

**Robot Orthogonal Defect Classification
Towards an In-Process Measurement System
for Mobile Robot Development**

Jack Silberman
jacks@ri.cmu.edu

CMU-RI-TR-99-05

January 1998

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890 USA

*Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
in Civil and Environmental Engineering*

Thesis Committee

John Bares, chair
Irving Oppenheim
Peter Santhanam, IBM T.J. Watson Research Center
Dan Siewiorek
William "Red" Whittaker

© 1998 Jack Silberman

Abstract

There is a current need in the mobile robot community for a measurement system that will transform mobile robot development into a measurable and controllable process. Experiences from previous developments are not being effectively recorded. Defects are sometimes quickly fixed and then forgotten. Certain defects recur repeatedly because new designers do not have the past experience or because a defect's cause was not properly recorded. This loss of information has a high cost and the trend must be reversed.

The method for addressing this problem involves collecting information regarding defects and their causes in the process of designing, producing, and using a product such as a mobile robot. When extracted and analyzed through the use of a data visualization and interpretation system, this information can be used to improve a product and process. Ideally, in the future this information will be provided to the development team during the development process (in-process) not just after the fact.

However, there are shortcomings of common analysis techniques (both quantitative and qualitative). Quantitative analysis does not consider origin, cause, or the effect of defects. Qualitative analysis does not abstract from details, so it is difficult to quantify process-related data. In order to improve a product, a methodology is needed that will draw on the advantages of these two systems while minimizing the disadvantages. The process measurement system developed in this thesis provides in-process feedback that takes advantage of the benefits of each method; that is, it extracts cause-effect relations and enables reliability predictions from quantifiable data. The method suggested here is Robot Orthogonal Defect Classification (RODC), which links quantitative and qualitative analysis in a systematic methodology. The goal of RODC is to generate an in-process measurement system that will extract information from classified defects with cause-effect relationships.

Supporting tools to enable data collection and feedback are developed based on the Internet World Wide Web technologies.

This research describes the RODC prototype developed at Carnegie Mellon University, Field Robotics Center and explores the future direction of this work.

“Being a scientist is like being a musician. You do need some talent, but you have a great advantage over a musician. You can get 99% of the notes wrong, then get one right and be wildly applauded.”

Dudley Hershbach

Acknowledgments

I do not think I can express in a text my gratitude and love to my wife Georgia for her love, support, patience, comprehension, ... (too many to list here!). She has great part of this Ph.D. Georgia, I hope I can pay back somehow your immensurable dedication and energy pushing me all the way up to the end of this work.

Thanks to my son and daughter who did not have a complete father during the past years. Pedro and Giovanna, I hope one day you will forgive me for not being present during this past 5 years. I will try hard to compensate this absence during these many years that we will have together.

To my parents David and Rosa, to my grandmother Chaya, and to my brother Jayme thank you for being there when I always needed. You are the best.

To my in-law family, especially Glycia, thanks for sharing the good and bad moments with me. Glycia's e-mail, talk, and webphone always kept me informed of what was going on with everyone in Brazil.

To Professor Sue McNeil, thanks for helping me start the dream.

To Dr. John Bares, thanks for making me develop and finish the dream. You sure made me more focused than lots of people I know.

To my thesis committee Dr. Dan Siewiorek, Dr. Peter Santhanam, Dr. William "Red" Whittaker, and Professor Irving Oppenheim, thanks for the great help.

To my "English guru" Julia, thanks for using your magic so many times on my text. It made it work.

To my friends Luciano and Cleia, Dimi, Eric, Ben, Deepak and Shivali, Sib, Mike, Kim, Matt, Jim, Fabio and Luciana, thanks for being what you are. It was a lot of fun sharing this experience with you. Special thanks to the Nomad people who had the patience to share information about Nomad development process defects in the good and bad moments.

To the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq-Brasil), thanks for the scholarship and financial support.

Even though RODC was not part of the primary objectives of the Nomad effort, the author received a great deal of support to implement the RODC system. To mention just a few individuals, the author gratefully acknowledges the following help: cooperation from developers; the loan of two computers used for development of software and the hosting of servers; acquisition of supporting documentation; and computer facilities fees. Once the RODC started to be understood by Nomad developers, cooperation in different phases of the RODC development significantly improved.

I am almost sure that I am forgetting someone. Well, thanks a lot for all of you that directly or indirectly helped me in this journey. Please forgive me for not mentioning your name here.

Table of Contents

Chapter 1: Introduction.....	1
1.1 Introduction.....	2
1.2 Motivation.....	4
1.3 Problem Statement	5
1.4 Measurement Systems Issues	6
1.5 Background.....	8
 Chapter 2: RODC Pilot.....	 34
2.1 Pilot Environment.....	35
2.2 Taxonomy Design.....	42
2.3 Data Collection	59
 Chapter 3: Data Analysis	 83
3.1 Validation	86
3.2 Information Extraction	107
 Chapter 4: Conclusions	 126
4.1 Lessons Learned	127
4.2 Contributions	132
4.3 Future Work	134
Appendixes	141
Appendix A - FMEA Tables.....	142
Appendix B - RODC Queries	144
Appendix C - RODC Survey	149
Appendix D - Logic Signatures Plots.....	151
Appendix E - Internet World Wide Web Based Tools.....	153
 References	 163

List of Figures

FMEA Steps	14
FMEA table with instructions	15
Security System block diagram and electrical schematic for the 5-volts Regulator	16
FMEA table 1-4	17
A FTA structure	19
Automatic air pumping system	19
Fault tree for the automatic air pumping system	20
Selection of FMEA vs. FTA	21
A reliability “bathtub” curve for electronics components	23
Failure and Replacement Report	25
The Two Ends of Reliability Analysis	26
Sample classification table	28
Defect type vs. triggers	29
Example of four defect types signatures	30
The Nomad robot developed at Carnegie Mellon University	35
The Atacama desert in Chile	36
Typical Mobile Robot Motion System	37
Typical Mobile Robot System	37
Components of a Typical Motion Subsystem	38
Nomad Motion Systems	38
Nomad Locomotion Subsystem	39
Nomad Pointing Subsystem	40
Parts for Nomad Motion Systems	41
RODC Classification Scheme Version 1.0	44
Final version of the RODC taxonomy interface.	46
Motion Systems Development Phases	52
Institutions Table	64
Personnel Table	65
Roles Table	65
Projects Table	66
Subsystems Table	67
Reliability Methods Table	68
Pictures Table	68
Projects and Institutions Table	69
Example of using the Projects and Institutions Table	69
Defects Table	70
Defect Type Table	70
Defect Source Table	71
Defect Cause Table	71
Development Phases Table	71

Defect Severity Table	72
Person Impact Table	72
Defect Triggers Table	72
Electrical Components Table	73
RODC table relationships	74
RODC tables in spreadsheet format	75
Interface using forms in Access	76
Example of a form linking two tables	77
RODC User Interface	78
Partial results of a query	79
Visual interface to generate queries	80
RODC Queries	81
Report for showing defect types and causes	82
Logic Signature for Specification	89
Nomad Defects	90
Pointing Defects	91
Locomotion Defects	92
Defect Signature Missing	93
Defect Signature Interaction	94
Defect Signature Interface	94
Defect Signature Damage	95
Defect Signature Performance	95
Signatures for Defect Type Interaction	96
Signatures for Defect Type Interface	97
Defect Type Damage Signatures	98
Defect Type Performance Signatures	99
Simulating Performance Tests	100
Defects Interfaces	101
Defects Performance	103
Nomad Defects Severity Distribution	110
Reduction of the number of defects to be studied - Costly Defects for Reliability	111
Nomad Defects Impact distribution	112
Reduction of the Number of Defects to be studied - Costly Defects for Management. ...	113
Nomad Defect Triggers Distribution	115
Development Phases Where Triggers Were More Effective	115
Most Effective Triggers for Specific Defects	116
Nomad Effective Triggers for Costly Defects	117
Nomad Costly Defects (Management) vs. Development Phases	118
Nomad Costly Defects (Reliability) vs. Development Phases	119
Nomad Defect Causes	120
Causes of Nomad Defects	121
Electrical Components Defects Distribution	122

Causes of Electrical-Related Defects	123
Mechanical Components Defects Distribution	124
Causes of Mechanical-Related Defects	124
Example of a Standard Defect Type - Function Signature	135
Defect Type - Performance Signature	136
Example of Defect Rate Model	137
FMEA Table 1-4	142
FMEA Table 2-4	142
FMEA Table 3-4	143
FMEA Table 4-4	143
RODC Queries	144
Assembly Logic Signature	151
Damage Logic Signature	151
Documentation Logic Signature	151
Esthetic Logic Signature	151
Interaction Logic Signature	152
Interfaces Logic Signature	152
Missing Logic Signature	152
Performance Logic Signature	152
RODC Hardware Scheme	154
RODC Software Scheme	156
RODC Home page	160
WWW Interface to the RODC System	161

Chapter 1

Introduction

This chapter introduces the research developed to design and implement a measurement system called Robot Orthogonal Defect Classification (RODC). The goal of the RODC system is to transform a mobile robot development process into a measurable and controllable process. The research addresses this problem by collecting information regarding defects¹ and their causes in the process of designing, producing, and using mobile robots. Moreover, the method developed here is used to extract and analyze data collected through data visualization and interpretation. Ideally, this information will be provided to a development team during the development process (in-process) not just after the fact. Supporting tools to enable data collection and feedback are developed based on the Internet - World Wide Web technologies.

The chapter is divided in five parts:

- Introduction
- Motivation
- Problem statement
- Measurement systems issues
- Background

1. Any change in a project artifact.

1.1 Introduction

It is logical that for a development process, such as for the development of mobile robots, to be controllable, process information has to be available. This information can be extracted from a process using a measurement system. Such a measurement system should provide enough information so decisions can be made based on quantitative and qualitative analysis. Information such as the most common problems (defects), the most common causes of defects, and the cost of the defects on the development process are examples of information that, after analysis, can improve the development process. If reliability is an issue in the development process, then the implication of these defects on reliability has to be captured as well.

Without the availability of process information, proposed solutions to process problems are based on guesses rather than management engineering [Chillarege, 1997]. That is, management and developers base their solutions to process problems on their guesses (i.e., on what they think might be the best solution to what they think the problem is). On the other hand, when process data is available (e.g., a graph containing a defect distribution and their causes), it can enable engineering decisions.

This research involves developing a measurement system that can be used to extract process information from mobile robot developments. In the process of designing, producing, and using a product such as a mobile robot, information regarding defects and their causes can be acquired and analyzed through the use of data collection, data visualization, and data interpretation systems. This information can be used to improve a product. That is, processes can be improved based on analyses of defect data. Thus, defects found during processes may be viewed as evidence of process deficiencies [Bhandari, 1993]. A methodology for collecting and displaying defects can also be used to deduce changes in system reliability by tracking and analyzing the number of specific defects during the life of the system (e.g., system reliability is derived from the number of defects that fail the system during a specified period of time).

If experiences from previous developments of mobile robots are effectively recorded, new development efforts can save time and money because they can use previous experiences to adjust the development process and prevent defects. For instance, if information such as the most common causes of

defects, the most effective triggers for identifying the defects, and the components that presented more defects are available, the development team can use it to better focus their efforts and decide where and when to spend resources.

If defect models are available, in-process feedback can be accomplished. This is done by comparing the defect data being collected during the development of a project with defect models. The comparison may provide evidence of process deficiencies. For instance, if the number of mechanical structure defects is noticed to be above an ideal threshold for a specific development phase, this would indicate a process deficiency. This, early identification can then enable the development team to address the problem (e.g., bringing in structural analysts to apply Finite Element Analysis techniques to the mechanical structural parts). Identifying and correcting defects in earlier development phases is important because generally there is a cost multiplier associated with each later development phase for which the defect is propagated [Chillarege, 1997].

To accomplish these goals, the following actions were followed:

- Development of the RODC pilot (pilot environment, taxonomy design, data collection)
- Analysis of data (validation and information extraction)
- Development of World Wide Web (WWW) based tools (RODC Hardware, RODC Software, RODC on the WWW)

This research introduces the use of defect models and describes how they can enable in-process feedback.

To be effective the RODC methodology must provide a set of tools that enable data collection and feedback. This research describes how tools were developed based on the Internet-World Wide Web technologies addressing ease of use and distributed client-server issues.

1.2 Motivation

There is a current need in the mobile robot community for a measurement system that transforms mobile robot development into a measurable and controllable process. Experiences from development projects are not being effectively recorded. Defects are typically fixed and then forgotten. Certain defects recur repeatedly because new designers do not have past experience or because a defect was not reported because of the lack of a measurement system. For instance, a developer might fix a mistake after spending some time trying to fix it. If this defect is not recorded, then the experience is lost (i.e., other developers can not learn from this experience). Moreover, because defects are not being recorded, statistical inferences can not be made to identify parts that are presenting more defects.

By using RODC, developers can identify the most effective triggers to specific defects. Thus, these triggers can be used to trigger defects in earlier development phases. Therefore, defects that would normally appear in later development phases can be fixed earlier. For instance, once users discover a mobile robot defect and its triggers are identified, developers may be able to trigger and fix these defects before the mobile robot is sent to customers. Also, if parts that present more defects are identified, management and developers can better decide the need and where to apply resources to solve parts-related problems.

Designing for reliability should not be a new adventure for each new product. Ideally, information stored from previous projects can be used to minimize expense by allowing designers and developers to identify potential defect trends and take appropriate actions [Ireson, 1996].

Changes in mobile robot reliability can be analyzed by tracking the number of defects that failed the system during a specific period of time. If applied to the development of mobile robots, RODC can provide a measurement system with quantitative and qualitative characteristics because RODC collects defects, their causes, and their effects. Thus, RODC can become a reliability tool accessible to all developers.

Such a system should enable product maturity monitoring, in-process feedback, and data collection for reliability evaluation. For instance, the RODC system can identify the effectiveness of design reviews as triggers for detecting defects, the most effective defect triggers, growth in reliability, and parts of the project where problems have been observed (e.g., defect rates being different from a defect model).

Automated tools and the use of the Internet (e.g., WWW browsers) can enable quick response time, interactivity, and ease of use in gathering in-process feedback. The WWW can also enable the use of a system by a large number of users, therefore enabling a large amount of data to be collected.

1.3 Problem Statement

No procedures exist for in-process feedback for a mobile robot development process. While in-process feedback techniques for monitoring defect rate goals, maturity, and reliability have been successfully used in other areas, they have not yet been used in mobile robot development.

Since no defect data collection techniques are being used and developed by the mobile robot community, no data is being collected in the process of designing mobile robots. This is unfortunate, because information extracted from collected data can save time and money as well as prevent errors, since it can be used to adjust the design process and prevent new defects.

1.4 Measurement Systems Issues

If a measurement system such as RODC can be successfully applied to the development of mobile robots, management and design teams will be able to change development course quickly, as necessary, before it is too late in the process. Thus, resources that would otherwise be spent on late modifications and redesign can be saved. Using this methodology, defects should be identified earlier and the overall number of defects should be reduced.

A development of a measurement system, such as RODC, needs to address the following issues:

- Taxonomy development
- Data gathering
- Validation
- Information extraction

Taxonomy Development

A taxonomy contains a set of attributes which guide how process information is captured. A taxonomy contains not just the types of defects, but many other necessary attributes in order that information from a development process can be captured (e.g., defect type, trigger, source, impact, etc.). The challenge here is to design a taxonomy to collect enough process information so as to enable inferences about the development process and parts. Moreover, the taxonomy should collect information that is considered useful by developers and management.

Data Gathering

Data gathering consists of collecting defects and related process information. The data gathering is based on the taxonomy design. That is, it is the process of collecting data using the taxonomy attributes. The challenge here is to get developers to commit to the data gathering process. Another challenge is the creation of a data gathering scheme and tools that support data collection during all development phases and that can be used at different locations (e.g., design offices, machine shop, etc.).

Validation

The end product of the RODC system is information about the development process of mobile robots. But before one can extract useful information from the collected data, it is necessary to validate the data collection scheme. The challenge here is to create procedures that can show that the data collection scheme is valid, that is, to develop procedures that illustrate that the data collection scheme is capturing known characteristic of a mobile robot development.

Information Extraction

The process of information extraction on the collected data has to be able to extract relevant information. Relevant information here means information that developers consider important to the development of mobile robots. The challenge here is to create procedures that can extract relevant information from the collected data, for instance, identifying the taxonomy attributes that can be used together to allow the extraction of relevant information. For example, one could use a graph containing defects vs. triggers to identify the most effective trigger for a specific defect.

1.5 Background

A quality management system (or a quality control program) establishes defect prevention actions and attitudes within a company or organization for the purpose of assuring conforming products or services on a permanent basis. It includes measurement activities such as inspections, tests, software evaluation, product qualification, and more [Crosby, 1986].

A quality control program is related to process observation. It identifies the variable characteristics of its content, and then tracks these variables using statistical methods. The monitoring of these variables, while they are being collected, permits control of the process to its specified efficacy limits. This prevents defects from occurring or at least ensures that they are identified at an earlier stage.

1.5.1 Improving Quality

Improving the quality¹ of a product may require changes in the overall business strategy [Montgomery, 1996]. That is, improving quality may require changes in activities within a company's organization. The following are activities that a quality control program influences within an organization [Crosby, 1986]:

- Engineering and manufacturing
- Management
- Marketing and sales
- Purchasing
- Training

Engineering and manufacturing

A quality program provides data related to manufacturing and customer experience with product use. It also influences design reviews, product qualifications, performance measurements, manufacturing processes, and testing.

1. Quality is one or more desirable characteristics that a product should have

Management

A quality program indicates to management the areas where problems are occurring and their causes; it provides quality status information and status charts visible in work areas.

Marketing and sales

A quality program provides data to assist in product sales (e.g., reliability and performance data), provides quality seminars for customers, helps handle complaints about the product and identifies ways to prevent these complaints.

Purchasing

A quality program enables source control and inspection and assists purchasing in vendor selection.

Training

A quality program provides employee orientation programs and conducts quality awareness activities within the company.

It is beyond the scope of this research to investigate all the organizational activities that are influenced by a quality improvement program. The focus here is on engineering, manufacturing, and management activities. This choice of focus is both due to the necessity of narrowing the research topic and because of the structure of the laboratory in which this research is being conducted and so as to have a more manageable problem. As stated previously, improving quality may require changes in company organization. Therefore, improving quality may in fact make it difficult to convince management to adopt a methodology such as RODC to improve mobile robot development process. Thus, minimizing changes to organizational activities aids the deployment of the RODC.

1.5.2 Quality Evaluation

Traditionally, the definition of quality is based on whether the product or service meets the requirements and expectations of those who use them. So the question is then how to evaluate (or measure) quality.

The quality of a product can be evaluated in several ways [Montgomery, 1996]:

- Aesthetics
- Durability
- Features
- Perceived quality
- Performance
- Reliability
- Serviceability

Aesthetics

Aesthetics are related to product appearance. Customers often consider the visual appeal of the product when evaluating quality.

Durability

Durability is basically related to effective service life. Customers want products that perform satisfactorily over a long period of time.

Features

Customers usually associate quality with added features (beyond the basics offered by the competition).

Perceived quality

Customers rely on past company reputation to evaluate quality. This reputation is negatively influenced by large publicly visible product failures and the way in which the company respond to failure-related problems.

Performance

A product usually is evaluated to determine if it will perform certain specific functions (is the product able to do the intended job?). Also evaluation is performed to determine how effectively the product performs its functions (e.g., how fast the product executes the function).

Reliability

A complex product such as a mobile robot will usually require some maintenance over its service life. But, if the mobile robot requires frequent repairs and thus the number of hours it can operate without intervention is low, then this indicates unreliability.

Serviceability

Serviceability is basically related to how easy is to repair the product. Customers view quality by how quickly and economically a repair or maintenance can be accomplished.

In this research the evaluation of quality is determined by reliability. That is, evaluating the quality of a mobile robot means evaluating the robot's reliability.

1.5.3 Evaluating Reliability

Currently, a standard method for recording reliability data does not exist [Ireson, 1996]. Most companies apply the same principles, but each company designs its own recording, analyzing, and retrieving systems. This section will introduce and summarily discuss the most common methods for qualitative and quantitative reliability analysis. It will highlight the gap existing between these analytical approaches and show a methodology that can be used to fill this gap.

Two types of analysis are typically used to enable reliability estimates or evaluation: qualitative (causal analyses) and quantitative (statistical). These two types of analysis offer different benefits and potential problems. Qualitative analysis identifies the root or origin of defects and their severity. This analysis isolates defect origins, so action can be taken to prevent each defect's occurrence or propagation. Defects are analyzed on an individual basis by investigation teams. Therefore, the resources necessary to conduct this type of analysis on a large project are significant.

Statistical analysis is used to predict product reliability as measured in terms of number of defects, failure rate, time between failures, etc. The data generated in this method provides information about the reliability of the product, but does not significantly contribute to the current product development cycle. This is because data is collected during product development and analyzed later, that is, after the process is complete. Ideally, however, developers would be provided with feedback during the development process (In-Process Feedback).

While both of these methods are useful, there are problems with both statistical and qualitative analysis. For example, statistical analysis does not consider origin, cause, or the effect of defects. On the other hand, qualitative analysis does not abstract from details, so it is difficult to quantify process-related data. In order to improve a product, a methodology is needed that will draw on the advantages of these two systems while minimizing the disadvantages. Ideally, a process measurement should give developers fast feedback that takes advantage of the benefits of each method; that is, it should extract cause-effect relations and enable reliability predictions from quantifiable data.

Orthogonal Defect Classification (ODC), a methodology developed at IBM T.J. Watson, links statistical defect models and qualitative analysis in a systematic methodology [Chillarege, 1994]. The goal of ODC is to generate an in-process measurement system that will extract information from classified defects with cause-effect relationships. ODC extracts information from defects using a well-defined set of attributes that form a classification scheme. Measurements are extracted from classified defect data; data analysis can show how the product is progressing when compared with a process model. ODC is applied to all stages of a development project. Because ODC categorizes defects into classes that can collectively highlight project deficiencies, project resources can be expended where they are most needed.

The ODC method is divided in two parts. In the first, process information is captured (classification). In this classification not only defect data, but also other attributes such as triggers (what caused the defect to be discovered), are captured. The second part is the analysis, during which relevant information can be extracted (information extraction). This analysis is performed on data classified by different attributes. The results of the analysis are provided to developers for process improvement

[Bhandari, 1993].

This research extends the ODC concept to the development of mobile robots including the necessary tools for interactive use. The method created here is called Robot Orthogonal Defect Classification (RODC).

1.5.4 Qualitative Analysis

Qualitative analysis uses a systematic process which anticipates and prioritizes failure modes and causes associated with the development process of a product [Moss II, 1996]. Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) are the most commonly used tools in qualitative analysis [Lazor, 1996].

FMEA and FTA are used to identify potential failures modes and related causes. They can be applied in early phases of the development process and then progressively refined through subsequent phases.

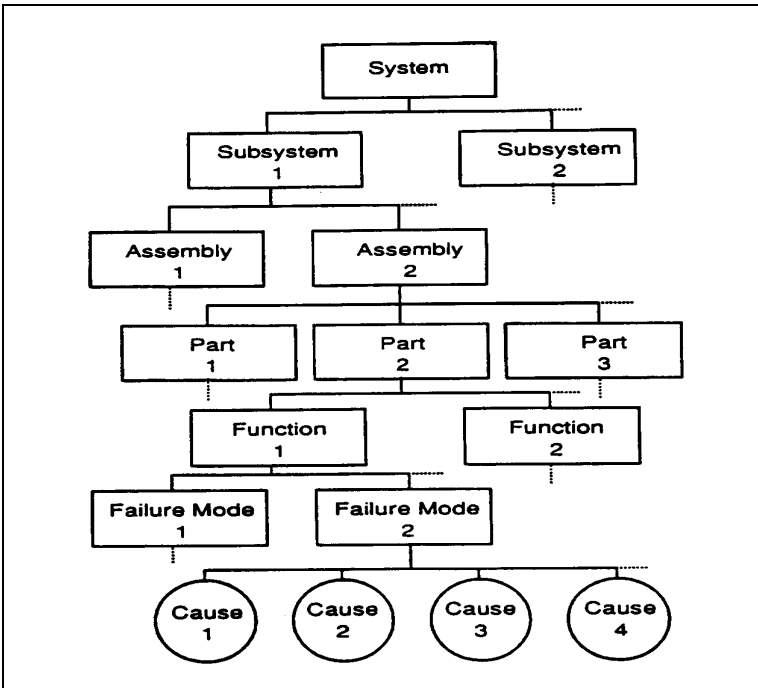
Failure Mode and Effects Analysis (FMEA)

The failure modes and effects analysis is a reliability evaluation and design review tool that analyzes potential failure modes from a system or its parts to determine the effects of failures on the system performance [MacDiarmid, 1997].

A typical FMEA analysis follows this sequence of steps [Lazor, 1996]: (Note: These steps are shown in Figure 1.1.)

1. Start at lowest level feasible for analysis (part level)
 2. Determine functional specification for the part
 3. Determine failure modes for each function
 4. Determine causes for each failure mode
 5. Determine effects for each failure mode on the next higher level, up to the overall system
 6. Analyze potential system failure modes
 7. Recommend actions that could eliminate or reduce chances of failure
-

FIGURE 1.1 FMEA Steps [Lazor, 1996].



The most common way of building FMEA is using custom designed tables. Figure 1.2 shows a typical FMEA table with general instructions in each cell. It is beyond the scope of this research to detail the FMEA and FTA building and analysis processes. The idea here is to illustrate the techniques and comment on qualitative analysis.

FIGURE 1.2 FMEA table with instructions [Lazor, 1996].

Page 1 Rows 1 through 2

Subsystem/Name 1 Suppliers and Plants Affected 4 Prepared By 7

Design Responsibility 2 Model Year/Vehicle(s) 5 FMEA Date (Orig.) 8 (Rev.) 8A

Other Areas Involved 3 Engineering Release Date 6

Part Name & Number Part Function	Potential Failure Mode	Potential Effects(s) of Failure	Severity	Potential Cause(s) of Failure	Occurrence	Design Evaluation Technique	D.P.N.	Recommended Action(s)	Area/Individual Responsible & Completion Date	Action Results				
										Actions Taken	S	O	D	R
<u>8A</u> <u>8B</u> Enter the Part Name and number. Also, enter the part function(s) being analyzed. If the part has several functions, list each function separately. Use the verb-noun format to describe a Part function.	<u>10</u> Enter the potential failure mode(s). Describe the failure mode as "loss of function" or in physical or engineering terms such as: -fractured -corroded -short circuit -loose	<u>11</u> For each Failure mode, list its consequences on: -Part function/performance -Next higher Assembly -System -Vehicle -Customer -Government Regulations		<u>14</u> Enter the 1st level cause(s) for each failure mode. List separately. For high Severity, list root causes (describe as a part characteristic). Assume: 1) Part is made correctly 2) Mfg/Assembly misbuild due to design deficiency (do not enter misbuilds due to process deficiency)		<u>16</u> Enter the methods, tests, or techniques to be used to detect the cause and/or failure mode before Engineering Release. Some Techniques may affect the occurrence of the cause. These are to be considered when estimating the Occurrence rating.		<u>19</u> List design actions that can reduce the Severity, Occurrence, then the Detection ratings. If no action, enter None at this time	<u>20</u> Enter: -Design Department -Design Engineer -Target completion data	<u>21</u> Enter a brief description of actions taken and their completion data.				
		Severity <u>12</u> → For each failure mode, rate the most serious effect. Enter rating in column 12. Use Severity Rating Table for Design FMEA If safety or compliance with government regulations are affected, enter a rating of 9 or 10		Occurrence <u>15</u> → Estimate the number of cumulative failures that could occur for a given cause over the design life of the part. Use Occurrence Rating Table for Design FMEA. ← <u>13</u> Potential Critical Characteristic		Detection <u>17</u> → For each Technique, estimate the likelihood the cause of the failure mode, or the failure mode, will be detected. If several methods are listed, enter the lowest (best) rating. Use the Detection Rating Table for Design FMEA.		← <u>18</u> RPN Risk Priority Number		Revised <u>22</u> → After actions have been taken, re-estimate ratings for Severity, Occurrence, and Detection. Enter revised ratings in columns to the right.				

The process of implementing FMEA has a high cost since the work is very labor intensive requiring that developers have high participation in the FMEA implementation. That is, not only developers work on the engineering aspect of the system development, but they are required to spend time participating in the FMEA implementation.

To illustrate this point, consider the following block diagram from a security system and the electrical schematic from a 5-Volt regulator. (See Figure 1.3).

FIGURE 1.3 Security System block diagram and electrical schematic for the 5-volts Regulator [Borgovini, 1993].

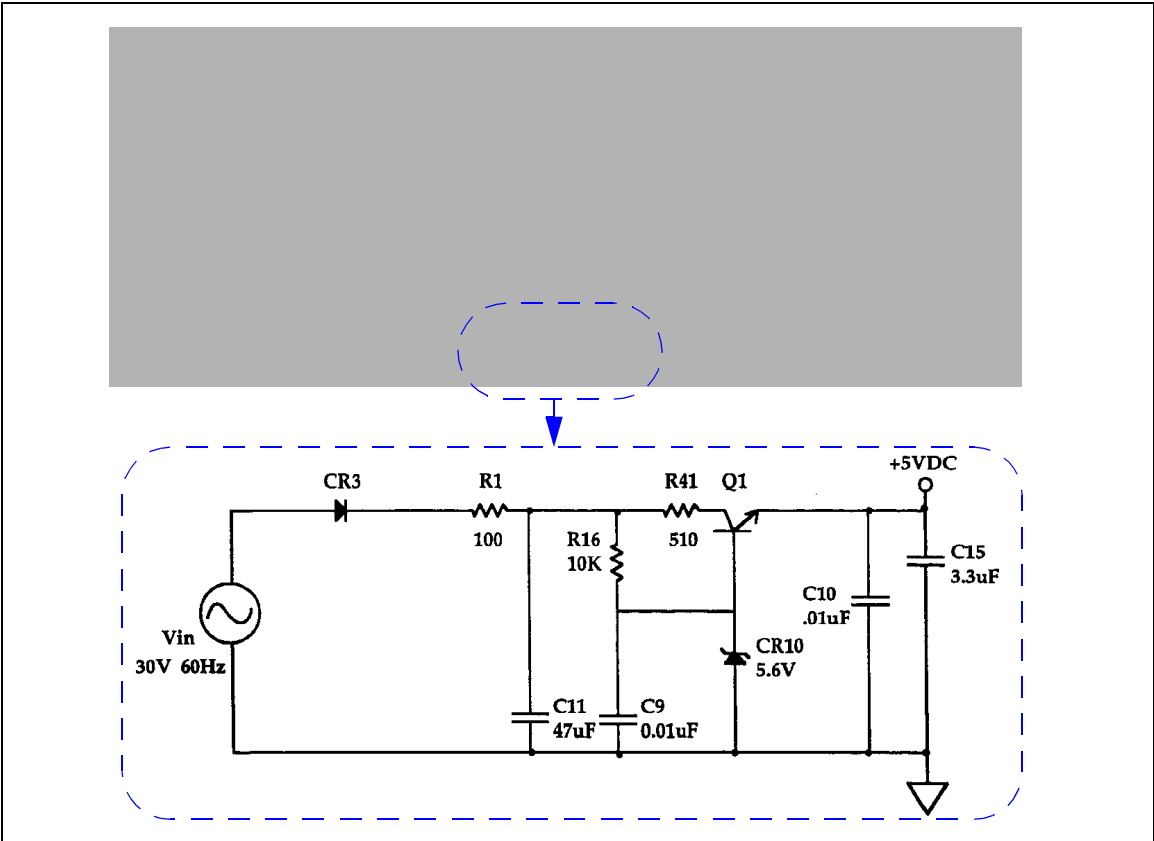


Figure 1.4 illustrate the implementation of a FMEA for the simple 5-Volt Regulator electronic circuit. Note: Figure 1.4 shows only one of tables of the FMEA. The complete set of tables can be found in Appendix A.

FIGURE 1.4 FMEA table 1-4 [Borgovini, 1993].

SYSTEM <u>Security System</u>					DATE <u>3/31/92</u>						
ASSEMBLY NAME <u>5VDC Regulator</u>					SHEET <u>1</u> OF <u>4</u>						
REFERENCE DRAWING <u>A123</u>					COMPILED BY <u>SLP</u>						
MISSION <u>Intrusion Detection</u>					APPROVED BY <u>RJB</u>						
I.D. Number	Item/Functional Identification (Nomenclature)	Function	Failure Modes and Causes	Mission Phase/Operational Mode	Failure Effects			Failure Detection Method	Compensating Provisions	Severity Class	Remarks
					Local Effects	Next Higher Level	End Effects				
001	CR3 Rectifier Diode	Half-Wave Rectifier	Short	Intrusion Detection	Loss of rectification of Vin	Loss of fixed 5VDC output	Loss of Alarm	None	None	I	
002			Open	Intrusion Detection	Loss of current to series regulator	Loss of output from 5VDC Regulator	Loss of Alarm	None	None	I	
003			Parameter Change	Intrusion Detection	Slight change in rectified voltage level	No change in output voltage	No effect	None	None	IV	
004	R1 Resistor Fixed Film 100 ohms	Current limit	Open	Intrusion Detection	Loss of current to series regulator	Loss of output from 5VDC Regulator	Loss of Alarm	None	None	I	
005			Parameter Change	Intrusion Detection	Slight change in input voltage level to Q1	No change in output voltage	No effect	None	None	IV	
006			Short	Intrusion Detection	Loss of current limiting protection	Possible overstress of circuit	Degraded operation	None	None	III	
007	C11 Capacitor Tantalum Elec 47µF	Filter	Short	Intrusion Detection	Loss of current supply to Q1	No output from 5VDC Regulator	Loss of Alarm	None	None	I	High current draw through CR3, R1, possible damage
008			Open	Intrusion Detection	Loss of filter for series regulator input	Possible instability in 5VDC output voltage	Degraded operation	None	None	III	

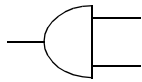
As can be noticed from this figure (and from the ones found in Appendix A), for the relatively small number of components present on the circuit the work of implementing the FMEA is very labor intensive. In addition, the maximum benefits of applying FMEA results are found in its early application in the development cycle rather than after the design is finalized [MacDiarmid, 1997]. That is, after the product is completed and is being used using FMEA provides no significant gain.

Fault Tree Analysis (FTA)

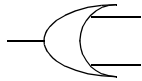
Fault tree analysis is a systematic methodology to determine all possible reasons (failures) that could cause a top event in a fault tree diagram. An FTA is an easier and faster method of analysis compared to FMEA [MacDiarmid, 1997]. An FTA can be a useful evaluation tool for aiding preliminary design modifications. For instance, developers can identify points of single failure mode.

FTA is derived from a logical schematic diagram used to represent faults and combinations of faults

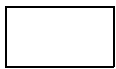
that can cause specific subsystem symptoms or higher level failures. The basic logic gates and event symbols used in an FTA are:



And - Provides an output event only if all input events occur



Or - Provides an output event if one or more of the input events are present



Rectangle - An event or a fault resulting from a combination of faults; also an event that still can be developed further



Circle - A basic event or fault that does not need to be developed any further

A typical FTA analysis follows this sequence of steps [Lazor, 1996]:

1. Identify and list top-level fault events to be analyzed
2. Do FTA for each identified fault
3. Use fault-tree symbols to present earlier and logic tree format; identify all contributing causes for the top-level event
4. Develop fault tree to lowest level of detail needed for analysis
5. Analyze corrective actions
6. Recommend necessary design changes

Figure 1.5 shows a FTA structure.

FIGURE 1.5 A FTA structure [Lazor, 1996].

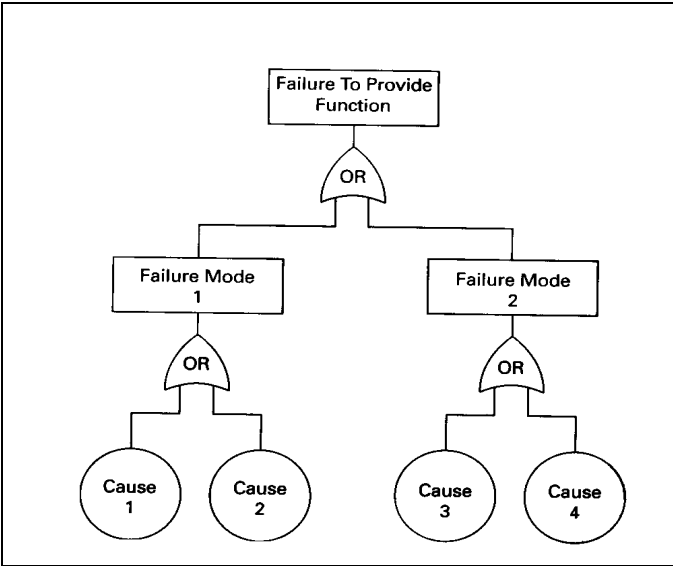


Figure 1.6 shows a component diagram for an automatic air pumping system. Figure 1.7 shows the FTA for this system.

FIGURE 1.6 Automatic air pumping system [Kececioglu, 1991].

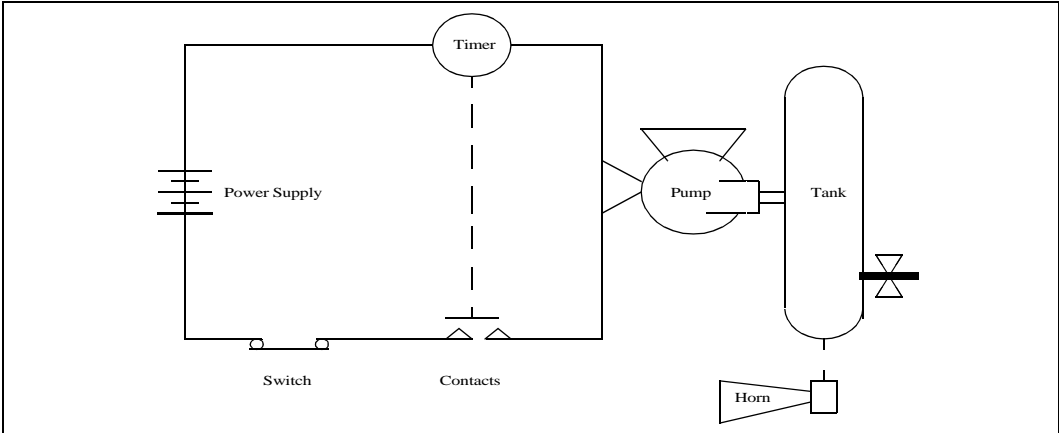
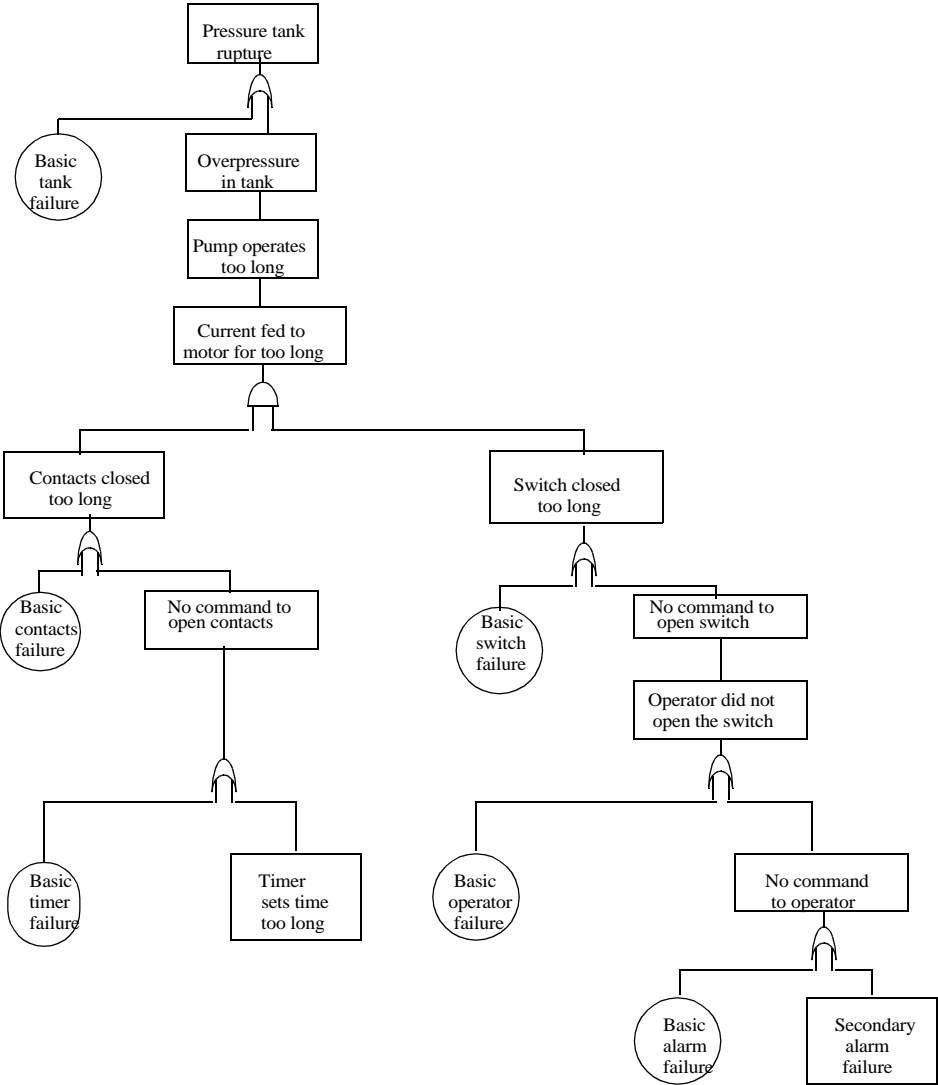


FIGURE 1.7 Fault tree for the automatic air pumping system [Kececioglu, 1991].



The FTA is a procedure that cannot be automatized easily. Therefore, it implies high costs (i.e., intensive manual labor).

Figure 1.8 shows a table with a comparison between FMEA and FTA [MacDiarmid, 1997].

FIGURE 1.8 Selection of FMEA vs. FTA [MacDiarmid, 1997]

Condition	FMEA preferred	FTA preferred
Primary concern is safety of public or operating and maintenance personnel		X
Primary concern is the identification of all possible failure modes	X	
Primary concern is a quantified risk evaluation		X
Completion of a functional profile is of critical importance		X
Multiple potentially successful functional profiles are feasible	X	
A small number of clearly differentiated top events can be explicitly identified		X
Top events cannot be explicitly defined or limited to a small number	X	
High potential for failure due to software error		X
High potential for failure due to human error		X
Product functionality is highly complex and/or it contains highly interconnected functional paths		
Product functionality is basically linear with little human or software intervention	X	
Product is not repairable once its function has been initiated (space systems)		X

Although the benefits of using qualitative analysis are proven, the methods require special training and are very labor intensive. Thus, the use of qualitative analysis in product development demands significant resources for implementation and maintenance. Also, quantification of process-related data is difficult when using qualitative analysis (e.g., validation of reliability programs).

1.5.5 Quantitative Analysis

Analysis of failure data is as important as analysis of the failures themselves [Moss II, 1996]. Using quantitative analysis, it is possible to identify frequent failures and efficiently apply resources.

Quantitative analysis produces numbers that can be used to compare two or more systems in terms of reliability. Also, qualitative analysis can be used to validate a reliability program (e.g., monitor a defect rate). Failure rate, reliability, mean time to failure (MTTF), and mean time between failure (MTBF) are commonly used parameters in quantitative analysis [Johnson, 1989].

Failure rate

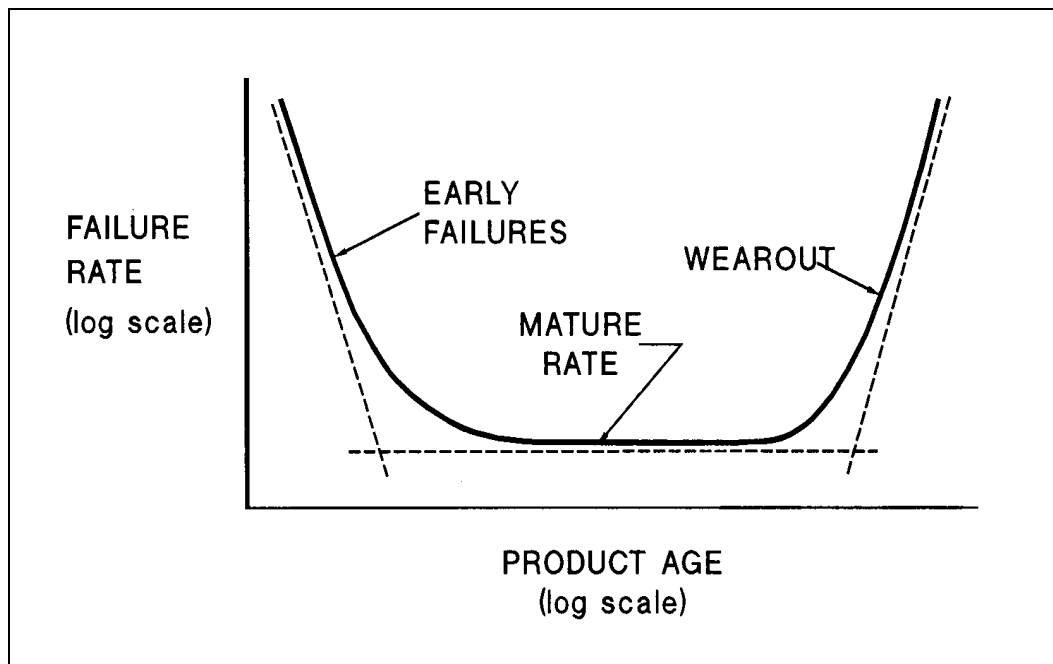
Failure rate is the expected number of failures of a device, subsystem, or system for a given period of time [Shooman, 1968]. For example, if a device fails, normally, once every 1000 hours, the device has a failure rate of 1/1000 failures/hour. The failure rate is expressed as λ .

Reliability

Reliability is the probability that a device, subsystem, or system will function correctly throughout the interval $[t_0, t]$, given that it was functioning at time t_0 [Johnson, 1989]. Reliability is expressed as $R(t)$. Many references show that if the failure rate (λ) is assumed to be constant the relationship between reliability and failure rate is represented as $R(t) = e^{-\lambda t}$. That is, for a constant failure rate the reliability varies exponentially as a function of time. This relationship is commonly used in electronic components to show the relationship between reliability and time. This is because experience has shown that the failure rate for electronic components does include a period where the value for λ is approximately constant [Johnson, 1989]. The relationship between the failure rate and time (for electronic components) is called the bathtub curve. (See Figure 1.9.)

If the failure rate can not be assumed to be constant, the above expression relating reliability and failure rate can not be used. Therefore another modeling scheme has to be used instead. For example, for software the failure rate should decrease as a function of time because, generally, software faults are discovered and fixed as the software is being used.

FIGURE 1.9 A reliability “bathtub” curve for electronics components [Moss II, 1996].



The bathtub curve can be interpreted as showing three phases. Early failures frequently result from failures occurring due to weak or out of standard components in the early life of systems. The mature rate assumes that the failure rate is constant. This phase indicates an expected useful life for the system. Wearout assumes that the systems have been operational for a long period of time and are beginning to experience failures. These failures are primarily due to the physical wear of components.

Plotting failure data enables various graphical analysis techniques. For instance, one could analyze a graph plotting failure data and infer whether a product is mature enough to be commercially released. That is, the failure rate is constant and low. As shown in Figure 1.9, an electronic product should be released when its reliability curve is at a “mature rate”.

Mean time to failure (MTTF)

Knowing about the mean time to failure is useful in specifying the quality of a system. Basically MTTF is the expected time that a system will operate before the first failure occurs. MTTF can be computed by averaging the time measurement of the first failure for identical systems placed into operation at the same time.

Mean time between failure (MTBF)

The mean time between failures is sometime used interchangeably with MTTF. The conceptual difference is significant but the numerical difference is small [Johnson, 1989]. The MTBF is the average time between failures of a system, whereas the MTTF is the average time until the first system failure appears. The MTBF is calculated by averaging the time between failures for identical systems. MTBF also includes the time to repair a system and place it back into operation. Calculating the average time to repair a system is often determined experimentally for different faults. As a result, it is difficult to estimate average time to repair a system.

Sources for reliability data

Sources for reliability data can be internal or external. Most companies place more importance on their own internally generated data than on data from external sources [Ireson, 1996]. Internally reliable data is typically generated by:

- Research tests
- Prototype tests
- Environmental tests
- Qualification tests

Manual methods are still important in the collection, retrieval, and analysis of reliability data [Ireson, 1996]. Generating reliability data (e.g., from reliability testing programs) usually requires that specific conditions be recorded in conjunction with failure itself in order to ensure test validity. A common use for internally generated data is the development of forms for classifying failures on an individual basis. (See Figure 1.10.) The forms are used to classify failures as they are identified and fixed. After the failures are classified and stored in some form of electronic format (e.g., databases or spreadsheet), qualitative analysis can be performed on the reliability data.

FIGURE 1.10 Failure and Replacement Report [Ireson, 1996].

ELECTRONIC EQUIPMENT FAILURE/REPLACEMENT REPORT DD-787 (PROPOSED)										REPORT BUSHIPS 10550-1				
1. DESIGNATION OF SHIP OR STATION					3. TYPE OF REPORT (CHECK ONE) 1. <input type="checkbox"/> OPERATIONAL FAILURE 2. <input type="checkbox"/> PREVENTIVE MAINTENANCE (POMSEE) 3. <input type="checkbox"/> PREVENTIVE MAINTENANCE (NOT POMSEE) 4. <input type="checkbox"/> STOCK DEFECTIVE 5. <input type="checkbox"/> REPAIR OF REPLACEABLE UNIT OR PLUG-IN ASSEMBLY 6. <input type="checkbox"/> OTHER					4. TIME FAIL. OCCURRED OR MAINT. BEGAN				
										MONTH	DAY	YEAR	TIME	
2. REPAIRED OR REPORTED BY NAME: _____ RATE: _____ AFFILIATION: _____					1. <input type="checkbox"/> U.S. NAVY 2. <input type="checkbox"/> CONTRACTOR 3. <input type="checkbox"/> CIVIL SERVICE					5. TIME FAIL. CLEARED OR MAINT. COMPL.				
										MONTH	DAY	YEAR	TIME	
6. MODEL TYPE DESIGNATION					9. FIRST INDICATION OF TROUBLE (CHECK ONE) 1. <input type="checkbox"/> INOPERATIVE 2. <input type="checkbox"/> OUT OF TOLERANCE, LOW 3. <input type="checkbox"/> OUT OF TOLERANCE, HIGH 4. <input type="checkbox"/> INTERMITTENT OPERATION 5. <input type="checkbox"/> UNSTABLE OPERATION 6. <input type="checkbox"/> NOISE OR VIBRATION 7. <input type="checkbox"/> OVERHEATING 8. <input type="checkbox"/> VISUAL DEFECT 9. <input type="checkbox"/> OTHER, EXPLAIN					10. OPERATIONAL CONDITION (CHECK ONE) 1. <input type="checkbox"/> OUT OF SERVICE 2. <input type="checkbox"/> OPERATING AT REDUCED CAPABILITY 3. <input type="checkbox"/> UNAFFECTED			11. TIME METER READING	
7. EQUIP. SERIAL NO.		8. CONTRACTOR (NAVY CODE OR COMPLETE NAME)			A. HIGH VOLTAGE		B. FILAMENT /ELAPSED		12. REPAIR TIME		MAN-HOURS	TENTHS		
REPLACEMENT DATA														
13. LOWEST DESIGNATED UNIT (U) OR SUB-ASSEMBLY (SA)	14. LOWEST DES. U/SA SERIAL NO.	15. REFERENCE DESIGNATION (V-101, C-14, R11, ETC.)	16. FEDERAL STOCK NUMBER	17. MFR. OF REMOVED ITEM	18. TYPE OF FAILURE	19. PRIMARY OR SECONDARY FAIL ?	20. CAUSE OF FAILURE	21. DISPOSITION OF REMOVED ITEM	22. REPL. AVAILABLE LOCALLY ?					
						P <input type="checkbox"/> S <input type="checkbox"/>			Y <input type="checkbox"/> N <input type="checkbox"/>					
						P <input type="checkbox"/> S <input type="checkbox"/>			Y <input type="checkbox"/> N <input type="checkbox"/>					
						P <input type="checkbox"/> S <input type="checkbox"/>			Y <input type="checkbox"/> N <input type="checkbox"/>					
						P <input type="checkbox"/> S <input type="checkbox"/>			Y <input type="checkbox"/> N <input type="checkbox"/>					
23. REPAIR TIME FACTORS								24. REMARKS						
CODE	DAYS	HOURS	TENTHS	CODE	DAYS	HOURS	TENTHS	(CONTINUE ON REVERSE SIDE IF NECESSARY)						

Also there are external data sources for reliability information. The most comprehensive of these data bases is the Government-Industry Data Exchange Program (GIDEP) [Ireson, 1996]. This database contains information on the design, production, and field operation of highly reliable products. A company can use the GIDEP by submitting all relevant reports and test results; it will then be eligible to receive the benefits of the program (e.g., access the database).

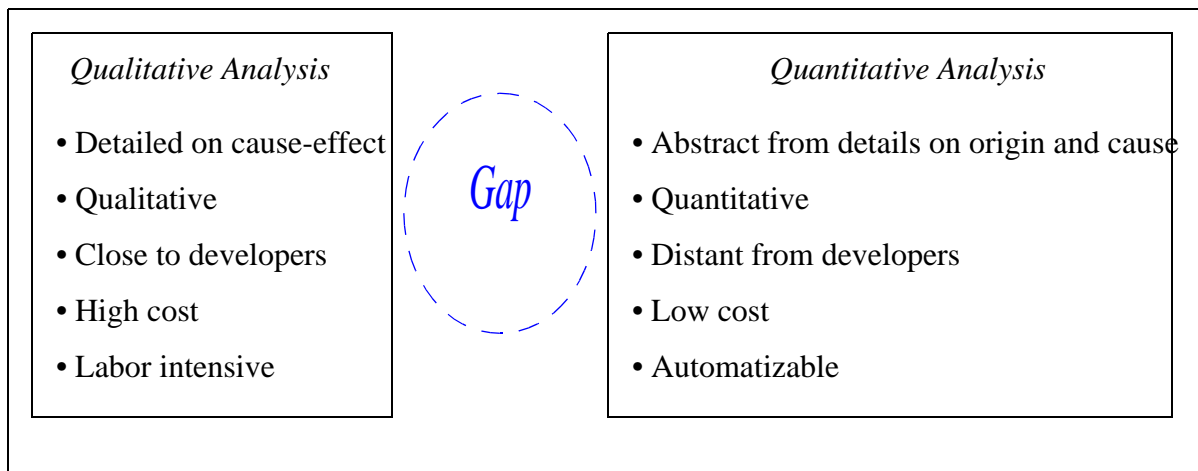
Data generated using quantitative analysis can produce information about the reliability of a product. However, this information usually is available after the fact, so it does not contribute to the current product development process. Therefore, in-process feedback is not available to developers. This is

because current practice does not try to change the development process based on data being collected from this same process.

1.5.6 The Gap Between Qualitative and Quantitative Analysis

Figure 1.11 illustrates two ends of the spectrum of reliability analysis. At one end is qualitative analysis which is detailed on cause-effect, close to developers, and it is labor intensive. At the other end is quantitative analysis which abstract from origin and cause, is distant from developers, and is automatable. These fundamental differences between analytical approaches creates a gap in the reliability analysis. This gap represents the need for a measurement system that can be used by developers and that can enable mathematical modeling.

FIGURE 1.11 The Two Ends of Reliability Analysis [Chillarege, 1996]



1.5.7 Orthogonal Defect Classification (ODC)

ODC has been applied to the development of software in over 50 projects at many IBM locations in recent years, ODC has reduced the cost of qualitative analysis by a factor of 10, while reducing software defects by a factor of 80 in a period of 5 years [Chillarege, 1996]. IBM claims that ODC provides fast feedback to developers and sufficient quantification to support management decisions while not overwhelming users with details. Therefore, ODC can be considered as a possible solution for filling the existing gap between qualitative and quantitative analysis.

ODC is based on a scheme for classification and analysis of software defects detected during all development phases [Halliday, 1993]. ODC enables fast feedback about a software development process during different development phases, not just after the project is completed.

The term *orthogonal* in the context of ODC is understood as follows: “*ODC essentially means that we categorize a defect into classes that collectively point to the part of the process that needs attention, much like characterizing a point in a Cartesian system of orthogonal axes by its (x, y, z) coordinates*” [Chillarege, 1992].

ODC is considered as an in-process system because it gives feedback to the developers during the process. Its methodology consists of two main parts: *Information Capture* and *Information Extraction*.

Information Capture

In information capture, process related data is captured using a classification scheme characterized by a well-defined set of attributes. This set of attributes captures defect information as well as process information. The following is an example of a set of attributes used on IBM’s ODC. (Note: these attributes are explained in more detail later in this document.)

Defect Type - The type of defect being classified (Assignment, Checking, Timing/Serialize, Algorithm, etc.)

Source - The origin of the defect (Reused-Code, Rewritten-Code, Refixed-Code, etc.).

Impact - How the defect impacts the system (Usability, Performance, Reliability, Instability, Migration, etc.)

Trigger - What has activated the defect (Design Conformance, Rare Situation, Concurrency, Operational Semantics, etc.)

Once attributes are chosen, the process of collecting defect data begins. For instance, a classification table such as the one shown in Figure 1.12 could be used to classify software defects. The idea here is to choose one, and only one option from each field of the table. That is, the ODC user classifying the defect is instructed to choose only one option from each available field. For example, the user can choose only one option from the Defect Type field (i.e., Assignment, Checking, Timing/Serialize, etc.). The data collected using the information capturer enables information extraction.

FIGURE 1.12 Sample classification table [Halliday, 1993]

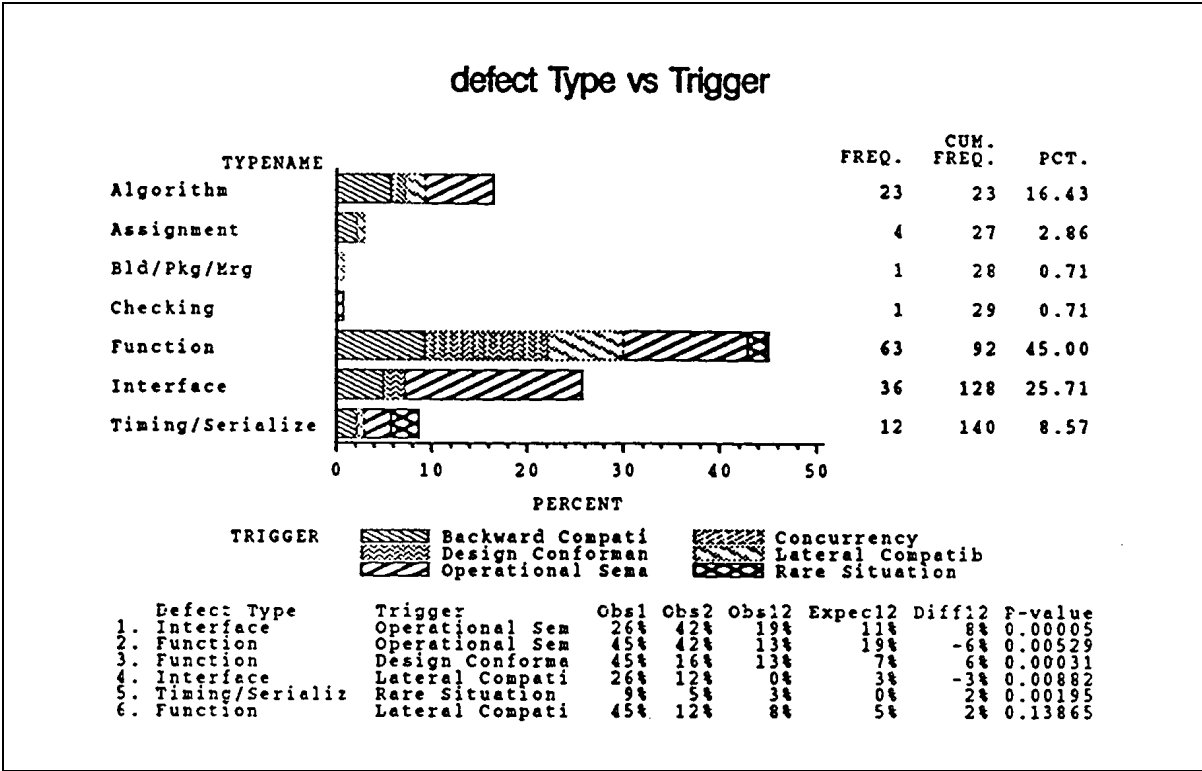
DEFCON 2.96- Defect Classification for project ATL		
Phase Found:	Defect ID:	Date:
Phase Intro:	Component:	Part:
Defect Type	Impact	Trigger
Assignment	Usability	Design Conformance
Checking	Performance	Rare Situation
Timing/Serialize	Reliability	Concurrency
Algorithm	Instability	Operational Semantics
Interface	Migration	Side Effects
Function	Maintainability	Backward Compatibility
Bld/Pkg/Mrg	Documentation	Lateral Compatibility
Documents	Integrity/Security	Language Dependencies
	Serviceability	Document Consistency
Source	Standards	
Reused-Code	Capability	
Rewritten-Code		
ReFixed-Code		
Vendor-Written		
Old-Function		
New-Function		
Scaffolded-Code		

Complete all fields and press ENTER to validate
F1= Help F3= Exit F12= Cancel

Information Extraction

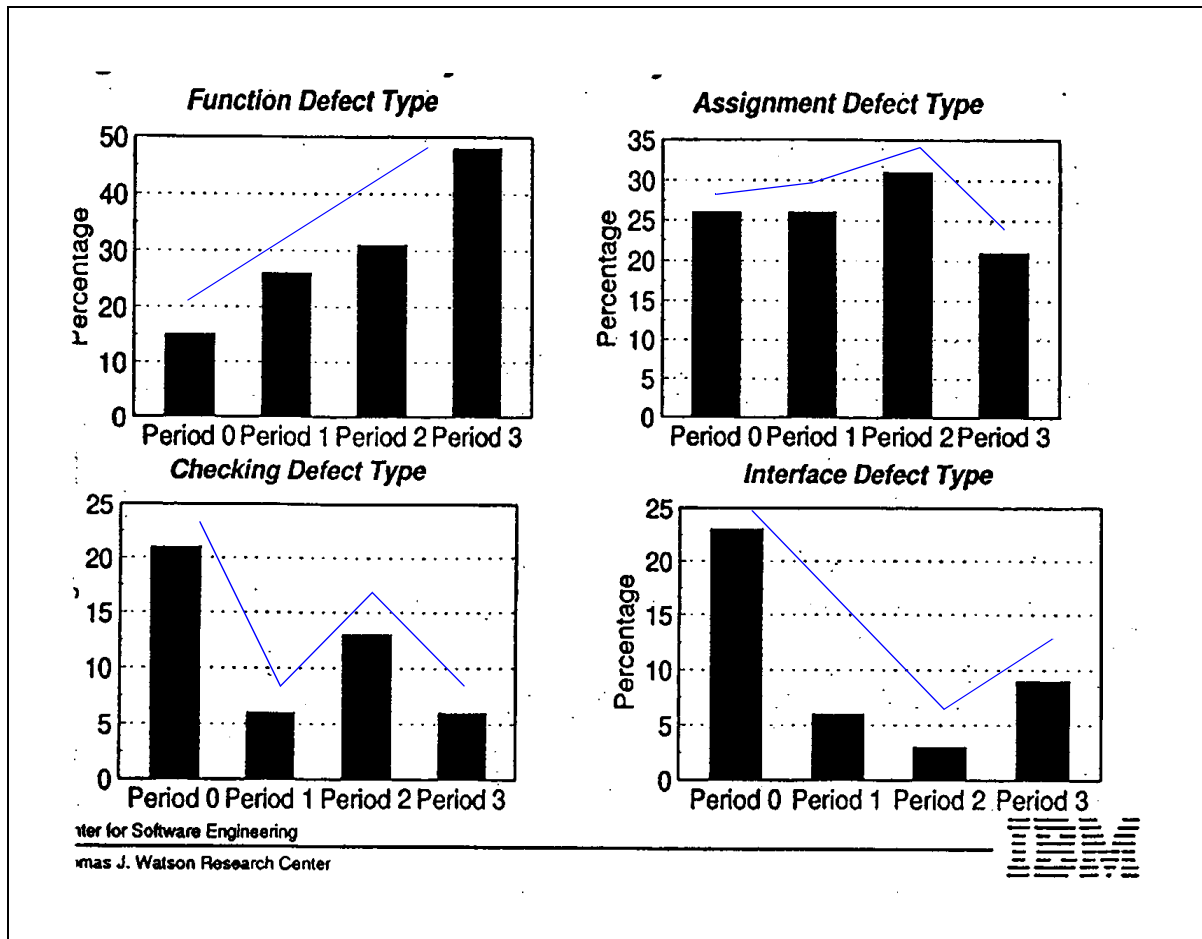
In information extraction, the classified data is analyzed. This analysis is performed using the different classification attributes (e.g., Defect Types and Triggers). Useful information can be obtained by displaying two attributes in a graph (e.g., what trigger was most effective for a defect type). Figure 1.13 shows a graph plotting defect types vs. triggers.

FIGURE 1.13 Defect type vs. triggers [Halliday, 1993]



Using defect types distributions changes over time, making it is possible to generate defect signatures. These signatures can be used as basis for processes health monitoring (i.e., comparing the monitored process signature with a model). Figure 1.14 shows four plots of defect types that can be used to extract defect signatures.

FIGURE 1.14 Example of four defect types signatures [Santhanam, 1996]



ODC has been applied to the development of software in over 50 projects in recent years. ODC provides feedback to management and developers while not overwhelming users with details [Santhanam, 1996]. Unfortunately, applying IBM's ODC to the development of mobile robots is not a straightforward process. Many key differences between pure software and electromechanical systems render the existing ODC useful as guideline but not as a direct tool. For example, one problem with using the current ODC is the difference between the Defect Types¹ used for software development and the Defect Types used for the development of complex electromechanical systems such as mobile robots.

1. Defect Types are attributes of the ODC classification scheme.

At the time of writing, no research on the use of ODC's for complex electromechanical systems containing moving parts has been conducted, and only one report about ODC's use on electromechanical systems with non-moving parts is available. The report describes ODC's application to the design of Wearable Computers [Amezquita, 1996], where an experiment applying ODC to two generations of Wearable Computers was performed. Defects were collected through interviews with developers that worked on the projects sometime long after the fact, and thus were limited to the items each person interviewed could remember. This totaled only eighty defects. No in-process feedback was attempted, as the application of the ODC was made after completing the computer's development. This work showed the possibility of applying ODC to the development of electromechanical systems, but was very limited in size and scope.

Issues

In order to modify and extend the existing ODC methodology, the differences between software and electromechanical hardware must be understood. Understanding these differences will enable the design of a taxonomy for the RODC to enable capturing of process information relevant to the development of complex electromechanical systems (mobile robots). The design of the RODC taxonomy is explained in chapter 2 (RODC Prototype). Additionally, in order to validate the new methodology, a validation process should be designed and performed.

Key differences include:

- Degradation
- Recovery
- Environment
- Component Maturity
- Interfaces
- Physical Interaction
- Severity and Impact
- Cause

Degradation - Hardware degrades very differently than software. For instance, once software is stable (or reliable) it won't "wear out" after a specific amount of time or number of use cycles. Hardware (for instance, mechanical gear) has a limited life time and may "age" differently depending on the specifics of component design and conditions of use.

Recovery - Most of the time recovery from software defects is performed by restarting the program. For example, in the case of a software crash, no rework is done on the code; recovery can be obtained by simply re-starting the program or re-setting the computer. For hardware, however, maintenance and repair is usually necessary after a defect has halted the system.

Environment - Influences from the environment (e.g., temperature, vibration, dust, radiation, etc.) can severely effect hardware. A hardware component may work perfectly in a certain environment but fail in another. Software, however, does not fail because of a change in the environment (although the computer itself might).

Component Maturity - Software components for mobile robots are not at the same level of commercialization as hardware. At least for mobile robots, software is generally developed from scratch. On the other hand, much of the hardware may consist of off-the-shelf components for which sharing of information about component performance is possible and recommended. As software components are generally unique, sharing of component information among projects is less likely.

Interfaces - Software interfaces are heavily dependent on specifications. If these specifications are correctly implemented, then interface problems can be avoided. On the other hand, hardware interfaces are dependent on specifications, tolerances, materials, assembly practices, etc. Even with well-defined specifications, correctly manufactured hardware interfaces can present defects over time (e.g., lubricant problems).

Physical Interaction - Hardware interaction is much more complex than software. Software modules (or components) do not suffer from physical interferences from other modules as hardware does. For instance, a hardware component might function acceptably in a certain physical location; however, if moved to a new location it might fail (e.g., because of the interaction of electromagnetic fields).

Severity and Impact - The impact of software defects (as considered in IBM's work) is always in terms of the user's point of view (as expected). No dangerous situations are considered in case of software defects (or faults) because of the kind of applications studied by IBM. But in the case of mobile robots, the impact can be more severe. A defect, either in hardware or software, might be catastrophic

(e.g., a mobile robot used on highways might cause a fatal accident in the case of a defect or failure).

Cause - The cause of software defects (as considered in IBM's work) does not consider hardware related causes such as transportation, storage, manipulation, etc.

RODC Supporting Tools

Tools to support the methodology (for instance, data collection) have to be created since IBM's tools were developed for internal use and are customized for use in IBM's software laboratories [Santhanam, 1996]. Tools are defined here as computer programs (e.g., software for data entry, data manipulation, graphics generation and display, statistical analysis, etc.).

The following factors should be addressed during the development of these tools:

User interface - Ease of use is fundamental in order to minimize mistakes.

Interactive feedback - Interaction is very important for in-process feedback.

Automation - Automated tools are required to enable interactive feedback.

Generalizability - The tools should be applicable to many projects.

Portability - The tools should be usable on a variety of computer platforms.

After expanding and considering the issues described in this section, a methodology based on ODC applied to mobile robots can aid data collection for reliability evaluation, management, and in-process feedback.

The next chapter will describe the RODC Prototype developed in this research.

Chapter 2

RODC Pilot

This chapter describes the development and implementation of a measurement system pilot called Robot Orthogonal Defect Classification (RODC). The extraction of information from the pilot is described in a separate chapter (Data Analysis). First, the pilot environment where the RODC was implemented is described. “Pilot environment” refers to the specific robot subsystem testbed and the mobile robot development where the methodology was applied. Secondly, the taxonomy design is explained. A taxonomy contains attributes that detail how information can be captured and where information about a process can be extracted.

Then the data collection implementation is explained. Data was collected using a classification scheme (paper-based questionnaire and computer programs) based on the taxonomy design.

The chapter is divided into three parts:

- Pilot Environment
 - Taxonomy Design
 - Data Collection
-

2.1 Pilot Environment

The pilot environment consists of the following:

- Pilot mobile robot
- Pilot testbed

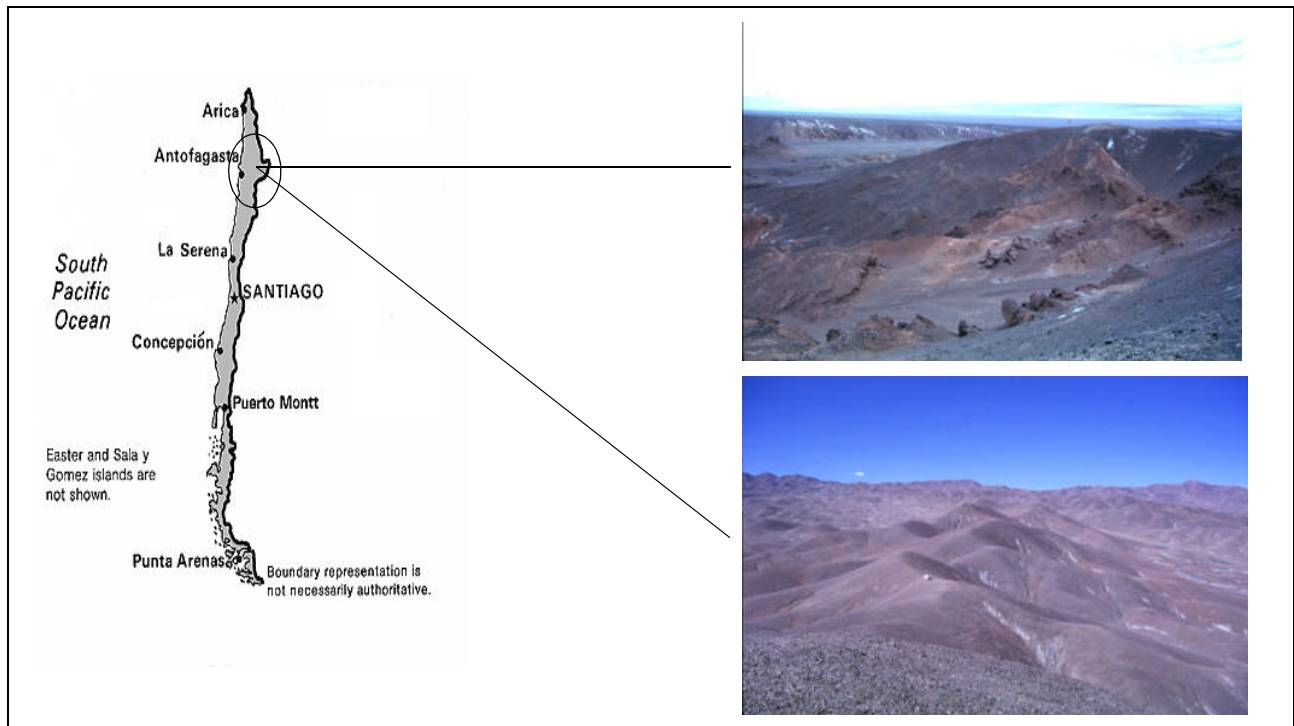
2.1.1 Pilot Mobile Robot

This research was developed at Carnegie Mellon University, The Robotics Institute, Field Robotics Center. The Pilot mobile robot where the research was applied is a mobile robot called Nomad. (See Figure 2.1.)

FIGURE 2.1 The Nomad robot developed at Carnegie Mellon University

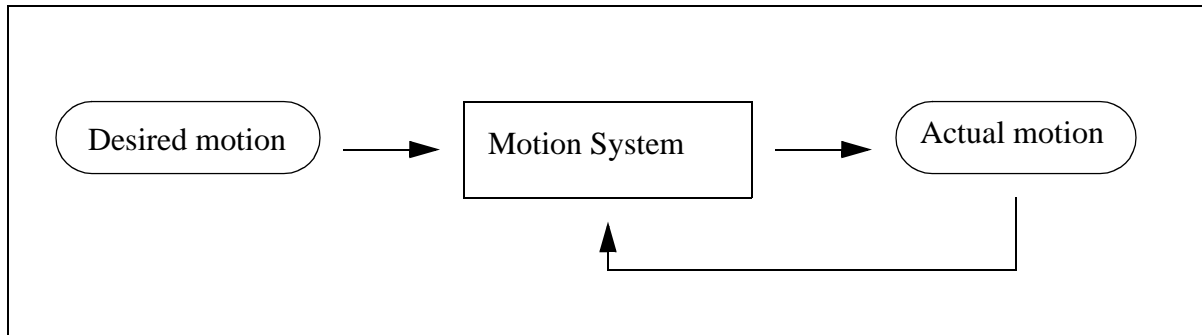


Nomad is a planetary-relevant mobile robot that traversed 200 kilometers across the Atacama Desert in Chile. (See Figure 2.2.) The decision to choose this machine as the pilot mobile robot was based on the author's participation as a member in the development team, and also because the development of the machine was almost exclusively in-house. Moreover, the majority parts of the analyzed sub-systems (Motion Systems) were manufactured in-house by the development team. Therefore, manufacturing defects were available.

FIGURE 2.2 The Atacama desert in Chile

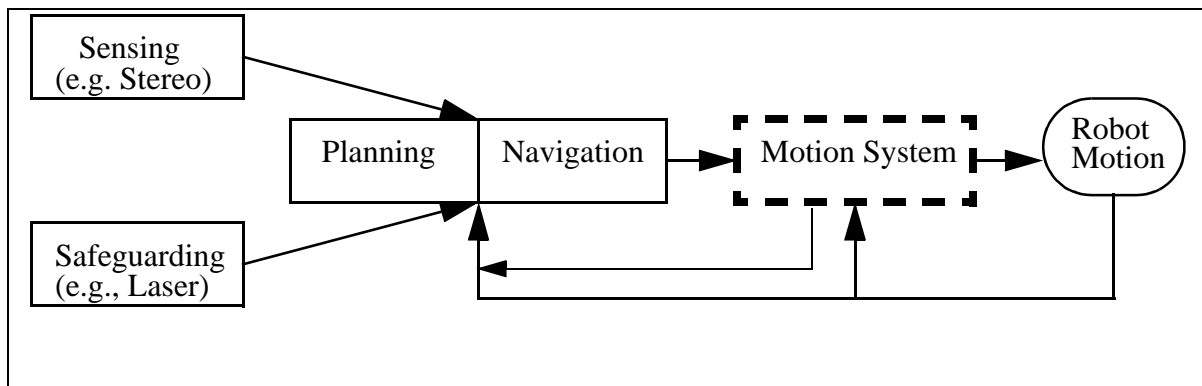
2.1.2 Pilot TestBed

This research created a RODC framework applied to the development of electromechanical motion systems. (See Figure 2.3.) The reason for using this particular type of system as a testbed is its common use on robotic systems. Most robotic systems, if not all, make use of a motion subsystem. Basically this motion subsystem enables mechanical motion from a received command. An example of this mechanical motion is the movement of the robot or part of the robot (e.g., wheels, legs, antennas, and laser scanner mirrors). By focusing the work on motion systems (motion generation and control), the scope of the research is clearly directed.

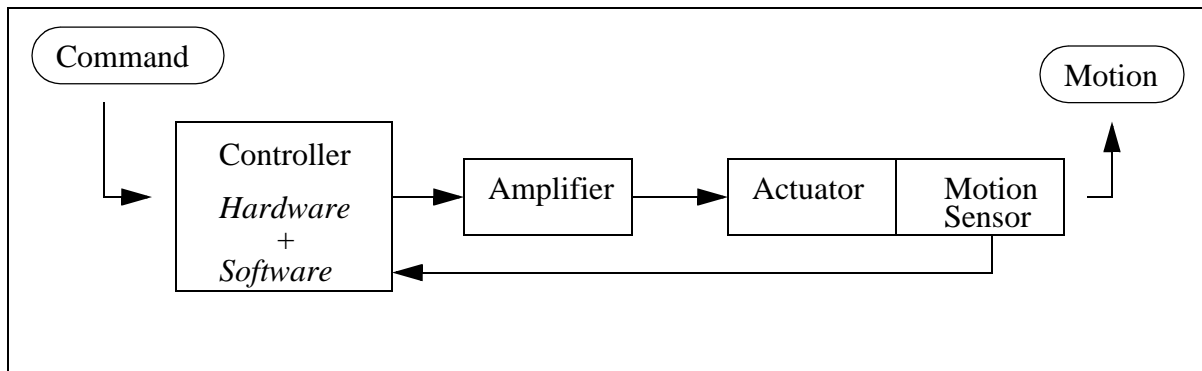
FIGURE 2.3 Typical Mobile Robot Motion System

With the focus on motion systems, subsystems that are heavily dependent on software and extremely diverse in function (such as navigation, safeguarding, and planning) were not considered. In doing so, the problem was made more manageable. (Figure 2.4 shows the relationship between a typical mobile robot system and the motion subsystem.)

Additionally, pneumatic or hydraulic motion systems are not addressed.

FIGURE 2.4 Typical Mobile Robot System

The RODC work described in this thesis does not consider software defects. Previous research on ODC applied to software exists [Bhandari and Halliday, 1993], so one can directly adapt IBM's scheme to the development of software intensive mobile robots subsystems in future work. In the future, generalization to other robotics subsystems should be possible because the motion subsystem consists of electronic components, electromechanical components (e.g., actuators), and software - the basic elements found on mobile robots. (See Figure 2.5.)

FIGURE 2.5 Components of a Typical Motion Subsystem

Two Nomad motion systems (that were developed in-house) were used as a source of data for this research: locomotion and antenna pointing. (See Figure 2.6 to Figure 2.8.)

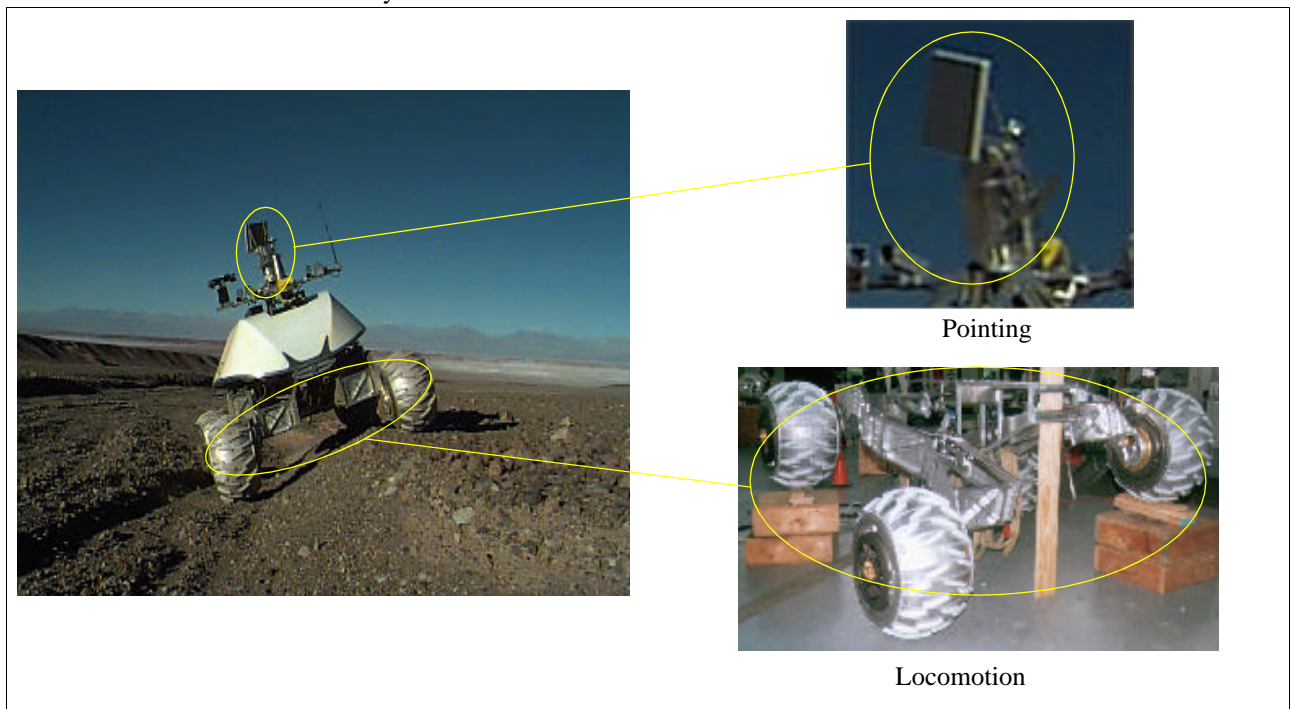
FIGURE 2.6 Nomad Motion Systems

FIGURE 2.7 Nomad Locomotion Subsystem

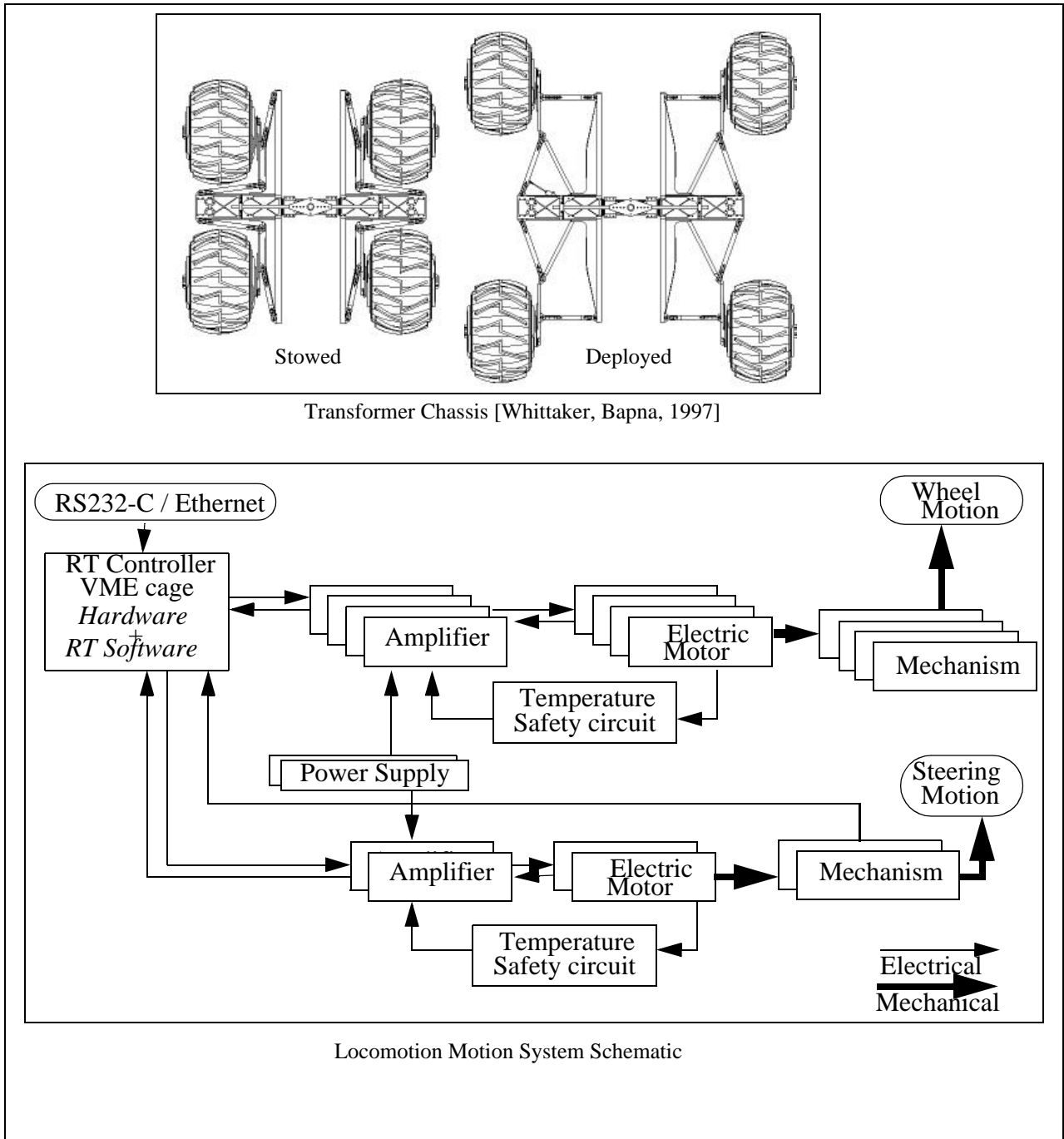
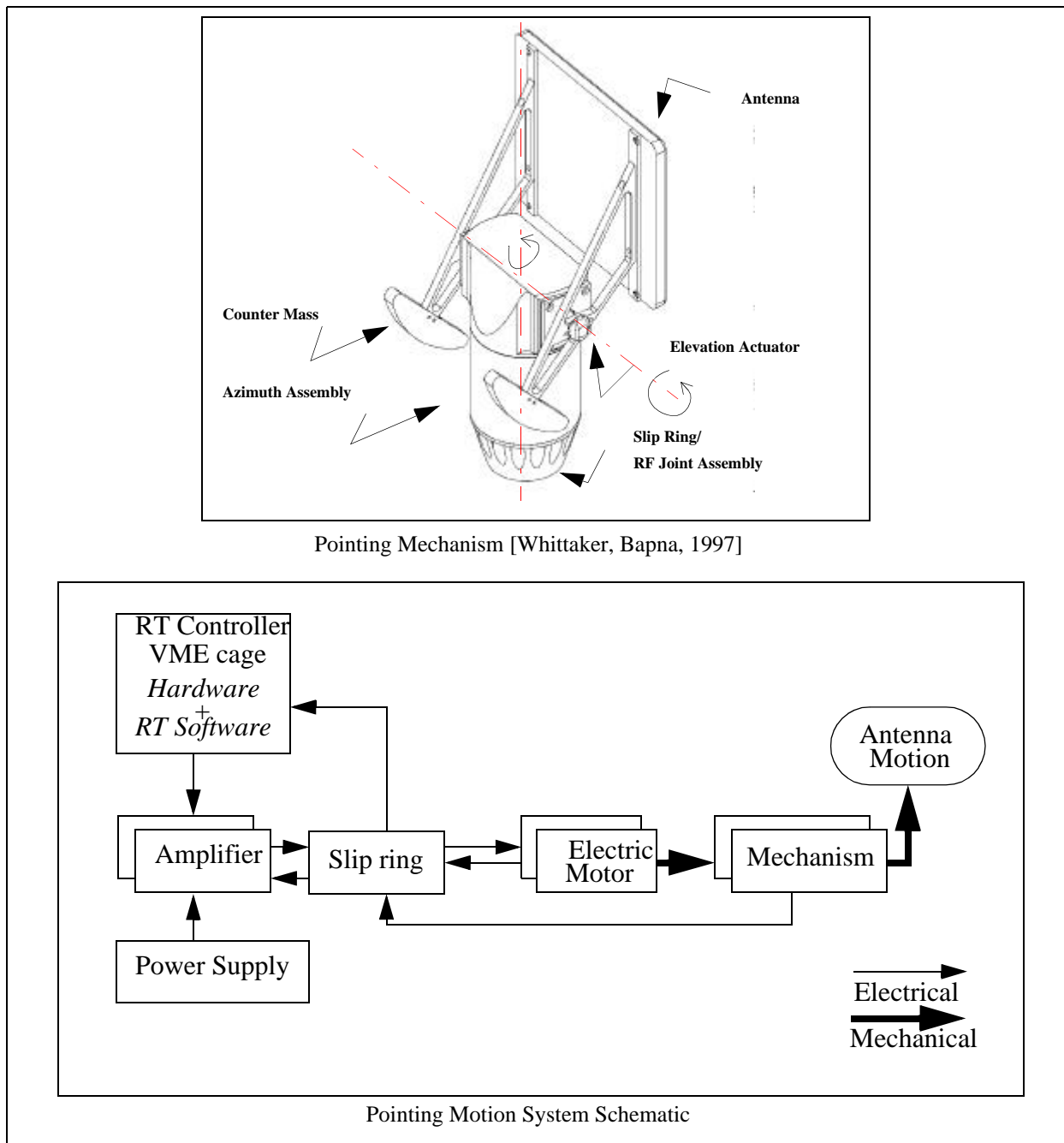
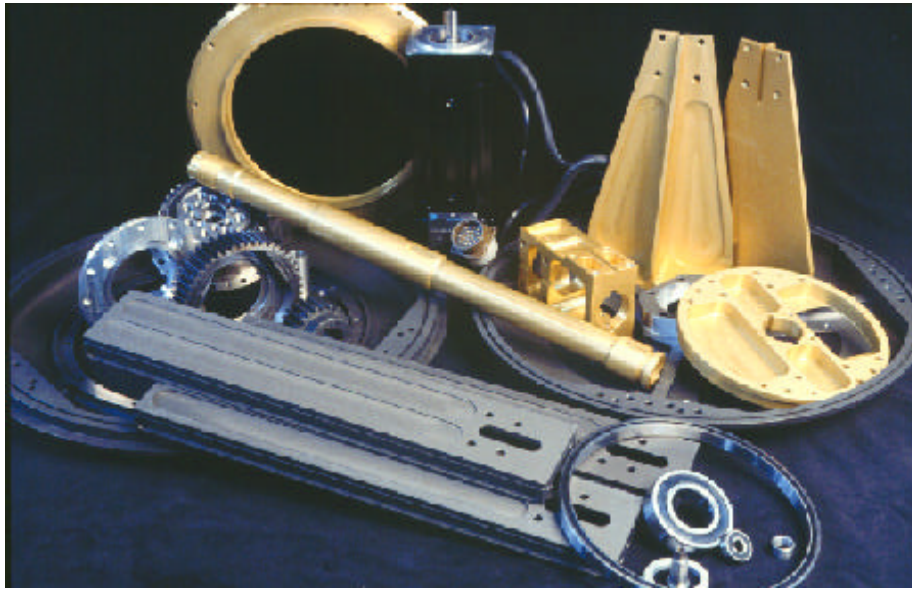
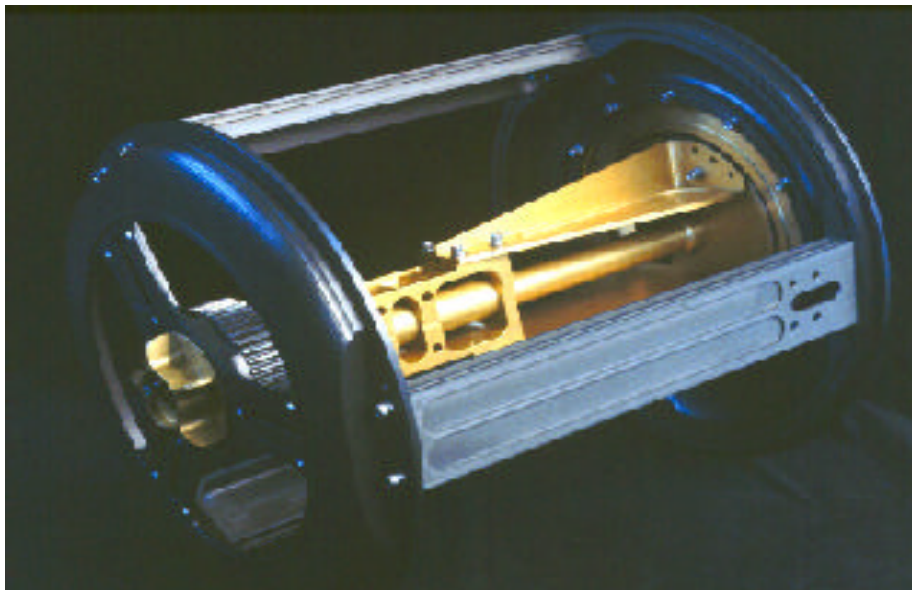


FIGURE 2.8 Nomad Pointing Subsystem

Having access to the entire process of designing, manufacturing, and assembly enabled collection of data for multiple development phases. Parts for the motion systems were mostly manufactured in-house. (See Figure 2.9.)

FIGURE 2.9 Parts for Nomad Motion Systems**Wheel Parts****Wheel Partially Assembled**

This section has described the pilot environment from which data collection was possible using the RODC measurement system. The next sections contain the design and implementation of the RODC taxonomy and the data collection scheme.

2.2 Taxonomy Design

In this section the taxonomy design is explained. This taxonomy includes not just defect types, but many other attributes needed for information extraction. Ideally the taxonomy should be independent of the specifics of a subsystem or robot design. Human error and confusion should be minimized by using a simple classification process that allows easy data entry. The goal in this work is to collect enough process information to enable inferences about the development process, not just defects.

The activity of collecting data represents one of the most important segments of a reliability program. A well-organized system for collecting reliability data is challenging [Lloyd, 1991]. For instance, having insufficient quantities or insufficiently detailed data can result in not being able to analyze the data effectively.

In designing a data collection system one should be concerned about the kind of data to be collected and why this data is necessary (i.e., what kind of data is important and how this data can be analyzed). Also, consideration should be given to how broad and detailed the coverage should be (i.e., the minimum amount of information required to satisfy needs [Lloyd, 1991].)

The design of a form should be tailored to the type of devices being analyzed. For instance, a form used for electronics components might not be useful for mechanical components. This is especially important for electromechanical systems, such as mobile robots. The need is for a simple form that can be used to classify both electronics and mechanical defects, thereby reducing effort needed to maintain the form and train developers.

Forms should be easy to use so errors can be minimized. One way to minimize errors is the use of checkmark options instead of description fields [Lloyd, 1991]. That is, the user does not have to type text for each option. This approach was used throughout the RODC development.

2.2.1 RODC Taxonomy Version 1.0

To illustrate the point that the taxonomy design should be well thought through, the evolution of the RODC will be explained next. Figure 2.10 shows the taxonomy version 1.0. A short description for the taxonomy attributes follows. (Note: These attributes are explained in more detail for the final version later in this section.)

Source - Origin of the defect component or module (vendor, new design, reused design)

Development Phase - Development phase where the defect was detected

Severity / Impact - How the defect impacts the system

History - Defect history (considered, not considered, known, supposed to be fixed, repeated)

Trigger - What caused the defect to be discovered (formal review, informal review, component test, system test, stress test, field test, user operation)

Defect Type - Classification of the defect (function, environment, interface, interaction, assembly, intermittent, damage, manufacture, documentation)

Electrical, Mechanical - Hardware component that presented the defect.

Software (taken from IBM Watson Research) - In this classification a working model adapted from IBM [Chillarege, 1992] was used.

Short Description - This information allows a better understanding of how the classification tool is being used by the user.

FIGURE 2.10 RODC Classification Scheme Version 1.0

<p>Project / Subsystem</p> <p>_____</p> <p>_____</p>	<p>Designer / Maintainer</p> <p>_____</p> <p>_____</p>	<p>Source</p> <p><input type="checkbox"/> Vendor</p> <p><input type="checkbox"/> New Design</p> <p><input type="checkbox"/> Reused Design</p>	<p>Date: __/__/__</p> <p>Page: __ of __</p>
<p>Development Phase</p> <p><input type="checkbox"/> Configuration Design</p> <p><input type="checkbox"/> Detailed Design</p> <p><input type="checkbox"/> Fabrication</p> <p><input type="checkbox"/> Assembly</p> <p><input type="checkbox"/> Integration</p> <p><input type="checkbox"/> Redesign</p> <p><input type="checkbox"/> Field Operation</p> <p>N/C _____</p>	<p>Severity</p> <p><input type="checkbox"/> Severity 1 (Failure -> Redesign is imperative)</p> <p><input type="checkbox"/> Severity 2 (May fail the system -> Redesign is recommended)</p> <p><input type="checkbox"/> Severity 3 (Will not fail the system -> Re-evaluate the design)</p> <p>N/C _____</p>	<p>History</p> <p><input type="checkbox"/> Considered</p> <p><input type="checkbox"/> Non Considered</p> <p><input type="checkbox"/> Known (Not new)</p> <p><input type="checkbox"/> Supposed Fixed</p> <p><input type="checkbox"/> Repeated</p> <p>N/C _____</p>	<p>Trigger</p> <p><input type="checkbox"/> Formal Review</p> <p><input type="checkbox"/> Informal Review</p> <p><input type="checkbox"/> Component Test</p> <p><input type="checkbox"/> System Test</p> <p><input type="checkbox"/> Stress Test</p> <p><input type="checkbox"/> Field Test</p> <p><input type="checkbox"/> User Operation</p> <p>N/C _____</p>
<p>Defect Type</p> <p><input type="checkbox"/> Function (Marginally used)</p> <p><input type="checkbox"/> Environment (e.g., Heat)</p> <p><input type="checkbox"/> Interface (Not compatible)</p> <p><input type="checkbox"/> Interaction (Interferences)</p> <p><input type="checkbox"/> Assembly (improper)</p> <p><input type="checkbox"/> Intermittent</p> <p><input type="checkbox"/> Damage</p> <p><input type="checkbox"/> Manufacture (≠Datasheet)</p> <p><input type="checkbox"/> Documentation (schematic)</p> <p>N/C _____</p>	<p>Electrical</p> <p><input type="checkbox"/> Connector/Connections</p> <p><input type="checkbox"/> Cables</p> <p><input type="checkbox"/> Soldering</p> <p><input type="checkbox"/> Electrical Noise</p> <p><input type="checkbox"/> Ground Loops</p> <p><input type="checkbox"/> Short Circuit</p> <p><input type="checkbox"/> Timing</p> <p><input type="checkbox"/> PCB Design</p> <p><input type="checkbox"/> Power Supply</p> <p><input type="checkbox"/> Mechanical Assembly</p> <p>N/C _____</p>	<p>Mechanical</p> <p><input type="checkbox"/> Structural support</p> <p><input type="checkbox"/> Welding</p> <p><input type="checkbox"/> Rivets</p> <p><input type="checkbox"/> Fasteners</p> <p><input type="checkbox"/> Gear / Drive</p> <p><input type="checkbox"/> Bearing</p> <p><input type="checkbox"/> Sealing Component</p> <p><input type="checkbox"/> Motor</p> <p><input type="checkbox"/> Brake</p> <p><input type="checkbox"/> Clutch</p> <p><input type="checkbox"/> Appropriate material</p> <p><input type="checkbox"/> Lubricant</p> <p><input type="checkbox"/> Contact Sensor</p> <p><input type="checkbox"/> Vibration</p> <p><input type="checkbox"/> Noise</p> <p>N/C _____</p>	<p>Software(IBM®)</p> <p><input type="checkbox"/> Assignment</p> <p><input type="checkbox"/> Interface</p> <p><input type="checkbox"/> Checking</p> <p><input type="checkbox"/> Timing / Serial-ization</p> <p><input type="checkbox"/> Build / Package / Merge</p> <p><input type="checkbox"/> Documentation</p> <p><input type="checkbox"/> Algorithm</p> <p>N/C _____</p>

Short Description: (briefly describe the defect)

There were several problems in Taxonomy version 1.0:

Trigger Attribute- The power of capturing defect triggers is the capability of identifying the activities that discovered the defect. The set of options for the triggers in the RODC version 1.0 (e.g., formal review, informal review, etc.) did not explicitly identify the activity. For instance, to say that a formal review identified a defect is not good enough to be used as advice [Santhanam, 1997]. What was done during the review is what is important (e.g., design conformance, compatibility check, etc.).

Cause - To provide a deeper qualitative analysis characteristic to the RODC taxonomy, an attribute to capture the cause of the defects was added.

Defect Type Attribute - The defect type attribute covered some of the defects that were being found during the data collection. But modifications were necessary so defects could be classified. Modifications such as adding: Performance, Specification, Missing Component, Esthetic, and Unclassified were necessary to allow defect classification. (Note: These defect types will be explained in detail later in this section.)

Software - No software defects were collected (as explained in Data Collection). So the software part of the defect classification was removed.

Impact Personnel- An attribute called Impact Personnel was added. This attribute was used to capture the impact of the defect on personnel. This attribute is important for management feedback.

Development Phases - Changes were made to the development phases to better represent the development of motion systems on a mobile robot project.

As in the IBM ODC, the set of attributes used in the RODC was experimentally verified (e.g., are the current triggers sufficient to describe the activities that discovered a defect?, Is there any redundant information being collected?). The taxonomy was modified several times to include attributes that enabled orthogonal classification of defects and extraction of development process information.

Next the design of the final version of the RODC taxonomy is described.

2.2.2 RODC Taxonomy Version 2.0

Figure 2.11 shows the final version of the RODC taxonomy interface. (Note: The attribute software was removed from the taxonomy and some others were added.)

FIGURE 2.11 Final version of the RODC taxonomy interface.

The screenshot displays the 'tbIDefects' application window. The form contains the following fields and values:

- Defect ID: 30
- Defect grouped #: 1 of 1
- Date Defect Found: 2/28/97
- Subsystem's Name: Nomad Pointing for Communications - Nomad
- User Name: Jack Silberman
- Source: New Design
- Cause: Assembly (improper)
- Development Phase: (4) Integration of Components
- Severity: Prevented Assembly (needed Fix)
- Impact Personnel: 2 to 3 Persons
- Considered during Design:
- Time to Fix (Hs): 15
- Not Known by the Developer:
- Time to Prevent (Hs): 0
- Trigger: Inspection & Pre-Assembly
- Defect Type: Damage (need fix)
- Electrical Component: Slip-Ring
- Mechanical Component: None
- Unclassified fields Description: (Empty text box)
- Notes: Collected from Mark Sibenac
The slip ring inside the pointing mechanism broke after 3 cycles of assembling and disassembling.
The time that would take to prevent this problem is unknown -> (0), maybe it is no applicable here.
- Last Updated: 5/11/97
- Record: 29 of 465

The evolution of the taxonomy to the final version was based on a design process and on experimental verification. The goal during the taxonomy development was to verify that the RODC taxonomy allowed enough process data to be collected to: enable reliability information extraction and process data to aid in project management.

2.2.3 RODC Taxonomy Attributes

In this section the taxonomy attributes (or fields) will be explained. Some complex attributes (i.e., *Trigger*, *Development Phase*, and *Defect Type*) will be detailed later in this section.

Defect ID - An unique ID for the defect.

Defect grouped # - Field to aid data input. For instance, if the user needs to type 10 related defects, this field works as a counter.

Date Defect Found - Date when the defect was found.

Subsystem's Name - Name of the subsystem where the defect belongs (e.g., Nomad-Locomotion).

User Name - The person entering the defect data.

Source - Origin of the defect component or module:

- New design
- Reused design
- Vendor
- Unclassified

Cause - The cause of the defect:

- Miscommunication (human factors, documentation)
- Change or misunderstanding of requirements
- Design
- Manufacturing (<> datasheet)
- Assembly (improper)
- Environment (e.g., heat)
- Storage
- Unclassified

Development Phase - The development phase in which the defect was found:

- (1) Requirements and configuration
 - (2) Design
-

- (3) Fabrication
- (4) Integration of components
- (5) Performance test
- (6) Integration to robot
- (7) Field performance test
- (8) Long term operation

Severity - The severity of the defect to the development of the subsystem:

- Prevented design (forced a redesign)
- Prevent manufacturing
- Prevented assembly (needed fix)
- Failure (fix or redesign is imperative)
- May produce a failure in future
- No risk of failure in future
- Unclassified

Impact Personnel - The impact of the defect on personnel (how many developers were impacted by this defects):

- 1 person
- 2 to 3 persons
- More than 3 persons
- Unclassified

Considered during Design - Was this defect considered during design? (Did developers consider this possibility during design?)

Not Known by the Developer - Is this defect known by the developer? (Did developers see this before?)

Time to Fix (hrs.) - How long did it take the development team to fix the defect?

- 0.5 ~ 200 hours
- Unclassified

Time to Prevent (hrs.) - How long would it take to prevent this defect (e.g., rechecking the drawings)?

- 0.5 ~ 200 hours
- Unclassified

Trigger - What was the developer activity that revealed the defect?

- Requirements conformance
- Configuration conformance
- Design conformance
- Previous design check
- Compatibility check
- Documentation check
- Inspection & pre-assembly
- Component performance test
- System performance test
- Stress test
- Field test
- User operation
- Unclassified

Defect Type - The type of defect:

- Performance (not working as expected)
- Damage (needs repair)
- Specification (incorrect or change)
- Interface (incompatibility)
- Interaction (interferences)
- Documentation (schematics, instructions)
- Missing component
- Esthetic (appearance)
- Assembly process (improper)
- Unclassified

Electrical Component - The type of electrical component that presented the defect:

- None
 - Connector/cable
 - Soldering
-

- PCB
- Power supply
- Servo amplifier
- Servo motor
- Optical switch
- Mechanical switch
- Slip-ring
- Encoder
- Potentiometer
- Electromechanical brake
- Electromechanical clutch
- Unclassified

Mechanical Component - The type of mechanical component that presented the defect:

- None
- Structural support
- Welding
- Rivets
- Fasteners
- Gear / drive
- Bearing
- Sealing component
- Lubricant
- Axle / shaft
- Unclassified

Unclassified fields Description - Explanation of why a defect was not classified on all fields (attributes). For instance, it could be a request to add a new mechanical component to the list.

Notes: - This field is basically used to describe the defect during the development of the taxonomy (short description of the defect). For instance, if the taxonomy changes the new classification would still be possible by reading the “*Notes:*” field.

Last Updated - Last time the defect was updated.

It is important to describe some complex attributes (fields) of the taxonomy. The next section will detail the design of complex attributes.

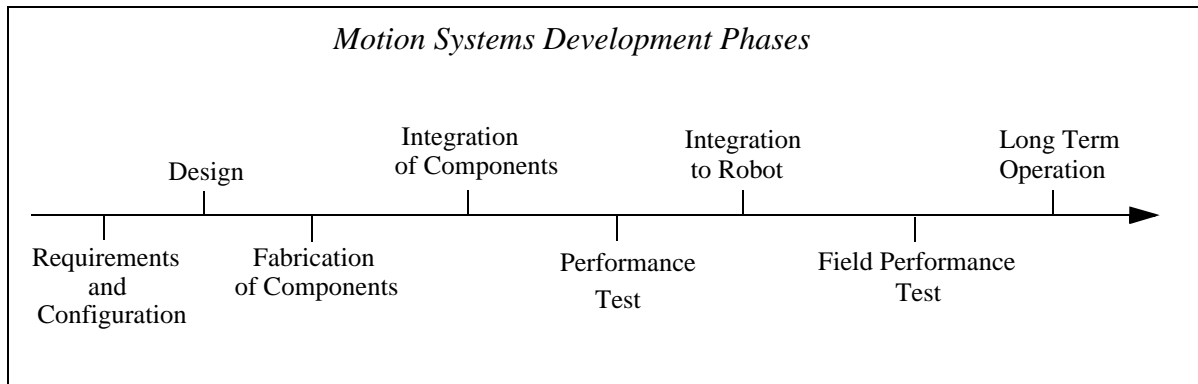
2.2.4 Complex RODC Taxonomy Attributes

In this section the design of the following complex taxonomy attributes will be detailed:

- Development Phases
- Triggers
- Defect Types

Development Phases

Motion systems development phases might start in a late phase compared to other subsystems of a mobile robot. This is because the motion system requirements (or problem definition) need data from other subsystems (e.g., what are the components that need to be controlled?, how many actuators?, how many sensors?, etc.). These phases usually happen at different times for different subsystems. For example, some parts of the Nomad locomotion system were being fabricated before production of the pointing mechanism (motion system) had begun. This means that if one was trying to classify defects from these two subsystems during the same development phases (i.e., a common scheme for the entire robot development phases) he or she would have problems classifying the defects in the development phases. Figure 2.12 shows the development phases used in this research. (Note: These development phases may not be general. They are typical for a particular research center; the Field Robotics Center.)

FIGURE 2.12 Motion Systems Development Phases

The development phases will be described next.

Requirements and Configuration - Requirements are studied and detailed. Different design alternatives are evaluated (the configuration is detailed enough to enable *Design*). A configuration design review is performed.

Design - Different design alternatives (that fulfill the requirements) are evaluated. Documents containing schematics and parts are generated to capture the final design (the design is detailed to component level enabling *Fabrication of Components*). A detailed design review is performed.

Fabrication of Components - Raw material and components are acquired. Parts are manufactured (structural, driving mechanisms, printed circuit boards, etc.). Parts are assembled forming Subsystem Components (i.e., mechanical and electrical). Pre-tests and modifications are performed.

Integration of Components - Components are integrated to form the Motion System. Pre-tests and repairs are performed.

Performance Test - Tests are performed to check subsystem requirements and design satisfaction. Repairs are performed fixing discrepancies.

Integration to Robot - The motion subsystem is integrated with the mobile robot system. Pre-tests and repairs are performed as needed.

Field Performance Test - The motion subsystem is tested during a mobile robot demonstration or test on field operations. Tests and repairs are performed as needed.

Long Term Operation - The Motion Subsystem is modified, if necessary, to fix problems or to adapt to commercial applications. Operations begin on a regular basis or commercially.

Triggers

Triggers activate and /or discover defects (or faults). Triggers might be the environment or other conditions that help or force a defect to appear. In this work, the focus is on the activity that the developer was doing when the defect was found. Identifying the most effective triggers can be extremely important. For instance, knowing the best triggers for specific defects might enable earlier detection. This can then ensure that fewer defects will appear in later development phases (i.e., Long Term Operation). Thus, fixing defects early on can improve robot reliability.

Management can make use of triggers to save valuable resources. For instance, a mechanical structural defect may cause a great number of resources to be consumed in a later development phase if this defect is not detected in earlier development phases. The use of triggers in this research is explored in the chapter Data Analysis.

Some of the triggers described here could be refined for more detailed activities (e.g., on *Component Test*, exactly what kind of test was performed), but this has to be carefully executed since many triggers will not allow appropriate quantification (few defects per trigger). The level of refinement is a starting point; it will be improved in future work according to future measurement system requirements.

There is a basic difference between the triggers that have “*check*” and the ones that have “*test*” words in their names. The triggers with “*check*” are related to reviews and discussions. Triggers with “*test*” are related to physical actions on components.

Requirements Conformance Check - Human factors that identify the defect by thinking about conformance to the requirements. For example, a developer may hold a discussion with another person to

search for different opinions or may have a discussion during a formal review, asking “Is this configuration fulfilling the requirements?”

Configuration Conformance Check - Human factors that identified the defect by thinking about conformance to the configuration.

Design Conformance Check - Human factors that identify the defect by thinking about conformance to the design. For example, in the design the system may use two power supplies while the current schematic may just show one power supply. The reviewer may ask, “Is this schematic describing the design correctly?” Or “Are these components assembled as depicted in the design?”

Previous Design Check - Human factors that identify the defect by thinking about previous experiences. For example, a comparison to a previous design may reveal the defect; similarly, it may be revealed by asking “This design was used before, did it work?”

Compatibility Check - Human factors that identify the defect by checking compatibility between components or subsystems; similarly, it may be revealed by informal or formal review.

Documentation Check - Human factors that identify the defect by checking documentation. This trigger is related to defects on documentation. For instance, a check on a drawing may reveal missing dimensions.

Inspection & Pre-Assembly - An inspection identifies the defect. For example, a visual inspection of a part may reveal damage. A pre-assembly operation may discover the defect. For example, the developer may have tried to fit the parts together before assembly, but there may have been an interference or parts may have been missing.

Component Performance Test - Tests of performance on individual components revealed the defect. The question asked may be “Is this component producing the desired results?”

System Performance Test - Tests of subsystems working together as a final product (integrated) discovered the defect. The question being answered is “Is this system producing the desired results?”

Stress Test - Submitting the subsystems/system to extreme conditions.

Field Test - Tests in the field discovered the defect.

User Operation - User operation revealed the defect. For instance, the user may have operated the system in a different way than during tests. For instance, the user gave a set of commands to the robot that were not tested before.

Unclassified - It is not possible to classify the defect using the current set of triggers.

Defect Types

The field Defect Type captures the nature of the defect. Defect Type captures neither the cause nor the consequences of the defect. Analysis of occurrences of each defect type over the project lifetime will provide a variety of process insights.

The following are the Defect Types and the process insight that might be gained by tracking the occurrences of this defect:

Performance (not working as expected) - The component is working as according to specifications (is out of specification).

What can be expected to be derived from this Defect Type?

- Is this a good development?
- Are the component technologies well understood?

Damage (needs repair) - The component is damaged. It needs to be fixed or replaced.

What can be expected to be derived from this field?

- This can be used to trigger a more detailed analysis. Was this a design problem? Why?

Specification (incorrect or change) - The specification is incorrect or has changed. Example: A misunderstanding or change in the requirements caused a developer to modify the design.

What can be expected to be derived from this field?

- Are the requirements well understood and translated to specifications?

Interface (incompatibility) - Wrong interfaces and / or incompatibility were found between components.

What can be expected to be derived from this field?

- Are the component technologies well understood?
- Was the design detailed enough (attention to detail)?

Interaction (interference) - The components might function independently, but they do not work as a system.

What can be expected to be derived from this field?

- Are the component technologies well understood?
- Was the design detailed enough (attention to detail)?

Documentation (schematics, instructions) - These are defects related to error in schematics, diagrams, instructions, procedures, and manuals.

What can be expected to be derived from this field?

- Are people being careful enough in the details?
- Trigger a more detailed analysis on what are the consequences (cost) of a bad documentation.

Missing Component- There is a component missing.

What can be expected to be derived from this field?

- Was this forgotten from the design? Or just not included on the documentation?

Esthetic (appearance) - There is an esthetic problem.

What can be expected to be derived from this field?

- Are people being careful during manufacturing, assembly, and operation?

Assembly Process (improper) - Improper or incorrect assembly was found.

What can be expected to be derived from this field?

- Are people being careful during assembly?
- What are the consequences (cost) of bad assembly procedures?

Unclassified - It was impossible to classify the defect using the current set of Defect Types.

A good taxonomy

A taxonomy identified as “good” should orthogonally capture process characteristics as well as enable the classification of the majority of defects without redundancy. Also the level of detail of the data collected should be carefully chosen. For instance, if too many types of defects are present in the tax-

onomy, quantification will be more difficult because the classified defects will be divided among the many defect types. On the other hand, if just a few defect types are present in the taxonomy, the extraction of information is difficult. This is because one defect type could represent more than one piece of information.

Another example of the necessity of carefully choosing the level of detail of the data collected is regarding the attribute (or field) component (i.e., electrical component and mechanical component). For instance, deciding if the type or class of components is sufficient or that manufacturer and the model of the component should be specified (e.g., microcontroller or a Motorola MC68HC11-E microcontroller, bearing or NSK-1165/2 ball bearing). In this research the level of detail was limited to the type of components (e.g., connector, power supply, encoder, fasteners, bearing, lubricant, etc.). The main reasons for this choice were:

Improve quantification and ease classification - As explained before, quantification can be improved by using few types of components (more defects per type of components). If using computer tools, quantification should not be a big problem since abstraction of the levels are possible (e.g., all kind of connectors add to a generic connector type). But this approach does not help classification because many more options would be available to users, resulting in more opportunities for error. To make available many kinds of components is very labor intensive because all possible component models from diverse manufacturers have to be present to enable defect classification.

Electronic components become obsolete relatively quickly - An electronic component commonly used now will be obsolete in a few years. Therefore the data collected about this component will not be useful after few a years. For example, a microcontroller from the Motorola MC689HC11 family is largely used on the industry today: however, this microcontroller architecture will be considered “old” five years from now.

Mechanical components - Mechanical components vary widely in part numbers and manufacturers for similar parts. Also, these components can be made from different materials. That is, developers can represent parts using different manufacturer reference code.

Another characteristic of a good taxonomy is that it is valid. The problem here is to validate the taxonomy. One way to validate a taxonomy is to relate the data collected to facts that happened during the development of a system [Santhanam, 1997]. This validation process is explained in Chapter 3, Data Analysis in the Validation section.

After the taxonomy is designed, the data collection process can begin. The next section will describe the process of data collection used in this research.

2.3 Data Collection

In a data collection system a large amount of data may be entered manually by support personnel or by developers. Support personnel collect data from customers, developers, or from any other source of defects (e.g., a manufacturing shop) and then classify the defects using the available classification tools. Developers can directly classify the defects if the classification tools are easy to use, or they can be interviewed by support personnel in order to supply defect data.

To enable correct data analysis many parts must work well together: a process and tools to collect data, an appropriate storage mechanism for the data, an environment that supports statistical methods for modeling and estimations, and a flexible data-oriented graphics tool [Jones, 1996].

This section describes the process of implementing the RODC taxonomy to enable data collection for the RODC prototype. Initially, data was collected using paper-based forms. Then the process was improved to include a more flexible parameterized process using computer tools. The focus of this section is on the computer tools; details about the design of the database (tables, forms, queries, and reports) will also be presented.

2.3.1 Data Collection Tools

According to [Halliday, 1993] “tools supporting the methodology” were the main obstacles to the deployment of the ODC method in IBM laboratories. An important finding from the IBM work was that tools for classification and analysis are necessary but do not necessarily need to be automated. In fact, in one case at IBM, a data collection scheme completely based on paper was deployed faster than many other automated data collection scheme tools [Halliday, 1993]. This led to the initial decision to not use automated classification or analysis tools in the RODC prototype.

The decision while enabling fast data collection, made it difficult to quickly modify the taxonomy.

Data collection in this prototype initially required paper forms (RODC V1.0) distributed to Nomad developers. In doing so, the implementation was accomplished quickly (since development on paper is faster than development of computer-based tools for data collection). That is, before any computer

tool was developed, defect data was being collected and the process of data entry was simple and fast. For each defect, one table was used. At the end of every day, the tables were collected.

This scheme did not work well for two reasons. First, people were just filling the forms when they thought it was appropriate (that is, when time allowed or when they were asked to do so); second, the taxonomy was not sufficiently mature (that is, classification was not very easy because the attributes were not orthogonal).

The second attempt to collect data using paper-based forms worked better. Developers were interviewed all day long to identify defects. The author walked around the laboratory (FRC) asking the developer questions and recording information on the forms. This scheme stopped working when changes to the taxonomy were necessary. That is, once the taxonomy was revised, data collected on previous forms had to be manually rewritten. (Note: no computer tools had been developed at that time. Focus was on the taxonomy design and the amount of data collection. As a result, attention was paid to minimizing missed defects.) The necessity for a more flexible method of data collection arose very quickly.

The third and successful method of collecting data was using a tape recorder. Instead of writing the defects according to the attributes available on the forms (thus limiting the amount of information that could be captured), general process information was collected. Flexibility was greatly improved and the time consumed during the interviews was greatly reduced. (This improvement was appreciated by the developers and led to them accepting the data collection scheme.) This turned out to be very effective because more information could be captured in less time.

Data continued to be collected during the development of the computer-based tools.

2.3.2 Computer Tools

To enable data analysis three criteria must be met: an appropriate storage mechanism for the data, an environment that supports statistical methods, and a data-oriented graphics tool. These requirements imply the necessity of using computer tools.

Some computer tools were analyzed (i.e., electronic spreadsheets, databases, and statistical analysis packages). Of these the Relational Database Management System (RDBMS or RD) was selected as most appropriate because it meets all the criteria: it can create the forms used to input data to the database, interact with the data stored in the database supporting statistical methods, and enable data-oriented graphics.

Relational Database Management Systems

A relational database management system is a software used to create, maintain, modify, and manipulate a relational database [Hernandez, 1997]. In a relational database, data is stored in tables containing records that contain attributes (or fields). Each record in a table has a field containing a unique value (ID) used for its identification. To access the data contained in a record it isn't necessary to know the physical location of the record; instead data can be retrieved by knowing the record ID or by running a query.

Relationships in a RD are specified by establishing relationships between RD tables. These relationships can be one-to-one, one-to-many, or many-to-many.

One-to-one - In this relationship a single record in one table is related to one and only one record in another table. This record on the second table can be related to only one record in the first table.

One-to-many - In this relationship a single record in one table is related to one or more records in a second table. This record on the second table can be related to only one record in the first table.

Many-to-many - In this relationship a single record in the first table is related to one or more records in the second table. One record on the second table can be related to one or more records in the first table. This relationship requires that a third table be used as an auxiliary table. That is, the third table stores relationship between IDs of the first two tables.

All three types of relationships are used in the design of this particular computer-based tool and will be described in the RODC Database Design section.

Once table relationships are established, data can be queried in several ways. For example, extraction

of data about the defects that caused a system failure can be accomplished by running a query that searches the database for defects that have the potential to cause the system to fail. Queries basically navigate through the relationships and return data that match certain criteria. Also, queries can perform actions on the database (for example, deleting or updating table information). The language used to perform database operations is called Structural Query Language (SQL). It is beyond the scope of this research to introduce SQL.

Many RDBMS are commercially available. Some examples are: Access, Oracle, MS SQL Server, DB2, Informix, FoxPro, FileMaker, etc. The software package chosen was Access from Microsoft. Access was chosen because of its advanced features and easy-to-use visual interface. Documentation for Microsoft Access, a desktop database used by millions of users around the world every day [Litwin, 1997], was readily accessible.

2.3.3 RODC Database Design

The most important step in designing a database is deciding how to structure stored data. Several references for RDBMS design are available, including [Elmasri, 1989], [Kroenke, 1995], and [Fleming, 1989]. The RODC database design is divided in the following:

- RODC Tables
- RODC Table Relationships
- RODC Forms
- RODC Reports

RODC Database Tables

As explained in the previous sections, the RODC method collected information about the development process of mobile robots rather than simply collecting information about defects. Therefore, tables were created to store information about different aspects of the mobile robot development. That is, tables were designed to implement the RODC taxonomy.

Other tables were created to capture process information. For instance, tables were created to store information about institutions and personnel working in the development of the robots (e.g., NASA

Ames - Hans Thomas).

Next, RODC tables are described.

The list of tables includes:

- Institutions
- Personnel
- Roles
- Projects
- Subsystems
- Reliability Methods
- Pictures
- Project & Institutions
- Projects & Subsystems
- Subsystems & Personnel
- Subsystems & Reliability Method
- Projects & Pictures
- Subsystems & Pictures
- Defects
- Defect Type
- Defect Source
- Defect Cause
- Development Phases
- Defect Severity
- Person Impact
- Triggers
- Electrical Components
- Mechanical Components

(Note: In the implementation of the tables in Access, the author used the prefix *tbl* in each table name to help in the differentiation of tables, queries, reports, etc., ex: tblDefetc, qryDefects, rptDefects.)

It is important to mention here that the tables were designed using parametrization. This enables references between tables using IDs (pointers). These references allow changes to field names on the

tables without having to change every record (one by one) in the database. For example, if the Taxonomy name for a Defect Type changes, it is not necessary to change all the records in the database, just the name on the Defect Type table. Moreover, this feature saves space since each database record stores an ID that points to a more complex structure (more costly in space).

Institutions

This table stores information about institutions such as addresses, contact names, etc. (See Figure 2.13.) The relationship to the database is that one project refers to one or more participating institutions. That is, a project can have one or more institutions participating in its development.

FIGURE 2.13 Institutions Table

	Field Name	Data Type	Description
	InstitutionID	AutoNumber	
	InstitutionName	Text	Name of the Institution
	Address	Text	
	City	Text	
	StateOrProvince	Text	
	PostalCode	Text	
	Country	Text	
	PhoneNumber	Text	
	FaxNumber	Text	
	EmailAddress	Text	
	URL	Hyperlink	
	Notes	Memo	
	LastUpdate	Date/Time	

Personnel

This table stores information about personnel working on the projects. (See Figure 2.14.) The relation to the database is that one subsystem refers to one or more members of the Personnel table. Also one member of this table can belong to one or more institutions. That is, a subsystem can have one or more developers. Also one developer can belong to one or more institutions (dual affiliation).

FIGURE 2.14 Personnel Table

Field Name	Data Type	Description
PersonnelID	AutoNumber	
FirstName	Text	First name
LastName	Text	Last name
InstitutionID	Number	Institution where affiliated
DepartmentName	Text	Name of the department in the institution
Title	Text	Job title on the institution
DegreeHeld	Text	Maximum degree earned
ExperienceDesigningRobots	Text	Aproximately in how many robots projects did this person worked already?
HistoryOfPreviousRobots	Memo	Name and intitutions of previous robots where this person worked
ExperienceWithReliabilityMet	Text	Aproximately in how many projects did this person used reliability technics already?
WorkPhone	Text	
WorkFax	Text	
EmailName	Text	
URL	Hyperlink	
Notes	Memo	
LastUpdate	Date/Time	

Besides contact information this table captures the experience of developers in working on robot development and also identifies whether the developers had used reliability improvement techniques previously. This data can be used on the reliability analysis of projects or subsystems. For instance, reliability problems in a development project could be related to personnel experience in working with mobile robot designs and experience with reliability improvement techniques.

Roles

This table stores information about developer roles on a subsystem (e.g., Electronic Leader). (See Figure 2.15.) The relationship to the database is that one member from the Personnel table can fulfill more than one role within a subsystem. For instance, one developer can have different roles in different subsystems. This table is associated with the Personnel table and can be used to infer relationships between reliability problems and personnel working on the project or subsystem.

FIGURE 2.15 Roles Table

Field Name	Data Type	Description
RoleID	AutoNumber	Unique Role ID generated automatically
RoleName	Text	Role's name, title on the project
RoleDescription	Memo	Description of the Role
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Projects

This table stores information about projects. (See Figure 2.16.) The relationship to the database is that one project can include one or more members of the Subsystems table and one or more members of the Pictures table. Members of this table can belong to one or more institutions. That is, a project can have one or more subsystems, one or more pictures (images, schematics, etc.), and the project may belong to one or more institutions. For instance, the Nomad robot has several subsystems, and includes several images and schematics representing the robot. As mentioned earlier, in this research two Nomad subsystems were used in the prototype: locomotion and pointing for communications.

FIGURE 2.16 Projects Table

Field Name	Data Type	Description
ProjectID	AutoNumber	ID for each project
ProjectName	Text	Name for each project (e.g., Dante II, Nomad)
ProjectDescription	Memo	Text describing the project
URL	Hyperlink	URL for the project. Note: This is not an URL for the institution
ContactID	Number	Contact person for this project
ManagerID	Number	Manager person for this project
ProjectBeginDate	Date/Time	When the project started
ProjectEndDate	Date/Time	When the project ended
PlannedProjectBeginDate	Date/Time	When the project was planned to start
PlannedProjectEndDate	Date/Time	When the project was planned to end
ComplexityIndex	Number	Computed complex index for the project & subsystem- To be implemented yet - See thesis proposal
SpentBudget	Currency	Budget spent on the project
PlannedBudget	Currency	Budget planned to be spent on the project
DesignPhasesAndDeadLines	Memo	Project's design phases and dead lines
Notes	Memo	Observations and notes
LastUpdate	Date/Time	

Also, included are fields for the project: URL, manager, contact person, complexity index, spent budget, planned budget, and deadlines. The idea here is to provide information that will aid management and future work.

URL - Pointer to the URL project. Developers will be able to access project documents using this pointer.

Manager - Project manager information (pointer to the Personnel table).

Contact person - Information documenting the name of the person to be contacted regarding this project. It might be the manager or any other person classified on the Personnel table.

Complexity index - The complexity index is used for data analysis. (See Data Analysis chapter - Defect Model.)

Spent budget - Developers can track the project budget. Access to this field can be restricted to authorized personnel.

Planned budget - Same use as for the spent budget.

Deadlines - A reinforcement of the project schedule. Developers can check upcoming deadlines.

Subsystems

This table stores information about subsystems. (See Figure 2.17.) The relationship to the database is that one subsystem can include one or more members of the Personnel table and one or more members of the Pictures table. Also one member of this table can belong to one or more projects. That is, the subsystem can have one or more developers and one or more pictures (images, schematics, etc.); the subsystem can also belong to one or more projects (e.g., re-used design).

FIGURE 2.17 Subsystems Table

	Field Name	Data Type	Description
	SubsystemID	AutoNumber	
	ProjectID	Number	Project Name
	SubsystemName	Text	Name for the subsystem (e.g., Real-Time computer, Locomotion)
	SubsystemDescription	Memo	Description of the subsystem (e.g., This subsystem includes all the onboard computers)
	URL	Hyperlink	URL for the project. Note: This is not an URL for the institution
	ResponsibleID	Number	Person responsible for the subsystem
	PlannedSubsystemBeginDate	Date/Time	Planned development start date
	PlannedSubsystemEndDate	Date/Time	Planned development end date
	SubsystemBeginDate	Date/Time	When the subsystem development started
	SubsystemEndDate	Date/Time	When the subsystem development ended
	PlannedBudget	Currency	Budget planned to be spent on the subsystem
	SpentBudget	Currency	Budget spent on the subsystem
	ComplexityIndex	Number	Computed complexity index for the subsystem - *** To be implemented yet - See thesis proposal ***
	DesignPhasesAndDeadLines	Memo	Subsystem's design phases and dead lines - How the project was divided
	Complete	Yes/No	Is the developments of the subsystem complete?
	Notes	Memo	Observations and notes
	LastUpdate	Date/Time	

As in the Projects table, data management supporting fields were included: URL, responsible, complexity index, spent budget, planned budget, and deadlines. The idea here is to provide information to aid management and future work. These fields will not be described here since they are very similar to the field on the Projects table described above.

Reliability Methods

This table stores information about reliability methods. (See Figure 2.18.) The relationship to the database is that one member of the Subsystem table can point to one or more members of the Reliability table, and one personnel member can point to one or more members of the Reliability table. That is, subsystems can make use of one or more reliability methods. Also personnel can have experience with one or more reliability methods.

FIGURE 2.18 Reliability Methods Table

	Field Name	Data Type	Description
PK	ReliabilityMethodID	AutoNumber	
	ReliabilityMethodName	Text	
	ReliabilityMethodDescription	Memo	
	URL	Hyperlink	
	Notes	Memo	
	LastUpdate	Date/Time	

Pictures

This table stores Pictures (photos, schematics, diagram, etc.). (See Figure 2.19.) The relation to the database is that one member of the Defect table, Subsystem table, and Project table can include one or more members of the Picture table. That is, a defect, subsystem, or project records can include a picture. For instance, a photo of a damaged axle can be include on the defect record. Projects and subsystems can have one or more pictures too.

FIGURE 2.19 Pictures Table

	Field Name	Data Type	Description
PK	PictureID	AutoNumber	Unique identification (automatically generated)
	SubsystemID	Number	Project Name where this picture belong
	PictureName	Text	Name of the Picture
	Photograph	OLE Object	Insert the picture file
	DatePhotograph	Date/Time	Date when the picture was taken
	PictureURL	Hyperlink	URL for this picture
	Schematic	OLE Object	Schematic representing the picture's content
	SchematicURL	Hyperlink	URL for this schematic
	Notes	Memo	Notes
	LastUpdate	Date/Time	

To optimize the manipulation of data in the database, pictures are included in “external” tables as references (i.e., just the picture ID is stored, no picture files are stored in tables other than in the Picture

table). For the user this issue is transparent. When a defect record is shown on the user screen, the RODC database reads the image from the Picture table and displays it on the appropriate field of the defect record. This feature is described on the RODC Forms and Reports sections.

Project & Institutions

This is an auxiliary table. This table stores auxiliary information so one record from one table can refer to multiple instances in another table. (See Figure 2.20.)

Databases make use of auxiliary tables to store multiple instances of records linked to different tables. For instance, one record of this table may contain a project ID pointer and an institution ID pointer. Doing so, it is possible to have one project that refers to many institutions and vice versa. An example of using the Project & Institutions Table can be seen in Figure 2.21.

FIGURE 2.20 Projects and Institutions Table

	Field Name	Data Type	Description
🔑	ProjectID	Number	
🔑	InstitutionID	Number	
	Notes	Memo	Observations and notes
	LastUpdate	Date/Time	

FIGURE 2.21 Example of using the Projects and Institutions Table

	Project Name	Intitution Name	Notes:	Last Updated:
▶	Nomad	Carnegie Mellon University	Nomad was built at CMU-FRC	4/24/97
	Nomad	NASA Ames		4/24/97
	FIRST-97	Carnegie Mellon University		4/25/97
	FIRST-97	NASA HQ		4/25/97
	FIRST-97	Schenley High School		4/25/97
*				12/13/97

The following five tables are also auxiliary tables: Project & Pictures, Projects & Subsystems, Subsystems & Pictures, Subsystems & Personnel, and Subsystems & Reliability Method.

No further description will be given here since the design used in these tables is basically the same as the one used on the Projects & Institutions Table.

Defects

This table stores information about defects. (See Figure 2.22.) This is the main table containing defect records. Most of the fields in this table are pointers to tables described in this section.

FIGURE 2.22 Defects Table

Field Name	Data Type	Description
DefectID	AutoNumber	Unique ID - Automatically Assigned
SubsystemID	Number	Subsystem's name
Date	Date/Time	Date defect found
UserID	Number	User Name who is collecting the defects
GroupDefect	Number	How many defects are grouped with this defect?
CountGroupDefect	Number	Count for the group defect
SourceID	Number	Source for the Defect
CauseID	Number	Cause of the defect
DevPhaseID	Number	Development Phase where defect was found
SeverityID	Number	Severity of the defect
ImpactID	Number	
DesignConsideration	Yes/No	Was this Defect Considered during Design?
NotKnown	Yes/No	Is this Defect Known by the developer? "Not known (never seen)"
TriggerID	Number	What have Triggered the defect
DefectTypeID	Number	Type of Defect
ElectricalCompID	Number	Detail Electrical Defect
MechanicalCompID	Number	Detail Mechanical Defect
TimeToFix	Number	Estimated Time to Fix the defect
TimeToPrevent	Number	Estimated Time that Could Prevent the defect
Unclassified	Memo	Description of Unclassified fields. Write the possible classifications in this space
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Defect Type

This table stores information about defect type. (See Figure 2.23.) The relationship to the database is that one defect can refer to one defect type. That is, a defect can be only one type.

FIGURE 2.23 Defect Type Table

Field Name	Data Type	Description
DefectTypeID	AutoNumber	
DefectTypeName	Text	Type of Defect
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Defect Source

This table stores information about the source of defects. (See Figure 2.24.) The relationship to the database is that one defect can refer to one defect source. That is, a defect can have only one source.

FIGURE 2.24 Defect Source Table

Field Name	Data Type	Description
SourceID	AutoNumber	
SourceName	Text	Source for the Defect
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Defect Cause

This table stores information about defect causes. (See Figure 2.25.) The relationship to the database is that one defect can refer to one defect cause. That is, a defect can have only one cause.

FIGURE 2.25 Defect Cause Table

Field Name	Data Type	Description
CauseID	AutoNumber	
CauseName	Text	Cause of the defect
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Development Phases

This table stores information about development phases. (See Figure 2.26.) The relationship to the database is that one defect can refer to one development phase. That is, only one development phase may be chosen per classified defect.

FIGURE 2.26 Development Phases Table

Field Name	Data Type	Description
DevPhaseID	AutoNumber	This field is automatically generated
DevPhaseName	Text	Development Phase Name
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Defect Severity

This table stores information about the severity of the defects. (See Figure 2.27.) The relationship to the database is that one defect can refer to one level of defect severity. That is, only one severity category can be chosen per classified defect.

FIGURE 2.27 Defect Severity Table

Field Name	Data Type	Description
SeverityID	AutoNumber	Unique identification, automatically generated
SeverityName	Text	Severity Name
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Person Impact

This table stores information about how defects impact personnel. (See Figure 2.28.) The relationship to the database is that one defect can refer to one defect impact. That is, only one impact is chosen per classified defect.

FIGURE 2.28 Person Impact Table

Field Name	Data Type	Description
ImpactID	AutoNumber	Unique identification, automatically generated
ImpactName	Text	Impact Name
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Defect Trigger

This table stores information about what triggers defects. (See Figure 2.29.) The relationship to the database is that one defect can refer to one defect trigger. That is, only one trigger is chosen per classified defect.

FIGURE 2.29 Defect Triggers Table

Field Name	Data Type	Description
TriggerID	AutoNumber	Trigger ID
TriggerName	Text	Trigger Name
Notes	Memo	Observations and notes
LastUpdate	Date/Time	Last Updated

Electrical Components

This table stores the name of electrical components. (See Figure 2.30.) The relationship to the database is that one defect can refer to one electrical component. That is, only one component can be chosen per classified defect.

FIGURE 2.30 Electrical Components Table

	Field Name	Data Type	Description
	ElectricalCompID	AutoNumber	
	ComponentName	Text	Detail Electrical Defect
	Notes	Memo	Observations and notes
	LastUpdate	Date/Time	Last Updated

Mechanical Components

This table stores the name of mechanical components. The relationship to the database is that one defect can refer to one mechanical component. This table was designed using the same scheme as in the electrical components table.

Tables on a DBMS are fundamental elements in data storage. Once tables are created then it is possible to make a relationship between them and thus to enable data retrieval.

RODC Table Relationships

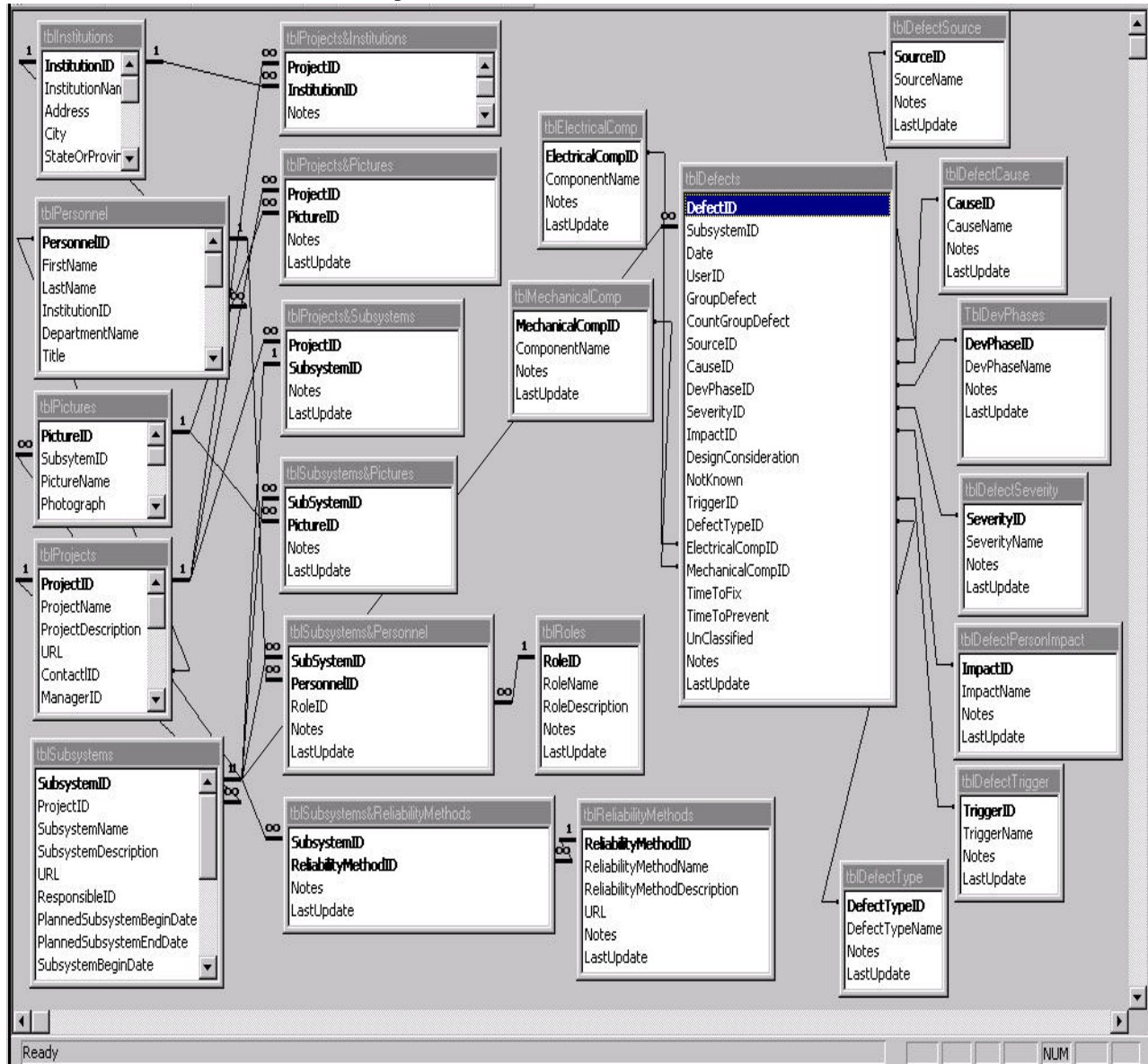
Relationships in a relational database are used to link tables and to allow “navigation” in the database. Using relationships, it is possible to keep different types of data in different tables (e.g., images, a personnel list, defect types, etc.). Doing so, the database becomes more manageable because different types of data and complex structures can be accessed by simply knowing the table name and the record ID.

Several types of information maybe needed to create a relationship: the name of the tables involved, the type of relationship desired, and the key field of one of tables. The key field is usually the ID field of the table (e.g., DefectID, CauseID, DevPhaseID, etc.).

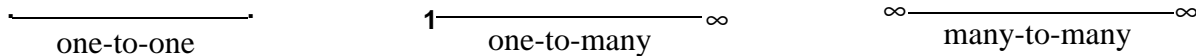
Access supports three different types of relationships: one-to-one, one-to-many, and many-to-many (requires an auxiliary table). These types are explained in the Relational Database Management Systems section.

Figure 2.31 shows the tables relationships in the RODC database.

FIGURE 2.31 RODC table relationships



The graphical links used in Access represent the type of relationship.



Access has a tool called *form* that enables the simultaneous use of tables and relationships. The next section describes how forms were used to collect the data used in the RODC prototype.

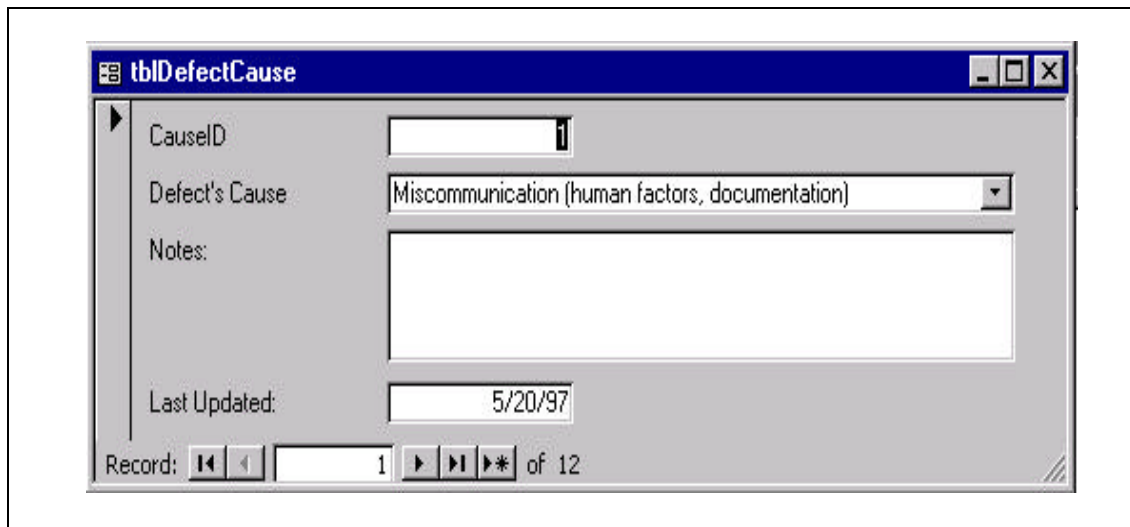
RODC Forms

Having an easy-to-use interface is fundamental to the data collection process. If the user spends a significant amount of time classifying a defect, it is likely that he or she will not classify all the defects found. A good user interface is also important because it can minimize human mistakes [Halliday, 1993]. In a typical RD, tables are shown to users as spreadsheets. (See Figure 2.32.) Users enter data into tables by typing information into the cells of the spreadsheet interface.

FIGURE 2.32 RODC tables in spreadsheet format

tblDefectCause : Table (Local)			
CauseID	Defect's Cause	Notes:	Last Updated:
1	Miscommunication (human factors, documentation)		5/20/97
2	Change or Misunderstanding of requirements		5/20/97
3	Design		5/20/97
4	Manufacture (<> datasheet)		5/20/97
5	Assembly (improper)		5/20/97
6	Environment (e.g., heat)		5/20/97
7	Storage		5/20/97
8	Transport		5/20/97
9	Manipulation		5/20/97
11	Defective Component		5/20/97
12	Unknown		5/29/97
13	Unclassified		6/12/97
*	(AutoNumber)		12/14/97

By using forms in Access, it is possible to generate a better interface to input data into the tables than the spreadsheet interface. (See Figure 2.33.) Using this interface the user is able to focus on the database register that he or she is inputting and therefore minimize confusion. Besides that, the “visual” space for the field *Notes:* is increased, so users can easily type and read text in this field.

FIGURE 2.33 Interface using forms in Access

The screenshot shows a Microsoft Access form window titled "tblDefectCause". The form contains the following fields and controls:

- CauseID:** A text box containing the value "1".
- Defect's Cause:** A dropdown menu with the selected value "Miscommunication (human factors, documentation)".
- Notes:** A large, empty text area.
- Last Updated:** A text box containing the date "5/20/97".

At the bottom of the form, there is a record navigation bar that reads "Record: 1 of 12". The navigation controls include buttons for first, previous, next, last, and refresh.

Another powerful feature of using forms is the possibility of displaying linked data. For instance, a table that is linked to a second table can be displayed in the same form as a single table form. That is, the information from the second table is linked to the first table and then displayed to the user. Figure 2.34 shows a form for the Project table that links the *project manager* and *contact person* fields to the Personnel table. Doing so the user can choose from a list of options presented in the form Projects built from the Personnel Table. Note the list of options available for the project manager when the user clicks on the pull-down menu. This data is being linked to the Personnel Table and is displayed to the user in a transparent way.

FIGURE 2.34 Example of a form linking two tables

The screenshot shows a window titled "tblProjects" with the following fields and values:

- Project ID: 1
- Project Name: Nomad
- Project Description: Carnegie Mellon University
A mobile robot - 4 wheel drive
Target 2 months 200 Km autonomous operation in Chile's Atacama desert
- Project's URL: <http://www.ri.cmu.edu/~nomad>
- Contact Person: Eric Rollins
- Project Manager: Eric Rollins (selected from a dropdown menu)
- Project Begin Date: Hans Thomas (NASA Ames)
- Project End Date: Jack Silberman (Carnegie Mellon University)
- Planned Begin Date: Jim Teza (Carnegie Mellon University)
- Planned End Date: Mark Maimone (Carnegie Mellon University)
- Complexity Index: [empty]
- Spent Budget: [empty]
- Planned Budget: [empty]
- Design Phases: [empty]
- Notes: [empty]
- Last Updated: 4/28/97

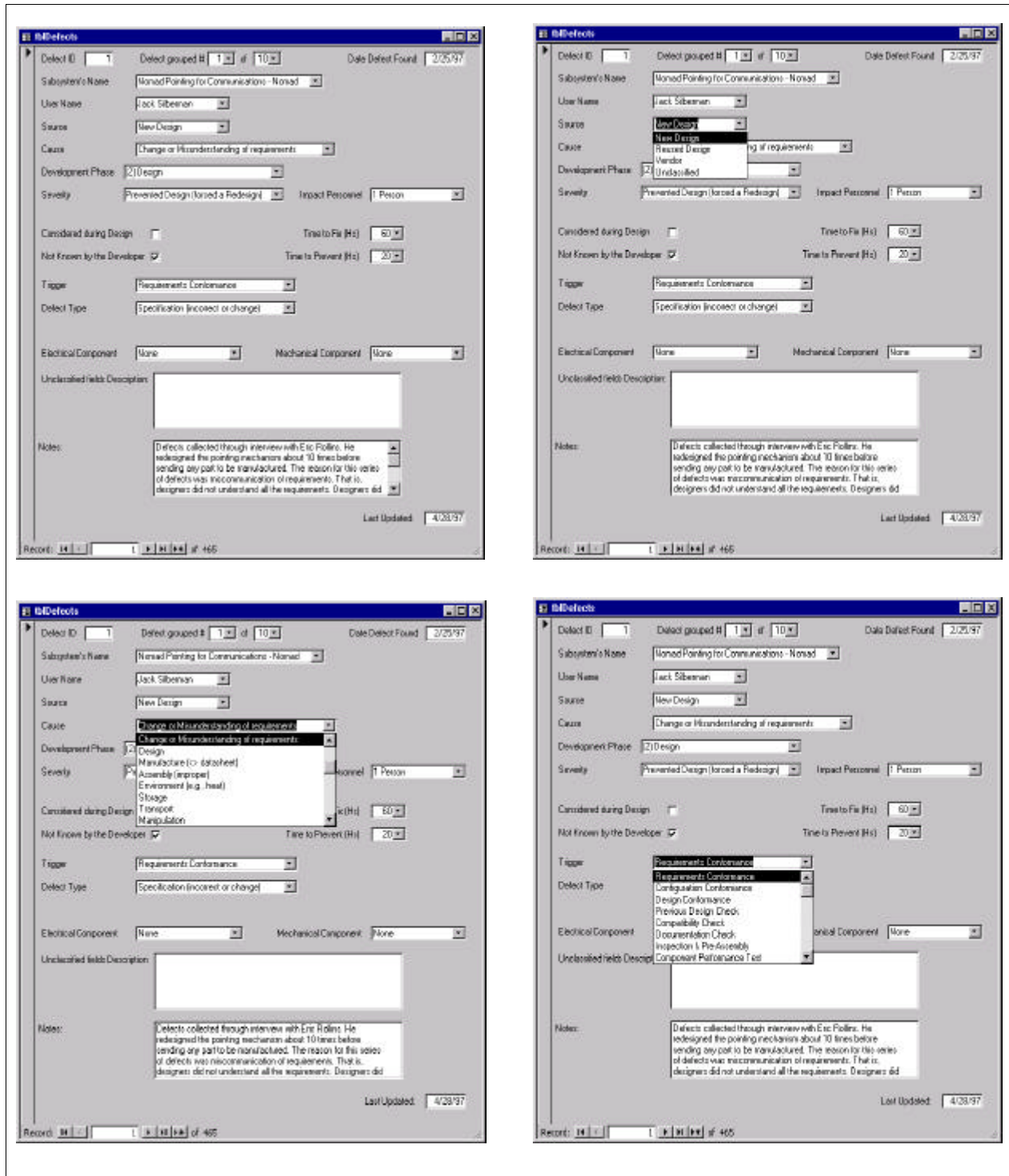
At the bottom, there is a record navigation bar showing "Record: 1 of 2".

Figure 2.35 shows the user interface for the RODC Defect Table. This main interface is used for classifying defects in the RODC database. The form designed here links several tables (as shown in Figure 2.31) to make options available to users.

The user has a pull-down menu for most options; the only fields in which the user needs to manually type in details are on the *Unclassified Description:* and *Notes:* field, but these fields are optional. (Note: Only a few examples of screen shots are shown here.)

The user can not type in the defect or other attributes. This feature minimizes confusion since it eliminates typing errors. In the case where the user is not able to classify the defect, he can type a short note explaining why classification is not possible. These notes were analyzed later determining whether changes to the taxonomy were necessary.

FIGURE 2.35 RODC User Interface



A database that only stores data would not be appropriate for use in the RODC method since information extraction is a main component of the method. A tool called Report, allows for the display of data contained in an Access database. The next section describes the reports designed for the RODC prototype.

RODC Reports

The primary use of reports is for output. Reports are bound to tables and queries. In the same way that forms can be used to link tables, reports are used to extract information from tables. Reports in Access can also generate data-driven graphics.

Queries

One of the most effective ways of extracting information from a database is by using queries. Generally, queries are used to “navigate” through the database and extracted data can then be used in reports. Figure 2.36 shows a partial result of a query executed on the defect data of the RODC database. The goal was to display defect types and then causes for each record.

FIGURE 2.36 Partial results of a query

Defect ID	Type of Defect	Defect's Cause
132	Interaction (interferences)	Assembly (improper)
133	Interaction (interferences)	Assembly (improper)
134	Interaction (interferences)	Design
135	Assembly Process (improper)	Assembly (improper)
136	Interaction (interferences)	Assembly (improper)
137	Interaction (interferences)	Assembly (improper)
138	Interaction (interferences)	Assembly (improper)
139	Interaction (interferences)	Assembly (improper)
140	Assembly Process (improper)	Miscommunication (human factors, documentation)
141	Assembly Process (improper)	Miscommunication (human factors, documentation)
142	Assembly Process (improper)	Miscommunication (human factors, documentation)
143	Performance (not working as expected)	Design
144	Performance (not working as expected)	Design
145	Assembly Process (improper)	Assembly (improper)
146	Assembly Process (improper)	Assembly (improper)
147	Interface (incompatibility)	Design
148	Interface (incompatibility)	Design
149	Interface (incompatibility)	Design

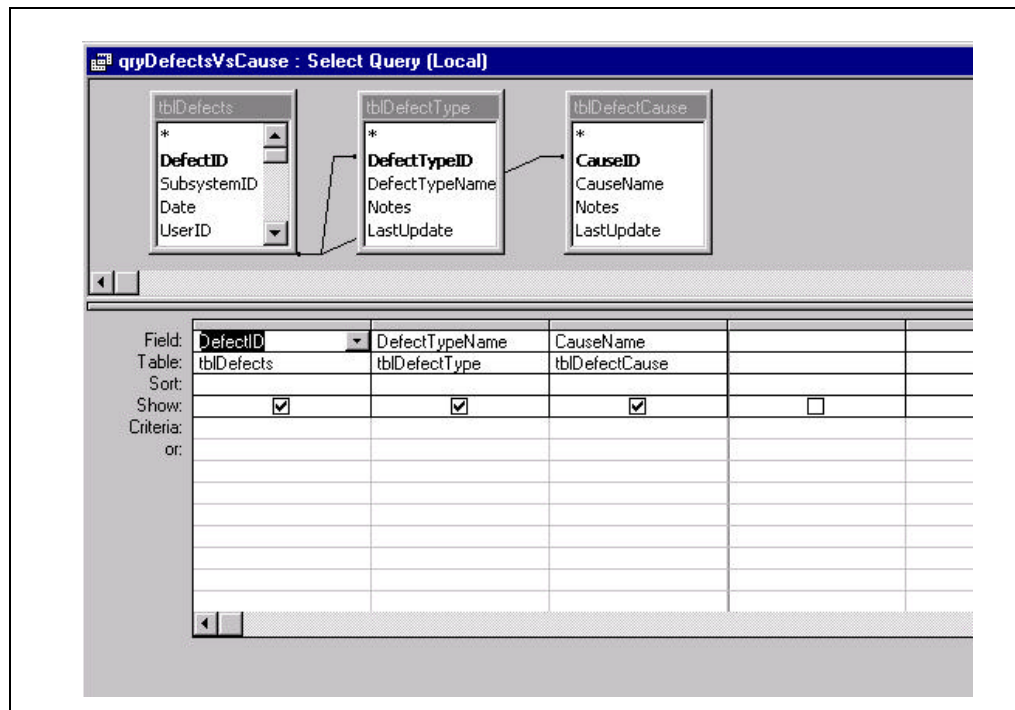
Record: 109 of 347

The following SQL code was used to generate the simple query shown above.

```
SELECT tblDefects.DefectID, tblDefectType.DefectTypeName, tblDefectCause.CauseName FROM tblDefectType INNER JOIN (tblDefectCause INNER JOIN tblDefects ON tblDefectCause.CauseID = tblDefects.CauseID) ON tblDefectType.DefectTypeID = tblDefects.DefectTypeID;
```

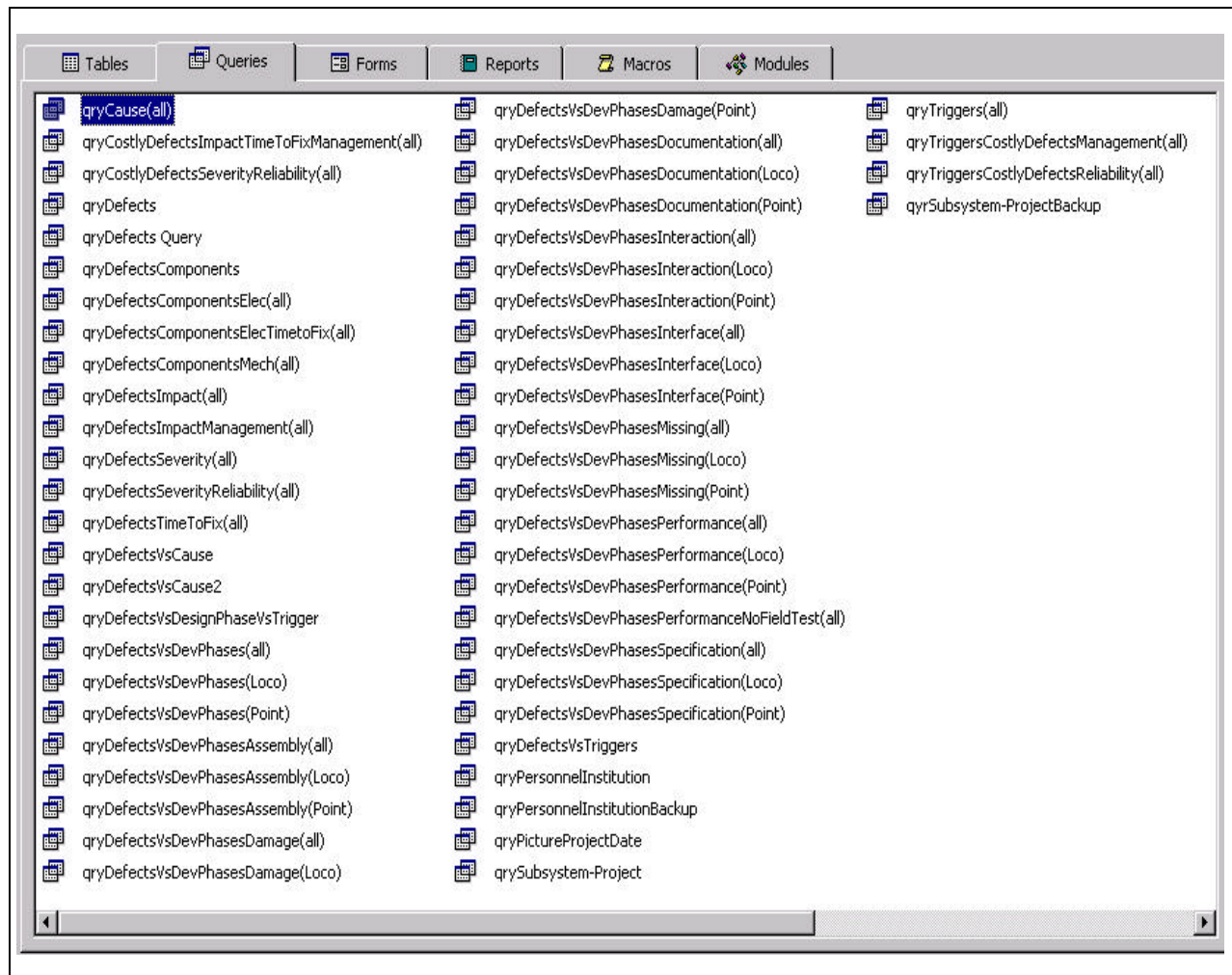
As one can imagine, developing the SQL code for a more elaborated query is relatively complex. Fortunately, Access has a visual interface to generate queries. Figure 2.37 shows the design of the same query used to generate the defect type and cause list shown before, but this time using the visual interface.

FIGURE 2.37 Visual interface to generate queries



The majority of queries developed for the RODC prototype were developed using the visual tools available in Access. Several queries were developed to enable information extraction. (See Figure 2.38.) These queries are described in the Appendix B.

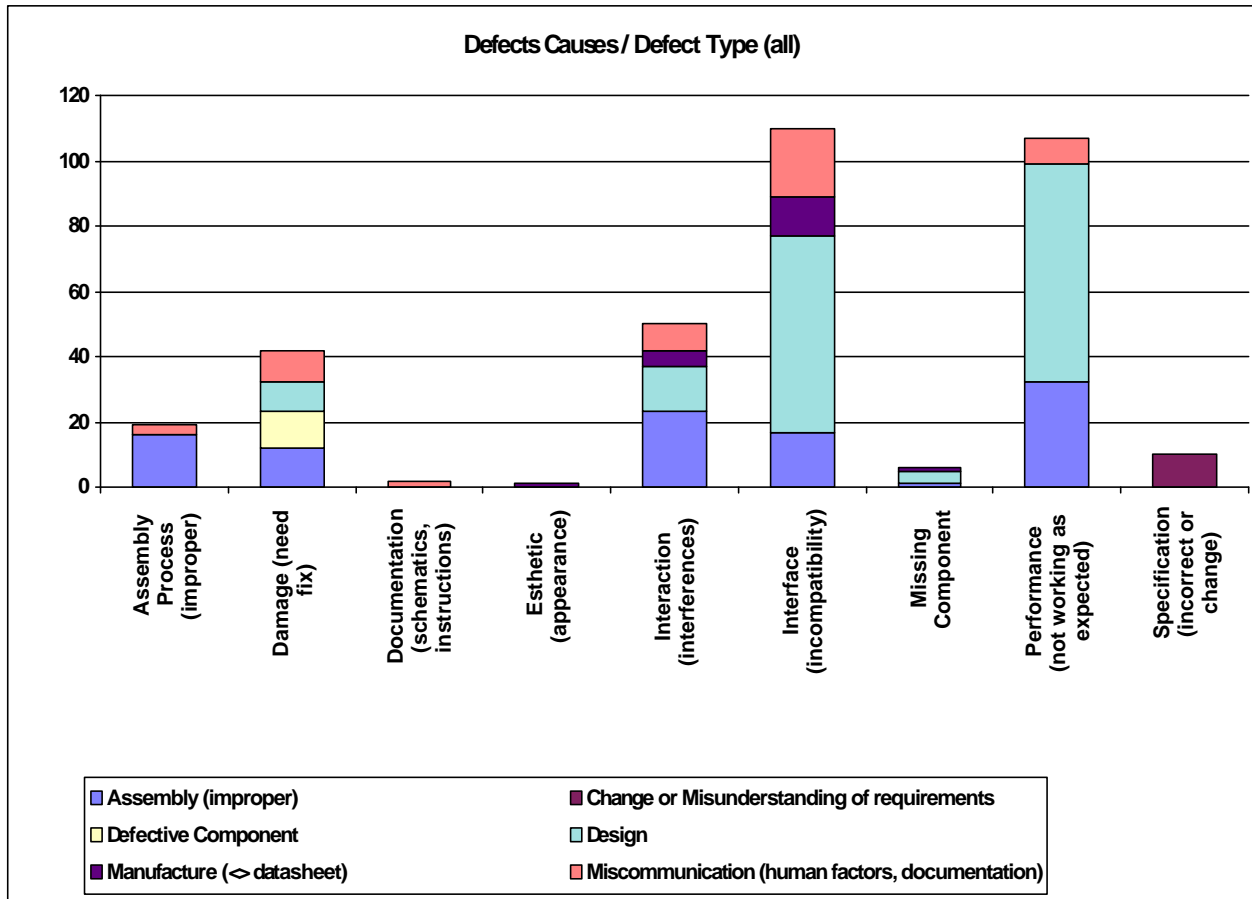
FIGURE 2.38 RODC Queries



Reports

The reports developed for the RODC are extremely important for the process of extracting information from the RODC data. Figure 2.39 shows a report with data-driven graphics for the query of defect type and cause for the entire database.

FIGURE 2.39 Report for showing defect types and causes



Reports can be generated to combine the data in many forms. In this case, the defect types were plotted on the horizontal axes and then the cause of each defect was plotted in the vertical axis. Instead of having one vertical bar for each cause per each defect, the author chose to plot the individual contribution of each cause for the total number of defects per each defect type. Looking at Figure 2.39, one can see that the main cause for the defect types interface and performance is design.

The next chapter explores the use of reports for extracting information from the RODC database.

Chapter 3

Data Analysis

This chapter describes the analysis performed on data collected during the development of motion systems on a mobile robot project. This analysis provides validation of the measurement scheme and information for the mobile robot developer team. The set of defect data here refers to the defect data collected from the development of the Nomad robot. As explained in the RODC Pilot chapter, two subsystems containing motion systems were investigated (i.e., Locomotion and Pointing Mechanism for Communications). Ideally, these sets of data should have been treated separately so as to incorporate knowledge gained from the analysis of the first set of data (e.g., locomotion defects) and then to apply that knowledge to the second set of data (e.g., pointing defects). As will be explained later in this chapter it was not possible to use this approach because the pointing defect data alone does not contain sufficient numbers of defect signatures.

A total of 465 defects were collected from the Nomad development process.

Two types of analysis will be performed on the collected data:

- Validation
 - Information Extraction
-

3.1 Validation

The objective of the validation analysis is to validate the measurement scheme. Such validation will be accomplished by comparing *defect signatures* to *logic signatures*. Logic signatures are signatures produced by a process of analyzing the probability (in each development phase) that defects will occur. They are explained later in this chapter. Moreover, validation can be accomplished by relating trends in the collected data to facts about the project (from which the data was collected). By analyzing the collected data and comparing facts about what happened during a project, one can tell if the measurement system is sound [Santhanam,1997].

Thus, validation will be accomplished by explaining the similarities and differences between the defects signatures and logic signatures. Validation will also be accomplished by relating analysis of the collected data to events that occurred during a project. That is, if the measurement system can capture development process characteristics, not just defects, that will enable more useful information extraction. Another way of validating a measurement system is to extract information. That is, if the measurement system provides information that really matters to developers, then this measurement system is said to be valid [Santhanam, 1997].

In order to validate the measurement system in the IBM Watson Research Center ODC work, a set of requirements needed to be satisfied [Chillarege, 1991]. This set of requirements, called Sufficient Conditions, was the third validation method used in the RODC.

Once validation is accomplished, we can say that we trust the data for performing *information extraction*. The following three sections will describe each validation method used in the RODC method.

3.1.1 Logic Signatures

The following are the steps for the validation using logic signatures:

1. Create logic signatures
 2. Create defect signatures from collected data
 3. Select defect signatures for comparison to logic signatures
 4. Compare signatures and explain differences
-

Create Logic Signatures

The name logic signatures refers to the likelihood that certain defects will happen at certain times. To allow the creation of logic signatures, a matrix including *Development Phases* and *Defect Types* was created. (See Table 3.1.) The idea here is to assign probable quantities of defects for each development phase. For instance, the defect type *Assembly Process (improper)* will not appear during the development phase *Requirements and Configuration*. Therefore, the matrix cell representing the intersection between these two fields will receive the lowest value possible for probable quantity (in our case 0). Logic signatures are not meant to be generic; for example, the set of logic signatures described here were created from the experience of developing robots in a specific research center, the Field Robotics Center at Carnegie Mellon University. A survey was sent to robot developers at the Field Robotics Center (see Appendix C). The results were used to populate Table 3.1. (As a result, they should be independently evaluated for use in other centers.) The idea here is to enable the use of logic signatures with a specific robot development in a specific research center. To generalize the logic signatures, one can study other research centers that develop such machines and use the tools generated in this work to aid the creation of appropriate logic signatures. This is because the same scheme can be applied to the development of other robots, thereby generating defect data for investigation. Generalization is not the goal of this research, future RODC implementations will have to address this issue.

In Table 3.1, a scale ranging from 0 to 3 is used to assign the probable quantity of defects. The lowest bound (0) means that no defect of this type should be found during a specific development phase. The highest value (3) means that a significant number of defects are likely during this phase. For instance, the defect type *Interfaces (incompatible)* will likely have a significant number of defects during the development phase *Integration of Components*. This is because during the development phase components will be integrated for the first time to form a subsystem. That is, chances of having interface incompatibility problems during this development phase are high.

The scale description follows:

- 0 - No defects should occur
- 1 - A small number of defects should occur

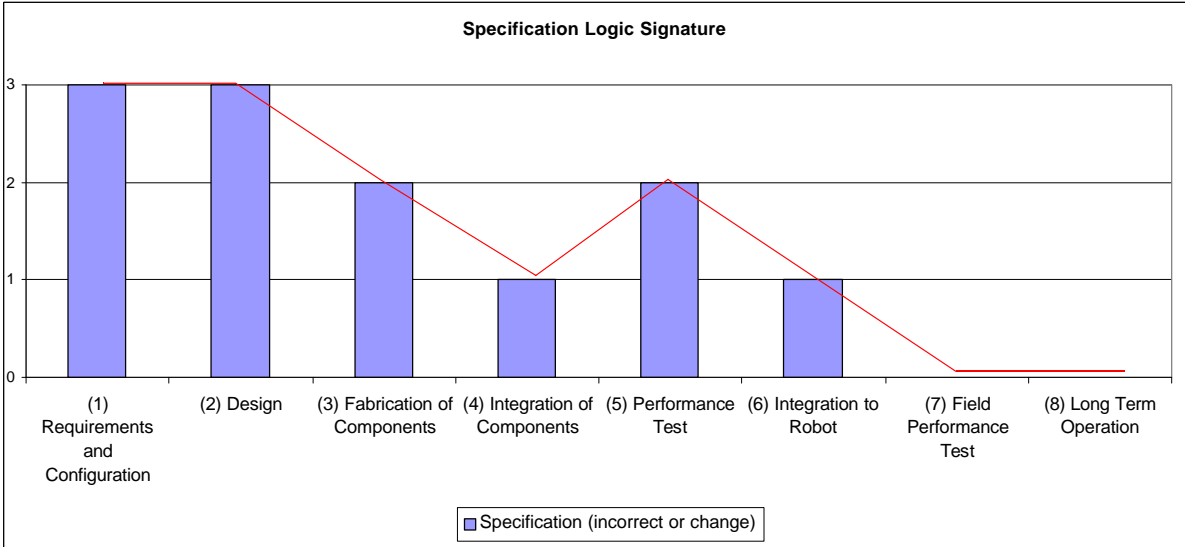
- 2 - A moderate number of defects should occur
- 3 - A significant number of defects should occur

TABLE 3.1 Probable Occurrence of Defect Types on Development Phases

Dev. Phase Vs. Defect Type	Requirements and Configuration	Design	Fabrication of Components	Integration of Components	Performance Test	Integration to Robot	Field Performance Test	Long Term Operation
Assembly Process (improper)	0	0	1	3	1	3	1	0
Damage (need fix)	0	0	2	1	3	2	3	1
Documentation (schematics, instructions)	2	3	2	2	1	2	1	1
Esthetic (appearance)	0	0	3	2	3	2	3	3
Interaction (interferences)	0	2	2	3	1	3	1	0
Interfaces (incompatible)	0	2	1	3	1	3	1	0
Missing Component	0	1	2	3	2	2	1	0
Performance (not working as expected)	0	1	1	2	3	2	3	2
Specification (incorrect or change)	3	3	2	1	2	1	0	0

After the matrix was completed, one logic signature was plotted for each defect type. Figure 3.1 shows the logic signature plotted separately for the defect type *Specification*. The other logic signature plots can be found in Appendix D.

FIGURE 3.1 Logic Signature for Specification

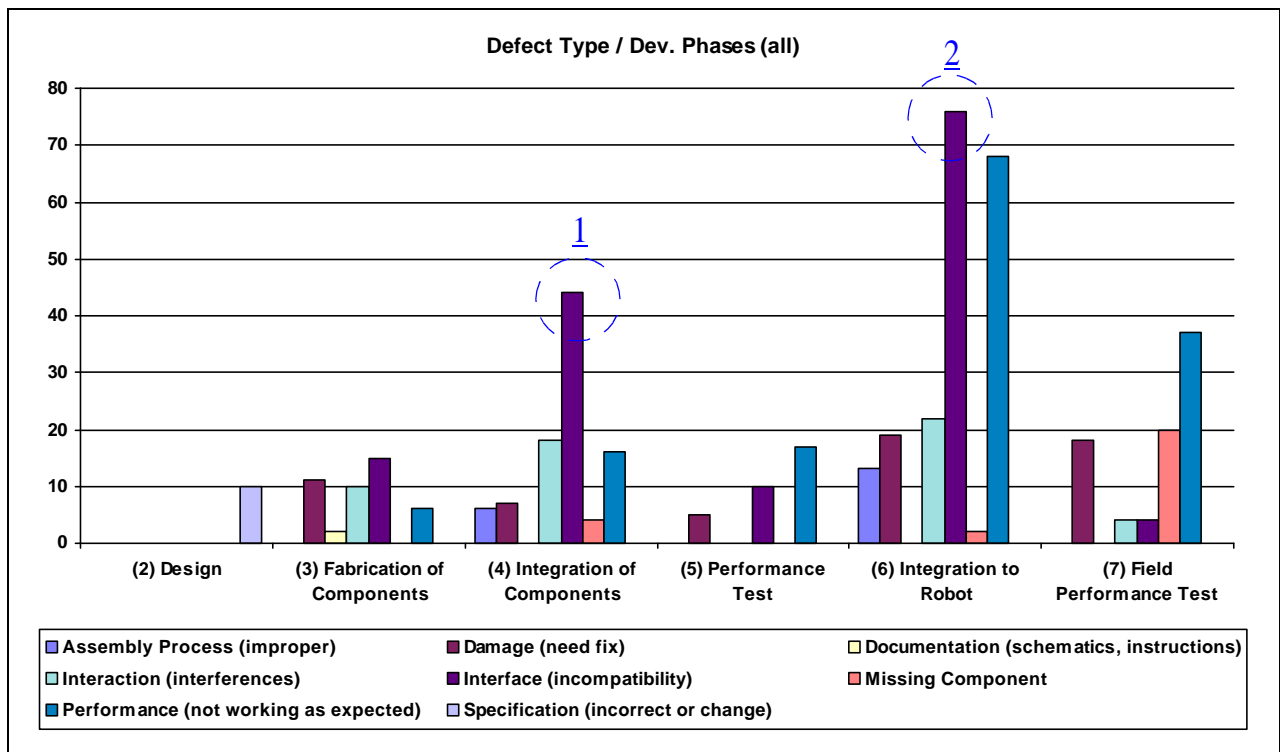


Create Defects Signatures from Collected Data

The name defect signatures refers to the occurrence of defects during various phases. To allow the creation of defect signatures, defect data was extracted from the defect database using queries. Figure 3.2 shows a plot containing all collected defects for the Nomad robot.

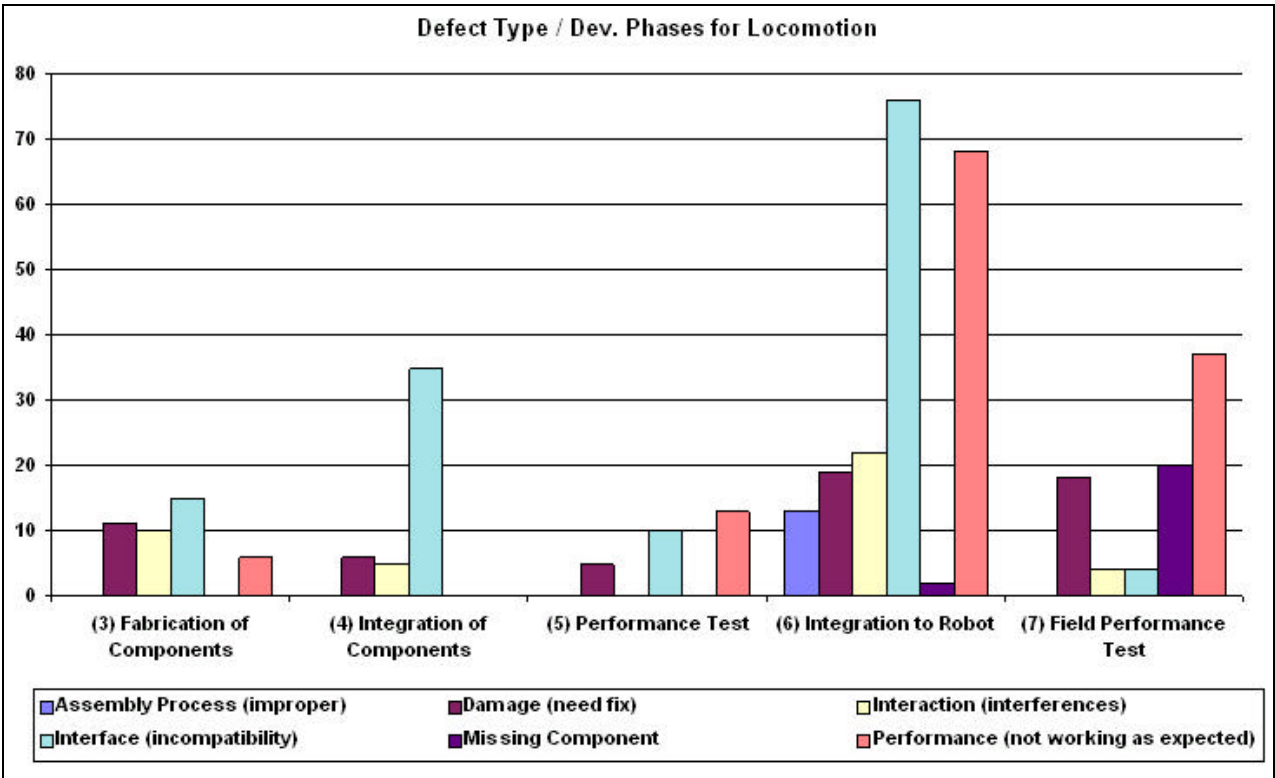
The plots are fundamental for data interpretation in this work. For instance, a quick analysis of Figure 3.2 reveals some key signatures. For instance, the defect type *Interface* has two peak values at the *Integration of Components* and *Integration to Robot* development phases. (See Points 1 and 2 in Figure 3.2. below) This is expected since in these two development phases, components are integrated to form subsystems and later to form the robot. More on the interpretation of Nomad defect data will be described later in this chapter.

FIGURE 3.2 Nomad Defects



Here it is important to mention that the term defect data in use in the analysis process contains both defects from Nomad's locomotion motion system and Nomad's pointing motion system. This is because these defects collectively represent a significant improvement in the quantification of defects. Figure 3.3 and Figure 3.4 show pointing and locomotion defects plotted separately. When compared to Figure 3.2 these plots lack signature representation. For instance, in the pointing defects no defect types are present during three development phases. This fact makes it difficult to create defect signatures.

FIGURE 3.4 Locomotion Defects



Select Defect Signatures for Comparison to Logic Signatures

Using defect data (locomotion + pointing defects) a plot was generated for each defect type. The goal of this validation step is to identify and select candidates for comparison to the logic signatures. Ideally, defect signatures should be generated from defect types that are represented in multiple development phases. This is because a defect signature can only be created from multiple points. But, defect types are not always present in each development phase. This problem has been caused by two factors: either because the classification scheme was not available during Nomad’s development phases or because specific defect types were not present. For instance, the measurement scheme was neither available during the Requirements and Configuration development phase nor during the Design development phase for the Nomad locomotion motion system. Therefore, defects were not collected.

The idea is to select defect signatures generated from the presence of defect types during multiple development phases. In this case, defect data from specific defect types would have to be collected in three or more development phases in order to be included. Moreover, it is necessary that a reasonable

number of defects be present in the signature. In this case, signatures should be based on at least 10% of the total defects collected. The rationale for using this 10% figure is based on preliminary observation of the defect data. That is, it seems that a defect type containing less than 10% of the total number of defects collected did not store enough information when compared to other defect types.

For instance, the defect signature *Missing* has defects present in three development phases, but this signature does not satisfy the minimum number of defects necessary in a signature (i.e., ~ 40). (See Figure 3.5.) Figures 3.6 to 3.9 show selected defect signatures.

FIGURE 3.5 Defect Signature Missing

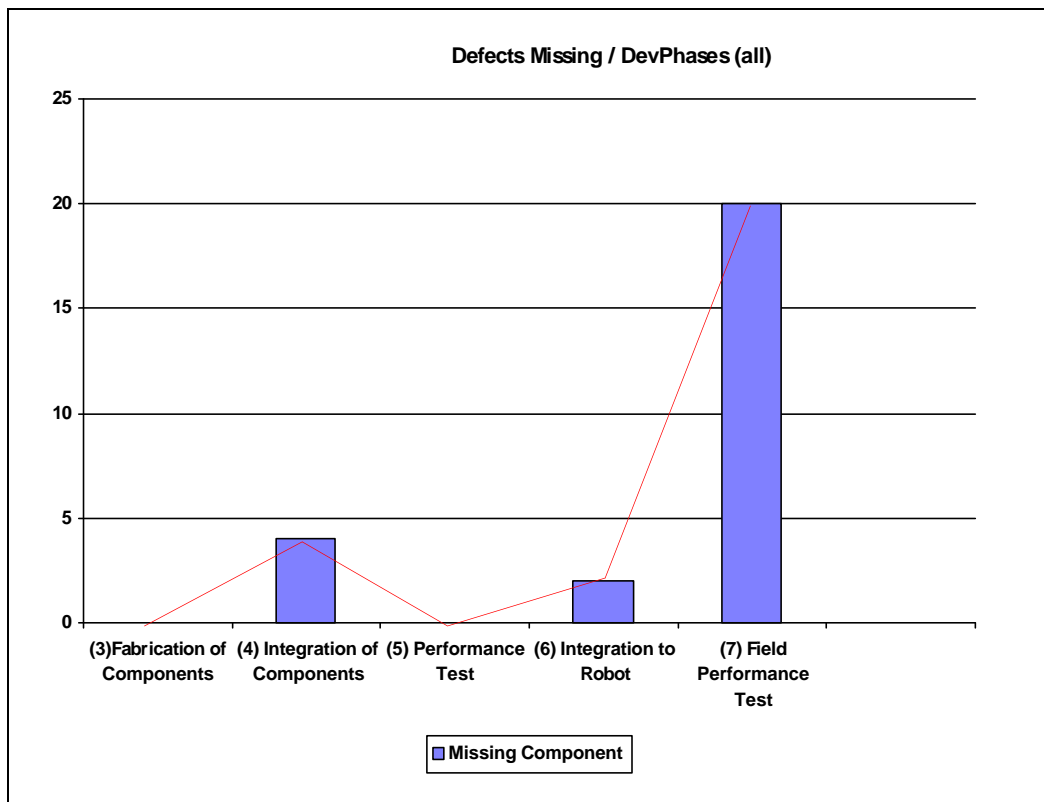


FIGURE 3.6 Defect Signature Interaction

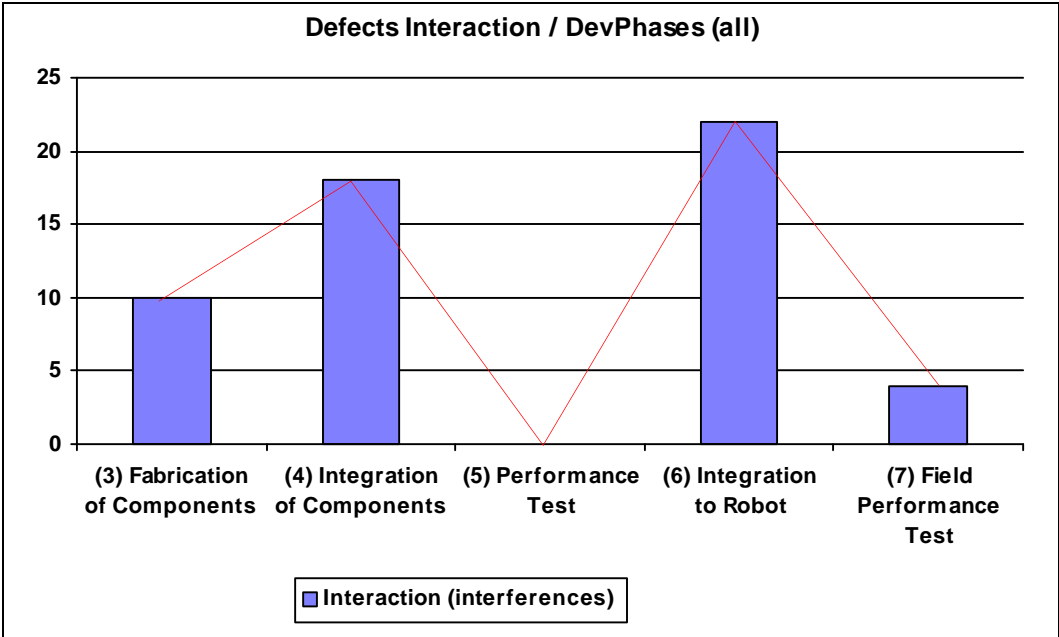


FIGURE 3.7 Defect Signature Interface

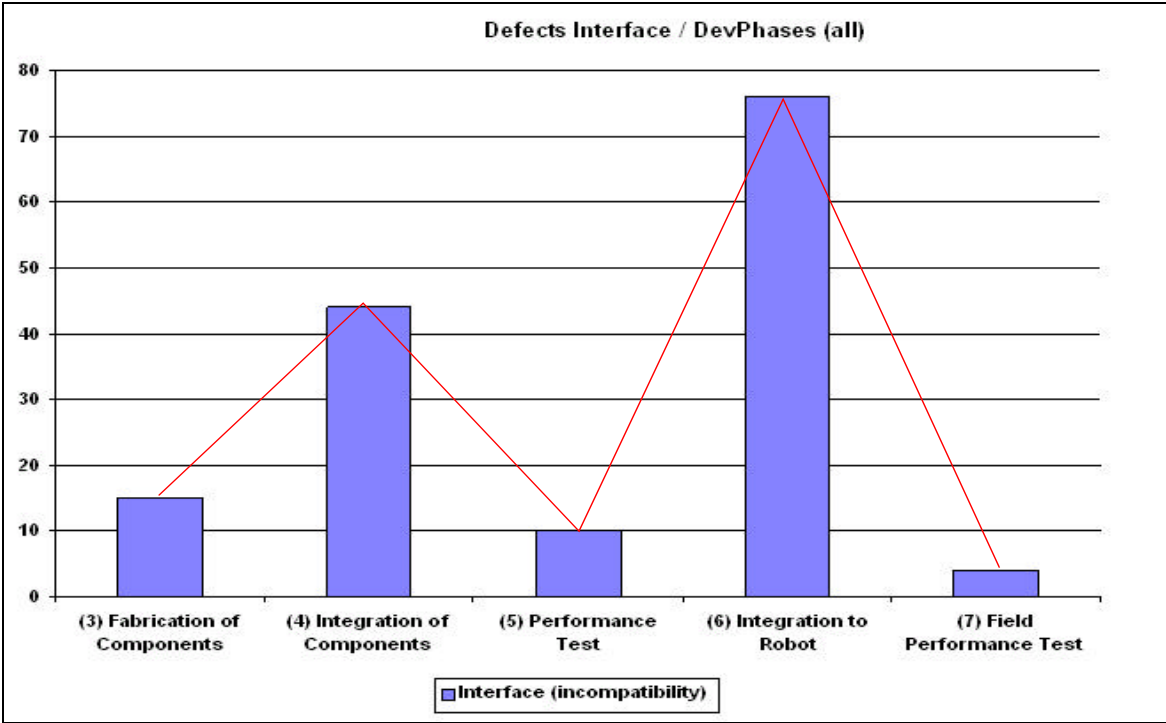
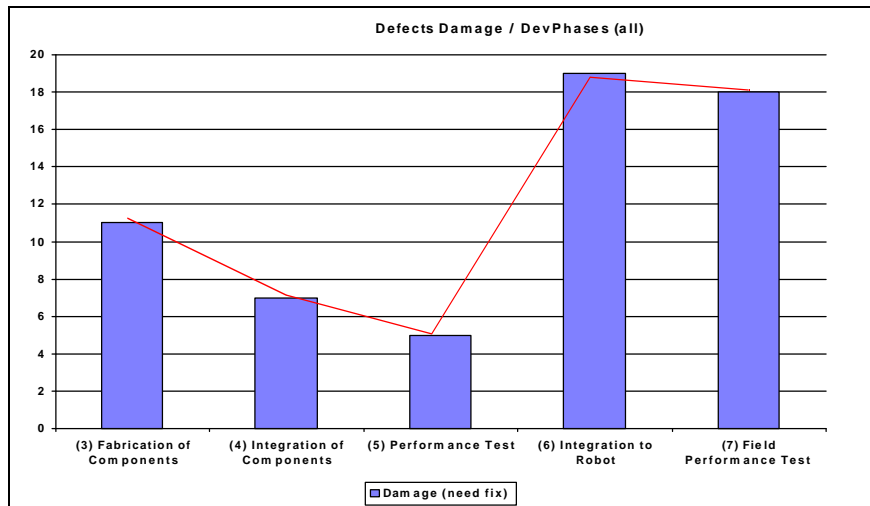
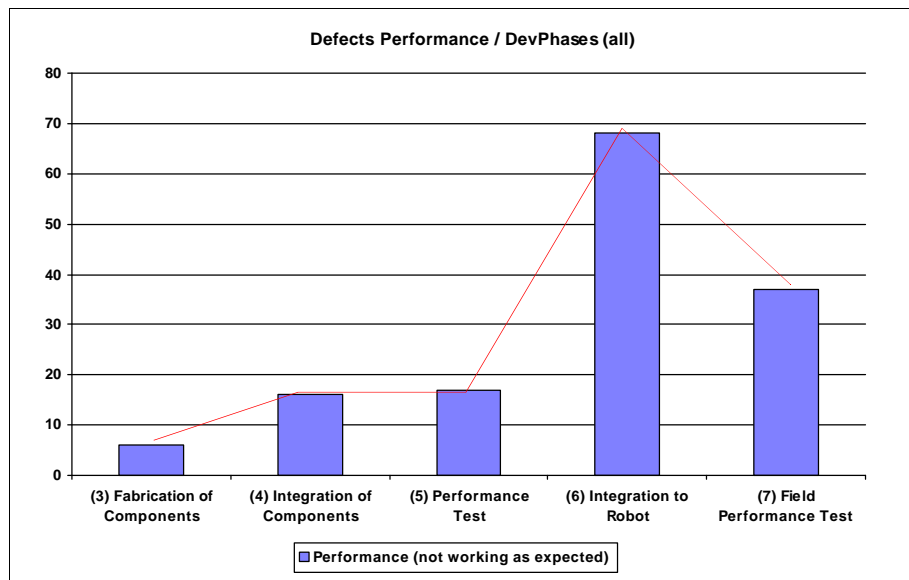


FIGURE 3.8 Defect Signature Damage**FIGURE 3.9** Defect Signature Performance

Compare Signatures and Explain Differences

In this step of the validation, logic signatures will be compared to defect signatures. The idea here is to accomplish validation by the interpretation of the similarities and differences between the logic signatures and the defect signatures selected in the previous section.

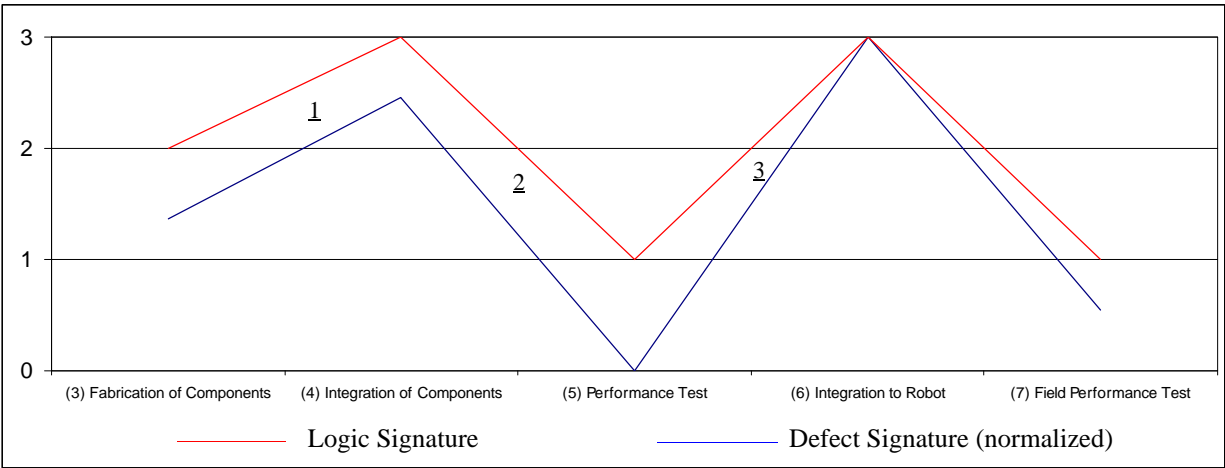
It is important at this point to mention that very few defects were collected during the development phase *Field Performance Test*. This is because defect classification during Nomad's field perfor-

mance test was not required. That is, developers had no obligation to collect defects; very little documentation is available concerning problems occurring during the field performance test in the Atacama desert in Chile. The defects collected were limited to what developers could remember weeks after the fact. This problem is addressed in Chapter 4: Conclusions, Lessons Learned section.

Defect Type Interaction

As can be seen in Figure 3.10 the two signatures have a similar appearance. As one might expect, the number of defect type *Interaction* increases from *Fabrication of Components* phase to *Integration of Components*. (See Point 1 in Figure 3.10.) The number of defects decreases from *Integration of Components* to *Performance Test* (see Point 2 in Figure 3.10), and then it increases again at the *Integration to Robot* development phase (see Point 3 in Figure 3.10). Therefore the signature seems logical and is thus a good comparison to the interaction logical signature.

FIGURE 3.10 Signatures for Defect Type Interaction



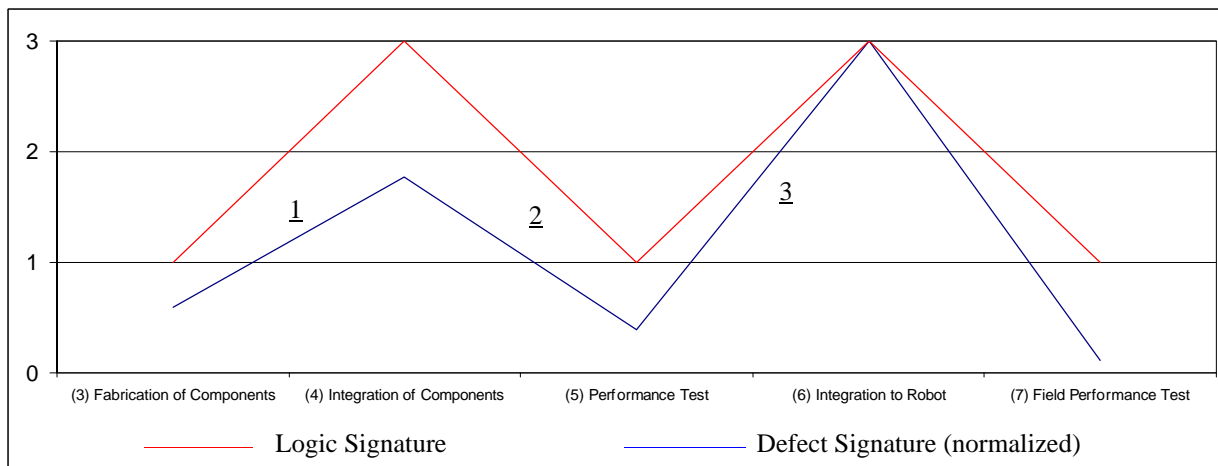
Defect Type Interface

As can be seen in Figure 3.11 the two signatures have a similar appearance. As one might expect, the number of defect type *Interface* increases from *Fabrication of Components* phase to *Integration of Components*. (See Point 1 in Figure 3.11.) The number of defects decreases from *Integration of Components* to *Performance Test* (see Point 2 in Figure 3.11) since less component integration is performed in this phase, and then it increases again at the *Integration to Robot* development phase (see

Point 3 in Figure 3.11). Therefore the signature seems logical and is thus a good comparison to the interaction logical signature.

It is important to mention here that the idea in this validation is to compare the general shape of the signatures and proportions of the number of defects. At any moment the author infers that the number of defects that will probably occur during the *Integration of Components* is equal to the number of defects that should occur during the *Integration to Robot* development phase. The numbers on the logic signatures indicate the likelihood that certain defects will happen at certain times. For instance, in the interface logical signature a development phase that has an index of 3 (e.g., *Integration of Components*) will likely have a significant number of defects when compared to development phases that have 0 and 1 as indexes (e.g., *Performance Test*).

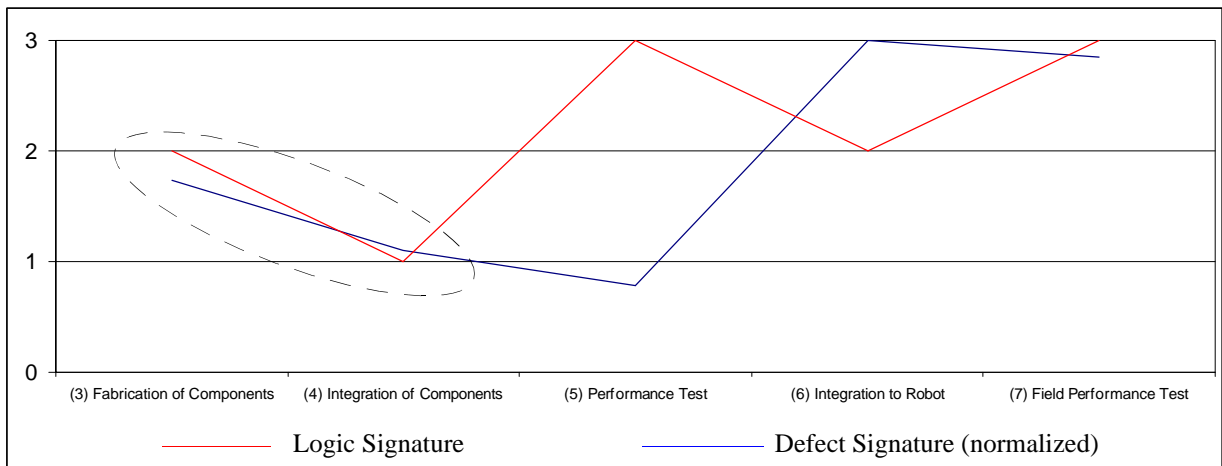
FIGURE 3.11 Signatures for Defect Type Interface



The next two signatures did not provide a good match to the logical signatures. But it is appropriate to describe them in this research because they illustrate that differences between signatures can be explained.

Defect Type Damage

As can be seen in Figure 3.12 the two signatures have a similar appearance for *Fabrication of Components* and *Integration of Components* but not for the other development phases.

FIGURE 3.12 Defect Type Damage Signatures

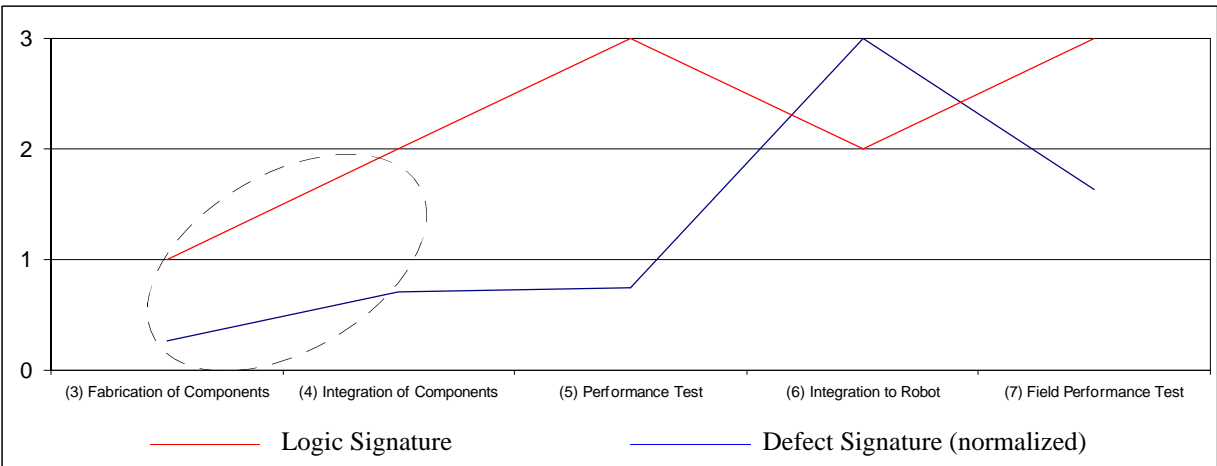
The difference in appearance of the signature for the *Performance Test* and *Integration to Robot* development phases is because the Locomotion and Pointing motion systems are subsystems that did not have extensive performance tests before being integrated into Nomad. Actually, the Locomotion Motion System was fully assembled for the first time during the *Integration to Robot* development phase. This fact pushed the majority of defect type *Damage* to the *Integration to Robot* development phase. That is, no tests were performed on the fully assembled Locomotion Motion System before the tests performed during the *Integration to Robot*. The reason for the difference on the signature in the *Field Performance Test* development phase, as explained before, is because few defects were collected in this development phase.

The author believes that if appropriate tests had been used to check the performance of Nomad's motion systems during the development phase *Performance Test*, defect signatures would have a better match with the logical signatures.

Defect type Performance

As can be seen in Figure 3.13 the two signatures have a similar appearance for *Fabrication of Components* and *Integration of Components* but not for the other development phases.

FIGURE 3.13 Defect Type Performance Signatures

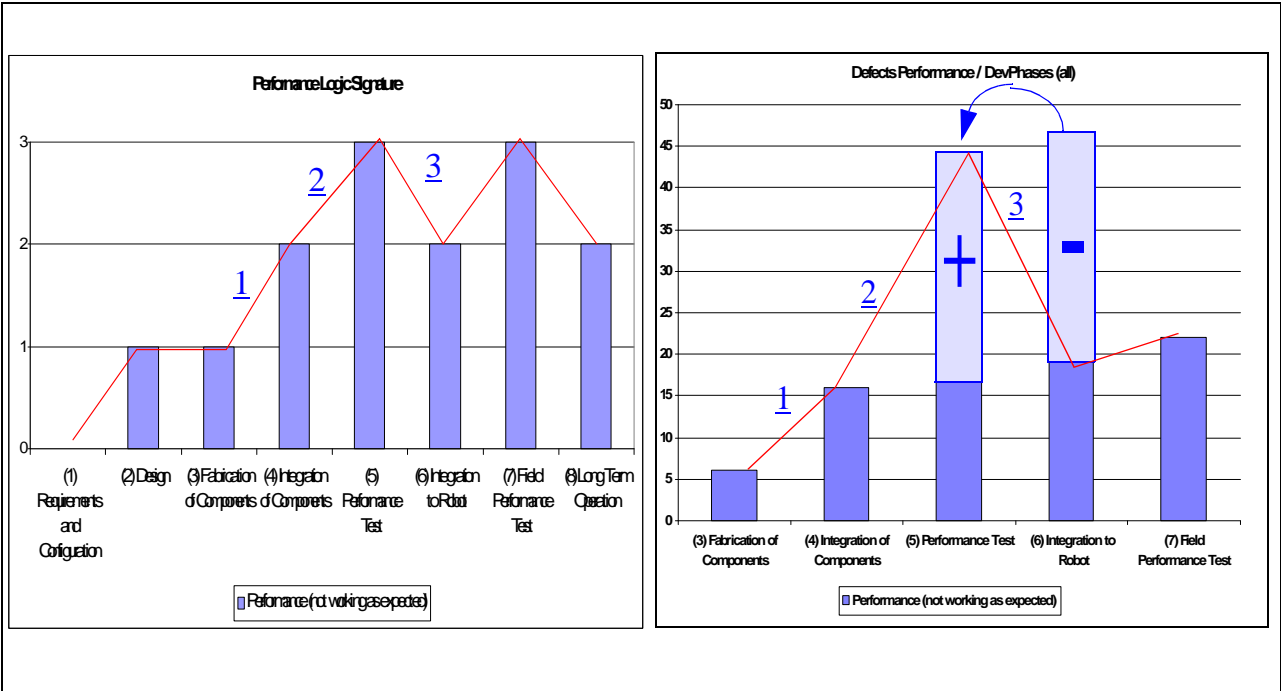


The difference in the appearance of the signatures for the *Performance Test* and *Integration to Robot* development phase is because no significant performance tests were conducted on the subsystems before the *Integration to Robot* development phase. The major components of the locomotion subsystem were integrated during the *Integration to Robot* development phase. Therefore, defects of type *Performance*, which should have been discovered during tests on the development phases *Integration of Components* and *Performance Test*, were discovered later during tests realized during the *Integration to Robot* phase.

Figure 3.14 shows how the signatures could have a better match if *Performance* defects were identified during the *Performance Test* phase. In this figure *Performance* defects were moved from *Integration to Robot* to *Performance Test* simulating that *Performance* defects are likely to be found in the *Performance Test* development phase rather than in the *Integration to Robot* phase.

The similarities on the signatures can be seen by comparing the format of the signatures constructed between Points 1, 2, and 3 on the two graphs shown in Figure 3.14.

FIGURE 3.14 Simulating Performance Tests



Basically, this section has shown validation through comparison. Defect signatures were compared to logic signatures, and differences between signatures were explained. This suggests that the measurement scheme is sound.

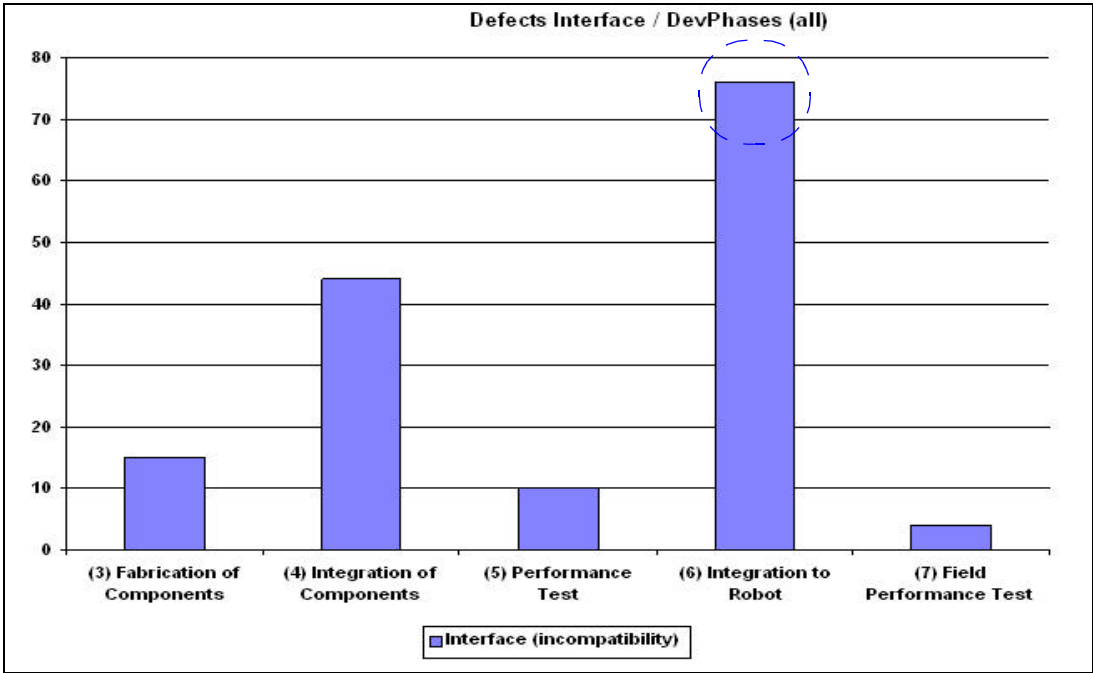
The next section will show validation through analysis of the defect data by relating it to events that occurred during Nomad development.

3.1.2 Defect Signatures and Facts on Nomad

A defect measurement system differs from a defect classification scheme because it captures development process characteristics [Chillarege, 1992]. In this section validation will be accomplished by relating process characteristics stored in the defect data to events that occurred during the development of the Nomad robot.

Looking at Figure 3.16 it is easy to identify the existence of a large number of *Interface* defects in the *Integration to Robot* development phase. It is known that one of the main reasons for interface problems is the absence of documentation containing detailed information on connectivity (i.e., how to interface to a particular component or subsystem) [Pahl, 1996]. So based on the large number of defects in the *Integration to Robot* development phase, we can assume that a problem existed in the Nomad development process. That is, this analysis identified a known development characteristic of the Nomad robot (i.e., lack of documentation). To check this hypothesis *Documentation* defects will be investigated next.

FIGURE 3.15 Defects Interfaces



Looking at the Defect Type Documentation plot (Figure 3.2 on page 90) and noticing that only two defects were reported one would guess that Nomad's development team did one of the following regarding documentation:

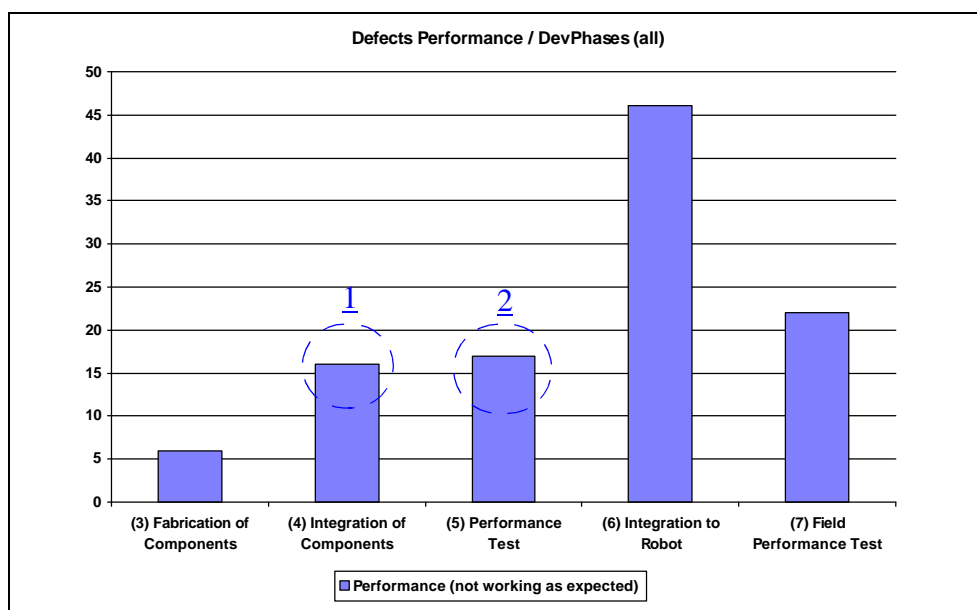
- The team did an outstanding job on documentation.
- The team did not collect *Documentation* defects.
- The team did not generate documentation.
- The team did not perform a documentation check.

Unfortunately, the two last reasons accounted for the number of *Documentation* defects shown. In fact, the only two *Documentation* defects collected were identified by an outside company hired to manufacture some parts. Since the great majority of parts manufactured for the Nomad were produced in-house, the lack of documentation was not a concern. That is, developers interacted frequently while developing the robot. This intensive verbal communication was necessary to compensate for missing documentation information. But, as expected, this lack of documentation had negative consequences on Nomad's development process.

It is important to mention here that this research does not intend to evaluate development procedures, the effectiveness of developers, optimization of designs, etc. Moreover, this is not a study of, or commentary on of the Nomad development process. It is beyond the scope of this research to identify or analyze these consequences. The goal here is to use the data generated by the measurement system to identify development process characteristics and refine the measurement tools.

Looking at Figure 3.16, notice the almost identical numbers of *Performance* defects in the *Integration to Robot* and *Performance Test* development phases. (See Points 1 and 2 in Figure 3.16. below.) It seems logical that a significant number of *Performance* defects are likely to be discovered during tests designed to verify performance (i.e., tests executed during the *Performance Test* development phase) but Figure 3.16 does not show an increase of the number of *Performance* defects for the *Performance Test* phase. Based on this lack of an increasing number of defects, we can assume that a problem existed in the Nomad development process. This assumption is supported because no significant performance tests were realized in the *Performance Test* phase.

FIGURE 3.16 Defects Performance



Assuming that different subsystems are developed by the same development team using identical schemes, one can expect that *Damage* defects found during the *Fabrication of Components* development phase will be present for one subsystem if the other subsystem has damage defects as well (for complex electromechanical systems).

There is no defect type *Damage* indicated during the development phase *Fabrication of Components* for the Pointing Mechanism subsystem. (See Figure 3.3. on page 91.) Several conclusions could be drawn from this:

- The component fabrication process was perfect.
- No *Damage* defects were collected.
- Fabrication of components was not performed in-house and thus no defect data was collected.

In the case of the Pointing Mechanism, parts were sent to be manufactured in an outside shop. Usually no defects are reported by manufacturing shops; parts are simply delivered as a product.

Thus, this indicates a characteristic of the development process. That is, the development process for the Locomotion subsystem was not the same one used on the Pointing Mechanism subsystem (i.e., the measurement system captured a process characteristic).

3.1.3 Sufficient Conditions

The sufficient conditions are a set of requirements that once satisfied suggest a valid measurement system [Chillarege, 1991].

Process sub-space fully covered by attributes

The taxonomy attributes of a measurement system need to be associated with the process that is to be measured. Moreover, the attributes should fully span the process sub-space so that a sufficient number of process characteristics can be collected to enable extraction of useful information about any part of that sub-space.

As the nature of this work is experimental it is difficult to guarantee that the first set of RODC attributes will satisfy the sufficient conditions. Some adjustments were required during the RODC development so that the attributes would span the entire problem sub-space (motion systems). All 465 defects were classified using the current set of attributes. Therefore, it is assumed that the current attributes adequately span the process sub-space.

Orthogonality

The second sufficient condition is the orthogonality during classification and extraction of information. Defects being classified should be placed in a distinct independent position in the classification scheme and information extracted (e.g., a process deficiency) should point to a distinct point much like points in Cartesian space.

After adjustments during the development of the RODC scheme, defects were classified without ambiguity. Thus, the classification scheme indicate orthogonality. Orthogonality in the extraction of information is addressed in Section 3.2.: Information Extraction.

Number of attributes adequate to make the necessary inferences

The third sufficient condition is the necessity of having an adequate number of taxonomy attributes so inferences can be made about the process. The process of identifying the number of attributes was described in Section 2.2: Taxonomy Design. The attributes provided information that developers needed. Actually, new information can still be gathered from the current RODC scheme.

Extracting information from the RODC is addressed in Section 3.2: Information Extraction.

3.1.4 Conclusions on Validation

The Validation performed here is not perfect. It is a starting point that should be refined in subsequent research. There are some weaknesses as described below.

Because no defect data were available prior to this research, the logic signatures are based more on Field Robotics Center developer's feelings and personal observations rather than on scientific data analysis. Therefore, these signatures are questionable. Nevertheless, they serve as a starting point. In the future these logic signatures will be replaced by defect models built from defect databases. Thus, they will become more dependable as additional data is collected.

Also because the number of defects collected enabled logic signature comparison only for a few defect types, the number of process characteristics that could be validated was limited. Moreover, no defects were collected in the initial development phases. Therefore, the validity may be questionable for these early stages.

As illustrated in previous sections, the measurement system appears valid based in the sufficiency tests and comparison to logical and actual data. Defect signatures generated from the measurement system were successfully compared to logic signatures and differences were explained. That is, analysis of defect data generated by the measurement system showed that the development of Nomad motion systems followed a logical path according to previous experiences from the development of mobile robots at the Field Robotics Center, Carnegie Mellon University.

This section concludes the validation steps. Now that the measurement system has been shown to adequately capture process information, information extraction can follow. That is, information useful to developers and management can be extracted.

3.2 Information Extraction

The objective of this section is to describe a process to extract relevant information from defect data collected with the measurement system. Relevant information here means information important to the development of mobile robots. In our case, the focus is on reliability and project management information extraction.

Analysis Without Math

In the late 1980's very few successful quality process control programs existed in the United States [Ellis, 1986]. The lack of success was due to the training executed in most programs. In most cases, training was executed by statistical experts with an emphasis on complex statistical and mathematical terms. These terms scared management and factory workers, so attempts to implement the quality process control measures failed. On the other hand, experience has shown that when statistical logic is used but statistical terms are not, quality programs are much more accepted since workers and management are able to see that statistical quality control is logical.

This research takes the approach of developing a measurement system that can be used by all developers. For this reason, tools were developed without the use of complex statistical terms or procedures. Doing so it is expected that the acceptance to the method will be reasonably fast and training can be kept to a minimum. If developers can understand the concepts quickly and realize that no significant extra work is required from them in order to use the system, then they will probably be willing to cooperate in the implementation and use of the RODC system.

A survey was sent to robot developers at the Field Robotics Center (see Appendix C) to inquire what information developers consider to be relevant in the mobile robot development process. The results of the survey indicated data of most use to developers:

- What are the most critical defects in a mobile robot development?
 - How can the most critical defects be evaluated? (i.e., what kind of tools should be used?)
 - How can the most critical defects be addressed or acted upon?
 - What are the most common defects for each development phase?
-

- What is the cost to fix the most common defects?
- What are the most expensive defects?
- What caused the defects to be found?
- What components present an uncommon proportion of failures in mobile robots?

These items will be addressed throughout this section.

It is beyond the scope of this research to indicate defect remedies. Therefore, the third item (how to address / act on the most critical defect?) will not be addressed in this research. The idea is to provide tools that will allow a development team to identify development problems. Solutions for these problems will depend on development characteristics (e.g., development team, resources available, previous experiences, etc.). The next sections show how the measurement system can be used to indicate development problems.

The process of extracting information is accomplished by running queries that allow the users to cross fields in the RODC database and then present the results with plots from which conclusions can be drawn.

The next sections show how mobile robot development process information was extracted using the RODC prototype.

3.2.1 Crossing Fields

The procedure of crossing fields consists of displaying the collected defect data in such a way that information can be extracted. For instance, simultaneously displaying fields from the defect database (e.g., Defect Type, Triggers, and Causes) enables cross information extraction. The list of important information (relevant to developers) extracted from the survey is developed inside the following divisions:

- Most Costly Defects
 - Most Effective Triggers
 - Development Phases with Most Defects
 - Most Common Causes for Defects
 - Components that Presented More Defects
-

Most Costly Defects

The relevant information according to developers addressed in this section are: the most critical defects, the most expensive defects, and the cost to fix the most common defects. Critical defects here are referred as Costly Defects.

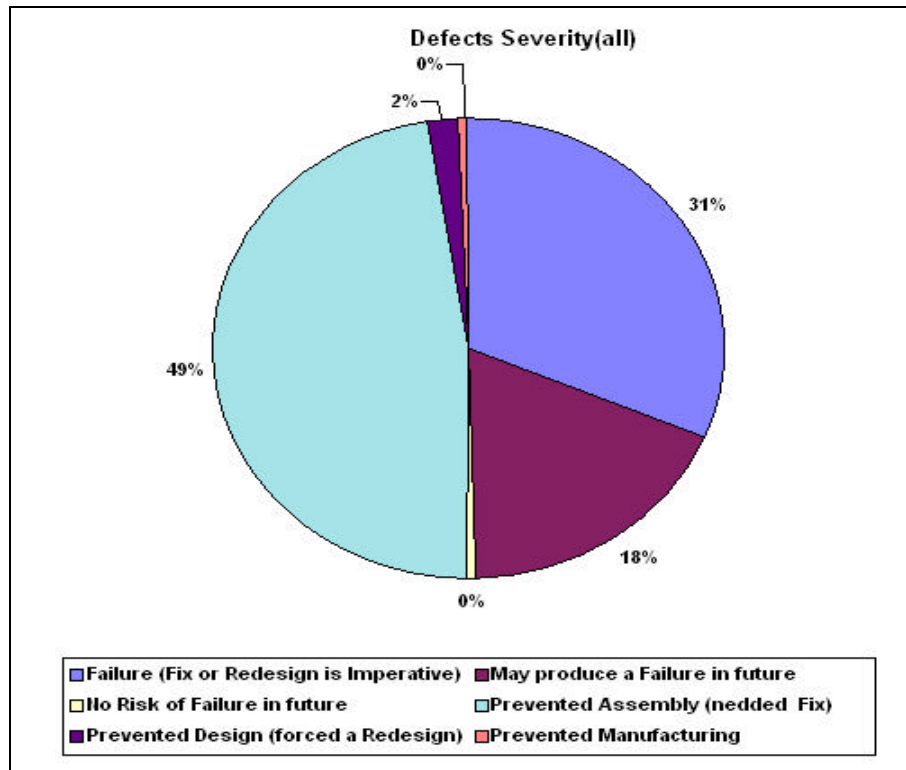
Two types of Costly Defects are investigated: reliability and management. Regarding reliability one can say that the most costly defects are the ones that have the highest severity (in terms of reliability): in our case *Severity - Failure*. But for management the most costly defects are those that impact personnel: in our case *Impact Personnel - 2 to 3 Persons*. Also, costly defects for management are the ones that require more time to be fixed, in this case using the field *Time to Fix (hrs.)*.

Another interesting aspect of the most costly defects is the weight effect that these can have in the data analysis. For instance, consider defect type *Damage* that might have dozens of defects and defect type *Interface* with hundreds of defects in the defect database. One could assume that because the interface defects occurred more often than damage defects, resources should be spent to prevent them. But in reality the damage defects were Costly Defects, ones that should receive priority in being prevented in the next development of a mobile robot, or ones that should be deal with in-process to change a current defect trend.

The following figures show examples of plots from where Costly Defects were identified in Nomad's defect data.

Figure 3.17 shows the distribution of Nomad's defects severity field. According to the definition of Costly Defects from the reliability viewpoint, three types of severity are relevant: *Failure*, *May Produce a Failure*, and *No Risk of Failure*.

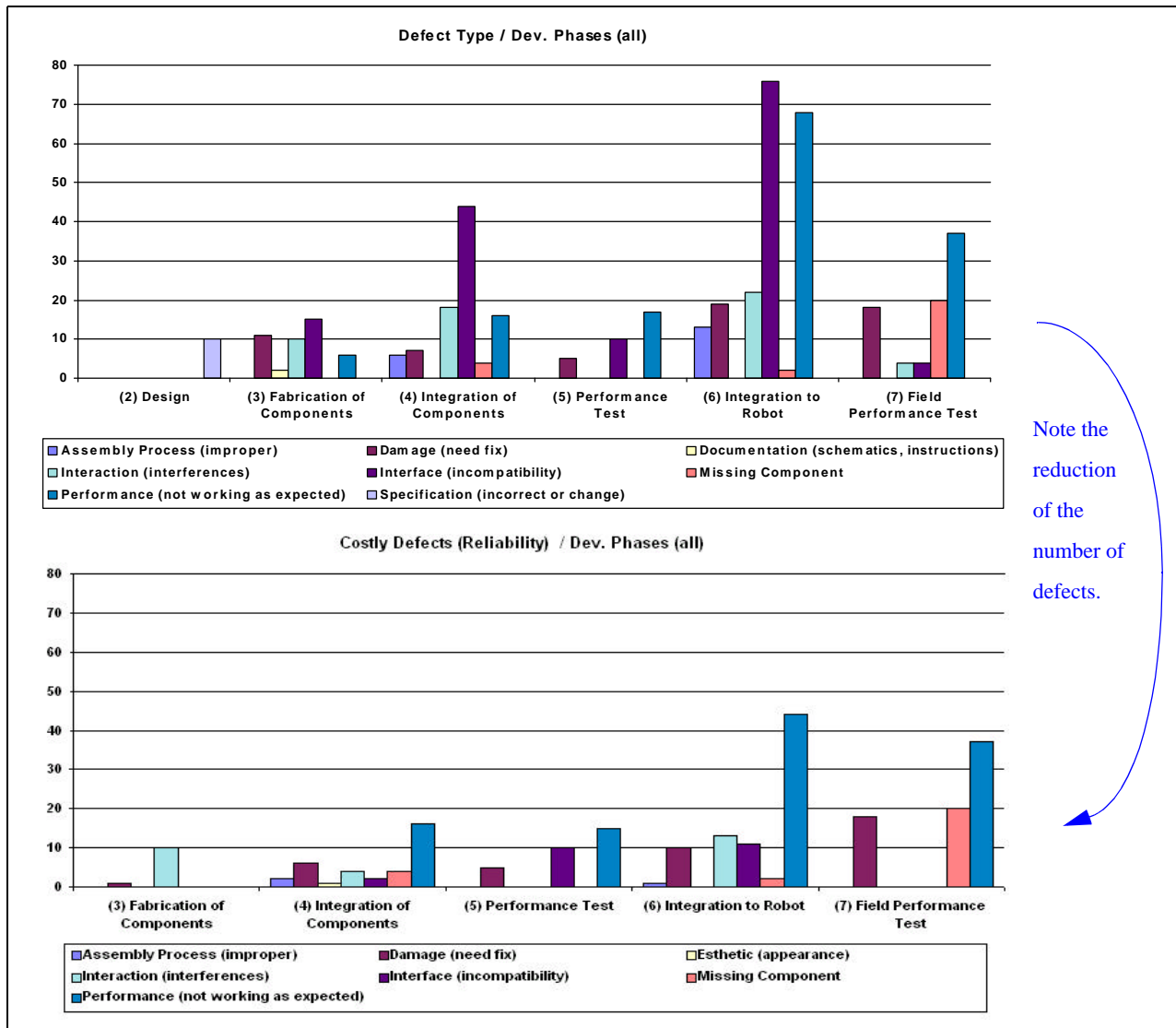
FIGURE 3.17 Nomad Defects Severity Distribution



A development team using this method can actually decide where they want to focus their efforts. If, for example, reliability is a prime consideration, this method would let developers see where to spend resources. To carry this example further, as a result of identifying the Costly Defects for reliability, the development team could focus on 49% of the total number of defects. That is, focus on defects *Failure* (31%) and *May produce a Failure in the future* (18%). In another example, assuming that the team is looking for *Severity - Failure* (that is, defects that failed the system), the number of defects is reduced to 31% of the total number of defects.

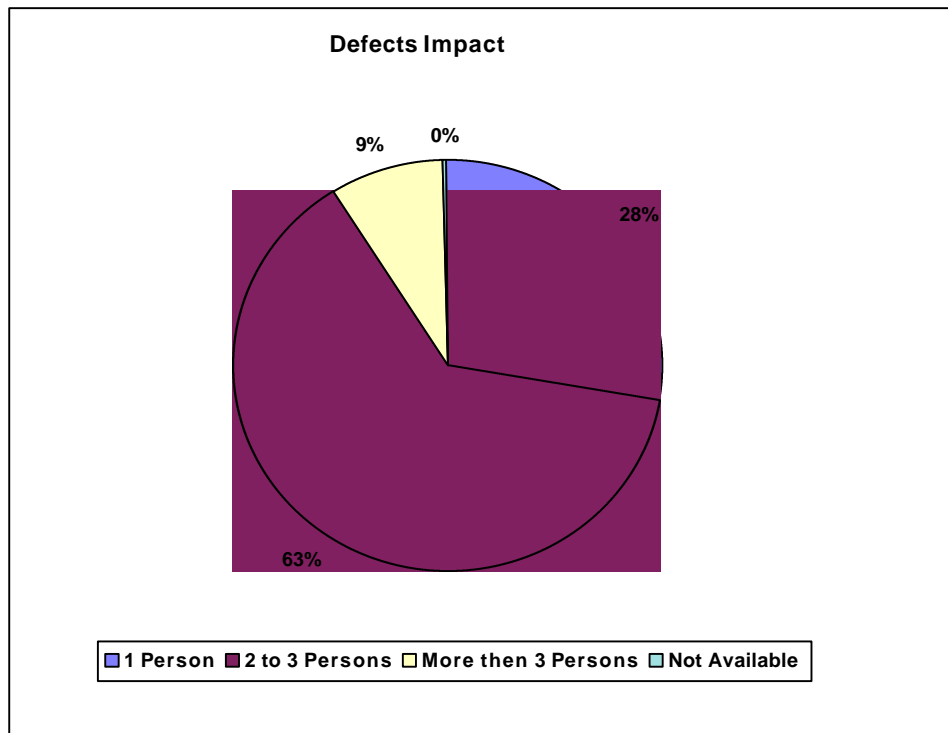
Figure 3.18 shows the entire defect set for Nomad and the Costly Defects for reliability.

FIGURE 3.18 Reduction of the number of defects to be studied - Costly Defects for Reliability



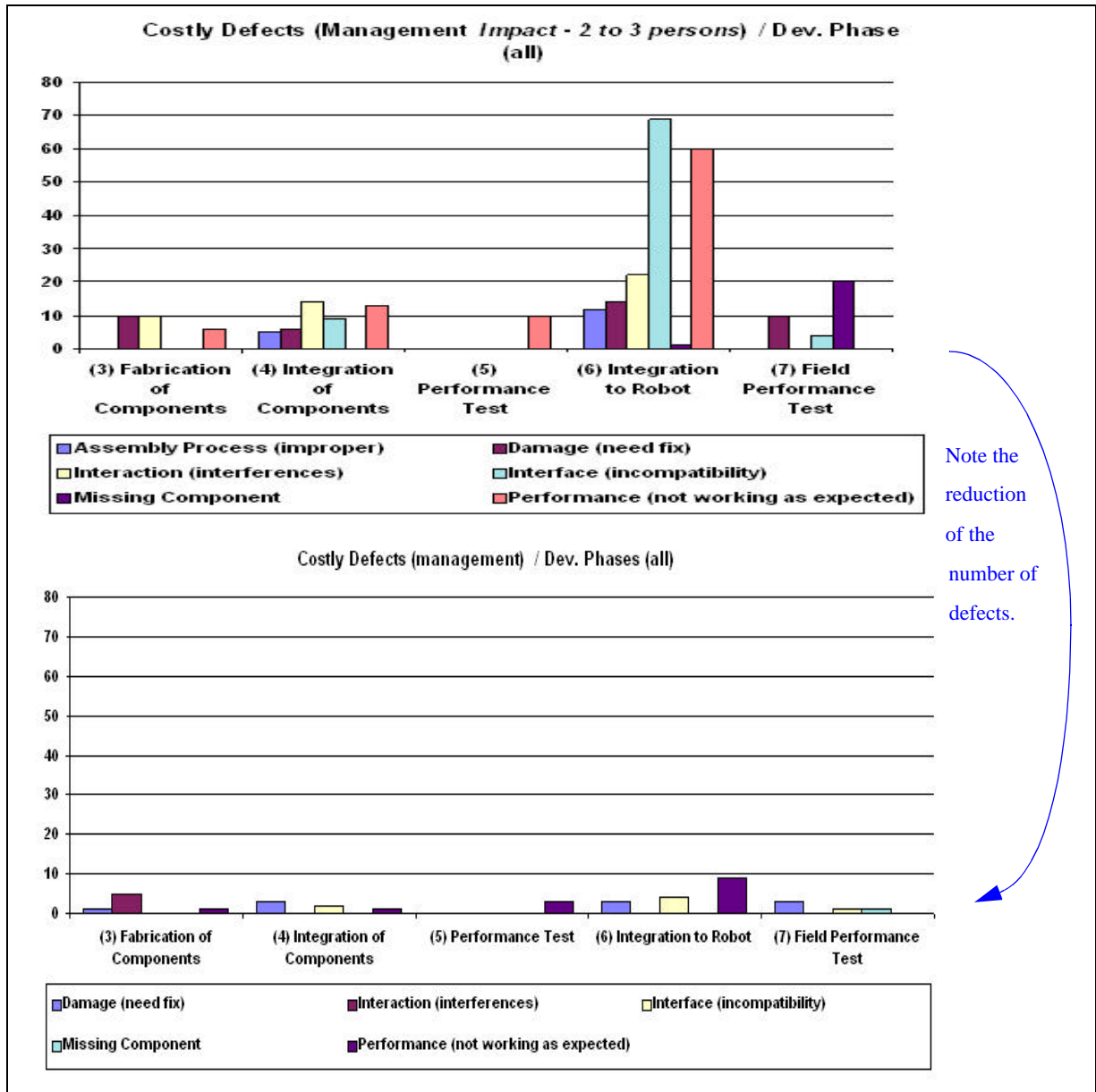
The reduction of the number of defects to study in this case can help focus attention on critical defects in terms of reliability.

Figure 3.19 shows the distribution of Nomad’s defects impact field. According to the definition of Costly Defects from the management viewpoint, the most costly defect is *Impact Personnel - 2 to 3 Persons*.

FIGURE 3.19 Nomad Defects Impact distribution

As a result of identifying Costly Defects for management, the development team would focus on 63% of the total number of defects. Actually, assuming that management is looking for *Impact Personnel - 2 to 3 Persons* and for defects that require significant time to be fixed (e.g., *Time to Fix (hrs.)* ≥ 5 hours), the number of defects to be investigated would be considerably reduced. Figure 3.20 illustrates this example. Upper plot is all defects with impact on 2-3 persons. The lower plot is the subset of these defects that required more than five hours to remedy.

FIGURE 3.20 Reduction of the Number of Defects to be studied - Costly Defects for Management.



This section shown how costly (critical) defects can be extracted from the RODC database. Identifying the costly defects helps developers focus on a smaller number of defects for data analysis. The next sections will show how the measurement system can be used to extract information that matters to developers. Also they will illustrate how costly defects can be used to narrow the focus of attention on defects.

Most Effective Triggers

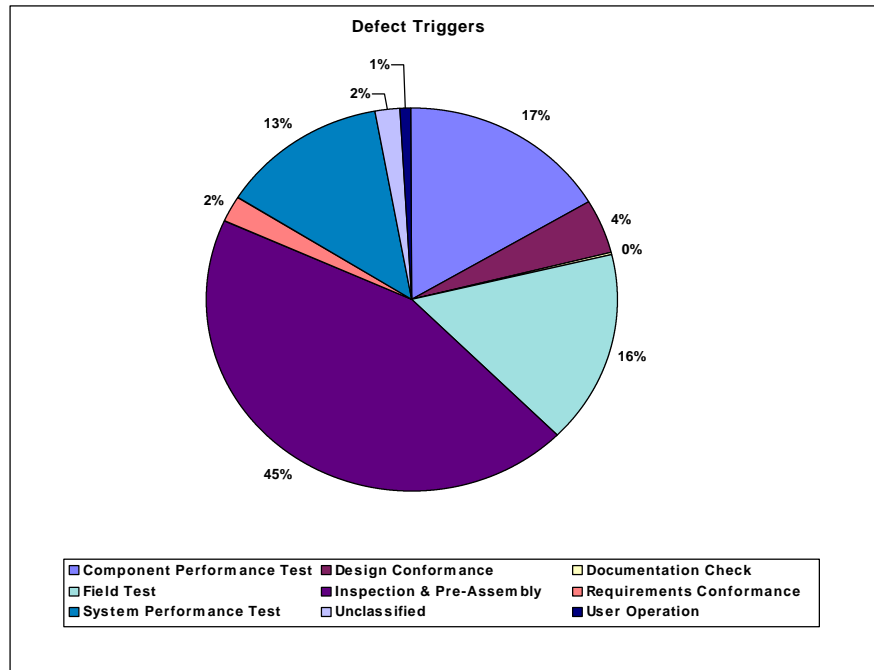
The relevant information according to developers addressed in this section is how defects were identified in each development phase. The expression “Identify a defect” is here referred to as the trigger for the defect (that is the condition or state that made it possible to discover the defect).

The idea is to extract the most effective triggers and the development phases where defects are discovered. Effective triggers can be used to reveal defects so they can be fixed in the early stages of a mobile robot development. From a reliability viewpoint, identifying and fixing defects in earlier development phases means ensuring that fewer defects will appear in a later development phase (i.e., *Long Term Operation*). Thus, fixing defects early should improve robot reliability.

Management can make use of the effective triggers to reveal costly defects. Identifying and fixing defects in early stages can save valuable resources. For instance, a mechanical structural defect may cause a great number of resources to be consumed in a later development phase if this defect is not detected in earlier development phases. That is, the work to fix the defect can take more time and/or resources (e.g., re-designing dependent parts, disassembling complex subsystems, preventing developers from working, etc.).

The following figures show examples of plots in which effective triggers were identified in Nomad’s defect data. As can be seen in Figure 3.21, the effective triggers for the Nomad project were: *Inspection and Pre-Assembly* (45%), *Component Performance Test* (17%), and *Field Test* (16%). That is, these triggers were the ones that revealed the majority of defects during the Nomad development. This kind of information is extremely important because it shows the kinds of activities that the development group performed that were effective in revealing defects. Therefore, these activities should be re-enforced in future developments.

FIGURE 3.21 Nomad Defect Triggers Distribution



Knowing when to apply the most effective triggers can aid in planning activities for the development group. Figure 3.22 shows when triggers were more effective.

FIGURE 3.22 Development Phases Where Triggers Were More Effective

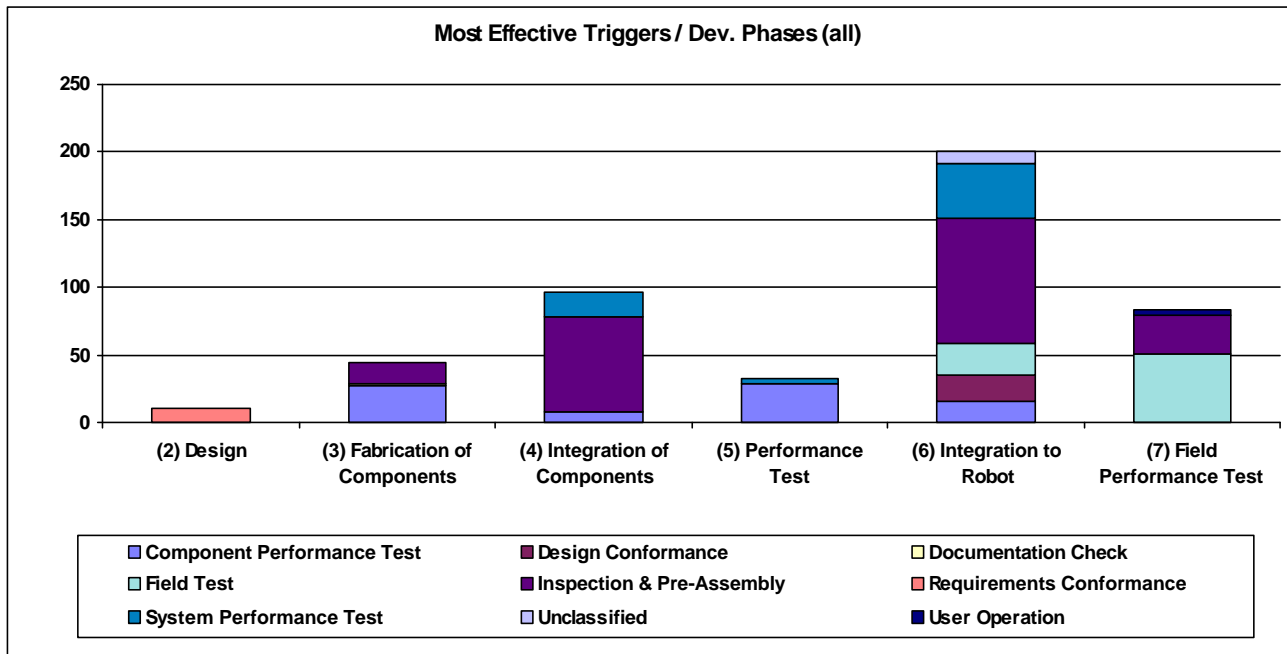
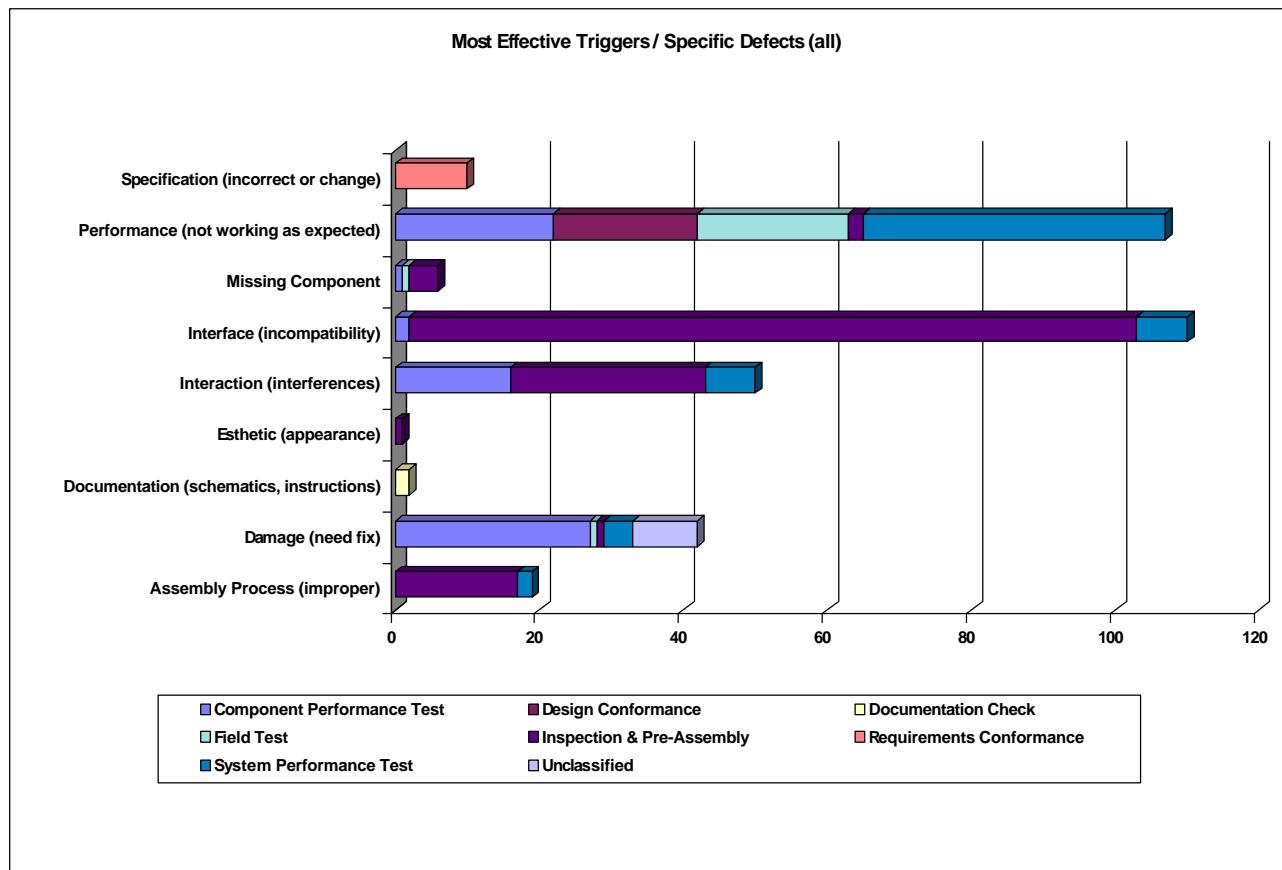


Figure 3.22 shows that *Inspection & Pre-Assembly* was the most effective trigger at *Integration of Components* and *Integration to Robot* development phases. This observation seems logical because these are the development phases where more assembly is performed. Inspecting and pre-assembling components before the final assembly process can reveal many defects (e.g., *Interface* defects) as expected.

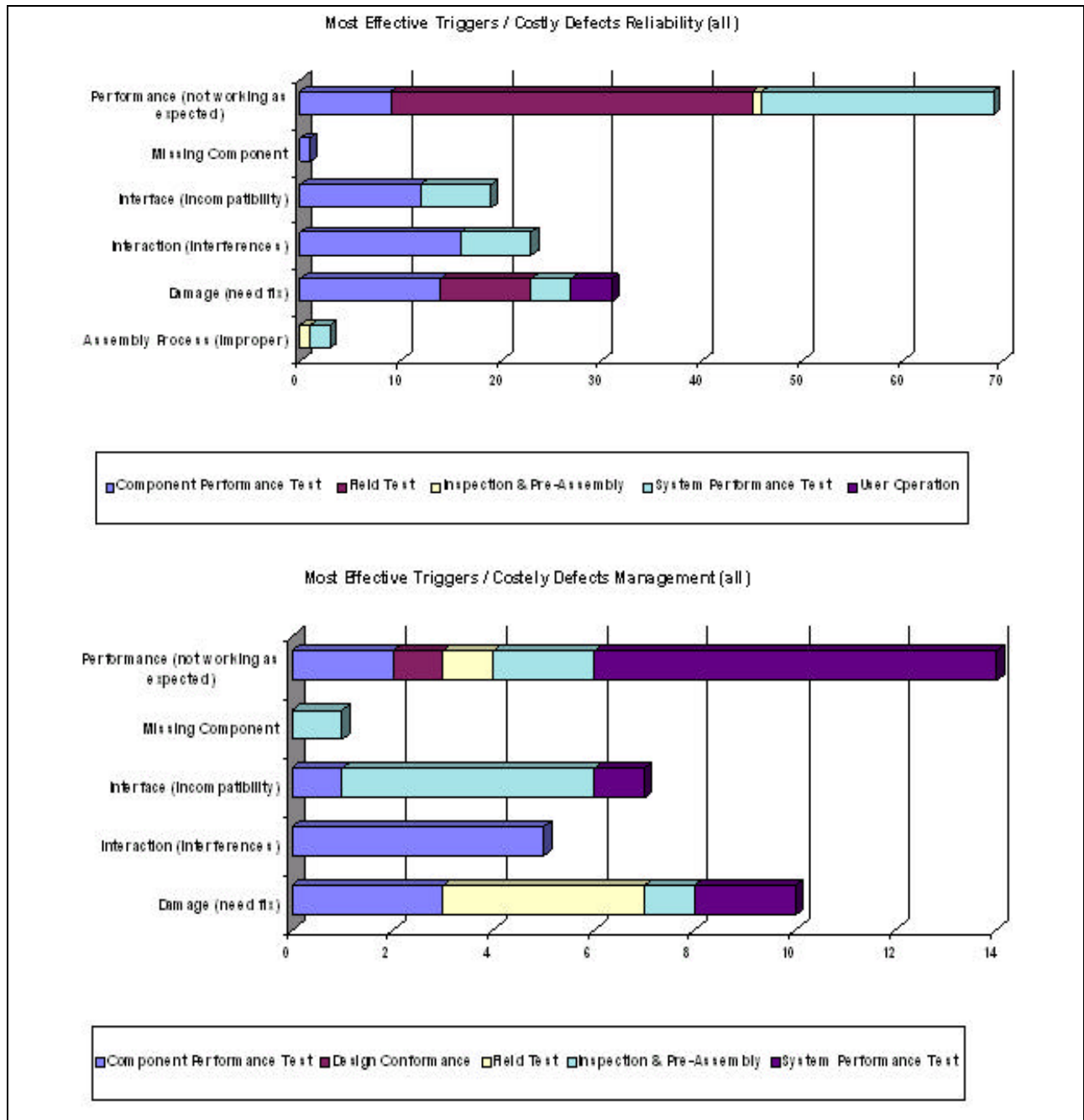
Another important issue regarding triggers is the identification of triggers that were more effective in revealing specific defects. Figure 3.23 shows a plot in which the most effective triggers for specific defects can be extracted (for instance, the most effective trigger for the defect type *Interaction* and defect type *Interface* is the trigger *Inspection & Pre-Assembly*). Thus, if a development team is analyzing these types of defects then the most effective activity to reveal these defects is *Inspection & Pre-Assembly*.

FIGURE 3.23 Most Effective Triggers for Specific Defects



Based on the approach for most effective triggers for specific defects one can identify the most effective triggers for costly defects. Figure 3.24 shows triggers for Nomad’s costly defects from which the most effective triggers can be extracted.

FIGURE 3.24 Nomad Effective Triggers for Costly Defects

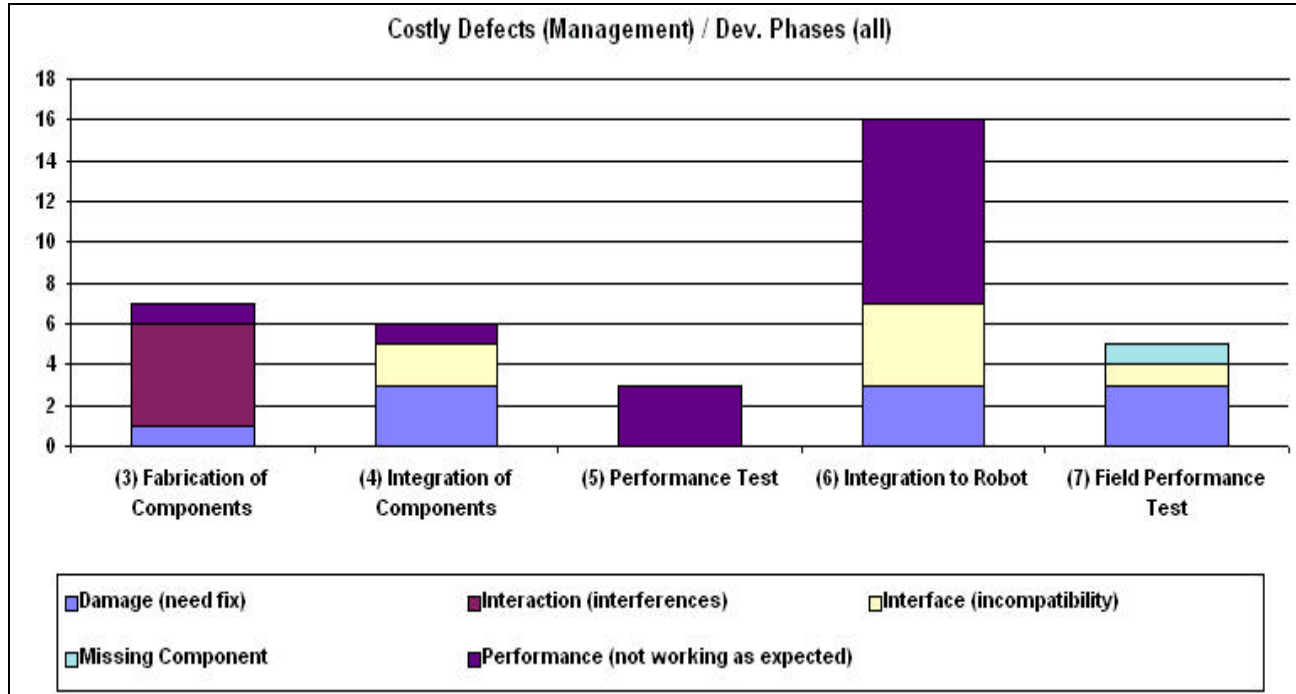


Development Phases with Most Defects

Identifying development phases that contain the greatest number of defects or specific defect types can aid management in deciding where, when, and how to spend resources. That is, this investigation enables the identification of development problems. One can visualize if the robot is becoming more reliable by tracking development phases with respect to most costly defects on reliability. Therefore, tracking reliability goals can be accomplished.

Figure 3.25 shows plots of Nomad's costly defects vs. development phases. One can notice that in the *Integration to Robot* development phase the number of defects increases considerably. Therefore, in the future management should consider bringing more developers to the team during this development phase. Also it is possible to note the contribution of each defect type, so management can better decide the type of developer who should be hired for specific development phases. That is, deciding the expertise necessary for specific development phases is made possible.

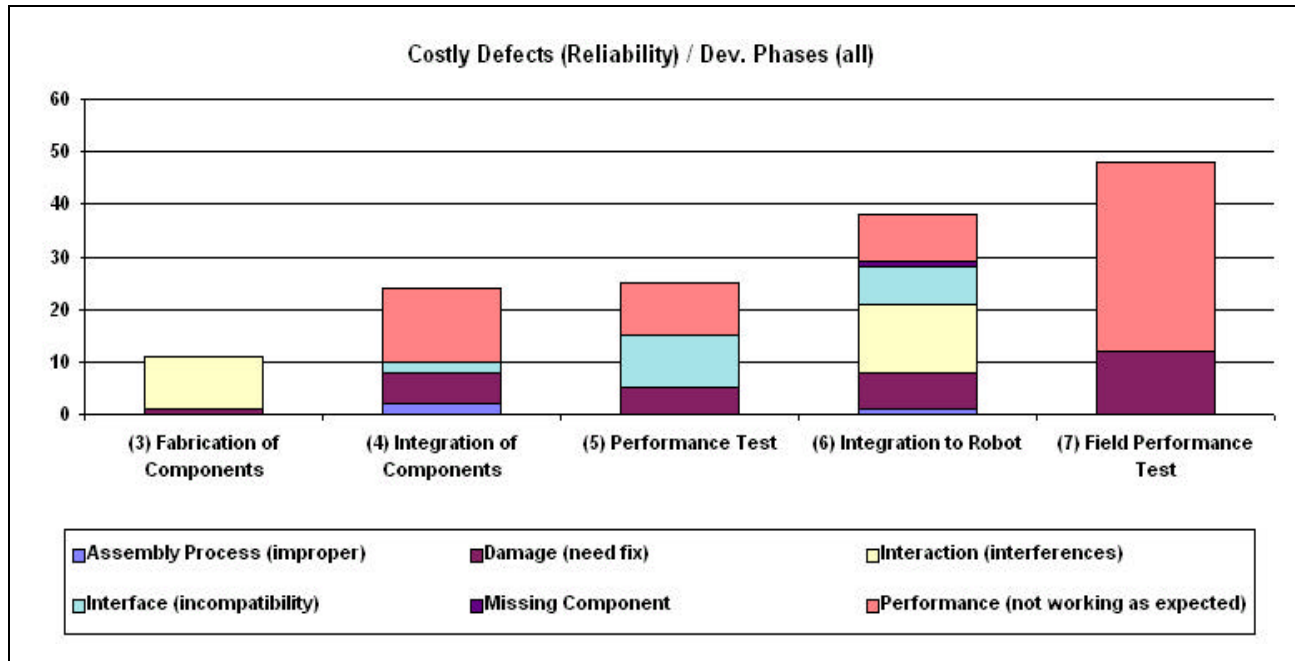
FIGURE 3.25 Nomad Costly Defects (Management) vs. Development Phases



Assuming that reliability can be tracked by quantifying the number of costly defects (on reliability) in

each development phase, one could use Figure 3.26 - Nomad Costly Defects (Reliability) for tracking reliability goals. For instance, tracking reliability can enable robot maturity monitoring (i.e., is the robot mature enough to be deployed?).

FIGURE 3.26 Nomad Costly Defects (Reliability) vs. Development Phases



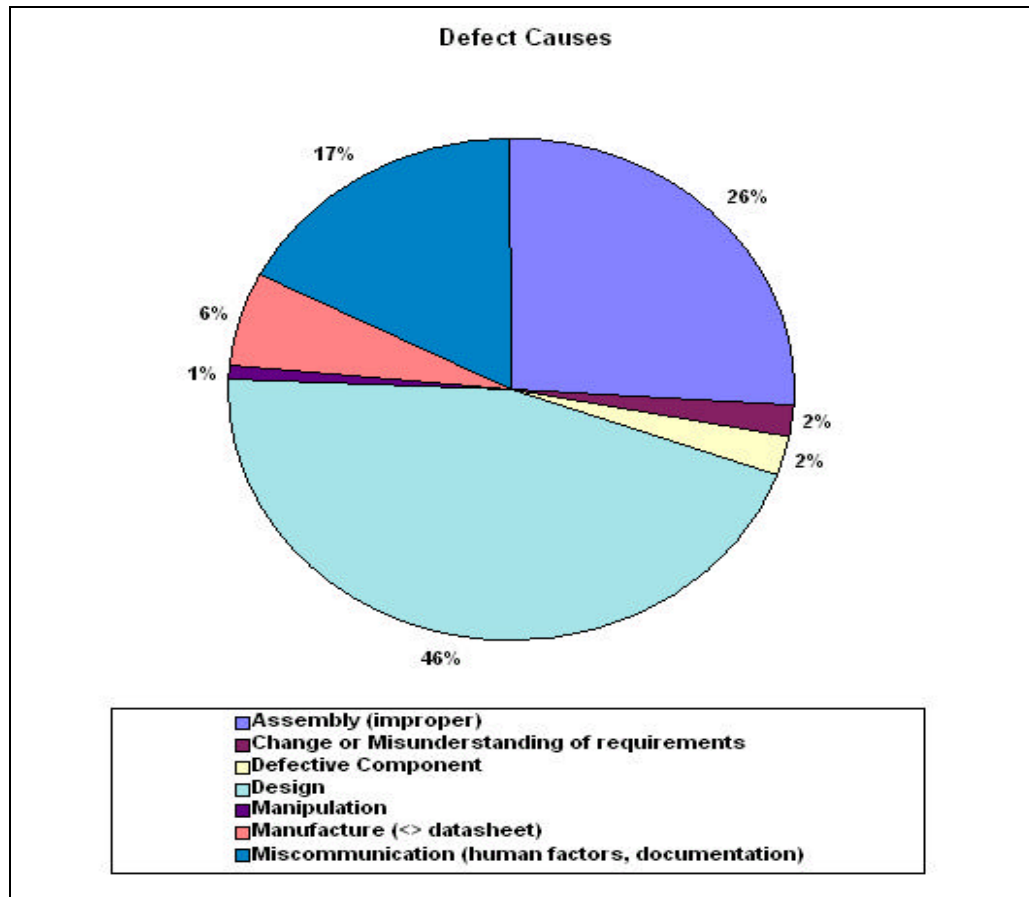
As can be seen at Figure 3.26 even though the number of defects collected during the *Field Performance Test* phase was very limited, the number of defects that were costly for reliability during the *Field Performance Test* was higher than in the other development phases. Ideally a robot should be deployed when 1) the number of critical defects becomes small (or decreases), thus showing improvements in the robot's reliability; or 2) a specific reliability goal is achieved.

Most Common Causes for Defects

As in the previous section, identifying the most common causes for defects can aid management in deciding where, when, and how to spend resources. Regarding reliability, knowing the most common causes for costly defects (on reliability) enables isolation of these causes. Therefore, reliability can be improved by diminishing the number of costly defects. Figure 3.27 shows that the majority of defects

during Nomad motion systems development were caused by improper design.

FIGURE 3.27 Nomad Defect Causes



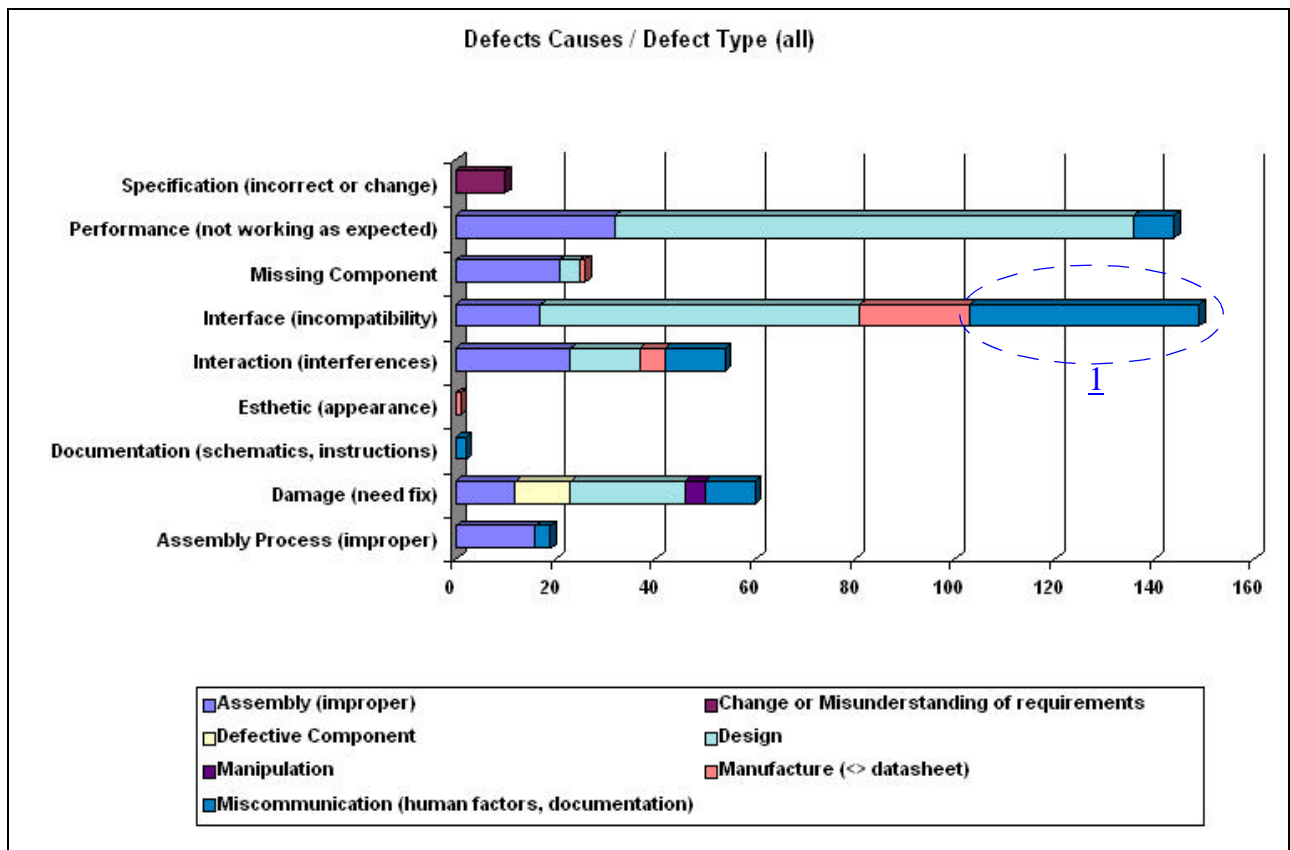
Saying that defects were caused by improper design is not very useful information for developers. Unless these design-related defects are detailed, little can be inferred from Figure 3.27. Unfortunately the current version of the RODC does not detail design causes. The main reason for this is that the RODC prototype was not available during the Nomad *Design* phase. Developers were not able to remember the exact reasons for design flaws. Therefore, the author decided not to detail causes since it would not be possible to experimentally validate the RODC taxonomy for design-related attributes.

Nevertheless the RODC method can be easily refined to include such details. That is, for future work RODC can be refined to include detailed description of causes for defects that are related to design. Then it will possible to use RODC to better help developers address design issues.

Figure 3.27 can be used to reinforce the importance of design reviews. The Nomad development process did not include design reviews. This fact may explain why the majority of Nomad defects were caused by design.

Figure 3.28 shows a plot of defect types and the causes of Nomad defects.

FIGURE 3.28 Causes of Nomad Defects



Note that in addition to besides *Design*, *Miscommunication (human factors, documentation)* was the other main cause for defects *Interface*. (See Point 1 in Figure 3.28.) As explained before in this research, there was a lack of documentation in the Nomad development process. This information shows that a good part of defect *Interface* could probably be avoided by adopting a documentation policy. The author believes that robot developers know the importance of documentation, but until the time of this writing no reports quantifying defects caused by documentation were available. The information presented here should help to convince developers of the necessity of thorough documen-

tation.

Components That Present Most Defects

Identifying the components that present the most defects can aid the development team in understanding what categories of components need special attention. For instance, Figure 3.29 shows that electrical *Connectors and Cables* are responsible for 64% of the total number of electrical related defects.

FIGURE 3.29 Electrical Components Defects Distribution

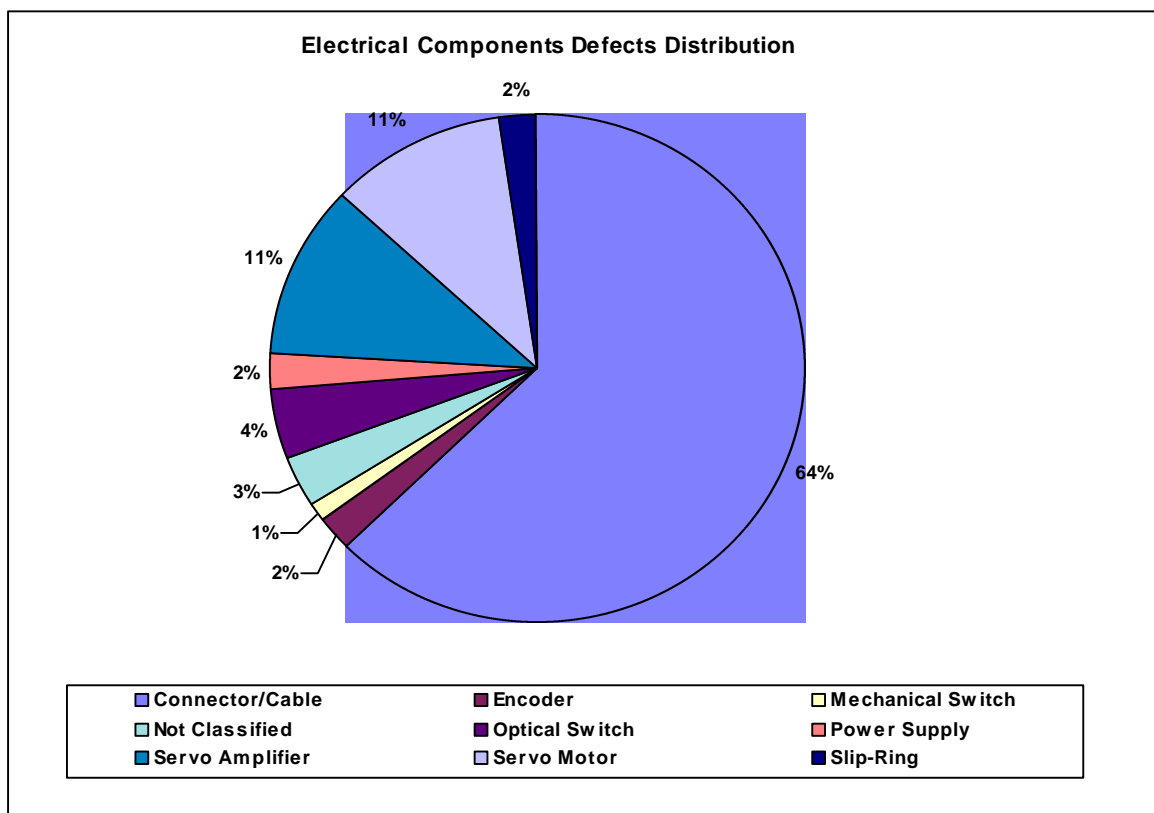


Figure 3.30 shows the causes of the electrical components related defects.

FIGURE 3.30 Causes of Electrical-Related Defects

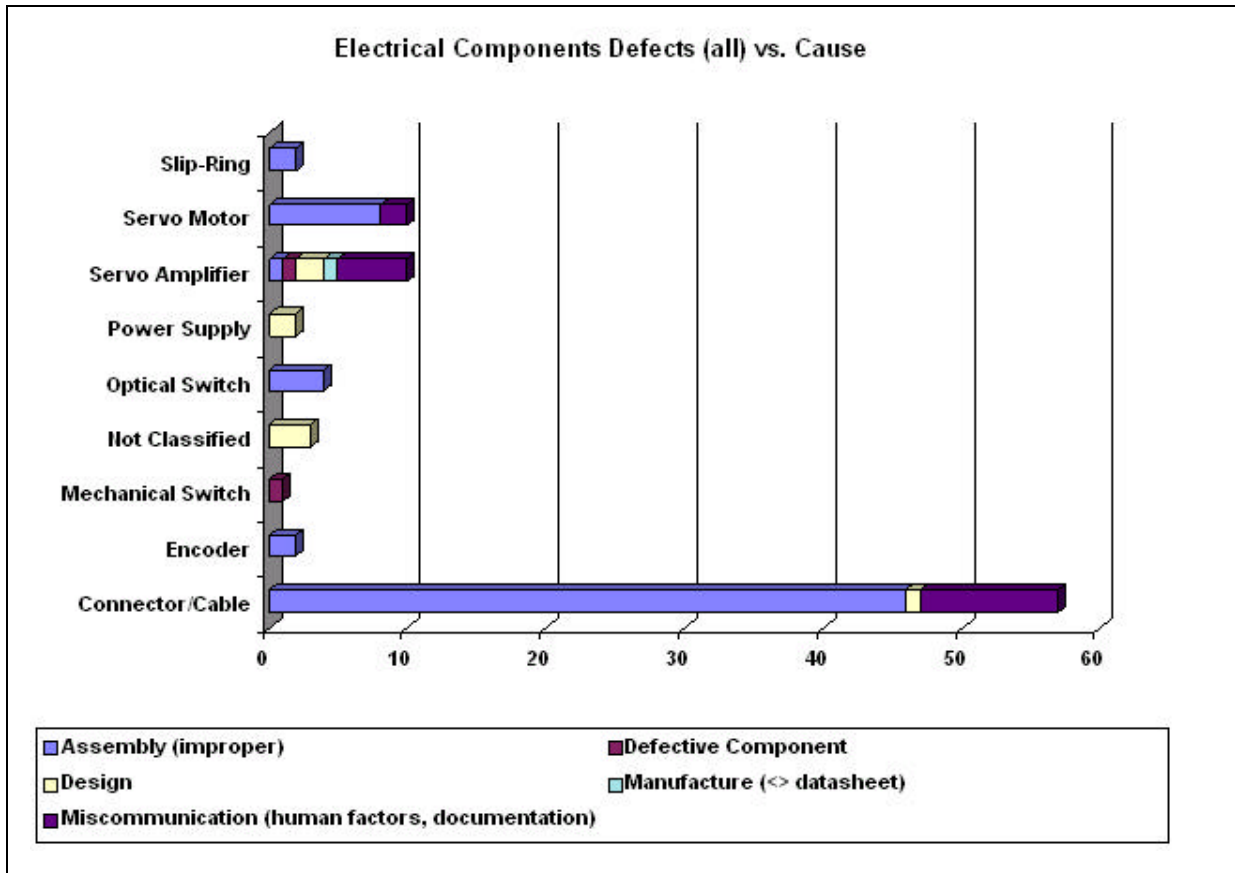


Figure 3.30 shows that the primary cause for electrical-related defects was *Assembly*. This is one more example of the kind of information developers need to help them decide where to spend resources.

Figure 3.31 shows that *Structural Support* is responsible for 64% of the total number of mechanical-related defects. Therefore, special attention should be given to these kinds of components. It is also interesting to note that 36% of mechanical defects on Nomad were on mechanical components *Fasteners*. This is an example of component technology that seems simple to developers but still keeps presenting problems in complex electromechanical systems such as Nomad.

FIGURE 3.31 Mechanical Components Defects Distribution

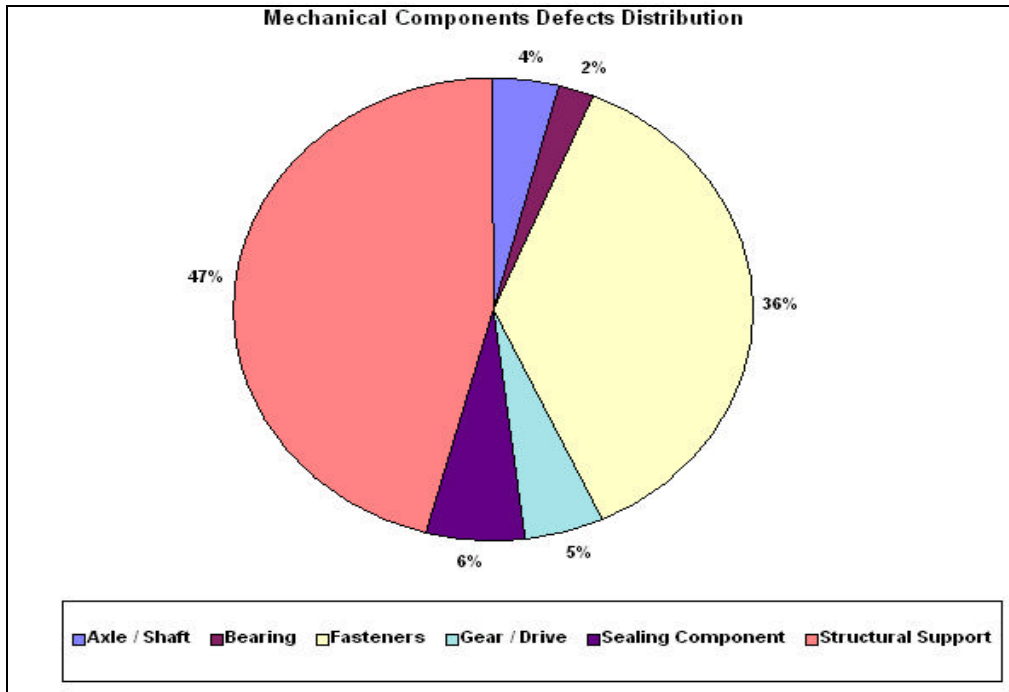
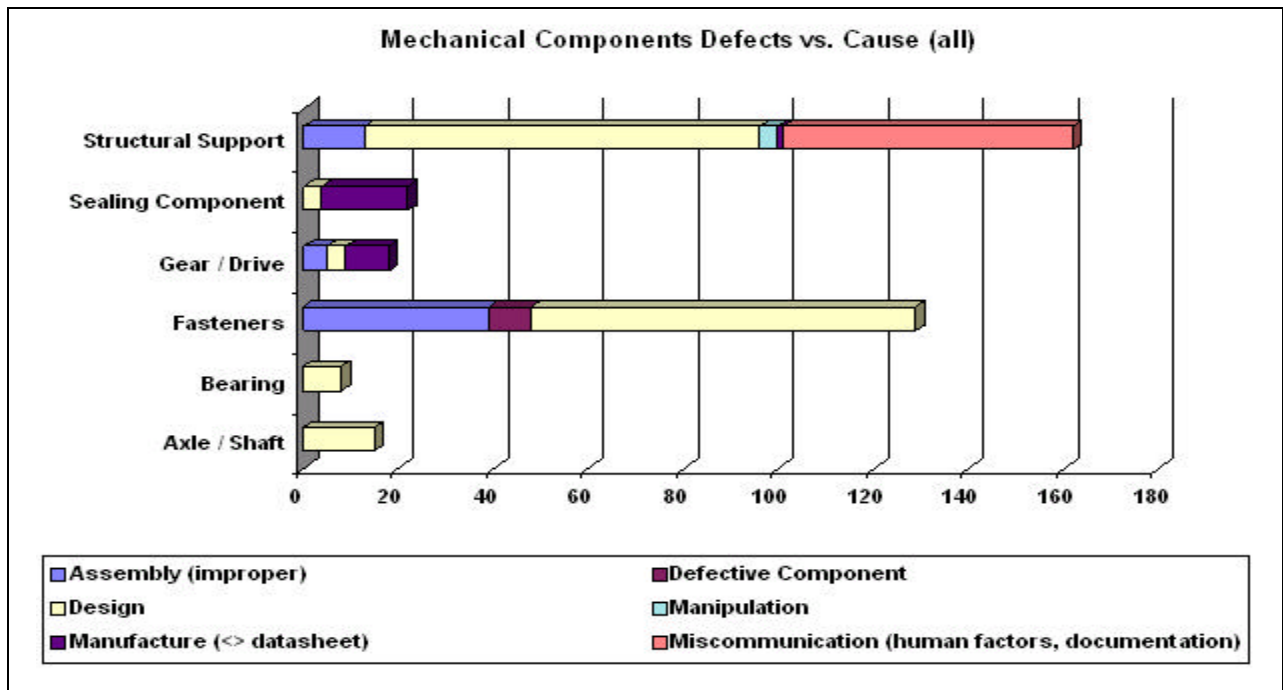


Figure 3.32 shows the causes for the mechanical components related defects.

FIGURE 3.32 Causes of Mechanical-Related Defects



3.2.2 Conclusions on the Information Extraction

This section has illustrated how to extract important information about the development of mobile robots from defect data collected with the measurement system (RODC) developed in this research. The focus is on identifying development problems, rather than recommending appropriate courses of action. The examples considered here are based more on Field Robotics Center developers' feelings and personal observations rather than on feelings and observations from the mobile robot community at large. But, as explained in the prototype chapter, the measurement system is parametrized. This parametrization allows a reasonably easy change to the prototype to accommodate changes without having to re-design the database. For instance, new database queries and crossing field procedures can be generated in a short time (minutes). Thus, using this research as a guideline, incorporating important information into the measurement system in the future should be straightforward.

The crossing fields procedure were developed in detail to illustrate how information can be extracted from real data. Although the procedures and results presented in this section are primarily a starting point for future work, the information extracted can already be useful in the future development of mobile robot motion systems. Effective triggers can be used to reveal defects in the next generations of mobile robots. For example, showing the most common causes for defects, the development team can better apportion resources to avoid defects. Finally, examples of how the information can aid management and improve reliability were described.

Chapter 4

Conclusions

This chapter describes lessons learned during the development of the RODC, contributions of the research, and proposes future work.

4.1 Lessons Learned

The opportunity to apply the RODC to the development of a mobile robot was fundamental to experimentally tune the measurement system and to draw the observations described in this chapter.

An important lesson learned in this research is the importance of commitment of management and developers to the RODC measurement system. Dr. Deming, a key player in the improvement of Japanese manufacturing organizations, strongly believed that the responsibility for quality stays with management, while very few responsibilities stay with developers [Montgomery, 1996]. The author believes that this statement is correct and applicable to the RODC experience. Because Nomad management had not been convinced of the utility of RODC, data collection was not always easy and in-process feedback did not occur.

The lessons learned are:

- Taxonomy Design
 - Data Collection
 - Data Analysis
-

4.1.1 Taxonomy Design

The main objective of a taxonomy design is to enable data collection. Not only defect types should be identified, but many other attributes used to capture process information should be included in the taxonomy. The goal is to enable the data collection necessary to extract process information without overwhelming the user with questions. That is, only the minimum amount of information required for analysis should be collected.

Experience with RODC use has shown that the number of attributes should be carefully chosen since the time spent for data entry can be long and can thus discourage users from classifying all defects. It was found that data entry time was inversely related to the user's tendency to record defects.

The set of attributes used in the RODC was experimentally verified. Using RODC in a mobile robot development turned out to be very important because it enabled modifications on the taxonomy as described in Chapter 2, Section 2.2. The IBM ODC taxonomy became stable after several years of modifications [Santhanam, 1997]. Based on IBM's experience and this first experience with RODC, it is expected that the RODC taxonomy will require modification during the next few applications before it can be expected to stabilize.

Taxonomy attributes that require developers to guess (for instance, Time to Prevent) should be removed because they are dependent on the developer's mood and physical conditions. Experience with RODC has shown that if the developer is tired or upset with a defect consequence (at the time of the defect classification), it is likely that the attribute that required a guess will not be completed or it will be incorrectly completed. Besides that, developers dislike the fact that they had to take the extra time guessing an answer to classify a defect. This fact was noticed in the following RODC attributes: *Time to Prevent (hrs.)* and *Considered During Design*.

Time to Prevent (hrs.). The idea here was to collect information that would enable comparison between the time used to fix a defect to the time that would have been required to prevent the defect (evaluating design procedures). For instance, a defect caused by an outdated document and that took 10 hours to be fixed could be prevented by having a document check procedure that would take one

hour of developer time. Some conclusions can be drawn from this kind of data if a significant number of defects with similar characteristics are found. Most of the time developers did not know the answer or did not want to talk about this attribute. Thus most of the time this attribute was not used to classify defects.

Considered During Design - Developers did not remember if the defect was considered during the design process to be either possible or likely to occur.

These two attributes could theoretically be used to evaluate design procedures of mobile robots but experience has shown that they had very limited utility because the data was based on a designer's guess or opinion rather than facts.

Some attributes caused confusion for developers. They required a classification policy to be implemented for data consistency. The attributes are: *Time to Fix (hrs.)* and *Impact Personnel*.

Time to Fix (hrs.) - It was not clear if the time spent to fix the defect should also include manufacturing time or just include project personnel time. The decision was made to consider only the time that project personnel spent working fixing the defect because some Nomad parts were sent to be manufactured outside FRC, thus making difficult to track the total time to fix the defects. The same decision was applied to the time spent waiting for parts to be ordered and received - This information was not recorded since the time could not be controlled internally.

Impact Personnel - It was not clear if the impact on personnel attribute should include the people required to fix the defect or whether it should also include the people who were not able to work because of that defect. The decision was made to consider only people who needed to fix the defect because determining the number of people who were influenced by the defect was very complex and involved people who were not necessarily working on the motion system (e.g., software developer for navigation).

The point here is that maintaining consistency along all development phases and making these policies clear for developers is essential for effective RODC benefit.

4.1.2 Data Collection

During the RODC data collection several lessons were learned. Without commitment to classify defects developers did not report every defect found. This problem was especially noticeable in the *Field Performance Test* development phase. Supporting material for data collection was sent to the developers, but no defects were collected during the field tests. The field performance test defects reported in this research were collected through interviews weeks after the fact. Developers believe that hundreds of defects happened during the field test but they had priorities other than collecting defects. This is unfortunate because this data could have been used to improve the development process of mobile robot motion systems.

During the process of collecting data an early start is recommended. Design phase defects are critical and should be recorded. The use of paper-based tables and a tape recorder was very effective for collecting defects in the initial phase of this work.

Less critical and quickly repaired defects are not easily remembered by developers. If the collection scheme does not allow for fast classification, then it is unlikely that developers will classify such defects.

When developers are under pressure they don't want to talk about defects (especially if something is not working). The classification scheme has to be efficient and fast so developers will not have the impression that they are wasting time. Special attention should be taken to keep the classification scheme working when close to deadlines. Developers tend to be under pressure before deadlines; thus, it is necessary to re-enforce the commitment to data collection.

The occurrence of repeated defects is a problem that must be addressed. For instance, 30 defects in one of the Nomad wheels meant in fact 120 defects total because all 4 wheels of Nomad had to be fixed. But they are repeated defects. They have the same defect type, cause, severity, and impact. Thus, little can be learned by classifying all was 120 defects in this case. Also the time to fix each defect is not the same since knowledge was gained whiling fixing the initial defects.

The approach taken in the RODC in this case was to count a repeated defect only once. This approach

has to be re-evaluated because of the total time required to fix multiple instances of the repeated defect.

By classifying all repeated defects, one could show the importance of details. That is, design and assembly that may have implications in many parts should get special attention because one mistake will be duplicated many times.

4.1.3 Data Analysis

The advantage of working as a developer in the Nomad project gave the author the possibility of relating defect signatures to facts that happened during the development process. The opinions Nomad developers had regarding data analysis were very important. The author believes that data analysis should always include feedback from developers. That is, the data analysis should be tuned to provide something that developers can learn from. The RODC method is easily understood by developers because no complex statistical terms are used throughout the data analysis.

Understanding that the goal of data analysis is to identify development process problems, not how to fix them, is fundamental. The work of remedying process problems is a separate activity from the RODC.

The option of using a database to be the main engine of the RODC proved to be extremely effective. Moreover, using the Access software provided the necessary computer tools for data collection forms, data storage, and data reports.

The development of tools based on the World Wide Web was appreciated by developers who intended to use the method in their based machines (UNIX, MACs, and PCs). Using its own domain name (www.in-process.org) provided easy access to the RODC site and thus improved the impression of having a more professional measurement system.

4.2 Contributions

At the outset of this work no process measurement system tailored to the development of mobile robots was available. Defect data had not been generated, collected, or analyzed during the development of mobile robots. Therefore, proposed solutions to process problems were based on guesses rather than management engineering. Moreover, many previous experiences have been lost because of the lack of a measurement system.

This research developed an easy-to-use measurement system that may transform the development of mobile robots into a controllable process. It extracts process information from mobile robot development using a proven taxonomy and shows how this information can be used to improve the process based on analysis of defect data.

Robot developers for the Nomad project were able to learn what the most costly defects were and discovered that procedures that they thought were already mastered were still causing numerous defects and consuming resources. For instance, the implication of a lack of documentation policy was shown during the RODC data analysis procedure.

The RODC is a system that can be used to test the effectiveness of development process activities such as project review and quality improvement efforts since it generates data-driven information. Also it can be used to monitor reliability (e.g., defect rates) to indicate when the robot has achieved a desired goal.

The RODC provides a measurement system that can be used simultaneously by many robot development centers to create the first multi-robot defect database available. The RODC database is capable of storing and providing information including defects, their causes, impact, severity, and component information. It also can store multimedia information such as schematics, pictures, movies, etc.

With the use of the RODC the creation of defect models is possible. In-process feedback will be enabled using these defect models. (See Section 4.3, Future Work.)

Because detailed steps regarding the experience of applying RODC to a mobile robot development

are described in this research, the process can be easily replicated. Interest has already been generated within the mobile robot community, and in particular in the National Robotics Engineering Consortium (NREC), to use the RODC as a standard tool for the development of mobile robots.

4.3 Future Work

The next step for further developing the RODC method is to provide process information during the development process of mobile robots (in-process feedback) not just after the fact. This section introduces how RODC can be used to enable in-process feedback. Also later in this section modifications and improvements to the RODC method will be proposed. This section is divided into two parts:

- In-Process Feedback
- Proposed Improvements and Modifications

4.3.1 In-process Feedback

In-process feedback will be introduced in this section by the use of signature analysis.

Signature Analysis

Signature analysis consists of using defect signatures to point to problems in the development of robots and to track defect goals. That is, it shows how defect signatures can be used to identify possible problems in mobile robot development.

Ideally, information stored from previous projects can be used to minimize time and expense and improve quality by allowing the development team to learn from previous experiences. This can be accomplished after the fact: by understanding development mistakes, correction of these mistakes is enabled for the next generation of robots; similarly, corrections may be accomplished in-process (during the development of the robot). In-process feedback can enable changes to be made during the development process to correct problems.

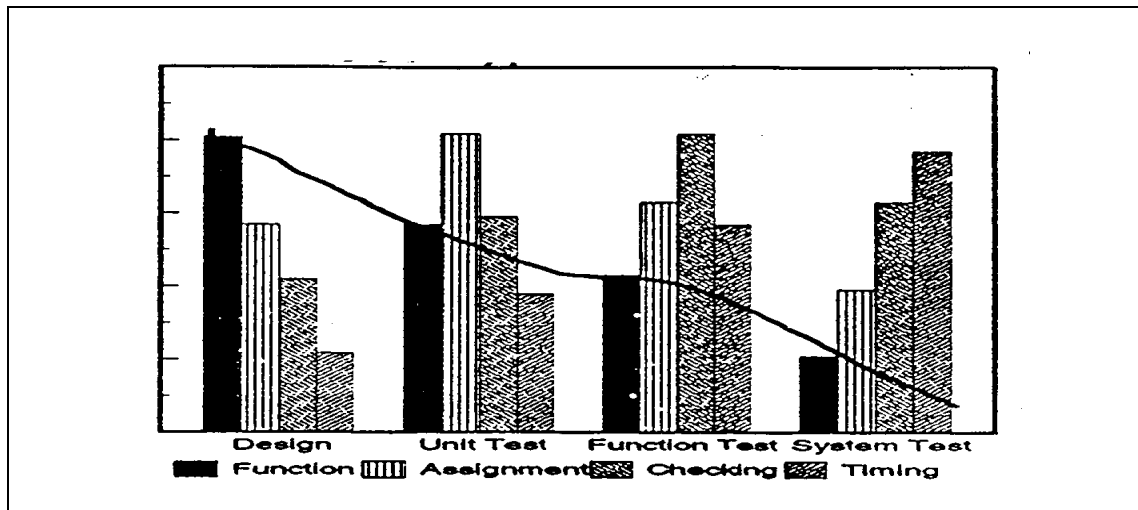
Signatures can be derived from graphic defect data. The analysis of these signatures provides information feedback. Two typical uses of signatures are:

- Comparing Patterns
 - Fitting Defect Rate (using a model)
-

Comparing Patterns

A standard signature and the defect signature of a project are compared in order to identify development problems. For example, if the project signature does not follow the standard signature pattern, then this discrepancy indicates that some problem might exist in the project development process. Defect signatures are derived from the distribution of defects through the development process. Figure 4.1 shows an example of a signature of *Defect Type - Function*. According to [Halliday, 1993] functional problems (*Defect Type - Function*) should decrease as the process proceeds. In application, if a defect signature for a *Defect Type - Function* rises in later stages of the project, it is likely that the project will produce a low quality product (i.e., it is unlikely that this project will deliver the desired functionality). Such a signature would indicate the need for a corrective action.

FIGURE 4.1 Example of a Standard *Defect Type - Function* Signature [Halliday, 1993]



Here the standard signature pattern is assumed to be reliable. That is, the standard signatures were derived from proven logic signatures or from previous defect signatures containing data from several development projects. Note that in this research what Halliday refers to as defects *Functions* are in fact referred to in this work as defects *Performance*.

At the time of this research, no defect databases were available from which defect signatures could be extracted. As a result, logic signatures were used as standard signatures to be compared to Nomad

defect signatures.

The following figures show an example of plots where defects signatures extracted from Nomad's defect data were analyzed.

FIGURE 4.2 *Defect Type - Performance Signature*

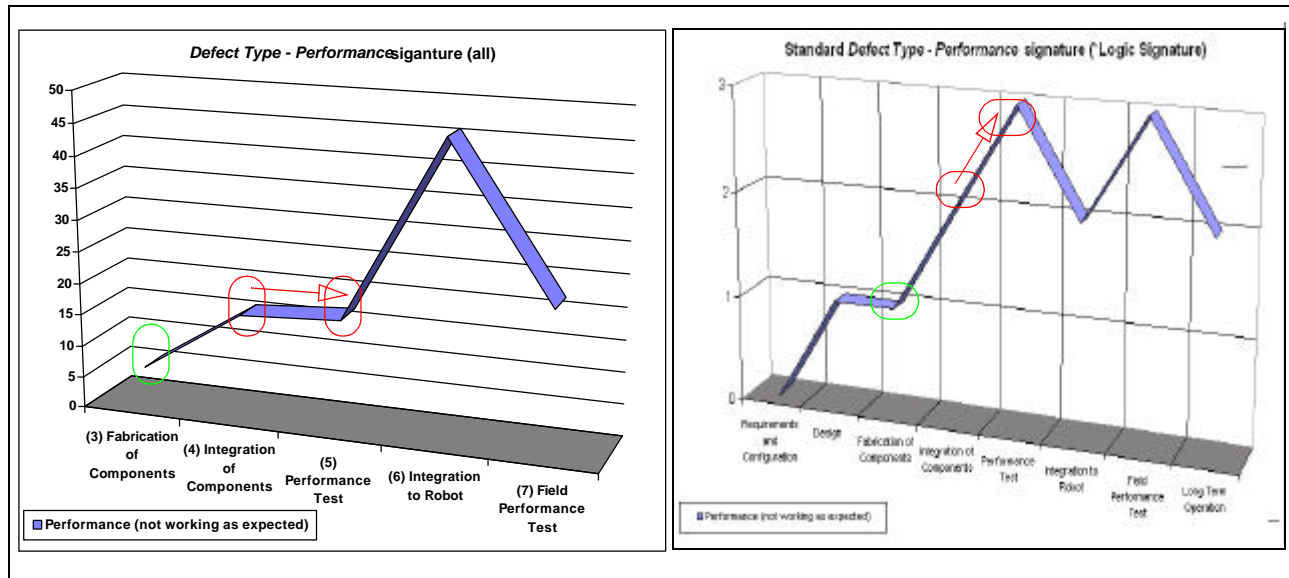


Figure 4.2 shows an example of *Defect Type - Performance* plots from which it is possible to compare signatures. (Note: the circles are used to identify corresponding development phases.)

If members of the Nomad development team tracked the *Defect Type - Performance* signature, they would have been able to observe that the signature did not follow the expected standard performance defect signature between *Integration of Components* and *Performance Test* development phases. That is, the signature is almost horizontal in these development phases for the Nomad signature but the standard (logic) signature has approximately 45 degrees angle. Therefore, in-process feedback would be possible.

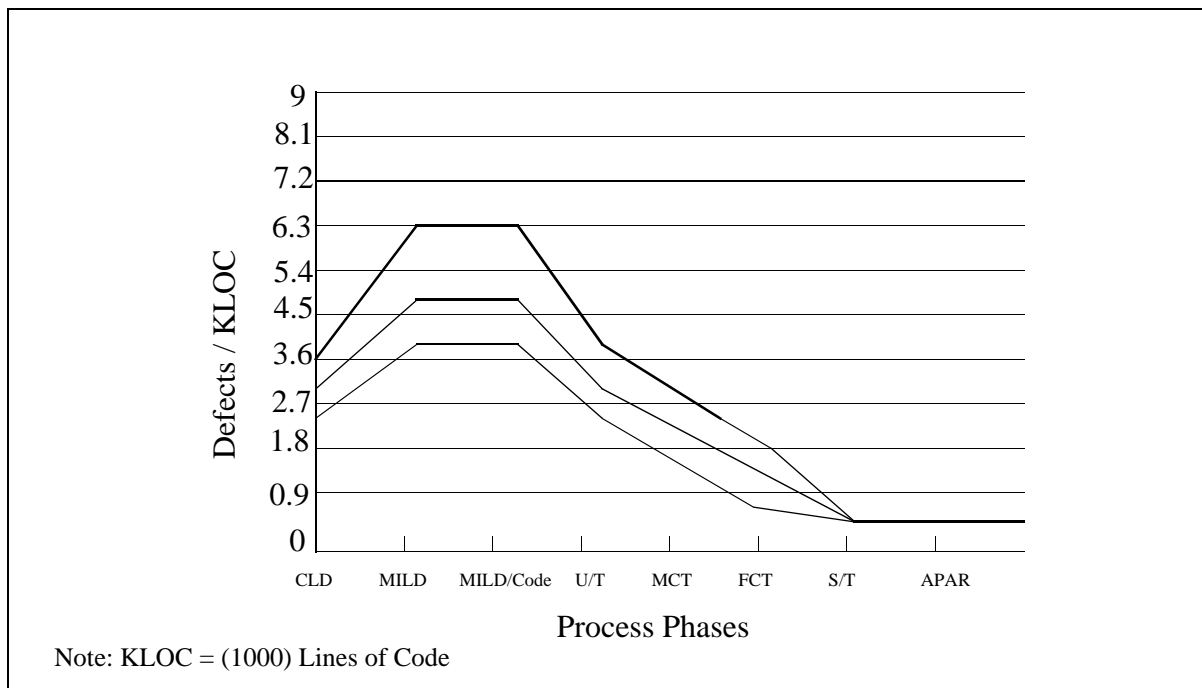
Fitting Defect Rates

In the fitting defects rates procedure, a defect signature is plotted against a defect rate model containing upper and lower limits. If the project signature exceeds a limit, an indication of process problems

exists (i.e., an investigation should be performed on the problematic area). Figure 4.3 shows an example of a defect rate model.

This research can be used as a starting point for building mobile robot defect rate models. As more defects are collected from many different projects, more defect rate signatures will be available

FIGURE 4.3 Example of Defect Rate Model [Bhandari, 1993]



Unfortunately at the time of this study no defect rate models for mobile robots were available. Therefore, no work on fitting defect rates was possible. This problem shows the opportunity to use this research as a starting point for building defect rate models.

4.3.2 Proposed Improvements and Modifications

During the development of the RODC system several modifications were made to the measurement system. But as expected, there is a need for further improvements and modifications. This section will describe these items.

Taxonomy

As explained in previous chapters the RODC taxonomy was experimentally modified. But further modifications need to be addressed:

- If RODC is used with other subsystems (besides motion systems) then the taxonomy will need to be re-designed to include relevant attributes. Following the steps presented in this research, the re-design of the taxonomy should be possible.
 - Attributes related to design can be refined. For instance, saying that a defect was caused by a design flaw is not detailed enough so developers can learn from previous mistakes. These attributes should be refined to address this problem. Similarly attributes related to the assembly process should be refined (e.g., What exactly went wrong during the assembly?)
 - A more detailed taxonomy should be developed to address defect types such as *Mechanical Structure*. It is likely that this defect type can be broken into more detailed defects related to structural problems.
 - The current set of attributes should be checked to see if they can be used in the analysis or to see if they can provide information relevant according to developers. For instance are the attributes *Not Known by the Developer* or *Considered During Design* useful? If not, they should be removed. This will save classification time and free space on the classification user-interface for new attributes.
 - The possibility of expanding multimedia tables such as the image table (*tblImages*) should be considered. They can be expanded to include voice or video in order to capture comments about defects. Instead of typing comments, developers can record a voice message, making the data collection process faster (assuming that multimedia-capable computers will be available for data collection).
 - Redundancy between attributes should be removed. For example *Triggers* and *Development Phases*. Some of the activities or triggers seems not to be orthogonal to development phases.
 - Complex attributes should be revised and separated into two or more attributes if necessary [Santhanam, 1997]. For instance, the activity that caused the defect to surface and the circumstances or conditions that revealed the defect can be separated from the *Trigger* attribute. The following example shows the separation of the RODC *Trigger*
-

attribute into two more detailed attributes. This example illustrates how an attribute can be refined to detail activities and triggers.

Activity: The activity that caused the defect to surface.

Inspection/Review of requirements and configuration

Inspection/Review of design

Fabrication of components

Integration of components

Performance test

Integration to robot

Field performance test

Stress test

Long term operation

Unclassified

Trigger: The circumstances or the conditions that revealed the defect.

Requirements conformance

Configuration conformance

Design conformance

Compatibility check

Documentation check

Basic test for component functionality under nominal unchallenging conditions

Extreme temperatures

Electrical noisy

Esthetic appeal

Unclassified

Data Analysis

The main future improvement in the data analysis is the implementation of the in-process feedback as explained in Section 4.3.1. Other improvements include:

- The RODC queries and reports can be modified according to developers feedback. That is, the extraction of information should constantly be evaluated in terms of relevance to developers.
 - The concept of costly defects (i.e., narrow the number of defects analyzed) can be used for design, fabrication, and assembly-related activities.
 - The standard used by commercial software packages used for quality improvement and control should be verified and then a filter should be generated that will allow the use of such packages with RODC. For instance, using Relx for reliability analysis.
-

Internet-Based Tools

The tools based on the Internet World Wide Web (WWW) were not tested to the extent that they are ready for a production environment. The goal in this research is to implement a working prototype.

This section proposes some improvements.

- In the present version of RODC, WWW pages use Active X scripts. This is not a problem if users have the right plug-ins in their WWW browsers or if they are using Internet Explorer browser. But if Active X scripts become an issue, then Java scripts should be used instead. Hopefully new tools will be available soon to translate from one type of script to the other.
 - The main upgrade that should be considered is in the database engine. The software database used (Microsoft Access) is able to support approximately 2000 hits per day. That is, approximately 2000 database accesses to the database per day are supported. For a greater number of hits, the software should be upgraded to a more capable engine such as Microsoft SQL Server, Oracle, or Informix. The advantage of using the SQL Server from Microsoft is the existence of translators that will transform Access databases into SQL Server databases (with some restrictions).
 - The RODC user interface can be improved by using some “intelligence” in the forms. For instance, more attributes (or fields) can be shown to the user if the defect is a costly defect. That is, for non-critical defects, the attributes will be limited and therefore may incline the user to enter more defects.
 - Grouping options by logical activities can help in classifying defects. For instance, if triggers are shown by logical groups (e.g., triggers that are design-related activities) classification will be simplified.
 - Help files were not implemented in the RODC. As in most software available now, help files containing examples should be developed for the RODC to improve training time and reduce mistakes.
-

Appendixes

Appendix A - FMEA Tables

FIGURE A.1 FMEA Table 1-4 [Borgovini, 1993]

SYSTEM <u>Security System</u>											DATE <u>3/31/92</u>
ASSEMBLY NAME <u>5VDC Regulator</u>											SHEET <u>1</u> OF <u>4</u>
REFERENCE DRAWING <u>A123</u>											COMPILED BY <u>SLP</u>
MISSION <u>Intrusion Detection</u>											APPROVED BY <u>RJB</u>
I.D. Number	Item/Functional Identification (Nomenclature)	Function	Failure Modes and Causes	Mission Phase/Operational Mode	Failure Effects			Failure Detection Method	Compensating Provisions	Severity Class	Remarks
					Local Effects	Next Higher Level	End Effects				
001	CR3 Rectifier Diode	Half-Wave Rectifier	Short	Intrusion Detection	Loss of rectification of Vin	Loss of fixed 5VDC output	Loss of Alarm	None	None	I	
002			Open	Intrusion Detection	Loss of current to series regulator	Loss of output from 5VDC Regulator	Loss of Alarm	None	None	I	
003			Parameter Change	Intrusion Detection	Slight change in rectified voltage level	No change in output voltage	No effect	None	None	IV	
004	R1 Resistor Fixed Film 100 ohms	Current limit	Open	Intrusion Detection	Loss of current to series regulator	Loss of output from 5VDC Regulator	Loss of Alarm	None	None	I	
005			Parameter Change	Intrusion Detection	Slight change in input voltage level to Q1	No change in output voltage	No effect	None	None	IV	
006			Short	Intrusion Detection	Loss of current limiting protection	Possible overstress of circuit	Degraded operation	None	None	III	
007	C11 Capacitor Tantakum Elec 47µF	Filter	Short	Intrusion Detection	Loss of current supply to Q1	No output from 5VDC Regulator	Loss of Alarm	None	None	I	High current draw through CR3, R1, possible damage
008			Open	Intrusion Detection	Loss of filter for series regulator input	Possible instability in 5VDC output voltage	Degraded operation	None	None	III	

FIGURE A.2 FMEA Table 2-4 [Borgovini, 1993].

SYSTEM <u>Security System</u>											DATE <u>3/31/92</u>
ASSEMBLY NAME <u>5VDC Regulator</u>											SHEET <u>2</u> OF <u>4</u>
REFERENCE DRAWING <u>A123</u>											COMPILED BY <u>SLP</u>
MISSION <u>Intrusion Detection</u>											APPROVED BY <u>RJB</u>
I.D. Number	Item/Functional Identification (Nomenclature)	Function	Failure Modes and Causes	Mission Phase/Operational Mode	Failure Effects			Failure Detection Method	Compensating Provisions	Severity Class	Remarks
					Local Effects	Next Higher Level	End Effects				
009	C11 (continued)	Filter	Change in Value	Intrusion Detection	Slight change in filter performance	No change in output voltage	No effect	None	None	IV	
010	R16 Resistor Fixed Film 10,000 ohms	Base current source for Q1	Open	Intrusion Detection	Loss of bias current to Q1	No output from 5VDC Regulator	Loss of Alarm	None	None	I	
011			Parameter Change	Intrusion Detection	Slight variation in current through CR10	Possible slight drift in output voltage level	Degraded operation	None	None	III	
012			Short	Intrusion Detection	Excessive current through CR10	Q1 biased at 30VDC-output to 30VDC	Loss of alarm	None	None	I	Damage to CR10
013	C9 Capacitor Ceramic .01µF	Filter	Short	Intrusion Detection	Loss of bias current to Q1	No output from 5VDC Regulator	Loss of Alarm	None	None	I	
014			Change in Value	Intrusion Detection	Slight change in filter performance	No change in output voltage	No effect	None	None	IV	
015			Open	Intrusion Detection	Loss of filter for biasing circuit	Possible ripple in 5VDC output voltage	Degraded operation	None	None	III	
016	R41 Resistor Fixed Film 510 ohms	Current limit for series regulator	Open	Intrusion Detection	Loss of current to series regulator	No output from 5VDC Regulator	Loss of Alarm	None	None	I	

FIGURE A.3 FMEA Table 3-4 [Borgovini, 1993].

SYSTEM <u>Security System</u> ASSEMBLY NAME <u>5VDC Regulator</u> REFERENCE DRAWING <u>A123</u> MISSION <u>Intrusion Detection</u>											DATE <u>3/31/92</u> SHEET <u>3</u> OF <u>4</u> COMPILED BY <u>SLP</u> APPROVED BY <u>RJB</u>	
I.D. Number	Item/Functional Identification (Nomenclature)	Function	Failure Modes and Causes	Mission Phase/Operational Mode	Failure Effects			Failure Detection Method	Compensating Provisions	Severity Class	Remarks	
					Local Effects	Next Higher Level	End Effects					
017	R41 (continued)	Current limiting for series regulator	Parameter Change	Intrusion Detection	Slight change in input current level to Q1	No effect	No effect	None	None	IV		
018			Short	Intrusion Detection	Loss of current limiting protection of Q1	Possible overstress of Q1	Degraded operation	None	None	III		
019	CR10 Zener Diode Voltage Regulator		Open	Intrusion Detection	Loss of regulated bias voltage to Q1	Output will rise to 30VDC. Possible damage to system components	Degraded operation	None	None	III		
020		Provides 5.6VDC bias voltage for Q1	Parameter Change	Intrusion Detection	Slight variation in bias voltage to Q1	Drift in output voltage level	Degraded operation	None	None	III		
021			Short	Intrusion Detection	Loss of bias voltage to Q1	Loss of output	Loss of alarm	None	None	I		
022	Q1 NPN Transistor 2N6428		Short	Intrusion Detection	Loss of regulation	Output will rise to 30VDC. Probable damage to system components	Degraded operation	None	None	III		
023		Provides output current at a regulated 5VDC	Open	Intrusion Detection	Loss of current to output	Loss of output	Loss of alarm	None	None	IV		

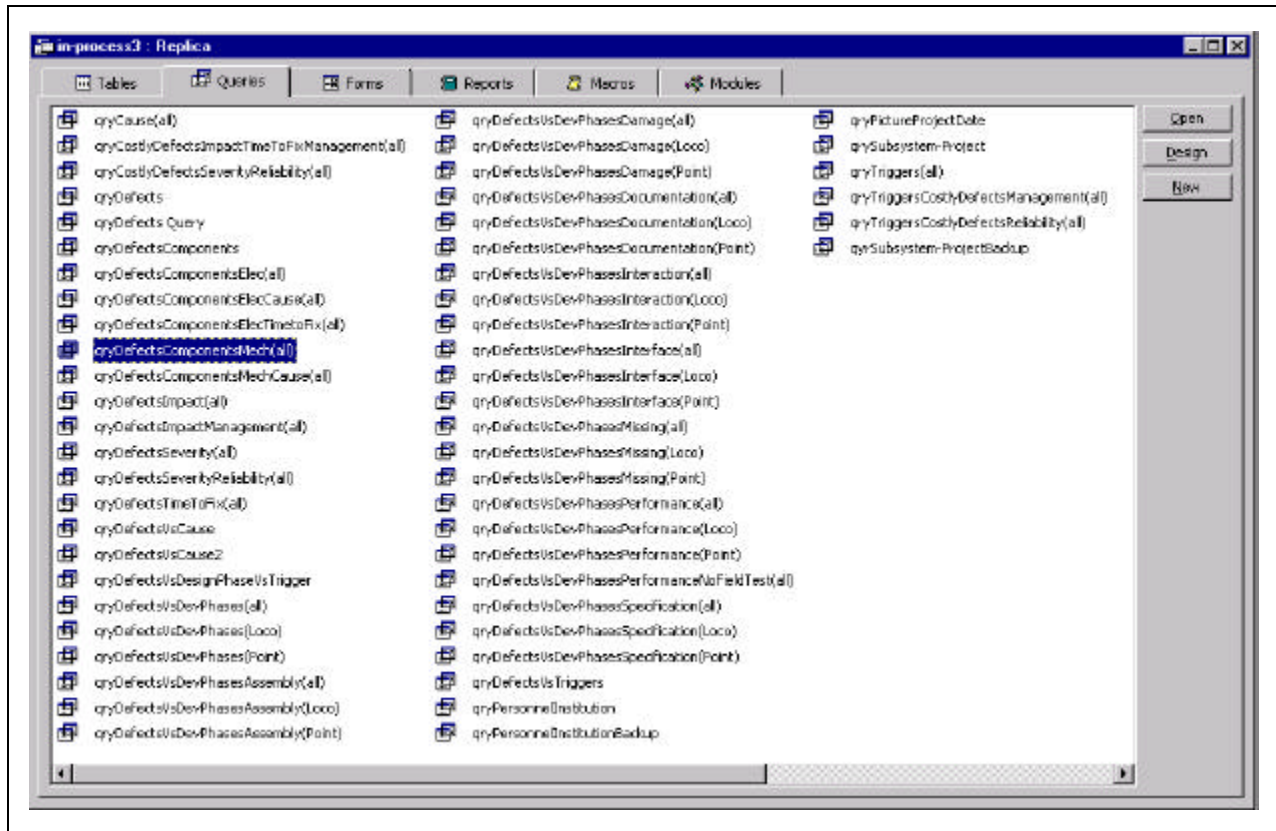
FIGURE A.4 FMEA Table 4-4 [Borgovini, 1993].

SYSTEM <u>Security System</u> ASSEMBLY NAME <u>5VDC Regulator</u> REFERENCE DRAWING <u>A123</u> MISSION <u>Intrusion Detection</u>											DATE <u>3/31/92</u> SHEET <u>4</u> OF <u>4</u> COMPILED BY <u>SLP</u> APPROVED BY <u>RJB</u>	
I.D. Number	Item/Functional Identification (Nomenclature)	Function	Failure Modes and Causes	Mission Phase/Operational Mode	Failure Effects			Failure Detection Method	Compensating Provisions	Severity Class	Remarks	
					Local Effects	Next Higher Level	End Effects					
024	C10 Capacitor Ceramic .01µF	Filter	Short	Intrusion Detection	Q1 output shorted to ground	Loss of output from 5VDC Regulator	Loss of alarm	None	None	I	Over current damage to Q1	
025			Change in Value	Intrusion Detection	Slight change in filter performance	No effect	No effect	None	None	IV		
026			Open	Intrusion Detection	Loss of high frequency filter on output	Possible high freq. noise in 5VDC output voltage	Degraded operation	None	None	III		
027	C15 Capacitor Tantalum 3.3µF	Filter	Short	Intrusion Detection	Q1 output shorted to ground	Loss of output from 5VDC Regulator	Loss of alarm	None	None	I		
028			Open	Intrusion Detection	Loss of ripple filter on output	Possible ripple in 5VDC output voltage	Degraded operation	None	None	III		
029			Change in Value	Intrusion Detection	Slight change in filter performance	No effect	No effect	None	None	IV		

Appendix B - RODC Queries

Figure A.5 shows the RODC Access database queries.

FIGURE A.5 RODC Queries



qryCause(all) - Extracts the cause for all defects from the RODC database.

qryCostlyDefectsImpactTimeToFixManagement(all) - Extracts the costly defects for management using the attribute *Time to Fix* as the impact for all defects from the RODC database.

qryCostlyDefectsSeverityReliability(all) - Extracts the costly defects for reliability using the attribute *Severity* for all defects from the RODC database.

qryDefects - Extract all defects from the RODC database.

qryDefectsComponents - Extracts all the defects related to components from the RODC database.

qryDefectsComponentsElec(all) - Extracts all defects related to electrical components from the RODC database.

qryDefectsComponentsElecCause(all) - Extracts the cause for all defects related to electrical components from the RODC database.

qryDefectsComponentsElecTimetoFix(all) - Extracts the time to fix for all defects related to electrical components from the RODC database.

qryDefectsComponentsMech(all) - Extracts all defects related to mechanical components from the RODC database.

qryDefectsComponentsMechCause(all) - Extracts the cause for all defects related to mechanical components from the RODC database.

qryDefectsImpact(all) - Extracts the impact for all defects from the RODC database.

qryDefectsImpactManagement(all) - Extracts the impact for management for all defects from the RODC database.

qryDefectsSeverity(all) - Extracts the severity for all defects from the RODC database.

qryDefectsSeverityReliability(all) - Extracts the severity for reliability for all defects from the RODC database.

qryDefectsTimeToFix(all) - Extracts the time to fix for all defects from the RODC database.

qryDefectsVsCause - Extracts the defect type and its cause for all defects from the RODC database.

qryDefectsVsDesignPhaseVsTrigger - Extracts the defect type, the development phase, and its trigger for all defects from the RODC database.

qryDefectsVsDevPhases(all) - Extracts the defect type and development phase for all defects from the RODC database.

qryDefectsVsDevPhases(Loco) - Extracts the defect type and development phase for defects related to the locomotion motion system for all defects from the RODC database.

qryDefectsVsDevPhases(Point) - Extracts the defect type and development phase for defects related to the pointing motion system for all defects from the RODC database.

qryDefectsVsDevPhasesAssembly(all) - Extracts the defect type *Assembly* and development phase for all defects from the RODC database.

qryDefectsVsDevPhasesAssembly(Loco) - Extracts the defect type *Assembly* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesAssembly(Point) - Extracts the defect type *Assembly* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsDevPhasesDamage(all) - Extracts the defect type *Damage* and development phase for all defects from the RODC database.

qryDefectsVsDevPhasesDamage(Loco) - Extracts the defect type *Damage* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesDamage(Point) - Extracts the defect type *Damage* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsDevPhasesDocumentation(all) - Extracts the defect type *Documentation* and development phase for all defects from the RODC database.

qryDefectsVsDevPhasesDocumentation(Loco) - Extracts the defect type *Documentation* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesDocumentation(Point) - Extracts the defect type *Documentation* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsDevPhasesInteraction(all) - Extracts the defect type *Interaction* and development

phase for all defects from the RODC database.

qryDefectsVsDevPhasesInteraction(Loco) - Extracts the defect type *Interaction* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesInteraction(Point) - Extracts the defect type *Interaction* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsDevPhasesInterface(all) - Extracts the defect type *Interface* and development phase for all defects from the RODC database.

qryDefectsVsDevPhasesInterface(Loco) - Extracts the defect type *Interface* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesInterface(Point) - Extracts the defect type *Interface* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsDevPhasesMissing(all) - Extracts the defect type *Missing Component* and development phase for all defects from the RODC database.

qryDefectsVsDevPhasesMissing(Loco) - Extracts the defect type *Missing Component* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesMissing(Point) - Extracts the defect type *Missing Component* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsDevPhasesPerformance(all) - Extracts the defect type *Performance* and development phase for all defects from the RODC database.

qryDefectsVsDevPhasesPerformance(Loco) - Extracts the defect type *Performance* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesPerformance(Point) - Extracts the defect type *Performance* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsDevPhasesSpecification(all) - Extracts the defect type *Specification* and development phase for all defects from the RODC database.

qryDefectsVsDevPhasesSpecification(Loco) - Extracts the defect type *Specification* and development phase for all defects related to the locomotion motion system from the RODC database.

qryDefectsVsDevPhasesSpecification(Point) - Extracts the defect type *Specification* and development phase for all defects related to the pointing motion system from the RODC database.

qryDefectsVsTriggers - Extracts the defect type and its trigger for all defects from the RODC database.

qryPersonnelInstitution - Extracts personnel and the institution he or she belongs to from the RODC database.

qryPictureProjectDate - Extracts pictures and the project that it belongs to from the RODC database.

qrySubsystem-Project - Extracts subsystems and the project that it belong from the RODC database.

qryTriggers(all) - Extracts the trigger for all defects from the RODC database.

qryTriggersCostlyDefectsManagement(all) - Extracts the triggers for the costly defects for management for all defects from the RODC database.

qryTriggersCostlyDefectsReliability(all) - Extracts the triggers for the costly defects for reliability for all defects from the RODC database.

Appendix C - RODC Survey

Dear robot developer:

I am working with Dr. John Bares on mobile robot reliability. He suggested that I contact you because of your expertise in the mobile robot community.

I would appreciate it if you could please fill in and return this survey at your earliest convenience. The results of this survey will be used to help implement a measurement system for mobile robot development (see <http://in-process.org>). It should take roughly 9 minutes of your time to complete this survey.

The survey is divided into two parts: *logic defects* and *process information*.

* Defects are any change in a project artifact (schematics, documentation, subsystems, etc.).

Logic Defects

Logic defects refer to the likelihood that certain defects will happen at certain times. The idea here is to assign probable quantities of defects to each mobile robot development phase. For instance, the defect type Assembly Process (improper) will not appear during the development phase Requirements and Configuration. Therefore, the matrix cell representing the intersection between these two fields will receive the lowest value possible for probable quantity (in our case 0). As another example, the defect type Interfaces (incompatible) will likely have a significant number of defects during the development phase Integration of Components. This is because during the development phase components will be integrated for the first time to form a subsystem. That is, chances of having interface incompatibility problems during this development phase are high.

The table scale description follows:

- 0 - No defects should occur.
- 1 - A small number of defects should occur.
- 2 - A moderate number of defects should occur.
- 3 - A significant number of defects should occur.

Please write the numbers you think should be in each cell of the table located in the next page. Use your own experience in developing robots to guide your decision.

Process Information

Process information here means what needs to be known during the development of mobile robots. For instance, what kind of information would improve the process of developing a mobile robot (e.g., causes for defects, defect triggers, etc.)?

Please write three items in the list that is located below the Logic Defect table (see next page).

After you have completed the survey, please send a fax including the next page with your answers to (412)268-5895.

Thank you very much for your cooperation.

-JACK SILBERMAN

Developer name: _____

1 - Logic Defect

TABLE A.1 Probable Logic Defects - *Please write your number in each cell*

Dev. Phase Vs. Defect Type	Requirements and Configuration	Design	Fabrication of Components	Integration of Components	Performance Test	Integration to Robot	Field Performance Test	Long Term Operation
Assembly Process (improper)								
Damage (need fix)								
Documentation (schematics, instructions)								
Esthetic (appearance)								
Interaction (interferences)								
Interfaces (incompatible)								
Missing Component								
Performance (not working as expected)								
Specification (incorrect or change)								

2 - Process Information (*please add the three most important items in your opinion*)

-
-
-

Please send a Fax with this page to Jack Silberman at (412)268-5895.

Appendix D - Logic Signatures Plots

FIGURE A.6 Assembly Logic Signature

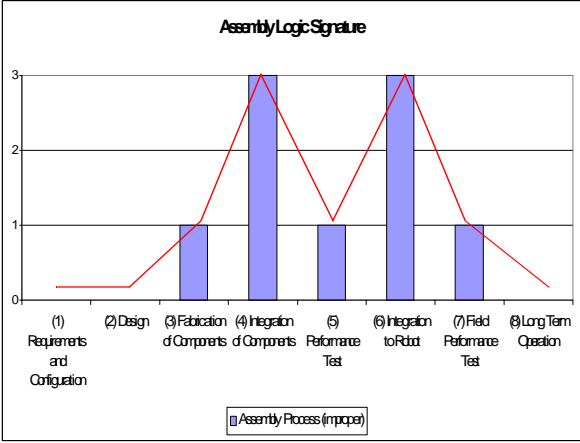


FIGURE A.7 Damage Logic Signature

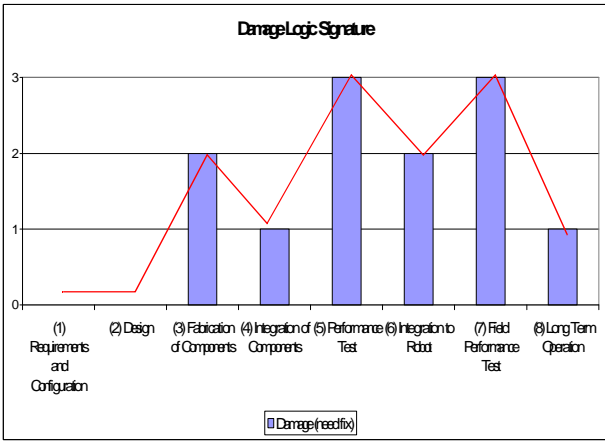


FIGURE A.8 Documentation Logic Signature

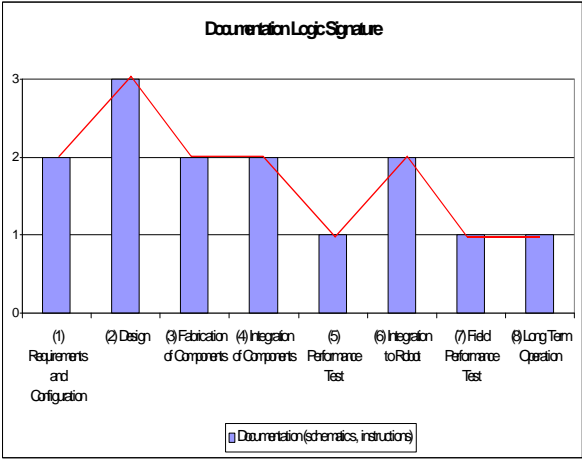


FIGURE A.9 Esthetic Logic Signature

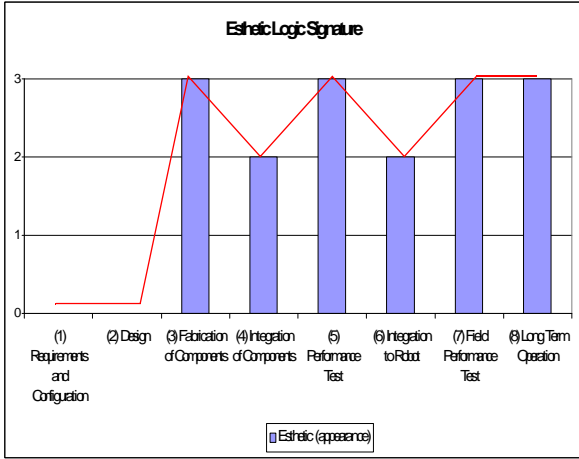


FIGURE A.10 Interaction Logic Signature

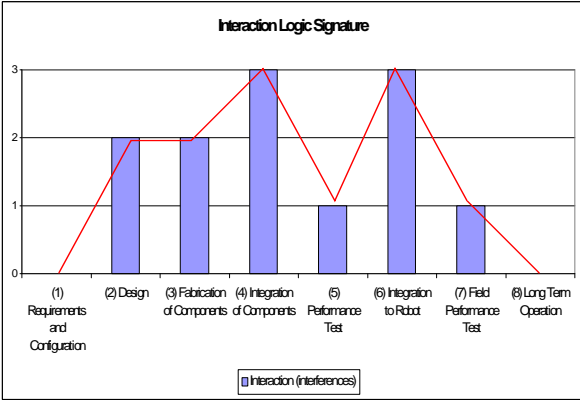


FIGURE A.11 Interfaces Logic Signature

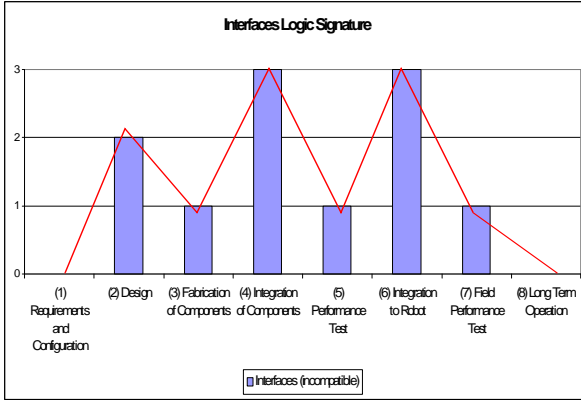


FIGURE A.12 Missing Logic Signature

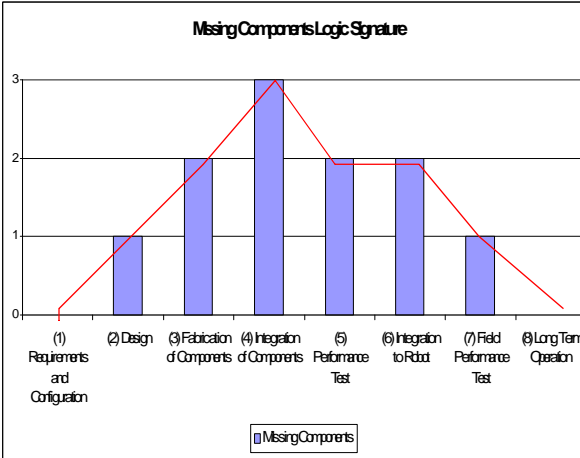
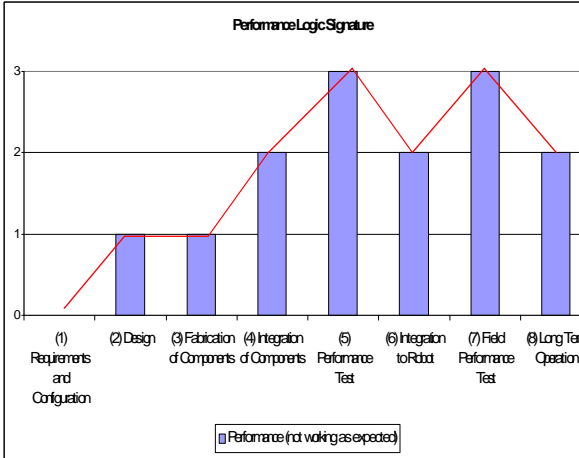


FIGURE A.13 Performance Logic Signature



Appendix E - Internet World Wide Web Based Tools

Tools based on server-client network and the Internet (e.g., WWW browsers) can enable quick response time, interactivity, and ease of use in gathering process data and feedback. The WWW can also enable the use of a system by a large number of users, therefore enabling a large amount of data to be collected.

Software and hardware are not necessarily independent in applications based on server-client technology and the Internet World Wide Web. This is especially true on the server side because not all software is available to all computer platforms. On the other hand, on the client side if applications are developed based on common standards such as the one used on the WWW (e.g., HTML and JAVA) then these applications are virtually independent of the hardware because different computer platforms running different operating systems can present the same user-interface to users.

The approach taken in this research is to develop tools based on the WWW so the effort for developing such tools can be reduced. RODC tools can be used by local users (in the Field Robotics Center) as well as users located remotely (e.g., anywhere on the Internet). That is, the goal is to develop one set of tools capable to be used by many different users.

This chapter is divided in the following sections:

- RODC Hardware
 - RODC Software
 - RODC on the WWW
-

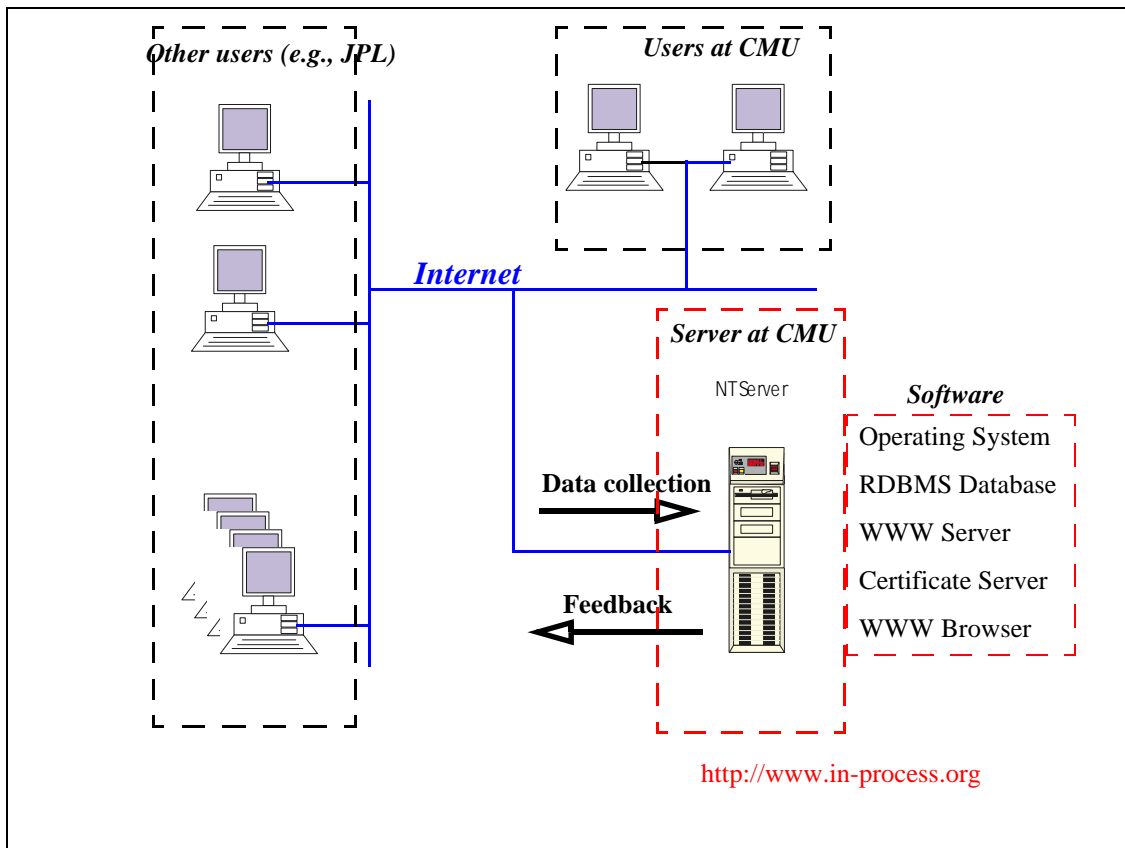
E.1 RODC Hardware

It is beyond the scope of this research to give detailed information on how to install the hardware or the software described here. Such information can be found in the manufacturer manuals.

The hardware requirements for hosting the RODC servers are reasonably small. The hardware has to be able to connect to the Internet serving files to RODC users. The term serving files here means hosting servers capable of working on the Internet.

Figure A.14 shows the RODC hardware scheme. (Note: software servers need to be located in just one hardware server.)

FIGURE A.14 RODC Hardware Scheme



The RODC hardware has to be capable of hosting the RODC database files, a WWW server, and certificate server. These servers will be explained in Section E.2 RODC Software.

Basically there were two options of hardware available: UNIX-based workstations (e.g., SUN and SGI) and Pentium-Pro (IBM PC compatible) machines.

Choosing between a UNIX machine and a PC sometime ago required compromise between speed and reliability vs. price. But today high-end PCs are fast and reliable, and they are less expensive when compared to UNIX workstations.

The decision was to use a Pentium-Pro based machine. This kept the development and server environments the same (Access runs on PCs). Besides that, the Pentium-Pro machine was faster than the available UNIX workstations.

The Pentium-Pro has enough resources to host the RODC servers. It has 200 MHz clock, 64 Mbytes RAM, 4 Gbytes hard disk, and a fast Ethernet interface (100 Mbits). The RODC does not require all these resources. The reason for these numbers is because the hardware donated by Intel came with this configuration not because of minimum resources necessary to host the RODC servers. More details on the minimum configuration required by the RODC software will be given in Section E.2 RODC Software.

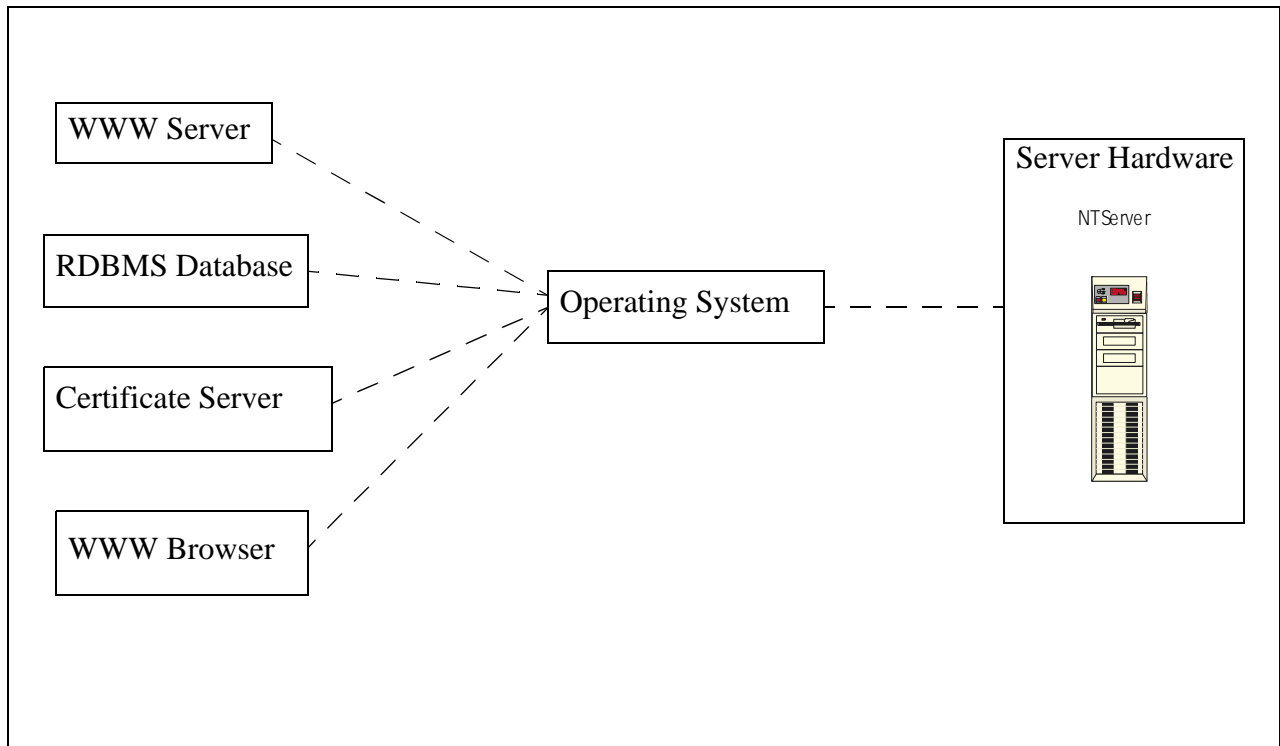
It is important to mention here that the RODC hardware is a regular desktop computer. It is not a special dedicated-server hardware that includes fault tolerance and other features. But it is capable of running the same software that runs on dedicated-server hardware. Nevertheless, by the time of the RODC development, the described configuration was considered a high-end PC.

The RODC hardware is capable of hosting the RODC servers and still leaving enough resources for future expansions.

E.2 RODC Software

The basic RODC software includes: an RDBMS database, a WWW server, a certificate server, and a WWW browser. Figure A.15 shows the RODC software scheme. (Note: all software uses the operating system to access the hardware).

FIGURE A.15 RODC Software Scheme



This section is divided into the following parts:

- Operating System
- RDBMS Database
- WWW Server
- Certificate Server
- WWW Browser

Operating System

The operating system is probably the only part of the software used by the RODC that is the most dependent on hardware. Few options were available for use with the Pentium-Pro hardware. The two

most reliable operating systems considered were the Windows NT 4.0 Server and Linux Redhat4.3. The decision was to use Windows NT 4.0 Server because of the availability of commercial WWW and certificate servers. The second reason for this choice was because the RDBMS used in the RODC (Access) runs on a Windows operating system. This fact allows some development on the server machine too.

RDBMS Database

As explained in a previous chapter, the RDBMS database used in the RODC Microsoft Access 97. Having Access installed in the RODC hardware server is not required since WWW servers support Open Database Connectivity (ODBC) protocol. An ODBC connection for Microsoft Access 97 requires the data source name (name of the file) and a user id that has the rights to open the RODC database. To allow some development on the RODC hardware, Access was also installed.

WWW Server

Two commercial WWW servers were considered for use in the RODC System: 1) Netscape Enterprise Server; and 2) Microsoft Internet Information Server (IIS). The reason for evaluating these two servers instead of all the diverse WWW servers available for Windows NT was because they were rated as being the most reliable and having better performance when compared to the other servers [Web Developer, 97].

The decision was to use Microsoft Internet Information Server (IIS) version 4.0 because it supports dynamic ASP files. Access has a feature to generate dynamic ASP files that simplify the process of making Access files available for the WWW. This process will be described later in this chapter.

Certificate Server

The Internet WWW is an excellent way to remotely use databases such as the RODC system. But because of the Internet architecture, raw data is not safe from eaves dropping. That is, at some point at the Internet malicious connections can be made to read files that are being transmitted through that point. One solution for this problem is to use Secure Sockets Layer (SSL). By using SSL, transmission between two points is encrypted. Therefore, it is much harder to be eavesdrop.

Security in the RODC system is an issue because project defect data may be confidential. If management and developers understand that project defect data is secure during connections to a WWW server then it is likely that they will be less concerned about security in using the RODC system.

The certificate server is used to create an electronic certificate that is issued to a WWW server and users. Using SSL for WWW servers and users significantly increases the security of an Internet-based system.

The process of encrypting connections using SSL is almost transparent for users because the encryption protocol and keys are automatically exchanged between the user's WWW browser (e.g., Netscape navigator) and the WWW server (i.e. RODC WWW server).

Two certificate servers were evaluated for the RODC system: 1) Netscape Certificate Server; and 2) Microsoft Certificate Server.

The decision was to use the Microsoft Certificate server because it has better integration with Windows NT 4.0 Server and it is easy to use.

WWW Browser

Any WWW browser that supports JAVA, Active X, and SSL can be used with the RODC system.

Two examples of such browsers are Netscape navigator (with Active X plug-in) and Microsoft Internet Explorer.

The choice is based on user opinion and availability of WWW browsers for their computer platform.

E.3 RODC on the WWW

The implementation of the RODC on the WWW followed these steps:

- Hardware Setup
- WWW Server Setup
- Database Setup for the WWW
- Certificate Server Setup

Hardware Setup

The hardware setup was accomplished by installing the Windows NT Server 4.0 operating system on Pentium Pro hardware. A name and IP address were issued from the computer science facilities at Carnegie Mellon University so the hardware could be connected to the Internet. The name of the machine where the RODC is running is *helena.frc.ri.cmu.edu* and the IP address is 128.2.196.36. After the setup of the hardware then the WWW server was installed.

WWW Server Setup

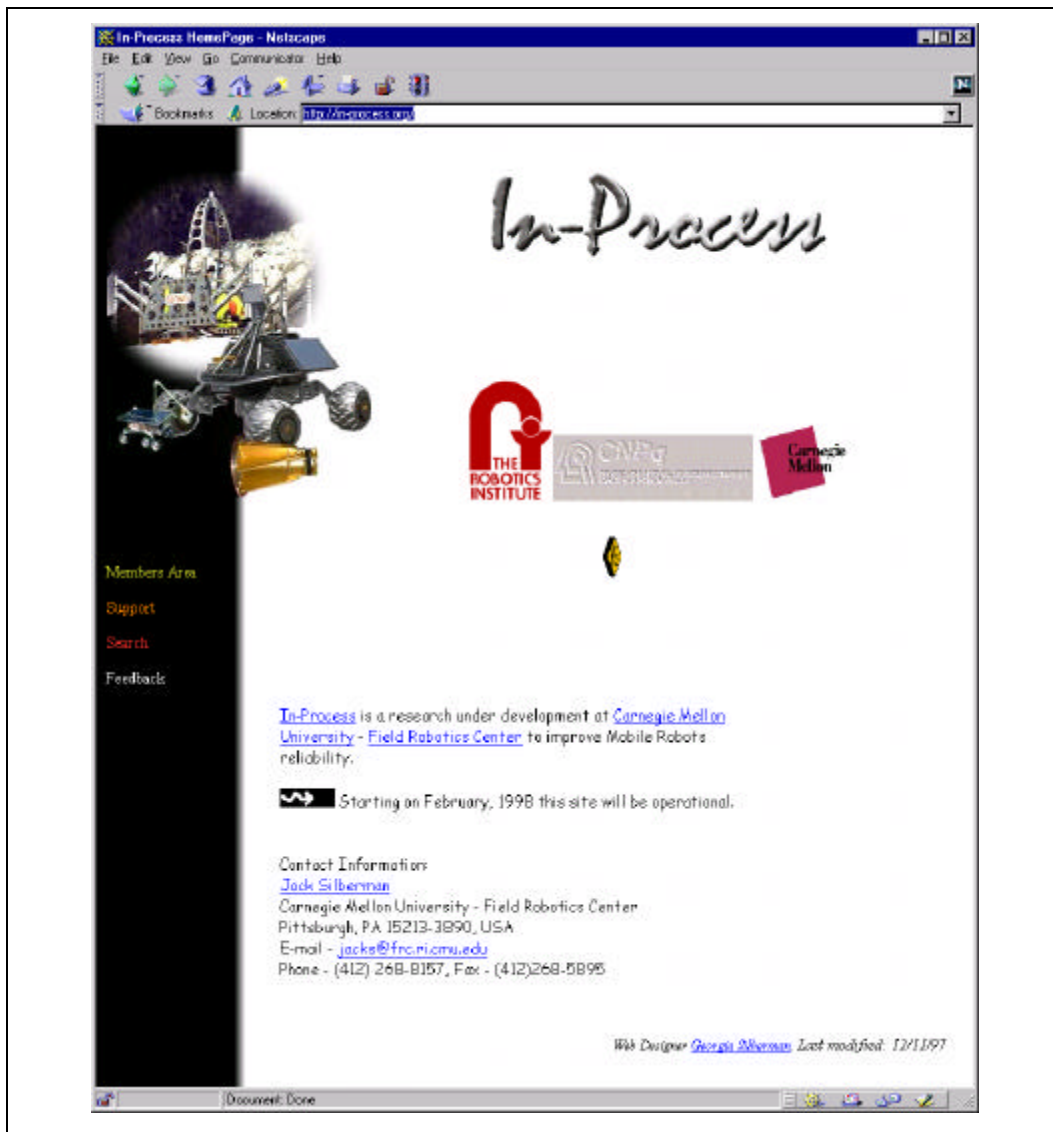
The WWW server was installed in the RODC hardware (*helena*) and a home page was created. The access to the WWW pages located in *helena* had the following pointer <http://www.helena.frc.ri.cmu.edu/>. As one can see, this is not easy to remember nor is it professional.

The author has registered an Internet Domain Name named *in-process.org* with an Internet authority so *in-process.org* points to *helena.frc.ri.cmu.edu*. The domain name is a description of a computer “location” on the Internet. Users neither have to type in the name of a machine (e.g., *helena.frc.ri.cmu*) nor their IP address (e.g., 128.2.196.36) to access the WWW pages located in the WWW server.

After this registration, pages located in the WWW server can be accessed by using the following pointers <http://in-process.org/> or <http://www.in-process.org>.

Figure A.16 shows the RODC home page.

FIGURE A.16 RODC Home page



The RODC home page has links to other WWW pages such as *Members Area*, which access is restricted to users registered to use the RODC system.

Certificate Server Setup

The certificate server was installed in helena using a procedure similar to the one used for the WWW server.

An electronic certificate has to be requested from the RODC administrator. To do so the user can send e-mail to support@in-process.org. In the case of the RODC system an electronic certificate was issued to the WWW server. Internet users can then make secure connections to the RODC WWW site.

The performance of the RODC system was not tested for multiple users accessing the database simultaneously. But according to reports, the scheme used in the RODC can handle such a situation.

References

- [Amezquita, 1996] - Alex Amezquita, Daniel P. Siewiorek, "Orthogonal Defect Classification for Electromechanical Systems", EDRC - Carnegie Mellon University, Pittsburgh, September 1996.
- [Bhandari and Halliday, 1993] - Inderpal Bhandari, Michael Halliday, "A case Study of Software Process Improvement During Development", IEEE Transactions on software engineering, Vol 19, No. 12, December 1993.
- [Bhandari, 1994] - Inderpal Bhandari, "In-process improvement through defect data interpretation", IBM systems Journal vol. 33, no1,1994.
- [Borgovini, 1993] - Robert Borgovini, Stephen Pemberton, Michael Rossi, "Failure Mode, Effects, and Critically Analysis (FMECA)", Reliability Analysis Center (RAC), Rome, NY, 1993.
- [Brown, 1996] - Christopher L.T. Brown, Scott Zimmerman, "Web Site Construction Kit for Windows NT", SamsNet Publishing, Indianapolis, Indiana, 1996.
- [Chillarege, 1992] - Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, "Orthogonal Defect Classification - A Concept for In-Process Measurements", IEE Transactions on software engineering, Vol 18, No. 11, November 1992.
- [Chillarege, 1994] - Ram Chillarege, "ODC for Process Measurement, Analysis and Control", The fourth International Conference on Software Quality, McLean, Virginia, October 1994.
- [Chillarege, 1994] - Ram Chillarege, "Orthogonal Defect Classification Data Definitions and Requirements", Continuous Availability and Quality Group, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, February 1994.
- [Chillarege, 1997] - "Orthogonal Defect Classification (ODC)", Center for Software Engineering IBM T. J. Watson Research Center, Hawthorne, New York, N.Y. 1996
- [Crosby, 1986] - Philip B Crosby, "Management and Policy", Quality Management Handbook, Hitchcock Publishing Company, Wheaton, Illinois, 1986.
- [Ellis, 1986] - Ott Ellis R., "Statistics Without Math", Quality Management Handbook, Hitchcock Publishing Company, Wheaton, Illinois, 1986.
- [Fleming, 1989] - J. Fleming, "Handbook of Relational Database Design", Addison-Wesley, Reading, Massasutest, 1989.
- [Halliday, 1993] - Michael Halliday, Inderpal Bhandari, Jarir Chaar, "Experiences in Transferring a Software Process Improvement Methodology to Production Laboratories", Proceedings 2nd International Conference on Achieving Quality in Software, Venice, Italy, October 18-20, 1993.
- [Hernandez, 1997] - Michael J. Hernandez, "Database Design for Mere Mortals", Addison-Wesley Developers Press, Reading, Massachusetts, 1997.
-

[Ireson-1, 1996] - W. Grant Ireson, Clyde F. Coombs, Jr. , Richard Y. Moss, "Handbook of Reliability Engineering and Management", 2nd ed., McGraw-Hill, New York, NY, 1996.

[Ireson, 1996] - W. Grant Ireson, "Reliability Information Collection and Analysis", chapter 10, Handbook of Reliability Engineering and Management 2nd ed., McGraw-Hill, New York, NY, 1996.

[Johnson, 1989]- Barry W. Johnson, "Design and Analysis of Fault Tolerant Digital Systems", Addison-Wesley Publishing Company Inc., New York, NY, 1989.

[Jones, 1996]- Wendell D. Jones, Mladen A. Vouk, "Field Data Analysis", Chapter 11, Handbook of Software Reliability Engineering, McGraw Hill, New York, 1996.

[Kececioglu, 1991] - J. Kececioglu, "Reliability Engineering Handbook", Vol.2, Prentice-Hall, Inc., New York, NY, 1991.

[Kroenke, 1995] - David Kroenke, "Database Processing Fundamentals, Design, and Implementation", 5th ed., Prentice Hall, Englewood Cliffs, N.J., 1995.

[Lazor, 1996] - J. D. Lazor, "Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) (Success Tree Analysis - STA)", chapter 6, Handbook of Reliability Engineering and Management 2nd ed., McGraw-Hill, New York, NY, 1996.

[Lemay, 1996] - Laura Lemay, "Teach yourself Java in 21 days", SamsNet Publishing, Indianapolis, Indiana, 1996.

[LeVitus, 1996] - D. Levitus, "WebMaster Windows, How to build your own World Wide Web Server without really trying", Academic press Inc.Chestnut Hill, MA, 1996.

[Litwin, 1997] - Paul Lotwin, Ken Getz, Mike Gilbert, "Access 97 Developer's Handbook", Sybex, San Francisco, CA, 1997.

[Lloyd, 1991] - P. Provost Lloyd, "Reliability: Management, methods, and mathematics", The American Society for Quality Control, Milwaukee, Wisconsin, seventh printing, 1991.

[MacDiarmid, 1997] - M. MacDiarmid, "Reliability Toolkit: Commercial Practices Edition", Reliability Analysis Center (RAC), Rome, NY, 1997.

[Moss II, 1996] - Richard Y. Moss II, "Design for Reliability", chapter 5, Handbook of Reliability Engineering and Management 2nd ed., McGraw-Hill, New York, NY, 1996.

[Montgomery, 1996] - Douglas C. Montgomery, "Introduction to Statistical Quality Control", John Wiley & Sons. Inc, New York, NY, 1996.

[Pahl, 1996] - G. Pahl, W. Beitz, "Engineering Design A Systematic Approach", second edition, Springer- Verlag, London, England, 1996.

[Santhanam, 1996] - PeterSanthanam, "Orthogonal Defect Classification (ODC)", Center for Soft-

ware Engineering IBM T. J. Watson Research Center, Hawthorne, New York, N.Y. 1996

[Santhanam, 1997] - Peter Santhanam, "Orthogonal Defect Classification (ODC)", Center for Software Engineering IBM T. J. Watson Research Center, Hawthorne, New York, N.Y. 1997.

[Shooman, 1968] - Martin L. Shooman, "Probabilistic Reliability: An Engineering Approach", McGraw-Hill, New York, NY, 1968.

[Siewiorek, 1982] - Daniel P. Siewiorek, "The theory and Practice of Reliable System Design", Digital Press, Bedford, Mass., 1982.

[Smailagic and Siewiorek, 1995] - Asim Smailagic, Daniel P. Siewiorek "Benchmarking An Interdisciplinary Concurrent Design Methodology for Electronic/Mechanical Systems", Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, 1995.

[Sun, 1996] - "JAVA SOFT", Sun Microsystems homepage about JAVA programing language, WWW 1996, <http://java.sun.com/aboutJava/>.

[Tittel, 1995] - R. Tittel, "Foundations of World Wide Web Programming with HTML & CGI", IDG Books, Foster City, CA, 1995.

[Web Developer, 97] - "Web Servers Benchmarking", Web Developer, Volume 3, Issue 2, Wesport, CT 1997.

[Whittaker, Bapna, 1997] - Willian "Red" Whittaker, Deepak Bapna, "Atacama Desert Trek: A Planetary Analog Field Experiment", Field Robotics Center, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA.
