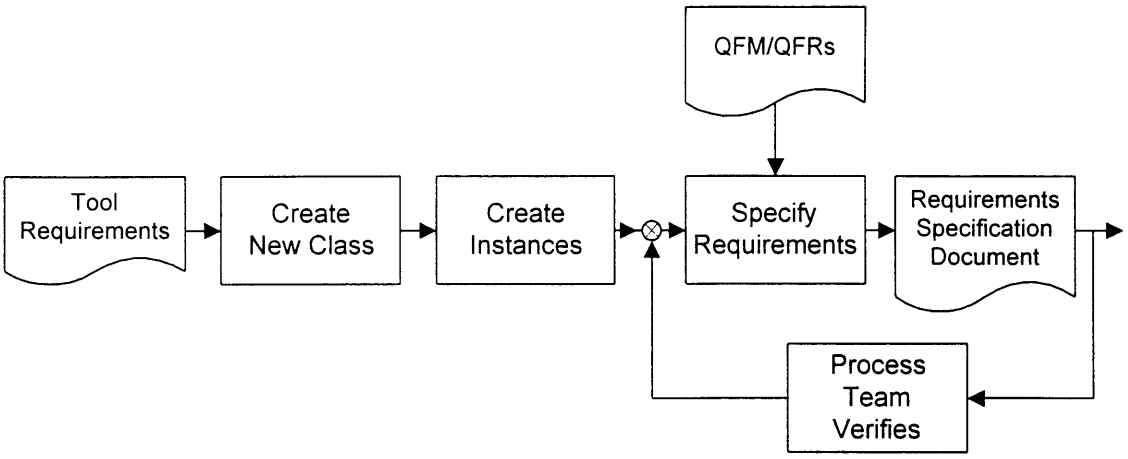



Figure 5.12 – The process flow for "Specify Requirements."

The transformation of an operational need for a production tool into a set of verifiable requirements requires knowledge of every aspect of production tooling. The operational needs of a production tool are often specified in terms of general, qualitative terms (e.g., "we want a tool that is reliable," "it should be easily maintained," and "it has to be usable"). These requirements are not verifiable even if the process team attempts to quantify them by assigning values to them (e.g., MTBF shall exceed 200 hours, MTTR shall not exceed 10 minutes). It is the tooling procurement engineer's job, as the requirements analyst, to transform these requirements into a requirements specification that will ensure their presence in the tool procured. In the transformation of a requirements specification, DMD's procurement engineers draw from their own internal knowledge of production tooling–we saw an example of this in the head-merge case study (Section 3.2). We can improve upon DMD's process if we create a process that draws from organizational learning instead of individual learning. This is the motivation for using the Quality Factors Matrix (QFM) developed in Section 4.3.1. By using the QFM, a procurement engineer can access the knowledge of the organization to create a verifiable set of requirements (e.g., "all locating points shall use carbide inserts" (reliability), "all wires shall be marked with a unique identification number" (maintainability), "the operator shall not have to apply more than two pounds of force to any component of the tool in order to operate it" (usability)). Moreover, because of the

object-oriented specification framework we have developed, the procurement engineer can focus his or her efforts on what is different about the tool being procured. Finally, before the requirements specification document can be considered complete, it must be validated by the same process team that participated in the preliminary concept review. This is an essential part of the verification activities that take place during the tooling development process. Errors found here are significantly less expensive to correct than they are downstream.

#### 5.3.3.2 Create New Class

The first step in "specify requirements" is to create a new class based on existing classification hierarchy. This is done using inheritance.

**Example:**

The tooling procurement engineer knows from the preliminary concept review that the new head-merge workcell must comply with both ESD and clean room specifications; it must have a graphical user interface with separate maintenance and operator sessions; it must include all assembly fixturing. The procurement engineer creates a new class of tool called **head_merge_workcell** which inherits the following specification classes:

1. workcell_assembly_fixture
2. esd_specification
3. clean_room_specification
4. operator_interface
5. maintenance_interface
6. gui_interface

In the above example, the new specification class is called **head_merge_workcell**. In a sense, the procurement engineer is "plugging" existing specification modules together to create a new one (Figure 5.13). This process will not create the entire specification needed for the head-merge workcell, but it will create a significant portion of it.[47] Figure

[47] This of course depends on how well the classifications it is inheriting have been defined.

5.14 shows the **head_merge_workcell** class and its full inheritance.[48] There are several benefits in this approach to requirement specification. They are:[49]

1. Rapid specification development. This is the most obvious benefit. By reusing existing specifications, the tooling procurement engineer does not have to reinvent a specification each time a tool is procured.
2. "Goodness" of specification. The specification classes become more rigorous as they mature; therefore, the requirements specifications that inherit from them are more verifiable.
3. Correctness of specification. A well defined class structure maintained by the organization has high visibility. Anyone in the organization can quickly locate and verify the correctness of a specification class.
4. Organizational change. There is less of a dependency on experience to develop a good requirements specification; this will enable the procurement organization to adapt to organizational change better.
5. Consistency. Instead of procurement engineers creating their own specifications, they create their specifications from a common set of "plug-ins."

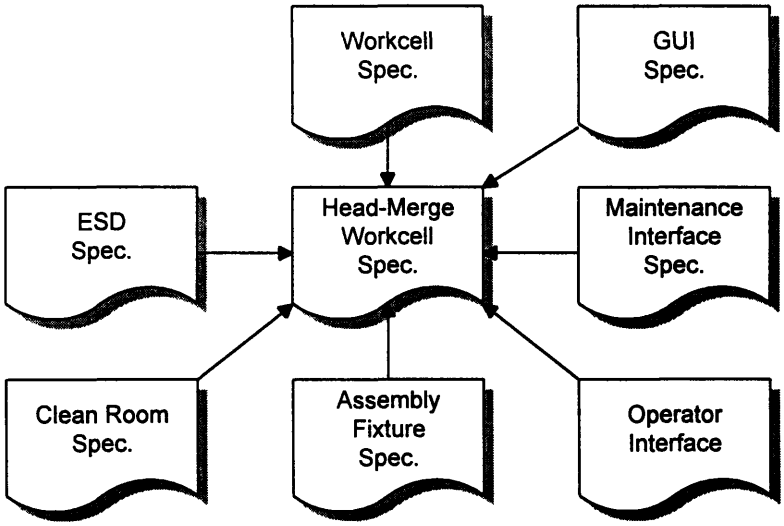



Figure 5.13 – Plugging existing specifications into the new head-merge specification.

[48] Full inheritance means it shows the classifications complete lineage. For example, it is only necessary to show "workcell" and not its inheritance. By inheriting the workcell class we inherit its children also.

[49] These benefits assume that the specification class definitions are maintained at an organizational level.

### 5.3.3.3 Create Instances

Some of the specification classes have requirements specification types that must be assigned values or specifics in each instance of their use.

**Example: Creating Specification Instances**

The procurement engineer expands the **head_merge_workcell** class to show its full inheritance (Figure 5.14). To inherit the class **workcell**, the procurement engineer must assign values to the following construction requirements from Specification Class 4 in Appendix A: cycle time, length, width, and height[50]. To inherit the class **automated_assembly_tool** the procurement engineer need not specify any instance requirements. And to inherit the class **assembly_tool** the procurement engineer must assign values to the constructors availability, foot_print, setup_time, and scheduled_downtime.

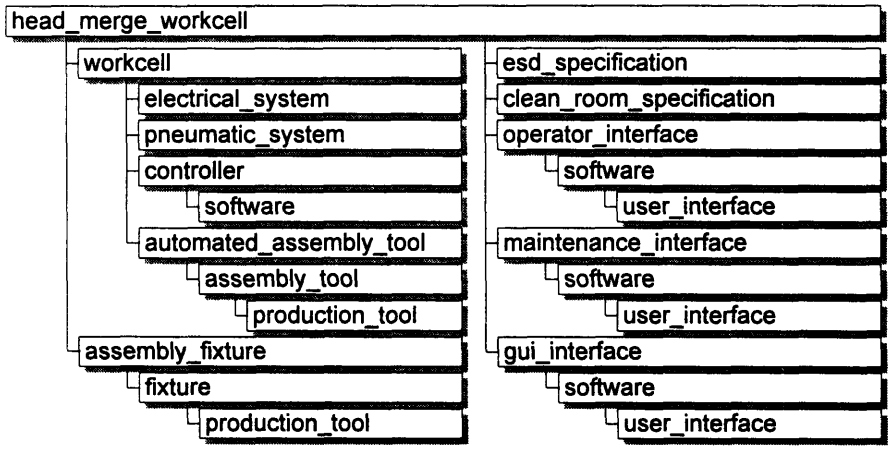



Figure 5.14 – Head-merge workcell class (fully expanded).

To understand how the requirements specification process benefits from using constructors, we turn our attention back to DMD's requirements specification process. In the author's investigation of DMD's requirements specification process, there were instances of requirements specifications where some of the most basic requirements of a tool were left unspecified (e.g., cycle time, Cpk's for critical dimensions, reliability, etc.).

[50] Inherits from workcell:automated:assembly:production_tool.

In talking with managers and engineers, one immediate solution was to create a "hot-sheet" of requirements that would be applied to all production tools. Of course this is not possible because what is a requirement for one tool might not make sense for another tool (e.g., cycle time of a torque driver, uptime of a fixture). By creating tool classifications that have constructors requiring the specification of certain requirements types we are in effect creating a "hot-sheet" that can adapt itself to the type of tool being specified.

#### 5.3.3.4 Specify Unique Requirements

Up to this point we have developed a hierarchy for the tool and performed all of the necessary construction within the hierarchy. We now turn our attention to those requirements that are unique to the tool being procured. These requirements must also transformed into a verifiable set of requirements. This is the purpose of having the Quality Factors Matrix (QFM) and Quality Factor Recipes (QFRs) described in Sections 4.3.1 and 4.3.2, respectively. The procurement engineer has direct access to an organizational knowledge base that has been developed from all of the lessons learned by the organization. Not every learning had been placed into the classification structures because they must be kept generic enough to be reused. The QFRs contain all of the organization's learnings, making it a valuable tool for the procurement engineer who must add rigor to a unique set of requirements.

#### 5.3.3.5 Create Requirements Specification Document

Once the requirements specification for the tool is complete, the requirements must be documented. The attributes of the document should be consistent with those listed in Section 2.4. As long as these attributes are present in the requirements specification document, it is safe to say that the document will serve its purpose which is to communicate requirements, support verification and validation activities, and control the evolution of the system. Various methods can be employed to create requirements specification documents with the desired attributes. In Chapter 6 we are going to show how the Standard Generalized Markup Language (SGML) can be used to support the

requirements documentation process. For now, we will assume that a requirements specification document has been developed using a process that results in the document meeting its required attributes.

#### 5.3.3.6 Verification

The output of the requirements specification process is the tool's requirements specification document. Rather than deliver it to the tool designer in its current form, it is important that this document be verified by the same process team that was present at the preliminary concept review. This is the first of the verification steps that will take place in the tool's development lifecycle. We now recall three of the points made by Davis (1990) in Section 1.4 (p. 17):

1. The later in the development lifecycle that a [system] error is detected, the more expensive it will be to repair.
2. There are requirements errors being made.
3. Requirements errors can be detected.

From Table 1.1, the relative cost of repairing errors at this stage of the development process is much less than they will be when the tool is delivered to the manufacturing floor. In Section 3.5.1.1, we described an incident where a requirements specification document containing multiple errors was about to be sent to a tool vendor. This type of incident motivates the need for an early requirements verification step.

All that is required for this verification process to take place is that the requirements document be distributed or made available to the members of the process team. Recall that one of the purposes of the requirements specification document is to support verification activities. This is one of those activities.

### 5.3.4 Develop Concept

The design process starts when the a tool's designer receives the requirements specification document from the purchaser. The first step is to develop the tool's concept

which was started back in the preliminary concept review. The process is identical to the one discussed in Section 3.3.2 and will not be restated. The tool's concept is not complete until it has been approved by the members of the process team.

### 5.3.5 Specify Requirements (Final)

Once a tool's concept is finalized, there may be instances where an additional set of requirements are required. This is because the tool's designer may make choices about the design that cannot be foreseen by the process team. As is the case with verifying the preliminary requirements specification document, the process team must approve the final version of the requirements specification document before the design of the tool can commence.

**Example: Specify Requirements Following Concept Review**

In developing the concept for the head-merge workcell, the tool designer has decided to use a vision system to measure dimensions G1 and G2. The process team has approved the use of this vision system and the tooling procurement engineer applies a set of requirements to it. In this case there exists a specification class for vision systems and the procurement engineer is able to "plug" this specification right into the existing one
(Figure 5.15).

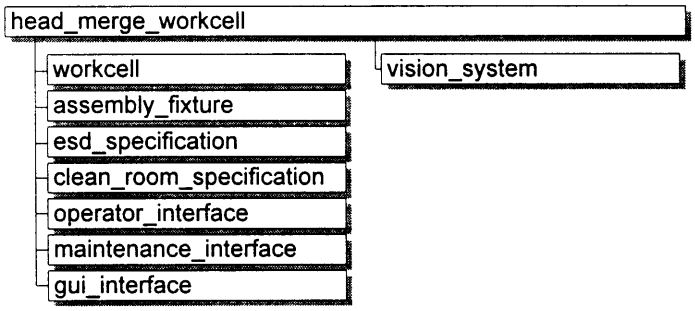



Figure 5.15 – Head-merge workcell class showing the vision system class "plugged in."

### 5.3.6 Design Tool and Verify

Once the tool's design is complete, the design of the tool is verified against the tool's requirements specification document. The design process is identical to the one

discussed in Section 3.3.2. The difference lies in the verification step. DMD uses a design review whereas this process is advocating a verification process. The design features of the tool should be checked against the requirements specification document. This increases the probability that errors will be detected before the build process actually begins. It will also help ensure the design of the tool is complete (a requirement for the tool development process to advance to the Build Phase).

### 5.3.7 Build Tool and Verify

Once the tool is built it is ready for final verification. If all of the requirements stated in the tool requirements specification are verifiable, then the verification process should be relatively straightforward.[51] A discussion on the techniques of verification is beyond the scope of this thesis.

### 5.3.8 Install Tool and Validate

Once a tool has been integrated into the manufacturing process for which it was designed, the validation cycle starts. Unlike the verification process whose purpose was to verify the requirements of the tool as stated in the requirements specification document, the purpose of validation is to understand how well the tool meets the needs of the manufacturing organization. It is very probable that not all of the tool's needs will have been satisfied, even if the tool had no problems meeting all of its requirements. How many of these needs are left unsatisfied will depend on three factors: (1) how well the process team communicated their needs, (2) how well the procurement engineer transformed these needs into a set of requirements, and (3) how thoroughly the process team verified the requirements in the requirements specification document (preliminary draft and final version). It is with the installation and validation of a production tool that

---

[51] Relative to trying to verify unverifiable requirements (e.g., MTTR shall not exceed 10 minutes, the tool shall be usable).

the greatest opportunities for learning are present. If the tool fails to meet a need, the question "why?" should be asked. It should be determined if it was (1) in the Preliminary Concept Review when the requirements were being specified, (2) in the transformation of requirements into a verifiable set of requirements, or (3) in the verification of the tool. If it is in the requirements transformation, the lessons learned from this experience should be entered on the appropriate QFR. The next section expands upon the importance of organizational learning and the use of the QFM.

## 5.4 The Value of Organizational Learning

In the description of the requirements specification process we talked about the benefits of using a QFM. The power of the QFM comes from its ability to convert unusable, loosely organized knowledge into a knowledge base rich with information that the procurement engineer can use to help transform tooling needs into a set of tooling requirements that will help ensure these needs are met. The QFRs that are referenced by the QFM contain the actual knowledge that will be used in the specification process. These tools, however, are only valuable if they are "filled with knowledge." The learning process that fills these tools with knowledge takes place during a tool's validation cycle. To show how the learning process works in the context of validating requirements we will use an example many people can relate to.

**Example: Maintainability of an Automobile**

Chris is purchasing an automobile for the first time in her life. She wants it to be maintainable because she plans on doing as much of the scheduled preventative maintenance herself. Prior to purchasing the vehicle she "checks under the hood" to verify that the engine meets her maintainability requirement. With no real indication of what to look for she is impressed with the overall appearance of the engine and it passes her maintainability test. The vehicle is purchased.

> One year later, Chris decides to change the spark plugs on her new vehicle only to find out that they are hidden away under the exhaust manifold. They are so inaccessible, it takes her 4 hours to change all six spark plugs. Chris vows that the next time she purchases a vehicle she will make sure all of the spark plugs are easily accessible.

There are three messages we want to pull out of this example. The first and most important message is that Chris has gone through a learning process that will enable her to do a better job of verifying that an automobile meets her maintainability requirement the next time she purchases one. The second message is that there is a significant difference between verification and validation. Chris verified the vehicle to meet her maintainability requirements and accepted delivery of the vehicle by purchasing it. She did this even though it did not truly meet her needs. And finally, the third message is that the validation process is not something that happens over night. It took Chris a year to realize that the vehicle did not meet her standards for maintainability.

During the author's interviews with the maintenance organization at DMD, it was clear that many of its personnel where frustrated with the maintainability of its production tooling. All of its personnel had very specific requirements they would like to see in the production tooling that DMD purchases. This is a case where each of the organization's members had some knowledge of what it takes for a tool to be maintainable. Unfortunately this knowledge was too fragmented and loosely organized to be of any value to the procurement engineer who needed to specify a set of requirements that would help the verification team determine objectively whether the tool met DMD's true standards for maintainability or not. The purpose of the QFM is to help build and organize this knowledge.

## 5.5 Summary

In this chapter we developed a new process for specifying and verifying production tooling requirements. The process is based largely on the process DMD uses, but we

have added strength to it in three separate areas: (1) it supports organizational learning, (2) it supports systems thinking, and (3) it takes a more proactive approach to requirements verification. These are the areas the author feels DMD's requirements specification and verification process can best be enhanced.

# 6

# Using SGML to Specify Production Tooling Requirements

This chapter shows how the Standard Generalized Markup Language (SGML) can be used to create a system that supports the requirements specification and verification process developed in Chapter 5.

## 6.1 Introduction

The goal of this chapter is to develop a system that supports the requirements specification and verification process described in Chapter 5. There are numerous possibilities for the implementation of such a system, but the author believes a system that incorporates the Standard Generalized Markup Language (SGML) will have several unique advantages over a system that does not. These advantages stem from the SGML's ability to seamlessly link documents together, and to rigorously define a documentation structure that supports the nine attributes a requirements specification document must possess to be effective (Section 2.4). Before getting started, a brief introduction to the SGML is in order. This introduction is very brief since there are books available that describe the SGML in detail. The author recommends that the reader interested in learning more about SGML obtain one of these books[52].

[52] The author recommends Practical SGML, Van Herwijnen, 1994.

## 6.2 SGML Basics

The Standard Generalized Markup Language (SGML) is an agreed upon set of rules for marking up character based text files (text files)[53]. These rules were established by the International Organization for Standardization (ISO) standard for document description in 1986 (ISO 8879). An SGML document is a text file that contains *markup,* where markup is nothing more than a set of predefined text symbols that are inserted into a text file. A document can be defined to have more than one type of symbol, and each type of symbol is called a *tag*. Tags are defined by the Document Type Definition (DTD) for a particular type of document and they can be used to format text or associate a command with selected text. If one examines an SGML document, there are two things that immediately standout: the markup, and the actual data. A tag is delimited using the start-tag open delimiter, "<", and the tag-close delimiter, ">". The name of the tag is contained between the two delimiters. Figure 6.1 show how tags can be used to format text. The tags used in this example are defined by the document type definition contained in the file named "html.dtd.[54]" Rather than explain what each of these formatting tags does, we will direct our attention to Figure 6.2 which is the way this particular SGML document is viewed from an application that interprets and displays SGML documents.

Tags can also be used to attach commands to selected text. These commands are limited only by the creativity of the person creating the document type definition (DTD) and by the application that supports it. A good example is the DTD for the HyperText Markup Language (HTML) and the popularity of the browsers that support HTML. HTML assigns network links to selected hypertext, thereby enabling users to navigate the world's largest network, the Internet, with an efficiency that allows users to find information on a plethora of subjects in minutes.

---

[53] By character based, we mean not binary. The most commonly used character set today is ASCII. Other character sets include EBCDIC which is the character set for IBM mainframes, and UNICODE, which is beginning to replace ASCII.

[54] Note the reference to the DTD in the header of the SGML document shown in Figure 6.1.

```
<!DOCTYPE HTML SYSTEM "html.dtd">
<HTML>
<HEAD><TITLE>Demo Document</TITLE></HEAD>
<BODY>
<H1>This is a level 1 heading.</H1>
<H2>This is a level 2 heading.</H2>
<H3>This is a level 3 heading.</H3>
<P>This is a paragraph containing text. It demonstrates how a
tag can be used to apply formatting to text such as
<B>bold face</B> and <I> italic</I>.</P>
</BODY>
</HTML>
```

Figure 6.1 – An SGML document showing how tags can be used to format text.

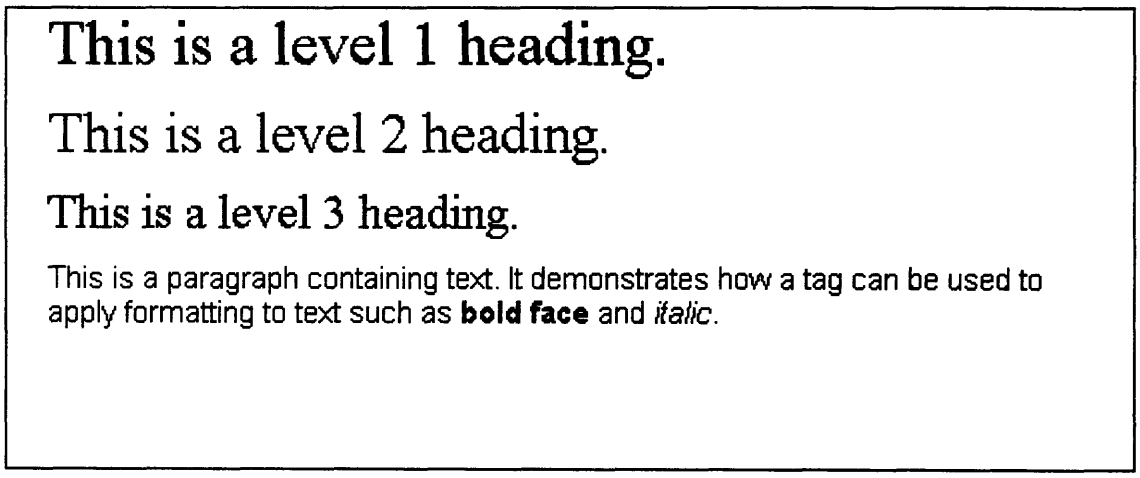



Figure 6.2 – An SGML document as viewed from an application that interprets and displays SGML documents.

## 6.3 Using HyperText to Support Organizational Learning

One advantage of hypertext is that it allows users to navigate through large volumes of information that have been linked together. Chapter 4 described an object-oriented framework that can be used to create requirements specifications. One of the reasons for having this framework is to create an intuitive index for the information regarding the requirements specification of production tooling. Figures G.1 through G.4 in Appendix

G show four SGML documents as they would be viewed by the user of an SGML system. Some of the data in these documents have been tagged as links to other documents. The hypertext (shown by that text which is underlined) is associated with the links contained in each document so that the user can process a command to move to another document by clicking on the hypertext using a mouse pointer (or equivalent device). This creates an interactive experience for the persons retrieving the information by allowing them to move seamlessly among all of the documents that have been linked together. Figures G.1 through G.4 show how a user can follow a set of linked documents and review the set of requirements that have been created for each of the specification classes in the requirements classification hierarchy. The figures that are shown map the classification hierarchy shown in Figure A.1 of Appendix A and in Appendix B. The navigational ability of such a system prompts its use; as the classification hierarchies are expanded upon and improved, the information contained within them will bring added value by reflecting the organization's increased knowledge.

We have just seen how hypertext can make the navigation of a linked set of documents simple. The advantages of hypertext, however, go beyond this ease of document navigation. One advantage of object-orientation is its support of information abstraction. This feature allows the person analyzing the construction of a system to ignore any aspects of the system which are not relevant to the current subject of concern so that he or she may concentrate more fully on what is important. We can use hypertext to take full advantage of this abstraction. We turn our attention to Figure G.4. A hypertext link has been created to a document containing information on emergency stop buttons. Rather than display this information on the screen, it is simple to create a hypertext link to a common document containing the requirements of the emergency buttons the organization uses on its equipment. If this information is of value to the person retrieving the information, it is a simple matter to select the hypertext in order to view it. Using hypertext, we have hidden, through abstraction, much the information regarding the requirements of automated_assembly_tool while at the same time making access to it quick and easy. The other advantage of this system is that a common document has been

created to contain all of the requirements for emergency stop buttons. A change to the requirement of an emergency button specification will be implemented immediately in all of the specifications referencing it.

## 6.4 Using SGML to Support the Requirements Specification Process

SGML documents are rigorously structured by a set of rules defined by the Document Type Definition (DTD). These rules define the legal set of tags that are allowed in a document as well as the order in which they are required to appear. The result is that the DTD sets forth the rules for constructing a specialized markup language (Van Herwijnen, 1994). We will use the DTD to design our own markup language called the Requirements Markup Language (RML). Just as the HyperText Markup Language (HTML) was uniquely designed to support Internet navigation, RML is uniquely designed to support requirements specification. The design of the RML was created with the following goals in mind:

1. Support the traceability of requirements.
2. Support the object oriented framework created in Chapter 4.
3. Support the requirements documentation process.

### 6.4.1 RML Document Type Definition (DTD)

The DTD defining the RML is shown by Appendices C and D. The tags defined are: **inherits, cname, parent, bname, children, child, requrmnts, construc, req, qf, qsf,** and **how**. Appendix C shows the tree and structure diagrams for the RML. An explanation stating the purpose and rule for each tag is provided in Appendix E. The rule set for the RML shows the degree of rigor with which we can create a document structure that supports the requirements specification process. Appendix F shows an RML document for the specification class **production_tool** (Specification Class 1 in Appendix B).

The RML supports the requirements specification process in a number of unique ways

which will be described briefly. First, by embedding the document's inheritance, we can automatically support the creation of links. In Figures G.1 through G.4, we saw how a user could follow the path of the classification hierarchy shown in Appendix A. To do this, the author had to manually create a link for each of the elements in the document. This is because the examples shown in Figures G.1 through G.4 where created with the HTML which has not been designed to meet our specific set of needs, although it does a satisfactory job in many respects. We can use inheritance data in each document to create the links we need automatically. Aside from automated link creation, the embeds inheritance in each document serves as a roadmap for the user of the system. Hammond (1990) points out why this is important:

> "It is certainly the case that learners can get lost or disoriented in large hypertext structures. The information base may be large and its structure unfamiliar, and the links provided will not be suitable for all individuals and for all tasks. Once in an unknown or unexpected part of the knowledge base, the user may have difficulty in reaching familiar territory, although the provision of backtrack facilities may alleviate this. More critical perhaps is that learners may also have problems finding specific information they know to be present (Hammond, 1990, p. 60)."

The RML with its embedded inheritance can be used to provide the user of the system with the information they need to keep from getting lost.

Secondly, the RML can be used to assign a quality factor and quality sub-factor to each of the requirements listed in a document. This feature can be used to support the automatic creation of a requirements specification document. One of the problems of using a classification hierarchy to create a set of requirements specifications is that there is not one contiguous document which contains all of the requirements of a given tool. It would almost certainly be difficult for a tool designer to create a tool design given a set of class specifications. With some of the elements of maintainability assigned to one class specification, and another set of elements assigned to another class, the essence of what is

required for the tool to be maintainable is lost. In constructing a requirements document, a parser can be used to scan through all of the requirements in class, including those requirements it has inherited, and organize them by quality factor and quality sub-factor.

Finally, the RML supports traceability of requirements. By giving the ability to specify how a requirement ensured, we are creating a system that supports top-down traceability. The traceability in our system is weak in that it only provides traceability for the requirements contained within each document. It shows, however, how SGML can be used to support traceability and the RML could be modified to support a more expansive traceability mechanism.

The purpose of creating the RML was not to establish the authoritative model for a markup language that supports requirements specification and verification. The purpose of creating this implementation was to simply show how the SGML can be used to add value to the requirements specification process.

## 6.5 An SGML Based Requirements System

Any SGML system consists of four subsystems: a database containing SGML data, a database of document type definitions (DTDs), an SGML parser, and an applications library (Figure 6.3). All of the system's data is stored in an SGML database which is made up of document instances. A document instance is a file containing the text to be processed, the SGML markup, and a DTD reference (Van Herwijnen, 1994).

### 6.5.1 The SGML Database

All of the SGML system's data is stored in a database which is made up of document instances. A document instance is a file containing the text to be processed, the SGML markup, and a DTD reference (Van Herwijnen, 1994). Because SGML files are text files, all that is needed is a file system; it can be any type.

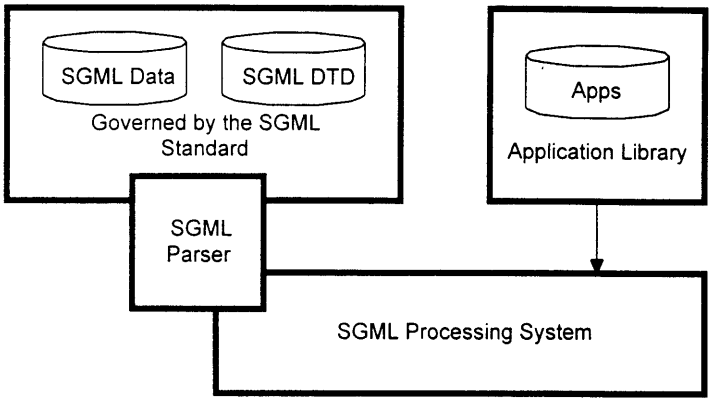



Figure 6.3 – The four basic subsystems of an SGML system.

### 6.5.2 The SGML Parser

The parser is the rule checking mechanism for SGML documents. It scans the document, identifies the elements (i.e., tagged text), verifies that no illegal elements are present, and then verifies that the ordering of the elements is correct. The rules used by the parser to verify the integrity of a document are defined by the DTD specified in the header of every SGML document. It is not necessary that a parser be present for an SGML system, but without it, there will be no rule checking.

With respect to requirements specification, the presence of a parser enforces the structure and content of a requirements specification document. In the case of an RML document, the presence of the parser will ensure that all of the requirements specification documents created by an organization share a common format and structure. More importantly, it makes it impossible for a person to unintentionally leave out important requirements specification information such as annotations and traceability linkages. By using a parser, we create a tool that helps ensure that good requirements specification documents are created.

### 6.5.3 Application Library

One of the advantages of SGML is that, through its openness, it is application independent. This means that the users of an SGML system can choose best in class

applications to view their SGML documents. A good example of this is Mosaic and Netscape, applications designed for viewing HTML documents (an SGML implementation just as RML is). Persons with information contained in HTML documents are not limited to a single application to view their data. Mosaic and Netscape are good examples. They are competitors in the business of creating HTML viewers and a customer can choose the viewer that best fits his or her needs.

## 6.6 Conclusion

SGML is well suited to creating a requirements specification and verification system based on the process developed in Chapter 5. First, we showed how hypertext makes it possible to seamlessly move along the object-oriented framework we developed for requirements specification. Finding information is quick and easy. Secondly, from SGML, we showed that we can define our own markup language that is tailored to the requirements specification process. Finally, by using SGML, the requirements specification process came to life by creating an interactive learning experience for the user while at the same time maintaining strict control over form and content of the requirements specification documents that are created.

# 7

# Conclusion

In this thesis, we examined the importance of specifying and verifying production tooling requirements. One of the difficulties in developing a good requirements specification comes from the knowledge required to transform a set of high level requirements such as maintainability, reliability, throughput, and safety, into a set of verifiable requirements. In our case study of DMD's head-merge workcell, we showed how the learning process helped improved the requirements specification for a second generation workcell. These learning's were funneled into the requirements specification for the second tool and were responsible for the improvements that took place. We also saw how DMD, as a whole, failed to learn from the lessons of the individual engineer. As a result, the same problems occurred on a similar production tool that was procured in another manufacturing group. The emphasis in this thesis, therefore, has been organizational learning and we developed a process and a set of tools that supports it. By creating a rigorous requirements specification up front, the manufacturer can adopt a more proactive approach to ensuring that the tools it procures meet its needs.

# References

1. Coad, Peter, and Edward Yourdon, OOA: Object-Oriented Analysis, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.

2. Davis, Alan M., Software Requirements: Analysis & Specification, Prentice-Hall Inc., Englewood Cliffs, NJ, 1990.

3. Dertouzos, Michael, Richard Lester, and Robert Solow, Made in America: Regaining the Productive Edge, Cambridge, MA, MIT Press, 1989.

4. Dorfman, Merlin, "System and Software Requirements Engineering", in R.H. Thayer and M. Dorfman (eds.), Tutorial: System and Software Requirements Engineering, IEEE Computer Society Press, Washington, DC, 1990.

5. Goodman, John, Hard Disk Secrets, International Data Group (IDG) Books Worldwide, Inc., Boston, 1993.

6. Hammond, Nick, "Learning with Hypertext: Problems, Principles. and Prospects," in C. McKnight, A. Dillon and J. Richardson (eds.), HyperText: A Psychological Perspective, Ellis Horwood, New York, 1993.

7. Institute of Electrical and Electronic Engineers, IEEE Guide to Software Requirements Specification, ANSI/IEEE Standard 830-1984, New York, 1984.

8. Keller, S.E., L.G. Kahn, and R.B. Panara, "Specifying Software Quality Requirements With Metrics," in R.H. Thayer and M. Dorfman (eds.), Tutorial: System and Software Requirements Engineering, IEEE Computer Society Press, Washington, DC, 1990.

9. Jacobson, Ivar, Maria Ericsson, and Agneta Jacobson, The Object Advantage: Business Process Reengineering with Object Technology, Addison-Wesley Publishing Company, Reading, MA, 1994.

10. Loy, Patrick H., "A Comparison of Object-Oriented and Structured Development Methods," in Thayer, Richard H.., and Merlin Dorfman (eds.), System and Software Requirements Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1990.

11. Martin, James, and James Odell, Object-Oriented Methods: A Foundation, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1995.

12. Mizuno, Y., "Software Quality Improvement," IEEE Computer, 15, March 1983, pp. 66-72.

13. Nelson, E. Dale, "System Engineering and Requirement Allocation," in Thayer, Richard H.., and Merlin Dorfman (eds.), System and Software Requirements Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1990.

14. Scharer, L., "Pinpointing Requirements," Datamation, April 1981, pp. 139-151.

15. Shapiro, Saul, "Process Transfer in Semiconductor Fabrication: An Evaluation of a Dry Etch Via Process in GaAs MMICs," MIT S.M. Thesis, Cambridge, MA, 1992.

16. Skinner, Wickman, "The Productivity Paradox", Harvard Business Review, Vol. 64, No.4, July-August 1986.

17. Thayer, Richard H.., and Merlin Dorfman, System and Software Requirements Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1990.

18. Thayer, Richard H., and Winston W. Royce, "Software Systems Engineering," in R.H. Thayer and M. Dorfman (eds.), Tutorial: System and Software Requirements Engineering, IEEE Computer Society Press, Washington, DC, 1990.

19. Thurow, Lester C., "A Weakness in Process Technology," Science, Vol. 238, December 18 1987, pp. 1659-1663.

20. Van Herwijnen, Erik., Practical SGML, MA, Kluwer Academic Publishers, Norwell, 1994.

# Appendix A – Example Classification Hierarchies for Two Different Industries

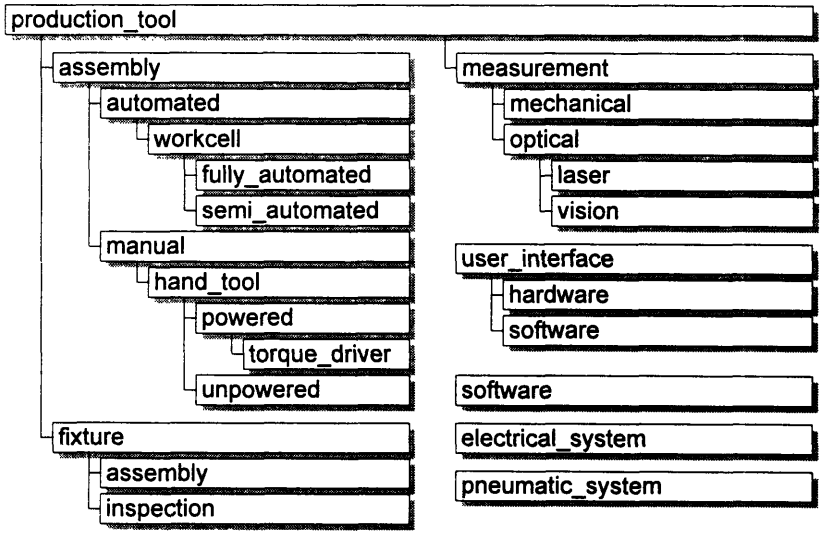



Figure A.1 – An example of a tooling specification class hierarchy for a hard disk drive manufacturer.

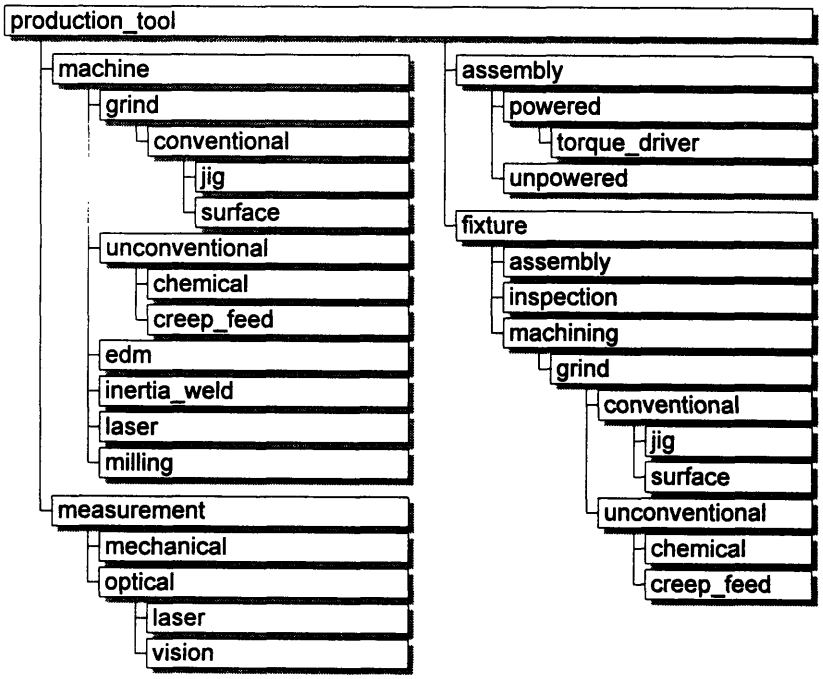



Figure A.2 – An example of a tooling specification class hierarchy for a hard disk drive manufacturer.

# Appendix B – Expanded Classification Hierarchy for a Hard Disk Drive Manufacturer

**class** production_tool

{

1. shall comply with OSHA regulations for health and safety
2. shall comply with corporate safety specification (doc 402)
3. noise levels shall not exceed 79dbA in operator work area or more that 4 feet away from any noise source.
3. shall comply with corporate ergonomic specification (doc 403)
4. upon delivery, workcell shall be accompanied by a list specifying all spare parts, quantities used, market price, and expected operating life in hours.
5. spare parts with lead times less than one week shall be accompanied with the tool in quantities equal to lead time (hrs) divided by expected life of spare (hrs).
6. tool shall require no more than 4 hours of training to operate at specified capacity and quality requirements.
7. no sharp or unfinished edges

**construction**:

}

Specification Class 1

**class** assembly_tool:production_tool

{

**construction**:

| | |
|---|---|
| 1. availability | percent |
| 2. foot_print | square_inches |
| 3. setup_time | seconds |
| 4. scheduled_downtime | hours_per_1000_hours |

}

Specification Class 2

**class** automated_assembly tool:assembly_tool:production_tool

{

1. tool shall conform to facility constraints
   a. 110 and 208 volts AC, 20 Ampere service
   b. 100 PSI, clean air (not dry)
   c. 20 hg vacuum source at 10 CFM
2. emergency stop buttons shall be placed in conspicuous and easily accessible locations.
3. emergency button activation shall immediately shut down all power, air pressure, and vacuum.

**construction**:

}

Specification Class 3

**class** workcell:automated:assembly:production_tool

{

1. mean time to repair shall not exceed 30 minutes
   a. where a choice exists, all workcell sub-components shall conform to spec 103.
   b. maintenance interface shall be used which conforms to spec 104
   c. workcell shall include a complete set of reference manuals describing theory of operation, hydraulic schematics, electrical schematics, mechanical drawings, trouble shooting tips, and error recover procedures.
2. workcell shall have light tower.

**construction**:

1. cycle_time seconds
2. length inches
3. width inches
4. height inches

}

Specification Class 4

**class** electrical_system

{

1. all wiring, power connectors, and switches shall be clearly labeled with amperage, voltage, frequency, and identifier
2. shall comply with NEC regulations
3. all wires shall be routed such that they will not interfere with any moving parts
4. all potentiometers used for calibration purposes shall have a tick mark that indicates the baseline for adjustment
5. all electrical connections shall use a mechanism that prevents an incorrect connection
6. all electrical components shall be grounded

**construction**:

}

Specification Class 5

**class** pneumatic_system

{

1. All tubing and gauges shall be clearly labeled with max. pressure (psi)

**construction**:

}

Specification Class 6

**class** hydraulic_system

{

1. All fluid reservoirs shall be clearly labeled indicating fluid type, max. level, and min. level

**construction**:

}

Specification Class 7

# Appendix C – Document Tree and Structure for the RML

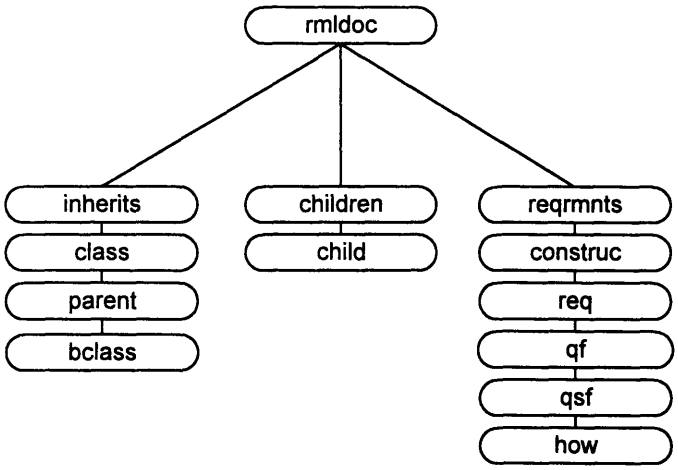



Figure C.1 – Tree diagram for the Requirements Markup Language (RML)

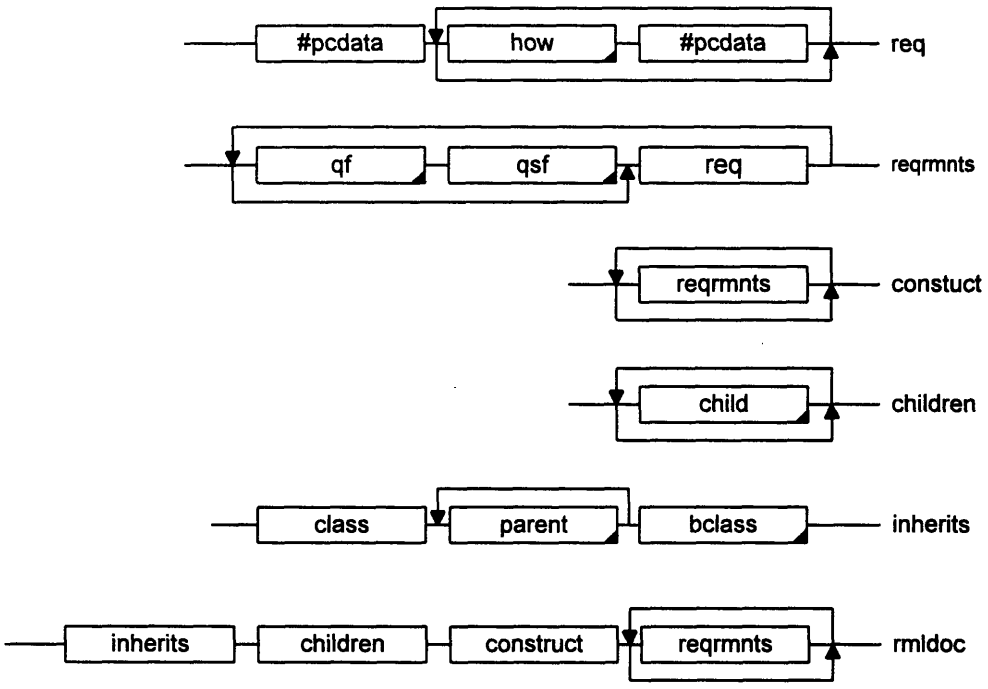



Figure C.2 – Structure diagram for the Requirements Markup Language (RML)

# Appendix D – Document Type Definition

```
<!--dtd for requirements specification markup language                    -->
<!entity % doctype "rmldoc"  --  document type generic identifier         -->
<!--        elements    min      content                    (exceptions)  -->
<!entity    %doctype    - -                                               -->
<!entity    inherits    - 1      (class, (parent)*, bclass)               -->
<!entity    children    - 0      (child)*                                 -->
<!entity    constuct    - 1      (reqrmts)*                               -->
<!entity    reqrmts     - 0      ((qf & qsf)?, req)                       -->
<!entity    req         - 0      (#pcdata, (how & #pcdata)?)              -->
<!entity    qf          - 0      empty                                    -->
<!attlist   qf          id id    #required                                -->
<!entity    qsf         - 0      empty                                    -->
<!attlist   qsf         id id    #required                                -->
<!entity    cname       - 1      empty                                    -->
<!attlist   cname       id id                                             -->
<!entity    parent      - 1      empty                                    -->
<!attlist   parent      id id    #required                                -->
<!entity    bname       - 1      empty                                    -->
<!attlist   bname       id id    #required                                -->
```

DTD for the Requirements Markup Language (RML).

# Appendix E – Explanation of RML Rule Set

1. **<inherits>**
   Specifies the complete inheritance of the document and must appear at least once inside the document. Legal elements within the tag are **class**, **parent**, and **bclass**.
2. **<class>**
   Specifies the name of the document class and must appear at least once inside the document and only inside the **inherits** tag. Must appear prior to **parent** and **bclass**.
3. **<parent>**
   Specifies the names of all the specification classes inherited starting with the most recent. **parent** must appear at least once inside the document and only inside the **inherits** tag. If the class does not inherit from another class, the attribute "this" shall be assigned to **parent**. Must appear prior to **bclass** and after **class**.
4. **<bname>**
   Specifies the name of the base class that **class** is inheriting from and must appear at least once inside the document and only inside the **inherits** tag. Must appear following **class** and **parent**.
5. **<children>**
   Delimits the names of the classifications that inherit from **class**. Is not required to appear but if it does, it must appear following **inherits**, and prior to **construct**.
6. **<child>**
   Specifies the name of each child class inheriting from **class**. Is not required to appear but if it does, it must appear inside **children**.
7. **<constru>**
   Delimits those requirements contained in **class** that are constructors. Even if the document contains no construction requirements, this tag must appear at least once.
8. **<reqrmnts>**
   Delimits each requirement in the document. This tag can appear an infinite number of times. It must follow **inherits** and **children**.
9. **<req>**
   Specifies a requirement. Must appear at least once inside **reqrmnts**.
10. **<qf>**
    Specifies the quality factor for the requirement (**req**) delimited by **reqrmtns**. Not required to appear but if it does, it must appear prior to **qsf**.
11. **<qsf>**
    Specified the quality sub-factor for the requirement (**req**) delimited by **reqrmtns**. Required to appear only if **qsf** does and it must appear after **qsf**.
12. **<how>**
    Specifies a requirement indicating how a given requirement (**req**) will be ensured. Not required to appear but if it does, it must appear inside **req**.

# Appendix F – Example RML Document Showing Markup and Data

```
<!doctype rmldoc system>
    <inherits>
        <class="production_tool">
        <parent="this">
        <bclass="this">
    </inherits>

    <children>
        <child="assembly">
        <child="fixture">
        <child="measurement">
    </children>

    <reqrmnts>
    <req><qf="safety"><qsf="">safety<how>
            <req>shall comply with OSHA regulations</req>
            <req>shall comply with corporate safety specification (doc 402)</req>
            <req>shall comply with corporate ergonomic specification (doc 403) <req>

            <req>no sharp or unfinished edges</req>
    </how></req>

    <req><qf="maintainability"><qsf="">maintainability<how>
          <req>upon delivery, workcell shall be accompanied by a list specifying all
                spare parts, quantities used, market price, and expected operating
                life in hours</req>
          <req>spare parts with lead times less than one week shall be accompanied
                with the tool in quantities equal to lead time (hrs) divided by expected
                life of spare (hrs)</req>

    <req><qf="throughput"><qsf="">throughput<how>
          <req>tool shall require no more than 4 hours of training to operate at
                specified capacity and quality requirements</req>

    </reqrmnts>
```

Figure D.1 – Example RML document containing Specification Class 1, form

# Appendix G – HyperText as a Learning Tool

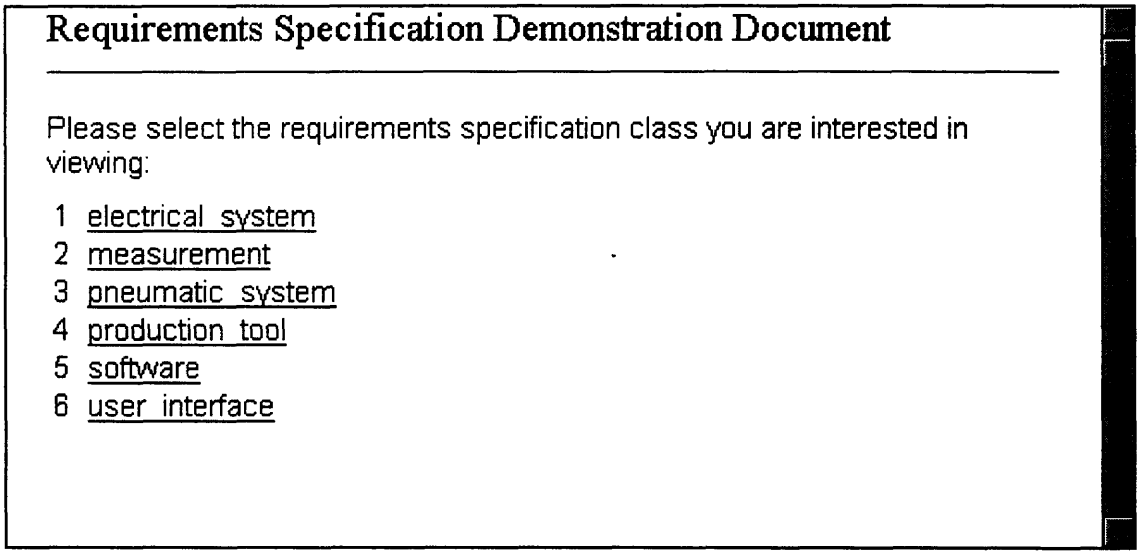



Figure G.1 – An opening SGML document presenting the user with six classifications to learn about.

## Production Tool Specification Class

**Class Name:** production_tool

**Inheritance:** none

**Children:**

1 assembly_tool
2 fixture

## Requirements Specifications by Quality Factor

**Safety, Health, and Ergonomics**

1 Shall comply with OSHA regulations for safety and health.
2 Shall comply with corporate safety specification. (doc 402)
    1 Noise levels shall not exceed 79dba in operator work area or more than 4ft away from any noise source.
3 Shall comply with corporate ergonomic specification.
4 No sharp or unfinished edges.

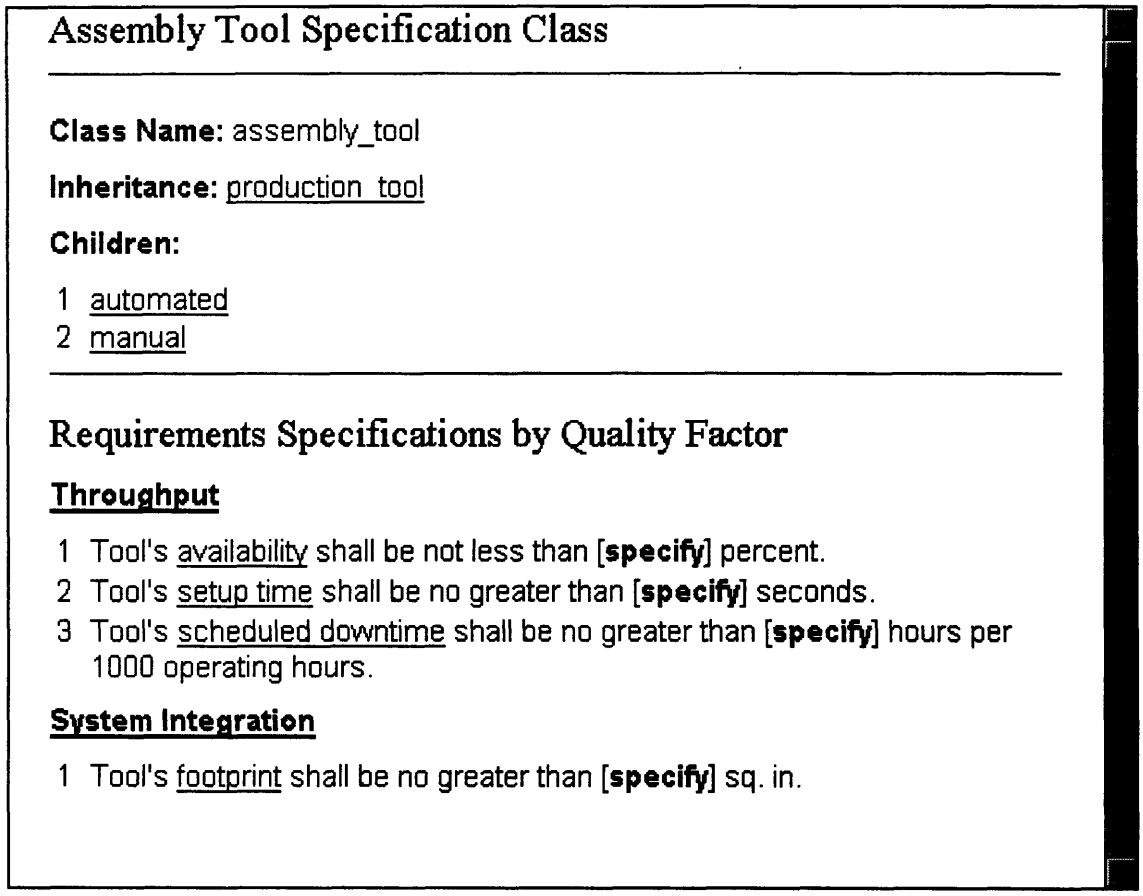
**Maintainability**

1 Tool shall be accompanied by a list specifying all spare parts, quantities used, market price, and expected operating life in hours.
2 Spare parts with lead times less than one week shall accompany the tool in quantities equal to lead time (hrs.) divided by the expected life of the spare (hrs.).

**Usability**

1 Shall require no more than 4 hours of training to operate at specified capacity and quality requirements.

Figure G.2 – User has selected Production Tool from the list of options in Figure G.1 to link here.



## Assembly Tool Specification Class

**Class Name:** assembly_tool

**Inheritance:** production_tool

**Children:**

1 automated
2 manual

## Requirements Specifications by Quality Factor

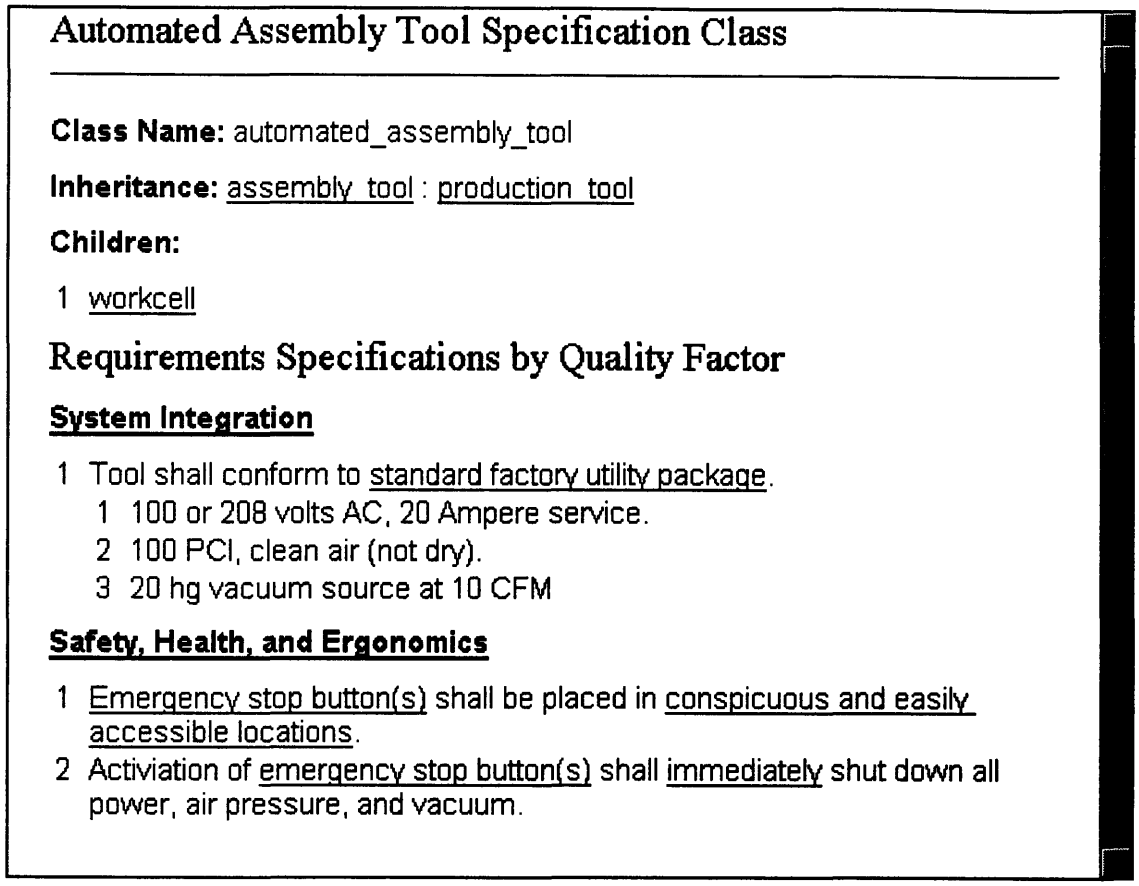
**Throughput**

1 Tool's availability shall be not less than [**specify**] percent.
2 Tool's setup time shall be no greater than [**specify**] seconds.
3 Tool's scheduled downtime shall be no greater than [**specify**] hours per 1000 operating hours.

**System Integration**

1 Tool's footprint shall be no greater than [**specify**] sq. in.

Figure G.3 – User has selected assembly tool in Figure G.2 and linked here.



## Automated Assembly Tool Specification Class

**Class Name:** automated_assembly_tool

**Inheritance:** assembly_tool : production_tool

**Children:**

1 workcell

## Requirements Specifications by Quality Factor

**System Integration**

1 Tool shall conform to standard factory utility package.
   1 100 or 208 volts AC, 20 Ampere service.
   2 100 PCI, clean air (not dry).
   3 20 hg vacuum source at 10 CFM

**Safety, Health, and Ergonomics**

1 Emergency stop button(s) shall be placed in conspicuous and easily accessible locations.
2 Activiation of emergency stop button(s) shall immediately shut down all power, air pressure, and vacuum.

Figure G.4 – User has selected automated in Figure G.3 and linked here.