12-2007

A CASE STUDY INVESTIGATING RULE BASED DESIGN IN AN INDUSTRIAL SETTING

Siva Chavali
Clemson University, schaval@clemson.edu

A CASE STUDY INVESTIGATING RULE BASED DESIGN IN AN INDUSTRIAL SETTING

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

---

by
Siva Rama Krishna Chavali
December 2007

Accepted by:
Dr. Joshua D. Summers, Committee Co-Chair
Dr. Gregory M. Mocko, Committee Co-Chair
Dr. Thomas R. Kurfess

ABSTRACT

This thesis presents a case study on the implementation of a rule based design (RBD) process for an engineer-to-order (ETO) company. The time taken for programming and challenges associated with this process are documented in order to understand the benefits and limitations of RBD. These times are obtained while developing RBD programs for grid assemblies of bottle packaging machines that are designed and manufactured by Hartness International (HI). In this project, commercially available computer-aided design (CAD) and RBD software are integrated to capture the design and manufacturing knowledge used to automate the grid design process of HI. The stages involved in RBD automation are identified as CAD modeling, knowledge acquisition, capturing parameters, RBD programming, debugging, and testing, and production deployment. The stages and associated times in RBD program development process are recorded for eighteen different grid products. Empirical models are developed to predict development times of RBD program, specifically enabling HI to estimate their return on investment. The models are demonstrated for an additional grid product where the predicted time is compared to actual RBD program time, falling within 20% of each other. This builds confidence in the accuracy of the models. Modeling guidelines for preparing CAD models are also presented to help in RBD program development. An important observation from this case study is that a majority of the time is spent capturing information about product during the knowledge acquisition stage, where the programmer's development of a RBD program is dependent upon the designer's product knowledge. Finally, refining these models to include other factors such as time for building CAD models, programmers experience with the RBD software (learning curve), and finally extending these models to other product domains are identified possible areas of future work.

DEDICATION

This thesis is dedicated to my parents, my aunt Ms Sarojini Devi and my uncle Mr. Rama Murty who helped me in getting to this stage.

ACKNOWLEDGMENTS

I would like to thank Dr. Summers and Dr. Mocko for their continuous support and the intellectual freedom they entrusted to me throughout this project without which the work would not have taken the present form. I would like to thank Dr. Kurfess for giving valuable feedback that helped me in strengthening the quality of work in the area of design.

I would like to thank Narasimhamurty Srirangam, Pavan Kumar, Sridhar Duddukuri, Abhinand Chelikani, Manoj Chinnakonda, Chiradeep Sen, Ben Caldwell, Suma Yaski, Kamesh Nara, Pavan Seemakurty, Priyanka Alluri, and my other lab members who helped me in reviewing this thesis. Your, encouragement, and friendship have been invaluable to me.

Finally yet importantly, I would like to thank Hartness International's management for giving me an opportunity to work on their grid assemblies. I would especially like to thank Mr. Mickey Dorsey and Mr. Clayton Rowley for sparing their precious time in giving me the valuable information about their grid designs. I would like to thank all other staff of grid change over parts department in Hartness for providing fun environment where I learned the fundamentals of their grid design.

TABLE OF CONTENTS

Table of Contents (Continued)

LIST OF TABLES

Table Page

LIST OF FIGURES

List of Figures (Continued)

Figure Page

CHAPTER 1

MOTIVATION AND BACKGROUND

Introduction

In today's competitive global market, the main goal of the manufacturer is to satisfy the needs of the customer. The individual customer needs and desires result in custom designs which the manufacturer provides, ideally, in short time by producing quality products at economic prices. The flexibility to adapt for custom changes necessitates in integrating the customer during the design process, which results in frequent changes in the product [42]. The method of producing custom products, known as mass customization, in exemplified in environments such as engineer to order (ETO) manufacturing organizations. Fast moving job markets have become difficult for the ETO industries to produce quality products especially given the current emphasis on shorter cycle times. One specific challenge to this approach is that ETO rely heavily on the internal corporate expertise of the product, typically found only within a few persons in the company. This suggests that the company can lose competitive advantage should the individuals with this knowledge leave the company for some reason. Rule based design (RBD) is one possible solution that helps in automating the design process as well as preserving the company's intellectual capital. The RBD process helps in reducing product development time by automating the low-value design activities like editing models, producing drawings, preparing bill of materials, and preparing quotations. Further, as the corporate knowledge is encoded in rules, the company is not as dependent on the expertise of the individual.

This research is focused on studying the RBD process, the stages, and level of effort that is involved for developing RBD programs in typical ETO companies. A case study is conducted to explore and understand these stages for identifying the parameters that are involved during the development of an RBD program. Case study research helps in exploring the "how" and "why" aspects of RBD program development [48]. It also helps in studying the previously identified low-value activities related to design.

The "how" and "why" aspects may include *how the design information should be exchanged* and why *it needs to be exchanged* or *how the design information should be stored* and *why should it be stored*. For addressing these areas, an ETO company is identified that deals with the design related activities and a case study is conducted to study the parameters that affect them.

Hartness International[1] (HI), headquartered at Greenville, South Carolina, is a family owned small-to-medium ETO enterprise that designs bottle case packaging solutions. Started in 1940, the company now employs about 450 employees, with manufacturing facilities in North America, Europe, and China and with regional offices around the world. They manufacture a wide variety of products including case packers, bottle packers, high efficiency conveyor systems, bottle filling and labeling systems, and shrink wrapping systems.

The HI "Investigation into Rule Based Design (IRBD)-2006" project is selected as the subject of case study, which is referred as "HI IRBD project" in the remainder of this thesis. This project is selected as the case study as many activities of design can be studied in various scenarios or products. The project is scoped to automate the design process of 32 different grid changeover parts, shown in Figure 1.1., using DriveWorks[2] (DW) RBD automation tool for the solid modeling package SolidWorks[3] (SW). The phrase "grid changeover parts" is interchangeably referred as "grid product" or "grid assembly" in the remainder of this thesis. Specifically, this project deals with the grid assembly, which interacts with bottles and cases in the HI product line of case packing. For every new design of a bottle and/or case, the grid assembly is replaced by a new one in order to run the new line of products through the existing machine.

[1] www. hartness. com

[2] www. driveworks. co. uk

[3] www. solidworks. com

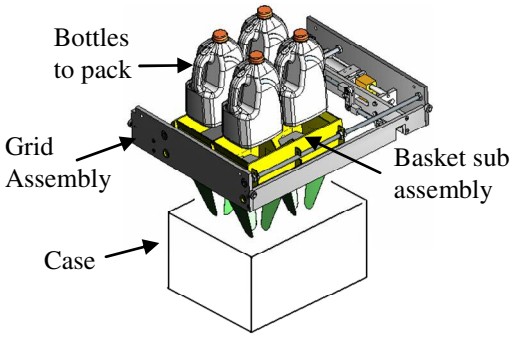


Figure 1.1: CAD model of grid assembly

A typical case packer machine with this grid assembly is shown in Figure 1.2. The packer consists of two conveyors running along the length of the machine where the top conveyor runs bottles to be packed and the bottom conveyor runs the cases that receive the bottles. This transfer of bottles from the top conveyor to the cases on the bottom conveyor is done by grid assembly as shown in Figure 1.3.

Figure 1.2: Typical HI case packing machine

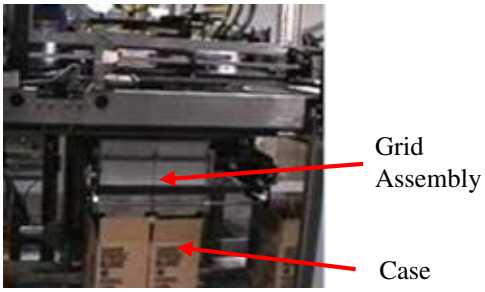


Figure 1.3: Grid assembly on the machine

The grid assembly needs to be designed for every new design of bottle or case. The HI grid designers must either develop custom solutions or modify the existing solutions to accommodate these new bottles or cases to run on the case packing machines. To reduce the effort and time required for designing new grids, HI uses standard design templates that can be modified without having to build new system. The HI grid changeover group is responsible for eighty eight products, of which thirty-two products are chosen for DW RBD automation. Either the remaining products were not chosen because they have homegrown programs for RBD automation (using SW application program interfaces and Visual Basic) or RBD automation is not required as their custom use is minimal at less than one redesign per year. Eighteen out of thirty-two products are selected for this case study and are shown in Table 1.1.

As the grid assembly plays a major role in the case packer machine, it must be changed for every new design of a bottle or case. Given that the changes in the bottles and/or cases are frequent, the aspects that the HI uses to accelerate the grid design process are commonality, modularity, and customization.

Commonality is the aspect of designing products with common functional and geometrical similarities. Designing common features aids in producing uniform designs that ultimately result in framing guidelines with consistent design rules. This can be seen in the design of components of basket sub assembly with the grid product as shown in Figure 1.4.

Modularity is the flexible arrangement of components that allow ease of assembly and disassembly. It also helps in reusing the existing designs for designing new products. Incorporation of modularity into designs assists in producing standard products in a short time. This aspect can be seen in HI designs as shown in Figure 1.4. The basket assembly is made up of modular components such as back part, front part, lane divider, and transversals.

Customization is the process of tailoring products to meet an individual's specific needs. Customized products can be developed by using the above-mentioned aspects of commonality and modularity in design. HI grid assemblies are designed for custom size pockets (L x B) that depends upon bottle dimensions as shown in Figure 1.4.

Table 1.1:Products that are of interest in this IRBD project

Sl. No	Product Name	Referred as
1	2800 Hybrid RH Metric	2800HRHM
2	2800 Hybrid LH Metric	2800HLHM
3	2800 Hybrid RH Inch	2800HRHI
4	2800 Hybrid LH Inch	2800HLHI
5	2800 Hybrid RH Metric Old	2800HRHMO
6	2800 Hybrid LH Metric Old	2800HLHMO
7	2800 Hybrid RH Inch Old	2800HRHIO
8	2800 Hybrid LH Inch Old	2800HLHIO
9	2800 PFinger Elevator Metric	2800PFEM
10	2800 PFinger Elevator Inch	2800PFEI
11	2800 PFinger Elevator Metric NB	2800PFEMNB
12	2800 PFinger Elevator Inch NB	2800 PFEINB
13	825 Platform Lower RH	825PLRH
14	825 Platform Lower LH	825PLLH
15	825 Laser Platform Lower RH	825LPLRH
16	825 Laser Platform Lower LH	825LPLLH
17	825 Platform Elevator RH	825PERH
18	825 Platform Elevator LH	825PELH
19	2800 Platform Lowering Inch	2800PLI
20	2800 Platform Lowering Metric	2800PLM
21	825 Laser Platform Elevator RH	825LPERH
22	825 Laser Platform Elevator LH	825LPELH
23	2800 WS Upper MFinger Elevator LH Metric	2800WSUMFELHM
24	2800 WS Upper MFinger Elevator RH Metric	2800WSUMFERHM
25	2800 WS Upper PFinger Elevator LH Metric	2800WSUPFELHM
26	2800 WS Upper PFinger Elevator RH Metric	2800WSUPFERHM
27	2800 MFinger Elevator Metric	2800MFEM
28	2800 MFinger Elevator Inch	2800MFEI
29	825 Hybrid Elevator RH	825HERH
30	825 Hybrid Elevator LH	825HELH
31	825 Hybrid Elevator RH Old	825HERHO
32	825 Hybrid Elevator LH Old	825HELHO

RH : Right Hand	LH : Left Hand	NB : No Basket
PFinger : Plastic Finger	MFinger : Metal Finger	WS : Wear Strip

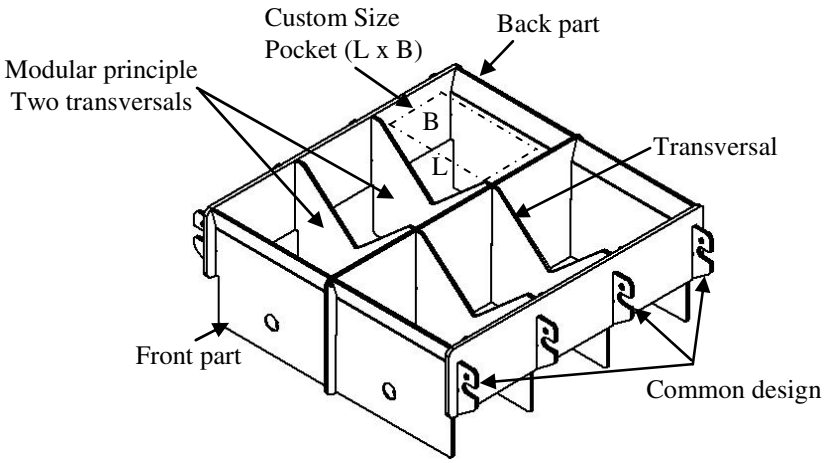


Figure 1.4: Custom modular basket that is commonly used in HI grids

These three aspects help HI to design grids that follow standard guidelines and principles which, in turn can be represented in the form of design rules. All of these rules taken together becomes a design knowledge repository from which RBD design systems can be developed. Therefore, a goal of this research is to convert the design process of various HI grid products that are shown in Table 1.1 into RBD automated programs. The above-mentioned aspects of the HI grid products result in many challenges that are similar to ETO companies. Before going into details of the challenges faced by HI, it is essential to establish a clear understanding of the term ETO as it is speculated that the HI design of grid products is similar to the design methods adopted by other ETO companies. An ETO is often referred as a *build-to*, *configure-to,* or *made-to* style of manufacturing. According to APICS dictionary [40], the ETO products are: "products whose customer specifications require unique engineering design or significant customization. Each customer's order results in a unique set of part numbers, bill of materials and routings."

Challenges of ETO companies

The challenges of ETO companies are to support high levels of customization. The challenges include high customization, producing successful products in one go, winning profitable work orders, producing consistent and reliable products, and preserving intellectual capital.

High Customization: Current markets tend to vary often and are striving to survive competition by providing greater quality, more customization, more innovative designs at affordable prices [43]. Flexibility is a key element to their success as ETO products are complex products tailored to meet individual's specific needs. ETO companies configure new products for different job order and the design may include changing well-known parameters like significant dimensions or influential features. Some other products may require greater product engineering with major alterations or redesign, which ultimately results in longer product developments, high production lead times, and high costs. These demands change from customer to customer and result in great uncertainties in design because of size limitations, governmental regulation, marketing strategies, environmental conditions, or operating loads [7]. In satisfying these demands, the ETO companies should utilize and store their product knowledge and experience for every variant product they design.

Producing successful products in one go: ETO companies differ from batch or mass production companies in building products. No prototype is made for every designs; therefore, product design and development must be carried out concurrently throughout the whole product life cycle. A guess is still never as good as knowing the answer which necessitates in building products in one attempt according to the customers' specification [7]. Thus building products in one go is a concern for ETO companies [47].

Winning profitable work orders: Often ETO companies need to submit full proposals which may include complete information regarding the proposed design [22]. These may include 3D CAD models, engineering drawings, detailed bill of materials, and other relevant engineering information. If they are conservative and overbid, they will not win much business. On the other hand, if they are liberal and underbid they end up in making products at the expense of profits that can hurt business over the long run.

Added to this short proposal time, which ranges from less than a day to couple of weeks, exerts extra pressures on ETO companies.

Producing consistent and reliable products: Customers judge products on every project and the manufacturers must ensure that their products adhere to consistent quality and reliability. ETO companies do not have the freedom of having product recalls or methods of using retro fittings to address quality issues in later developmental stages. To have the benefit of continuous business from long-term customers, ETO companies need to implement and enforce the best practices that they have learned in the development of products. Similarly, quality and manufacturing efficiency should be designed into a product. Manufacturing constraints need to be considered during the design stage itself, which avoids expensive surprises that occur later in the manufacturing.

Preserving intellectual capital: Dedicated employees and application of adequate knowledge becomes the most important success factors for any company [44]. Many ETO companies heavily rely on people dependant processes for producing custom products. Knowledge of an experienced human expert is an asset to an organization and incurs a substantial loss in intellectual capital when it looses this human expert. Retention of human expertise and knowledge is a major problem that enterprises are facing intoday's fast-moving job market. It is essential for the companies to capture their product knowledge and designers' experience. They should secure them in the company's databases to avoid loss of valuable product knowledge [7].

The above presented challenges are addressed in many different ways by researchers and are presented as literature survey in the succeeding sections.

Previous work in addressing ETO challenges

The following literature survey presents the outcome of the previous work done by other researchers in addressing the challenges of an ETO enterprises. The achievements of their research are [47]

design for modularity, parametric design, variant design, design for manufacturing, concurrent integrated product development, and rule based knowledge-base systems.

Design for modularity: Gu and Sosale [17] suggested integrated modular design methodology in which they describe life cycle engineering (LCE) objectives and identified customization is one of them. They suggested that modular design helps in accomplishing these objectives by providing customers with predefined choices and rearranging few optional modules. Huang and Kusiak [20] proposed that modules should be formed in the conceptual design stage for reaping the benefits of agile manufacturing. Functional interactions and physical interactions [17] should be considered in dividing into modules and based on these modules customized products can be developed. As suggested by Huang and Kusiak [20] variant products can be developed by: a) component swapping, b) component sharing and c) Bus modularity. They used a decomposition approach to detect modularity in a product set with the help of interaction and suitability matrices. Modularity concept will not fully help in customizing products but helps in designing standard components that goes into the products [47]. This all depends upon how unique a product is. Whitney [45] identified the physical limits to modularity in automating the design process. The term "ideal modularity" was introduced, wherein ideal conditions are described for using a design automation tool. Whitney identified that the multi-function nature of mechanical components is the primary reason that hinders in designing custom products. Therefore, if the multi - function nature is not present in mechanical components systems can be built that can automate the design process. Thus, modularity in design can help in producing some aspects of customized products and helps in building consistent and reliable products.

Parametric Design: Parametric design is often synonymously used with relational modeling or constraint based design [33]. Many people have done wider research in this field to automate the design process [28, 33, 49]. A CAD designing system should be able to capture the designer's intent and a system with such capability is observed as intelligent designer's assistant [41]. Such a system would automatically perform well-defined tasks, which are boring and time consuming such as creating drawings, preparing and quotations. Zalik [49] proposed acyclic constraint description graph (ACDG) for representing

geometric objects and solving constraints in parametric design problems for producing new geometric models. Zalik uses a 'black box' approach for initially defining the seed models with sketch carrying topological information and uses predicates for describing geometric objects. Later on, Lee and Kim [27] proposed knowledge based parametric design using graph representation for expediting the inference process i. e. constraint solving process. They used rules to represent constraints in a graph form. Parametric design is not applicable in situations where the designer is still conceptualizing the idea in which he refrains from assigning fixed values or constraints for concept models. In Monedero's [33] opinion it's a mistake researching advance integrated design methods for applying them in modeling without adequate 3D generating or modifying tools. Parametric design helps in building stable design rules, which helps in efficiently producing the new products. These rules can be used in generating precise quotations, which helps in gaining profitable work orders.

Variant design: Variant design is the process of adapting existing designs for developing new products [47]. It helps in relieving pressure from the designers from performing repetitive design tasks and reduces design cycle times. It also helps in developing customized products based on existing mature design [1, 2]. Study conducted by Wang [1, 2] reveals that there exists literature on component design and processing stages whereas there does not exist significant efforts in developing design methodologies for complex assembly variants. Systematic assembly variant design methodology was developed that helps ETO companies to develop new and individual design based on mature components' design [2]. Extensive database and sound reasoning methods are essential for variant design process. Group Technology (GT) is one way of creating databases that record product families and possible variations. The two important reasoning methods suggested by Fowler [13] include: a) Analogical reasoning applied to design and b) case-based reasoning applied to design. Variant design methods cannot be fully used for designing completely new products for the following reasons [13]: a) the fact that designers need to redesign the existing design depending upon the requirements and this knowledge cannot be captured beforehand. The database or knowledge-base needs to be updated every time to reflect the change and cannot update

automatically [13]. Assembly variant design methodology [1] developed by Wang *et. al*, addresses the area of complicated product data and they use assembly models, the assembly variants model, and assembly mating graphs in dealing with large assemblies in ETO industries [1, 2].

Design for X: Design for 'X' can be divided into three major headings [21]: a) Design for manufacturing and assembly, b) Design for life cycle, and c) Design for competitiveness. Design for manufacturing is addressed by Gupta et. al [18] and pointed out that the various DFM methods [4, 11] were cross functional teams of feature-based evaluation and empirical parametric evaluations [47]. Automatic feature recognition and feature-based methods are emerging technologies that are widely studied in DFM approaches. The research conducted by Lin [31] in studying computer-aided process planning for manufacturing automation resulted in integrating the above technologies in developing automatic extraction of manufacturing features from a design-oriented model. Later on, Lee and Kim [28] suggested a methodology for generating alternative ways for manufacturing a machined part. They used transformations and criteria techniques for generating alternative models by minimizing the number of tool/work piece accessibility directions by using reorientation, reduction, and/or splitting operations. Other advantages of using DFM as documented by [4, 35] indicates the possibility of : i) 61 % reduction in product assembly time, ii) 53 % reduction in the number of assembly operations, iii) 68 % reduction in the number of assembly defects, and iv)50 % reduction in time to market.

Design for life cycle includes design for dimensional control, design for inspectability, design for effective material storage and distribution, design for reliability, design for serviceability, design for ease of disassembly, and design for recycling. Design for competitiveness includes design for quality, design for modularity, design for optimal environment impact, and design for uncertainty. Quality and reliable products can be produced by integrating all the above aspects into the product during the preliminary stages of design [4, 11, 18].

Concurrent integrated product development: In this process, a simultaneous engineering (SE) approach is used which combines all evolutionary methodologies in iterative product development [9].

Evolutionary design methods as suggested by Bullinger [9] implies that "previously unrecognized product requirements or technological progress must be considered and incorporated" in later stages of design. During the conceptual design phase, it is difficult to understand how a design parameter or set of parameters affect other components of the system. The factors contributing [16] for this are: a) insufficient communication between customer and designer, b) evolving or changing customer requirements, c) difficulties in data and information exchange and d) inconsistencies in modeling approaches between various departments. In addressing these issues, SE and rapid product development (RPD) was concurrently used for developing framework. Complete product development is done by planning on the whole process in SE. Similarly, the RPD method violates the traditional approaches of systematic design [36] and is based on the approach of evolutionary design cycle [26]. In a RPD environment physical prototypes are replaced by digital prototypes which can be produced faster by integrating CAD technologies such as rapid prototyping, virtual realty, and reverse engineering [9]. Recent advances in the internet have accelerated the advancement in concurrent product development that bridges the gap between customer and the manufacturer.

Rule based knowledge systems: Knowledge-base systems play an indispensable role in concurrent product development [9, 16, 22, 27, 34, 39]. Some examples of knowledge bases include tool database, design database and manufacturing database. These knowledge bases also help in areas of order acquisition and order fulfillment by automatic activities like preparing quotations, BOMS, drawings and routings [22]. All of these innovative technologies use predefined rules for making design related decision that exists in specific knowledge bases. The systems that use rules for generating CAD models are known as rule based knowledge systems[37]. In order to develop these systems, the company has to identify the product related design information and practices for embedding them into rule-based shells that are supplied by CAD vendors [23]. This is done by defining and summarizing company related design tasks and practices in the form of design rules for developing such automated design systems. Interdependencies, applicability, and rationale behind the product design are important for building a rule

based knowledge system. The research conducted by Institute of Defense Analyses (IDA) [23] identified that commercial rule based systems are essential for reducing design cycle times and for storing company specific design information to improve product performance and quality. The other advantages of using such systems is that, they are capable of integrating the previously discussed areas of design for modularity, parametric design, variant design, design for manufacturing and concurrent integrated product. This is because; the principles used in those areas can be represented as design rules from which rule based design systems can be developed. Some examples that use rule base systems include email filtering clients like Microsoft Outlook and Eudora, electronic hardware configurators such as XCON systems [3],and enterprise level application servers such BEA's WebLogic and ILOG for their use in logistics and supply chain [14].

Research Opportunities

From the above literature review, it can be seen that rule based knowledge systems are essential in developing automation systems for producing customized products. These rule based knowledge systems are referred as rule base design (RBD) systems in the remainder of this thesis. RBD is the process of designing artifacts by using rules for taking design related decisions. Literature exists in efficiently processing the data within the knowledge base, but researchers have done less work in explaining the intermittent stages that are involved in building knowledge-base systems. Knowing the intermittent stage helps ETO companies in estimating the time required for RBD program development, which helps them in automating the design process.

The research questions are presented in the next sections and are framed for specifically addressing the challenges of ETO industries. The proposed research question work will address these challenges by using RBD methods in commercial automation software (DW). A major section is presented on representing knowledge in the form of rules. This research also presents best practices in CAD modeling for RBD method, which is a result of this case study. Further, estimated time calculations can be used in calculating the return on investment (ROI), which is left for future work.

Research Questions

With respect to the previous discussion on research opportunities and motivations, three research questions are proposed. Addressing these research questions will help ETO companies to overcome the challenges that they face. The scope in which this research is conducted is RBD method of designing variant artifacts. How designers make decisions related to design are explored and these design decisions are represented as rules in this research. The HI case study is explicitly used to explain the elicitation and implementation of rules in designing grid assemblies. Different grid products are studied and analyzed to identify various stages and their times in developing a RBD program using DW.

The first research question is formulated for explaining the stages that are involved RBD program development process. The remainder of the thesis discusses these stages and the activities that are involved in them. The factors that affect these individual stages are also explored for developing time estimations. The resulting time calculations help ETO companies in ROI estimations on allocating resources for converting frequently changing design of a variant product into RBD automation systems.

RQ 1: What are the stages that are involved in the DW RBD program development process?

The second research question is framed to identify the parameters that affect the RBD program development process. There is an opportunity to study different HI grid assemblies, which has a potential for many parameters that need to be controlled and rules can be written for them. The parameters that are of interest for developing RBD programs are dimensions, and features. Different parameters affect different stages of the RBD program development process in terms of time and discussion on these parameters is presented in the remainder of this thesis.

RQ 2: What are the programming parameters that affect the stages in the RBD program development?

The third research question is formulated to develop the empirical models that predict time estimations. Time is dependent upon parameters that are involved in various stages of the RBD program

development process. Since multiple HI grid assemblies are present, precise relations can be chalked out between the interested parameters and equations related to time calculations. In this research, numerous RBD programs are developed for various HI grid products and times are recorded for activities that involve parameters of interest.

RQ 3: How time estimations for RBD program development process depend on CAD parameters?

Thesis road map

The remainder of this thesis is organized as follows: Chapter 2 deals with the need of knowledge base system and it elaborately discusses on rules and presents the stages that are involved in the RBD program development process. Chapter 3 discusses the second stage of RBD program development because the first stage of CAD modeling is not part of HI IRBD project. This chapter discusses about interviews as the method of knowledge acquisition process. It also describes the method adopted in the HI IRBD project and ends with the estimated time calculation models for capturing design information. Chapter 4 discusses the preprocessing stage of DW programming, where it presents the best practices in CAD modeling for capturing parameters. This chapter also explains the individual activities that are involved in the third stage of RBD program development process where it presents the time calculations. Chapter 5 describes the corresponding individual activities of fourth stage of DW program development. It presents the important aspects of creating a graphical user interface, the corresponding data, and the rules that are essential for creating the DW program. This chapter ends with presenting time calculation models of the DW RBD programming process. Chapter 6 deals with the debugging and testing process of RBD programs, which need to be verified before releasing the programs. Chapter 7 deals with a demonstration study where it conglomerates all equations presented in earlier chapters for calculating the time estimates for various stages. The obtained estimated time is validated against the observed time by actually programming a new grid product. Chapter 8 concludes the thesis by presenting the conclusions and future work that is required for refining these time models.

CHAPTER 2

RBD PPROGRAM DEVELOPMENT

Need for knowledge base tools

Knowledge about the desired domain is an essential element in designing a RBD system. Vajna [44] describes the knowledge as the sixth and most important production factor besides people, machines, material, money, and information. The understanding of design information helps in making decisions, and the designers attain this knowledge through experience. Knowledge of an experienced designer is an asset to an organization which forms the intellectual capital [10]. The organization incurs a substantial loss when it loses this expertise. Building CAD modeling systems that can capture, represent, store, and reuse corporate design knowledge would solve the problem of losing this intellectual capital. One of the foci of this research is on capturing and representing design information. The storage and reuse aspects of knowledge are shown by developing RBD systems for automating the design process of HI grid assemblies by using DW and SW.

Traditional CAD models represent geometric information about the products and it changes for every new design specification. The result is to do repetitive design tasks for generating production related information such as geometric models, CAD drawings, BOM and process sheets. Repetitive work can be automated by building system that use stored knowledge for making decisions related to design. For building these systems, there is a need for representing machine recognizable information that is interoperable with industry's existing software. DW is one such system that uses simple rules for generating CAD related information in SW. These simple rules can be used for driving the design parameters such as dimensions or features.

Knowledge Representation: Rules

RuleStream[4] defines a rule as any thought that can be quantified [5]. A rule is obtained when one can express his thoughts so that another human or machine can obtain the same results [5, 6]. Rules can be represented by using simple equations and can be classified as: functional and physical properties rules, parametric rules, feature rules, part selection rules, manufacturing rules, safety rules, check rules, price-determining rules, regulation rules, and so forth. Parametric rules constitute about 20 - 25% of all the rules [6] while rest of them deal with other concerns related to design and manufacturing.

In the HI IRBD project, knowledge is represented in two ways. The first, are the *algebraic equation rules* in which mathematical relations are used to obtain value of CAD parameter (parametric rules). Assignment operator, the "equal to (=)", and the algebraic operators: the plus (+), minus (-), multiplication (*), and division (/), are used for writing the parametric rules. The second are traditional *IF-THEN conditional rules.* The antecedent (the "IF" part) and consequent, (the "THEN" part) are required for building a conditional rule. The syntax for a simple conditional rule is:

$$\begin{aligned} &IF \ \ < antecedent > \\ &THEN <consequent\_1> \\ &ELSE <consequent\_2> \end{aligned} \qquad \textbf{Equation 2.1}$$

An antecedent is an algebraic equation, which drives the consequent. The resultant consequent is applied to design parameters (dimension or feature) in the form of value. Here value refers to the numerical number or state of the feature such as suppression state or unsuppression state. Operators are used in writing algebraic equations. Conditional operators such as not equal to (!=), greater than (>) or less than (<) are used to compare the value against a reference, while assignment operator is used to assign a value to a design variable. Multiple antecedents are joined by logical operators such as AND (conjunction), OR (disjunction) or combination of both. Equation 2.2 and Equation 2.3 shows the

[4] http://rulestream. com/

algebraic equation rule and conditional rule for controlling the transversals in basket sub assembly which is shown in Figure 2.1.

$$Transversal\_Number = Bottle\_per\_Lane - 1 \qquad \textbf{Equation 2.2}$$

$$\begin{aligned} &IF\ Transversal\_Number < 1 \\ &THEN\ \text{"Suppress Transversal"} \\ &ELSE\ \text{"Unsuppress Transversal"} \end{aligned} \qquad \textbf{Equation 2.3}$$

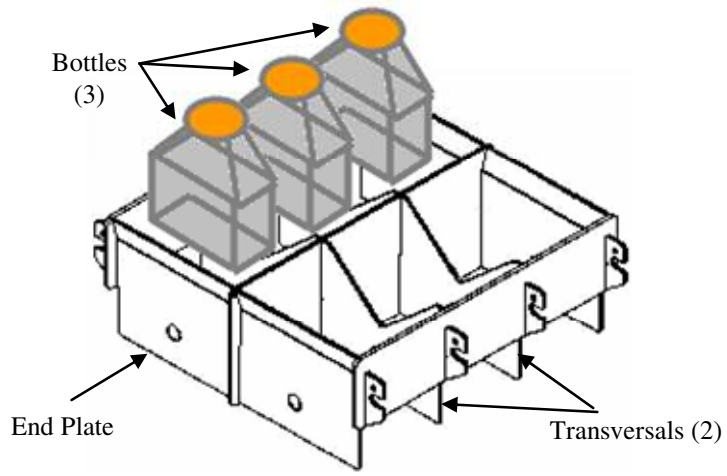


Figure 2.1: Basket sub assembly within a grid product

Sometimes transversals need to be suppressed even though the number of transversals is not less than one as in the design where transversal are not required for a particular basket sub assembly. In this case, logical operators are used with more than one consequent as shown in Equation 2.3.

$$\begin{aligned} &IF \left(\begin{array}{c} Transversal\_Number < 1 \\ OR \\ Transversal\_are\_not\_required \end{array} \right) \\ &THEN \quad \text{"Suppress Transversal"} \\ &ELSE \quad \text{"Unsuppress Transversal"} \end{aligned} \qquad \textbf{Equation 2.4}$$

Apart from design decisions, design constraints can also be encoded in RBD systems by writing verification rules. Typically, problems that are constrained with in boundaries need these rules. Primary set of rules are first used to produce the design, which is then checked against the constraints encoded in

the verification rules. For example, the end plate of the basket assembly that is shown in Figure 2.2 requires these verification rules.

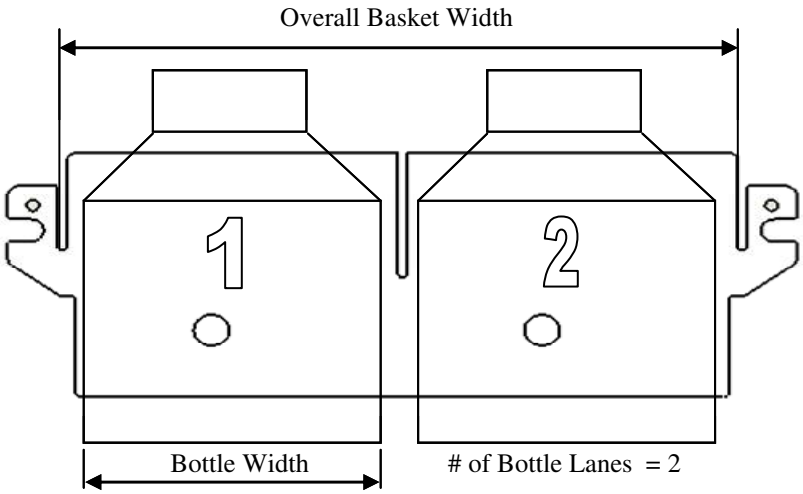


Figure 2.2: End plate of basket assembly

The overall basket width is computed by using the bottle width, clearances between bottles and, number of bottle lanes as shown in Equation 2.5.

$$Overall\_Basket\_Width\ (OBW) = \begin{pmatrix} Bottle\_Width \\ + \\ Bottle\_Lane\_Clearance \\ + \\ Number\_of\_Bottle\_Lanes \end{pmatrix} \qquad \textbf{\textit{Equation 2.5}}$$

The resultant value should be less than the overall width of the grid (15 inches in this case) for holding the basket assembly as shown in Figure 1.1. The rule for this verification is given by Equation 2.6 as shown below:

$$\begin{aligned} &IF \quad (OBW < 15) \\ &THEN \ "Design\_Possible" \\ &ELSE \ "Design\_Not\_Possible" \end{aligned} \qquad \textbf{\textit{Equation 2.6}}$$

In a RBD program, knowledge is encoded by a collection of rules for making design decisions. RBD programs are developed using system shells [25] which provide a skeletal structure for writing rules in common programming syntax. These shells are platforms on which a RBD system can be built without

any knowledge in them [23]. When rules are explicitly embedded into shells, they are ready to take up the task for the desired application of generating CAD related information. DW is one such software where SW seed CAD models are driven to produce the variant CAD models and are called "clones" in DW terminology.

Limitation of RBD programming

In software engineering perspective, RBD programming has some limitations because systems developed in it are difficult to maintain, test, and are unreliable [30]. The roadblocks to RBD programming are:

Knowledge sharing: Knowledge acquisition is essential for developing RBD systems. The programmer gains this knowledge from the expert designer who has experience about that product. It is difficult to capture 100% of the designer's experience in one attempt and the designer sometimes cannot express some types of thoughts in words. Some other designers may not disclose full information about the product because they are afraid that the new systems would make lose their jobs.

Maintainability: Maintenance of the software product consumes up to 60% of the total product cost in its entire life-cycle [30]. In Li's view, maintaining a RBD system is almost impossible. Jacob and Froscher [24] have mentioned that, in order to upgrade the knowledge base of a RBD system, a knowledge engineer with extensive understanding of the system design and structure is required, which implies the person originally developed the system. This is a major drawback of RBD systems because more than one person involved in developing them. Highly declarative applications are well suited for RBD programming architecture that primarily consists of a rule base, a database, and an inference engine (DW). Most of the real world problems, which fall into evolutionary applications, do not deal with RBD paradigm. Adequate control of non-trivial applications is a major problem in RBD paradigm.

Testability: A conventional program written in C or C++ is easier to grasp the flow of control and data, because of well-structured and nice looking statements [30]. Single entries and single-exits are the

characteristics of well-structured program while proper indentation and segmentation are the characteristics of nice looking program. Many real world applications cannot be coded in RBD paradigm by these well-established requirements of high quality software's. The main problem identified by Li [30] in terms of its untestability and unmaintainability is loss of modularity (or encapsulation) and abstraction (hierarchical representation) which are the essential requirements for quality software.

Reliability: RBD programming is unacceptable since it is non-deterministic in nature. For applications that evolve continuously, the testing mechanism to prove the absence of errors is low [30]. The reliability of the expected situations should be consistent before dealing with the unexpected situations. Lee [30] pointed that, "such systems must overcome their inherent control uncertainties before they can deal with the uncertainty of an application". RBD systems are good for giving expert advices that do not require any common sense as in diagnosis and scheduling systems.

Unless reliability and maintainability issues are addressed, the maintenance of RBD systems becomes difficult. Pure declarative statements are the bottlenecks in RBD programming in terms of representing how well a human thinks and how smart in problem solving.

Commercial RBD Approach

The above limitations can be mitigated by using commercial RBD systems. They offer a structured way of programming RBD applications. DW is one such RBD automation system used for generating CAD related data in SW. In HI IRBD project, DW was used for driving SW CAD models. The rules are encoded in DW with the help of customer specifications that are obtained from the user interface (UI) forms. These design specifications in conjunction with other product related information such as data, constants, and variables are used in writing rules to drive CAD models. For every run of design specifications, the DW will generate **. mdb** (Microsoft access) and **. xml** (MS excel) files as outputs. These files are used by DW model generator to produce the variant CAD models which are a result of driving CAD seed models. The entire process of producing clones is shown in Figure 2.3.

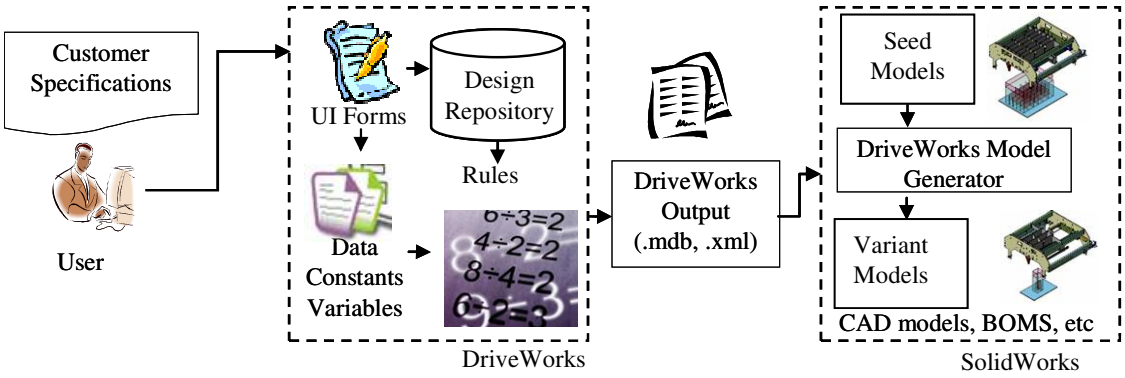


Figure 2.3: The Commercial RBD Approach

As seen in previous sections rules play an important role in building RBD systems. This approach of using rules has the following advantages:

1. Rules can be captured in a dedicated RBD software tool making declaration, access, and retrieval of rules easy.

2. Rules can be declared and can be edited using the DW graphical interface. This greatly enhances rules' readability and interpretation while eliminating the designers' dependency on programmers.

3. The declaration syntax is simple, typically stated in the form of equation or IF-THEN format, with only one variable being controlled in each statement. This makes the rules structure modular for having easy control of them.

4. The seed models in SW can be built in sync with the DW output format. This needs special attention toward modeling, which is covered in Chapter 4 under the heading best practices in CAD modeling.

5. The rules can be stored and managed in an environment outside the CAD system, which enforces design standards and reduces the risk of errors by inexperienced designers. At the

same time the rules can be simply declared, which allows an experienced designer, often the administrator, to edit these rules.

RBD programming and its stages

From the HI IRBD project, it can be seen that DW RBD program development typically consists of six stages. They include: 1) SW CAD modeling, 2) Knowledge Acquisition, 3) Extracting CAD parameters that need control, 4) DW Programming, 5) Debugging and Testing, and 6) Production Deployment. Figure 2.4 shows the process of DW RBD program development process. Stages I and II are interchangeable and the sequence depends upon the type of product involved. For example, if a company starts to build a new product and if it wants to automate the design process for related future variants, knowledge acquisition becomes the first stage and CAD modeling becomes the second stage. In a case, where CAD models are already present and later decided for automating the design process, such as in the case of HI IRBD project, Stage II follows Stage I. The next stage in this process is to gather the product design knowledge, which is called knowledge acquisition stage. In this stage, the programmer collects the design information and identifies the dimension and features that vary with order specifications. Knowledge acquisition is done by conducting interviews with people who have thorough knowledge about the product. After fully understanding the product, the programmer extracts the CAD parameters that need to be controlled. Extracting parameters is done with the help of DW Model wizard, which can be found in DW tool bar with in SW. The next stage in this process is to elicit rules for extracted parameters. This is done in the DW Administrator, where there is a provision to create user interface forms. The forms are modeled to capture all design specifications (customer inputs). By using these specifications and assembly related constants, rules are built for frequently used variables and for extracted parameters. Rule development is an intellectual task, and the time required for developing rules varies greatly from person to person. Programmer's mathematical concepts and experience play a vital role during this process. Once

the program is developed, it is thoroughly debugged and tested for eliminating bugs. After testing, the program is ready to release for applying it in production.

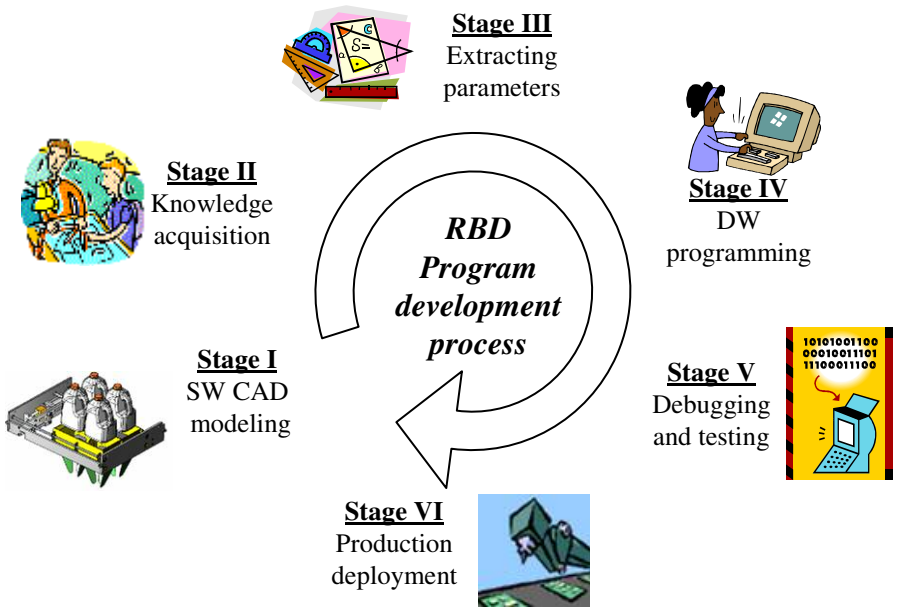


Figure 2.4: Stages in RBD program development

Chapter Summary

This chapter discusses the need for knowledge-base systems in current ETO markets and presents the RBD method. It describes about the types of rules and shows their use with an example for controlling the transversals within a basket sub assembly of a grid product. It describes the advantages and limitation of RBD systems. Later it introduces the commercial RBD system (DW) that can be used for writing rules and the various stages that are involved in program development process.

CHAPTER 3

KNOLWDGE ACQUISITION

The previous chapter dealt with the RBD programming process in which there was a major discussion on rules. A rule is built by using information related to design and becomes the knowledge about the product. Knowledge is an essential element in building a RBD program and the process of collecting product related design information is known as *Knowledge Acquisition [23]* and is the Stage II of RBD program development process as shown in Figure 3.1.

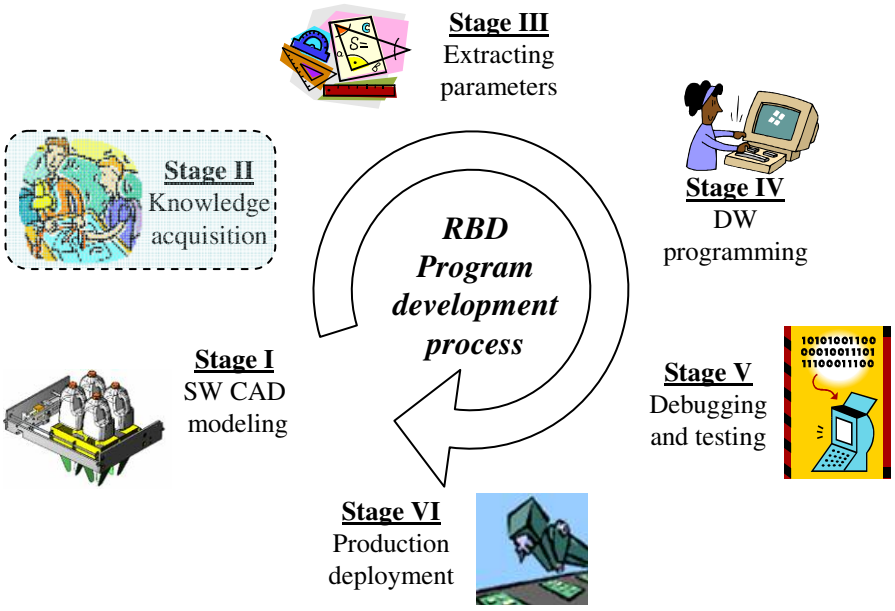


Figure 3.1: Knowledge acquisition stage in RBD development process

Knowledge Acquisition

Knowledge acquisition deals with the subset of the world's collected knowledge in which knowledge engineers play a dominant role. A knowledge engineers profession is to collect design information for building knowledge-base systems and they collect design related information from the designers who have thorough knowledge about the product. In general, the knowledge engineer is

interested in collecting, the following categories of information [15]:

Requirements: Requirements are the objectives that need to be met for a design problem. It is typically a design brief, which specifies the statement of intents. The requirements are guided by set of constraints or specifications. In the HI IRBD project, the requirements are to automate the design process of grid change over parts with the help of DW.

Principles: Principles are the organized design guidelines of specific domain. These principles may be a set of equations for designing a particular product and are obtained with experience. The principles may include the design clearances or standard dimensions that are specific to a given company or product.

Resources: In order to obtain the principles defined earlier, it is essential to identify the resources. The resources include company personnel, design documents, past design, vendors, and internet search engines.

Limitations: Limitations are the constraints of a particular design. The knowledge engineers should have a strong understanding of these limitations in order to implement strategies to avoid them. These limitations may be due to the available resources, manufacturing limitations, and machine limitations on which grids are assembled.

The above categories of information can be collected by conducting interviews. In HI IRBD project, interviews are used to collect the information about the grid products. In addition to these interviews, telephonic conversations and email exchanges are also used for gathering the missing product information. Later, all of the captured information is preserved in the form of a design document to show design information about the product. Desk research becomes the part of the documentation process, which includes gathering the missing information that is not captured during the interviews. Detailed discussion about the documentation and desk research is presented later in this chapter.

Interviews

Initiating a new knowledge acquisition process is a difficult task and the knowledge engineer has to obtain product related design information from various sources. Interviews are helpful in obtaining the first hand information about the product and are particularly helpful in gathering knowledge from an experienced designer. The interviewing options that are available for knowledge engineers' include [32]:

1. Informal, conversational interview approach,
2. General interview guide approach,
3. Standardized, open-ended interview approach, and
4. Closed, fixed-response interview approach.

In an *informal conversational approach*, no predetermined questions are asked and the answers lead to new questions. An interview takes place with the conversation of flow In the *general interview guide approach*, predetermined questions are asked to the respondents, which are more focused about the domain than the former approach. This type of interview approach is the best method for collecting domain dependent knowledge. In the *standardized open-ended interview approach*, the same open-ended questions are asked to each respondent. The open-ended questions are the ones which do not restrict to '*yes'* or '*no'* answers. Finally, in the *closed fixed response approach*, standard questions are asked to various respondents and they are given a choice to pick from set of alternatives. This is used in surveys for getting feedback about the product. In any interviewing approach, for reaping the maximum benefits, it is important to have a good relation with the interviewee. The following are the strategies for building a good rapport with the people [15]:

Technical Jargon: The knowledge engineer should become familiar with technical jargon of the product domain and should avoid using programming terms like loops, classes and objects. Instead, the

knowledge engineer should use terms such as patterns, features, and entities that are familiar to technical people since they use them frequently in their day-to-day activities.

Treat equally: It is important to show respect to the technical people. Without them, the knowledge engineer cannot gather any information related to the product. Interviewees' time is more important than the knowledge engineers' time, since they are doing a favor by letting out the useful information about the product.

Involvement: The knowledge engineer should always make an eye contact with the interviewee and should ask follow up questions if the answers are not clear. Writing quick notes is a good idea and helps to avoid asking again the same question. The knowledge engineer should make a friendly interview environment so that both are comfortable in asking and answering the questions. The knowledge engineer should ask questions related to domain so that they stay involved and focused.

Be reassuring: The knowledge engineer often needs to collect information from the current employee. The employee might be afraid of losing his or her job because of new technologies while the knowledge engineer should assure that the new systems are used for doing repetitive tasks. Knowledge engineers should make it clear that the employee would have more time to concentrate on how to improve the existing and new designs.

In the HI IRBD project, the Clemson students used the interviewing methods of general guide approach and closed fixed response approach for gathering the design information related to the grid product. The following sections explain in detail the interviewing method adopted in the HI IRBD project.

Method adopted in interview process

Two Clemson students are involved in HI IRBD project. One of them had trained to work on HI grid assemblies and DW for about three and half months with in the company. This student had an opportunity to understand design and manufacturing process of grid assemblies. With the gained

experience, this student helped the other student to learn about grid design. Later both of them visited HI on Fridays and gathered information about the remaining grid products for about 3 hours. The focus was to gather rules for a particular grid product that was on a priority list for RBD programming. The students requested the grid designer to keep the sample models in company's' their ftp site so that Clemson students can have an idea about the product before the actual interview process. Similarly, the grid designer kept all of the rules ready for the products and explained the rules with the help of CAD models and drawings. The interview process used to typically last for 2-3 hours and helped to gather the required information about one grid product. In addition, students used to get any missing information about the previously collected grid products. After the interview process, grid designer handed over the CAD drawings, which depicts the parameters that need to be controlled and their corresponding rules.

To gather the information about the similar product, it took less time since the students are conversant with the product. During these situations, the students collected design information for more than one grid product. This helped in reaching a conclusion that the time required for gathering information about new products is greater than the time required for gathering information about similar products.

Before starting the programming process, Clemson students are supposed to prepare and submit a design document to HI for approval. This submitted design document depicts all the assemblies and components that change with specifications. The document acts as a design journal, which explains all the rules about the assemblies and components. HI grid designer are supposed to verify these design documents for accuracy before approving them. The next task after getting approval for submitted documents is to start the RBD programming process in DW. For getting any missing information, the Clemson students had to contact the grid designer through emails and by WebEx meetings. Meanwhile within Clemson, the students did desk research for finding out the missing information. The term desk research is loosely used in referring to the collection of the secondary data that helps to find the missing

information from the primary collected data during the interviews. Desk research is augmented with legacy research or background research.

In HI IRBD project legacy research and background research was done by going through the existing AutoCAD drawings and SW CAD models. This was mainly done for figuring out the missing information of the grid components without contacting the grid designer. HI Company's website also helped in understanding the various products that they manufacture and the working of their machines. They have video files on their website that helped in better understanding the working principle of the case packing machines. The succeeding sections deal with the parameters that affect the knowledge acquisition process by using grid products as examples.

Three products are selected from Table 1.1, in showing that the time taken for capturing design information about similar product to that of already captured product will take less time than for capturing unique product. The three products chosen are shown in Figure 3.2, Figure 3.3 and Figure 3.4 respectively, and are:

1. 2800 Hybrid RH Metric (2800HRHM)
2. 2800 PFinger Elevator Metric (2800PFEM), and
3. 2800 PFinger Elevator Metric No Basket (2800PFEMNB)

The 2800 Hybrid RH Metric (2800 HRHM) is a large assembly when compared with the other two assemblies. It has three major sub assemblies that include the upper basket assembly, the shift frame assembly, and the lower basket assembly. Figure 3.2 shows the 2800 HRHM grid product with its sub assemblies. Figure 3.3 shows the 2800 PFinger Elevator Metric (2800 PFEM) grid product. It has two major sub assemblies, which include the upper basket assembly and the lower basket assembly. Finally, the 2800 PFinger Elevator Metric No Basket (2800 PFEMNB) is shown in Figure 3.4. The only difference between the second and the third products is the upper basket.

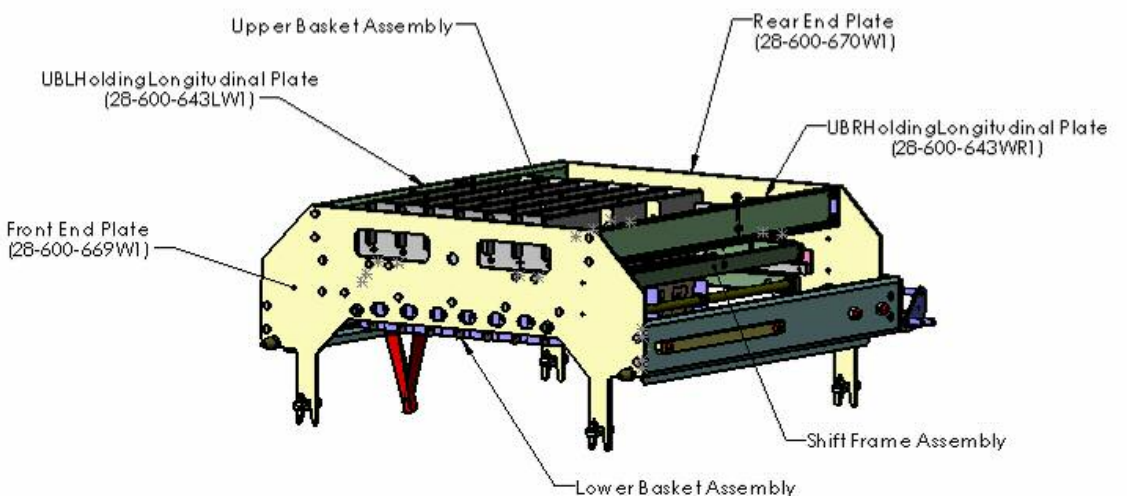


Figure 3.2: 2800 Hybrid RH Metric

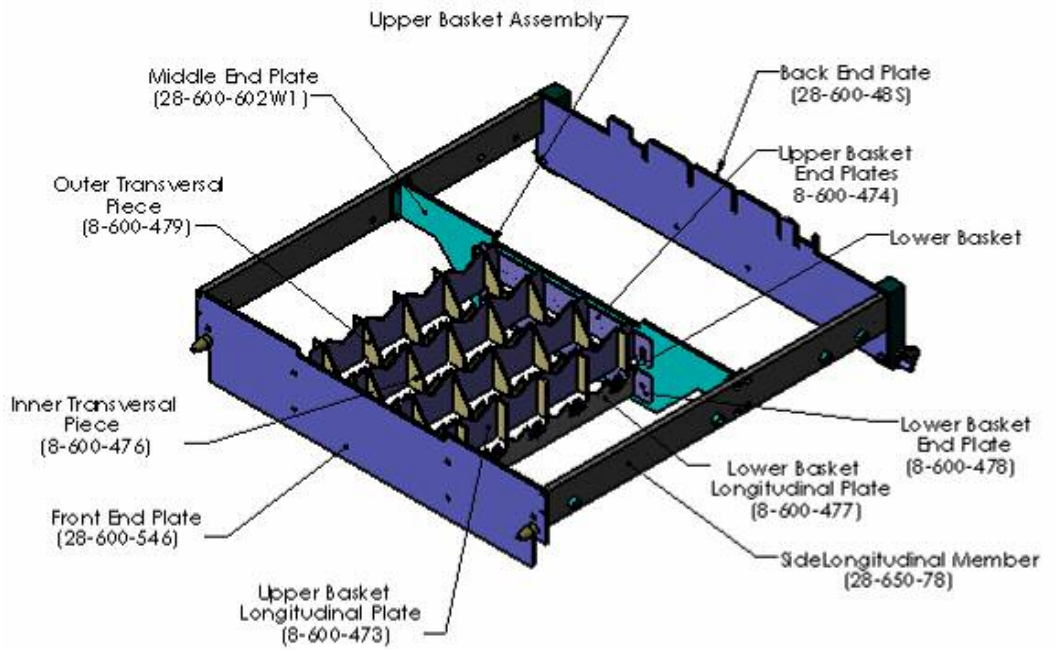


Figure 3.3: 2800 PFinger Elevator Metric

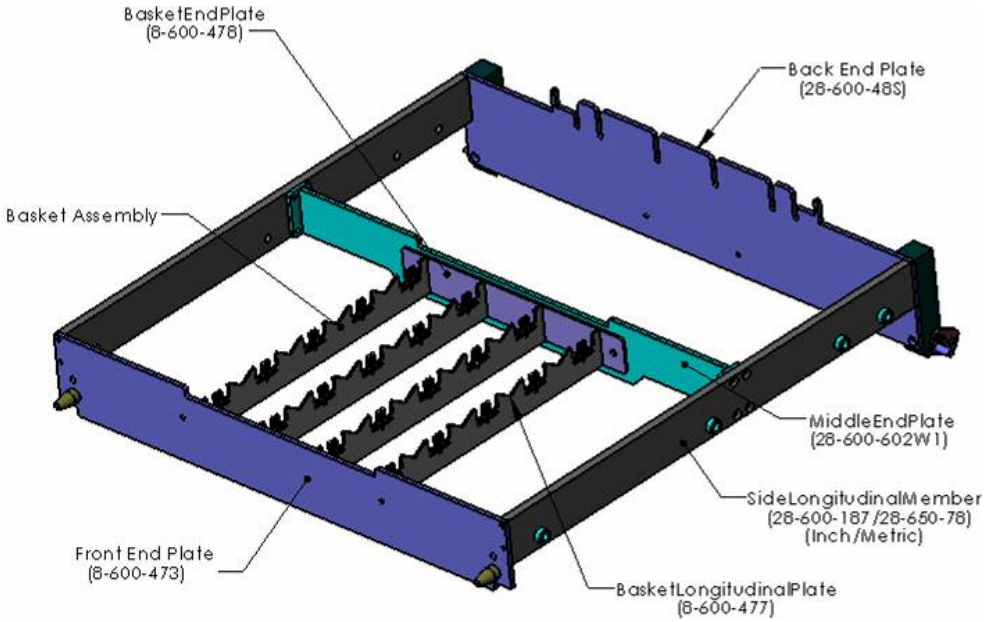


Figure 3.4: 2800 PFinger Elevator Metric No Basket

Table 3.1: Various parameters for three grid products

Sl. No	Parameters	Product Name		
		2800 HRHM	2800 PFEM	2800 PFEMNB
1	Total number of Assemblies	32	10	5
2	Total number of Parts	95	20	14
DW Programming statistics				
3	# of assemblies that need control (A)	14	7	5
4	# of components that need control (P)	25	10	6
5	Gathered dimensions rules (d)	155	41	20
6	Gathered feature rules (f)	89	12	6
7	Time in min for capturing MSA	360	120	15
8	Time in 'min' for capturing one rule	2	3	1
9	Time in 'hours' for preparing design document	30	12	4

The metrics from the Table 3.1 suggest that the time required for knowledge acquisition depends upon the number of dimension (d) and number of features (f). The recorded observations depict that, for capturing design knowledge of a sub assembly will take on average about one hour for grid products. This includes opening models in SW, reviewing the components' construction history, selecting the features, explaining the dimensions that need control and making printouts of CAD drawings. If one considers

information at the lower level, it will take on average about 2-3 minutes to collect information about a particular dimension or feature. On the other hand, for capturing information about similar products, it will take on average about half an hour for the grid products. This time includes opening the drawings and explaining the missing information from the previously collected information or suggesting the changes between the existing and new designs. The other observation is that, for capturing the information at parameter level, it will take on average about one minute to explain that the components are similar or dissimilar. These observations are only valid if the knowledge engineer has some prior knowledge about grid products. In addition, these are valid only for capturing information about grid products and may not be suitable for capturing any other products.

From the Table 3.1, it can be seen that the time taken for collecting the information is dependent upon low-level parameters such as dimensions and features. The other observation is that it takes roughly about 1-3 minutes for capturing the information about a particular parameter. Therefore, for time calculations one can takes 3 minutes for capturing a particular dimension (d) or feature (f) of a grid product and depends upon the number of parameters altogether. The activities that are involved in this process include opening of the CAD model, browsing through the feature construction history, opening the sketch, selecting the dimension, and explaining about that dimension. Therefore, the total time for capturing the design knowledge (T_I) for the grid products by using interviews can be given as:

$$T_I = 3 * (d+f) \qquad \textit{Equation 3.1}$$

Where,

T_I = Time required for capturing design information using interviews

d = # of dimensions that needs control

f = # of features that needs control

Documentation of captured information

After interviewing the designers, the knowledge engineers should prepare a design document that depict all of the collected information. Preparing a design document is a tedious task since the knowledge engineer has to explain all the assemblies, parts, dimensions, and features that vary with design specifications. This process includes preparing CAD drawings and annotating parameters such as dimensions and features in detail. In addition to these parameters to control, it should show the corresponding information required for programming. The corresponding information will include what constants need to be used, what variables need to be used and other information related to data and tables. Finally, this design document should include the rules for the parameters to show how the specifications affect the related components' construction information (dimensions and features). In brief, this design document becomes the design bible for rest of the DW rule based program development process.

Preparing a design document can be avoided in the DW RBD programming stage, when a designer produces this document for every new design that is produced in SW. Therefore, if the design document is present beforehand, the knowledge engineer can capture most of the design knowledge from them reducing the dependence upon the designer. This results in a significant amount of time saving during knowledge acquisition.

The knowledge engineer may prepare a rules document rather than a design document. The former document can be considered as the condensed version of the latter which emphasizes on rules for driving parameters. Preparing a rules document is an intellectual task because of coming up with the constants and variables that are used in writing rules for the extracted parameters. The time for preparing the rules document can be minimized by using the standard template documents. The figures from the design document can also be directly copied into rules document, eliminating the creation of CAD drawings from scratch that results in saving a considerable amount of time.

Based on the experience from the HI IRBD project the time required for preparing the grid design document can be approximated as shown in Equation 3.2.

T_{DD} = 10 hrs for assemblies that have components < 20
= 20 hrs for assemblies that have components in range of 20-40
= 30 hrs for assemblies that have components >40 ***Equation 3.2***
This T_{DD} is called grid design documentation constant C_1 in the remainder of the thesis

Therefore, the time taken for knowledge acquisition can be given by Equation 3.3, which is the sum of time taken for capturing the design information using interviews (Equation 3.1), and the time taken for preparing a design document (Equation 3.2) as given below:

$$T_{KA} = T_I + T_{DD}$$
$$\Rightarrow T_{KA} = 3 * (d+f) + C_1 \qquad \textit{Equation 3.3}$$

Where,
T_{KA} = Time for knowledge acquisition
T_I = Time for conducting interviews
T_{DD} = Time for preparing the design document (C_1)
C_1 = Grid design documentation constant

The values of C_1 are applicable only for HI grid products since the total number of components does not exceed seventy.

Chapter Summary

This chapter discusses one of the important stages of the DW RBD programming process i. e. knowledge acquisition. It presents the method of capturing design information and suggests that interviews can be used for capturing the design information. This chapter also discusses the post processing stage of the interviews, which include preparation of design documents as part of desk research. It also presents the estimated time calculations for conducting an interview and for preparing the design document based on the parameters that need control.

CHAPTER 4

EXTRACTING PARAMETERS TO CONTROL

In order to start the RBD programming in DW, the dimensions and features that are identified in the knowledge acquisition need to be extracted from SW models. This task of extracting parameters is done after getting the approval for design documents that are prepared during Stage II of RBD program development. The Stage III of parameters extraction is shown in Figure 4.1. Extraction of the SW parameters is done by the DW model wizard, which can be found in the DW tool bar within SW. It is easy to extract parameters from the SW CAD models that have additional design details of the product rather than having simple sketches or features. These details should be added by the designer and are advised to follow these guidelines while building CAD models for RBD programming. These guidelines are presented as best practices in CAD modeling in the succeeding section.

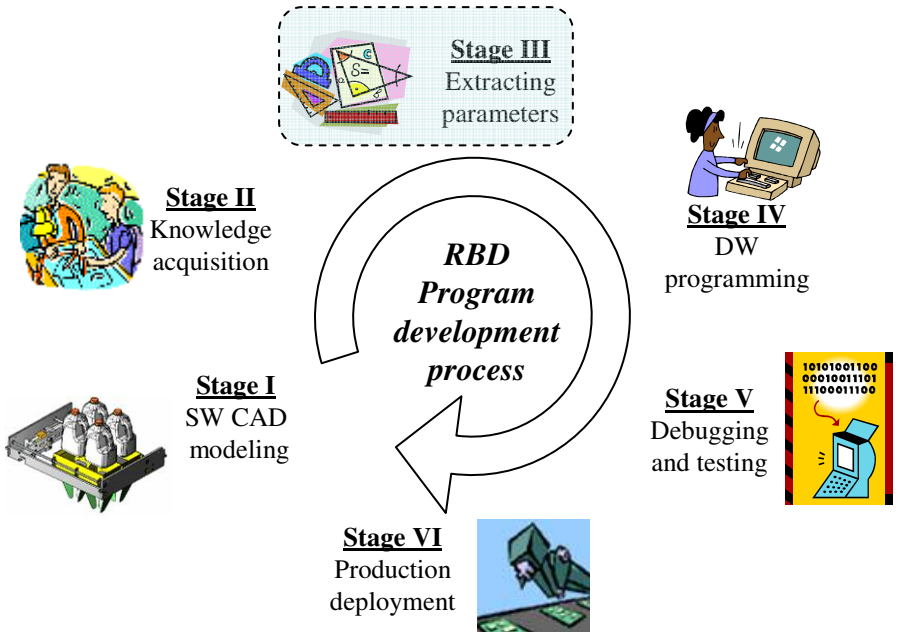


Figure 4.1: Stage of extracting parameters in RBD program development process

Best practices for CAD modeling

In order to make models compatible with RBD for being able to be driven by DW, designers should follow certain guidelines during CAD modeling. These guidelines help in future stages of RBD programming for easy capture of parameters and for debugging these programs.

Relative referencing: During the creation of sketches, the designer should avoid dimensioning the detail entities from absolute reference frames. This helps to maintain the parametric relation in terms of dimensions. The incorrect and recommended ways of creating sketches for building a feature is shown in Figure 4.2. In Figure 4. 2a, four rules are needed for driving the feature and in Figure 4. 2b, only one rule is required for driving the feature in terms of dimensions and position. In the latter method when DW imposes a change in driving parameter, the driven dimensions are properly updated with respect to each other, ensuring successful model regeneration.

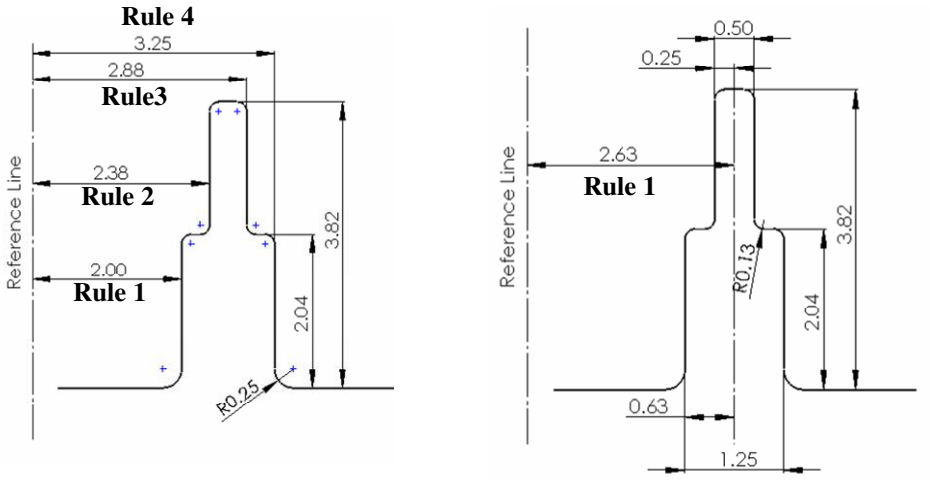
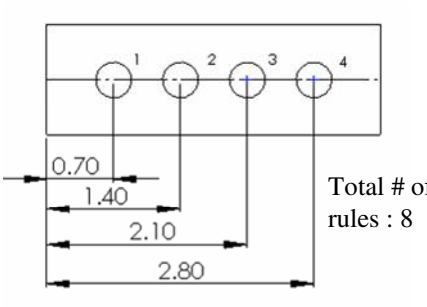


Figure a: Incorrect **Figure b: Recommended**

Figure 4.2: Relative referencing

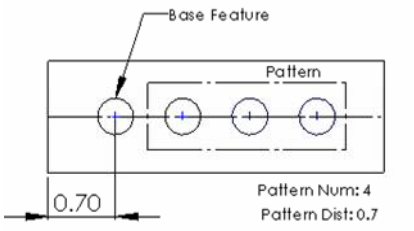
Use of pattern option: For multiple occurrences of the same features (e. g. an array of holes), the designer should use the pattern tool. This allows DW to control the pitch and number of occurrences in the

patterns by using less number of rules. In the example shown in Figure 4.3, if the features are modeled individually, see Figure 4.3a, eight rules are required for controlling four holes in terms of their position and state (suppression or unsuppression). On the hand, if one uses the pattern tool the control of these four holes can be achieved by only five rules as seen in Figure 4.3b. Moreover, if the features are modeled individually, the number of rules required for controlling them is directly proportional to the number of features instances. In Figure 4.3a, the number of rules required will be two times the number of features instances. However, if the pattern tool is used, only five rules are needed for controlling any number of feature instances, which saves a lot of time in capturing and writing rules.



Total # of rules : 8

Figure a: Incorrect



Pattern state : Suppress/Unsuppress

Total # of rules : 5

Figure b: Recommended

Figure 4.3: Use of pattern tool

*Creating independent features :*SW allows combining the profiles of multiple features in a single sketch. The designer should avoid combining them in single sketch, unless they do not need any control by DW. This incorrect way of modeling is shown in Figure 4.4a. On the other hand, the features that need control should be decoupled and created as independent entities, which helps DW to drive them individually. The correct way of modeling is shown in Figure 4.4b.

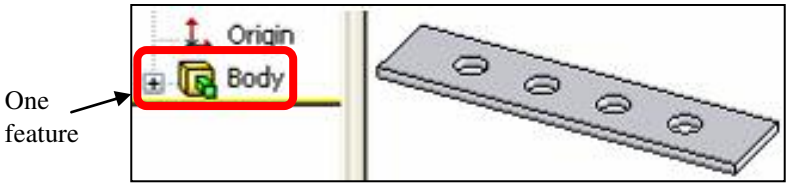


Figure a: Incorrect

Individual control of holes is not possible

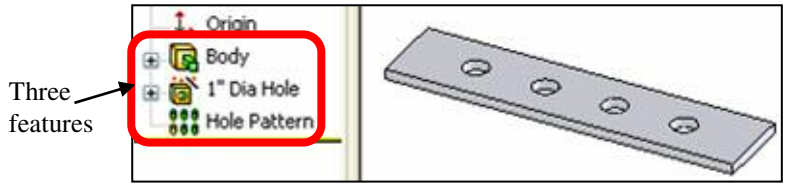


Figure b: Recommended

Individual control of holes is possible

Figure 4.4: Creating features independently

Naming the dimensions: The dimensions that need to be controlled by DW should be labeled by meaningful names so that capturing them would be easy in Stage III. This way one can avoid capturing the wrong dimension in a larger sketch that has many dimensions. It also helps in Stage V for debugging the program. Figure 4.5 illustrates the incorrect and recommended way of modeling.

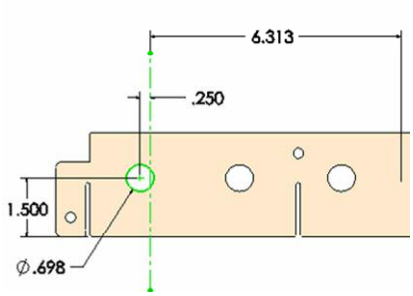


Figure a: Incorrect

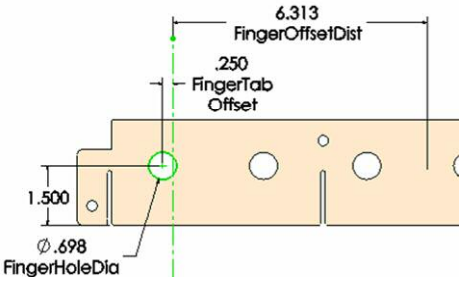


Figure b: Recommended

Figure 4.5: Naming dimensions in sketches

Naming the features: Naming the features with meaningful names helps in locating the feature in the construction history within larger components. This also helps in capturing the right feature during the

capturing (Stage III) and in fixing the bugs within program during debugging (Stage V). Figure 4.6 shows the incorrect and recommended ways of modeling.

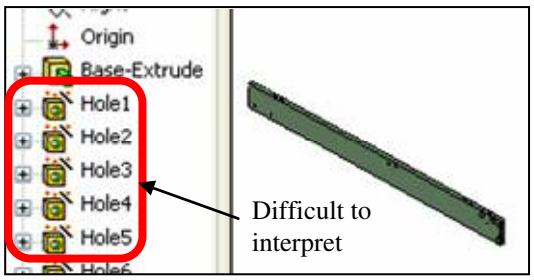


Figure a: Incorrect

No naming convention for features

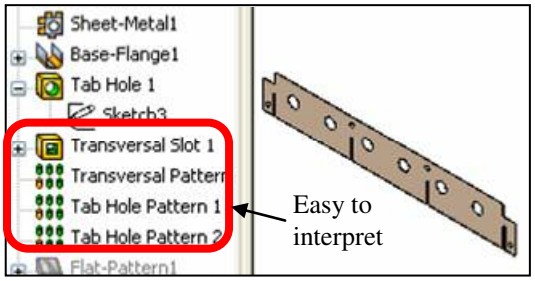


Figure b: Recommended

Naming convention for features

Figure 4.6: Assigning meaningful names to features

Use of dummy references: Dummy reference lines help in referencing entities that are not present in the models. This helps in simulating other component's reference features in the current model as shown in Figure 4.7. It also helps in building uniform rules among different components by avoiding unnecessary constants and variables in DW, while holding them back in SW sketches. This way the parameters that do not change remain untouched in SW CAD models. Dummy references help programmers to use similar rules for different components while preserving the design intent. The advantages of creating dummy references are explained below.

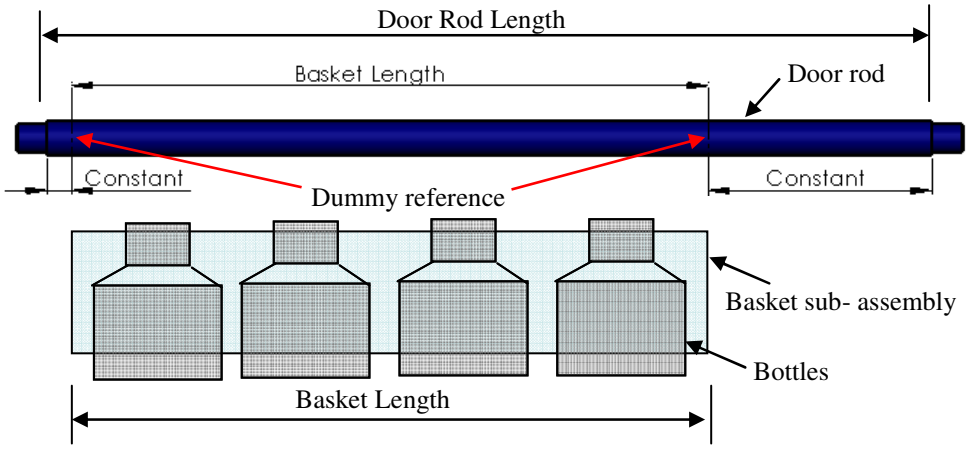


Figure 4.7: Using dummy references for preserving design intent

If the dummy references are not used, the length of the door rod is given by the Equation 4.1 and a rule should be written in DW by using a constant. On the other hand, if dummy references are used while creating the door rod in SW sketch, the *"basket length"* dimension can be created to mimic the basket sub-assembly, which in turn drives the *"door rod length"* as seen in Figure 4.7. The same rule that drives the basket sub assembly for calculating the *"basket length"* can be applied to calculate the length of the door rod, which results in less number of rules to be written in DW.

$$Door\ Rod\ Length = Basket\ Length + const.\ on\ either\ side \qquad \textbf{Equation 4.1}$$

Combining non variant features: Entities such as chamfers, rounds, and tool clearances that do not change with specifications should be modeled within the sketch itself. This guidelineis applicable for laser and conventional machining types of manufacturing processes. This is not applicable for components that are produced on a CNC machine in which the corresponding code is generated automatically using CAM software such as MasterCam or Pro CNC. This method of creating CAD models is shown in Figure 4.8.

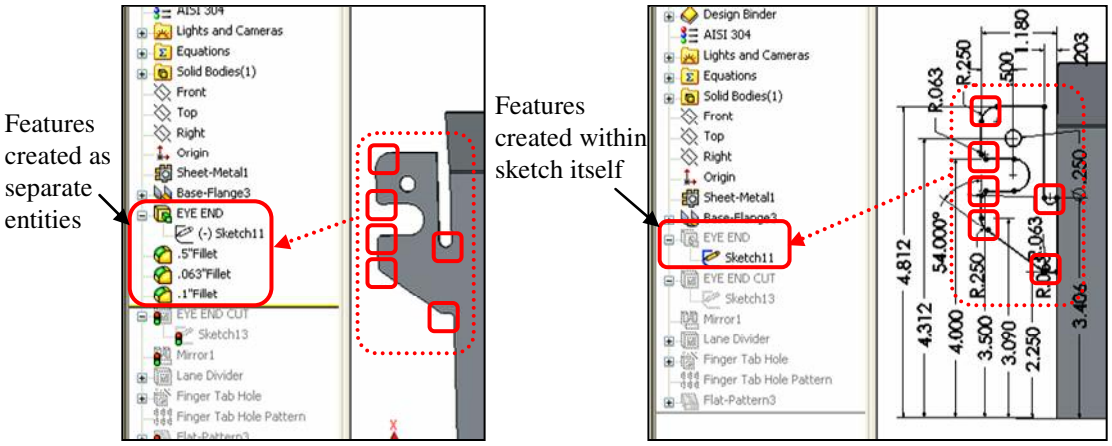


Figure a: Incorrect **Figure b: Recommended**

Figure 4.8: Combining non varying parameters into sketch

Process for extracting parameters using DW

The DW RBD programming process starts with the creation of a new group in the DW Administrator. The design of the assemblies, which have to be automated are added to this group as projects. In DW terminology a project is defined as a collection of items that include CAD models, user

forms, documents, data and rules [12]. . The DW model list only recognizes the assemblies that are added to the projects. The DW model list can be found in the DW toolbar within the SW menu and is used for extracting SW parameters.

The process of extracting parameters consists of six steps as shown in Figure 4.9. The process begins with selecting the assemblies and components that need to be controlled. To do this, the programmer should know the file names of the CAD models that need to be controlled. This information can be obtained from the design document. The time required for selecting the components (T_{EP1}) in the DW model wizard is negligible when compared with the other steps in this stage, and therefore not explicitly considered in the time calculations.

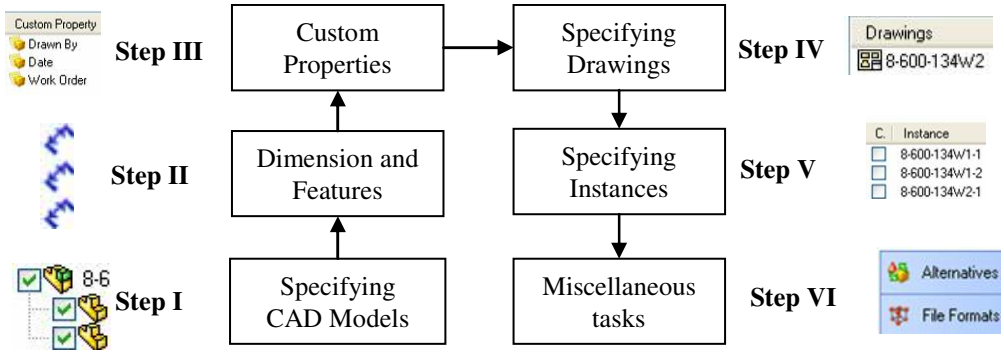


Figure 4.9: Steps in extracting parameters for DW programming

The next step in this process is extracting the actual parameters that vary with design specifications. A parameter is defined as entity that needs control, and it includes dimensions, features, and custom properties. Extracting the parameters is an important step, since most of the rules are written for them in the DW Administrator. The previously presented modeling guidelines help in quickly identifying the features and dimensions that need to be controlled. Extraction process is done by selecting the parameters (dimensions and features) in SW CAD models and by assigning recognizable names to them. For example, in Figure 4.10 *"D1@LaneDividerPattern@8-600-478.Part"* is named as *"LaneDividerPatternNum"*.

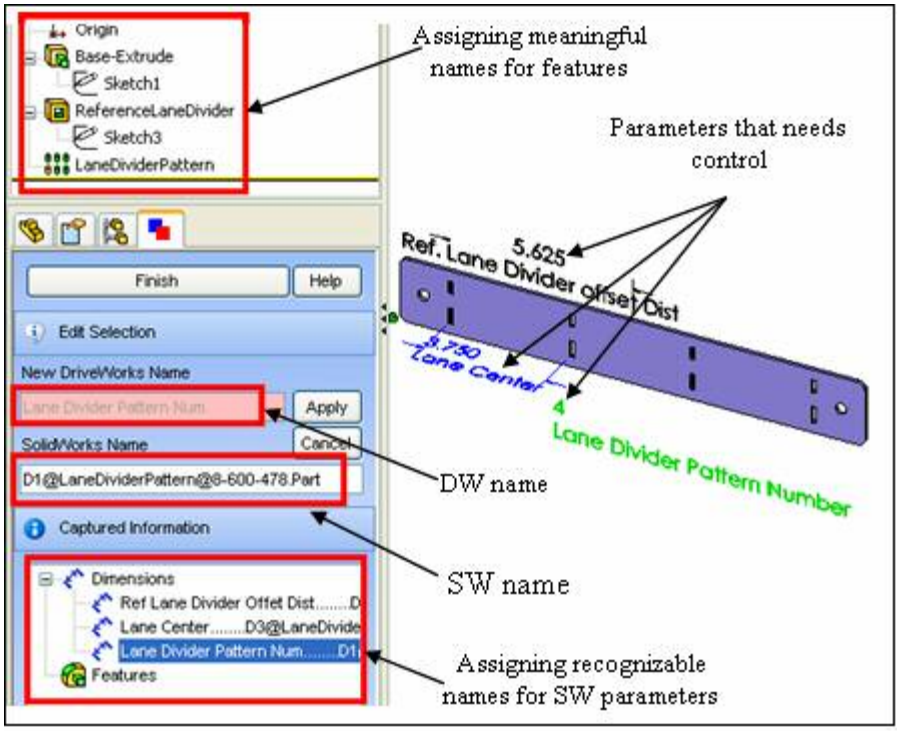


Figure 4.10: SW screen shot in the process of extracting parameters

In the HI IRBD project, the times for extracting various parameters (dimension 'd' and features 'f') are recorded and are shown in Table 4.1. Regression analysis is done on these recorded times, using MS Office Excel.

Table 4.1: Recorded times for extracting dimensions and features

Observation	Dimensions (d)	Features (f)	Total parameters (d+f)	Time (minutes)
1	3	3	6	9
2	3	3	6	4
3	4	1	5	4
4	4	1	5	2
5	2	3	5	4
6	1	1	2	2
7	2	1	3	2
8	3	1	4	1
9	4	0	4	4
10	1	0	1	1
11	15	15	30	12
12	15	15	30	16
13	15	15	30	15
14	10	4	14	6
15	10	4	14	4
16	0	4	4	1
17	3	0	3	3
18	3	0	3	1

It is observed that the time required for extracting parameters (T_{EP2}), is directly proportional to the number of dimensions and features that require control and can be seen from Figure 4.11.

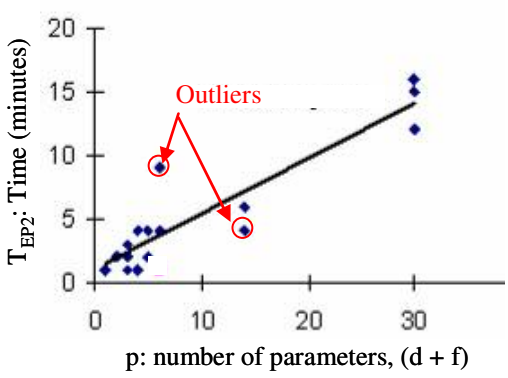


Figure 4.11: Linear plot between parameters and time

From the Figure 4.11, it can be seen that not many outliers are present except for the observation 1 (parameters =6, time =9) and observation 15 (parameters = 14, time =4). The presence of these outliers depicts that other activities like fixing the CAD models are involved during the process of extracting dimensions or features. These outliers can be eliminated if the CAD modelers use the best practices that are presented before while creating the CAD models. Further, the relation between parameters 'p' (which include dimensions 'd', and features 'f') and, time 't' in minutes is given by the Equation 4.2.

$$T_{EP2} = 0.5 * p + 1.0 \Rightarrow 0.5\,(d+f)+1.0 \qquad \textit{Equation 4.2}$$

Where,

T_{EP2} = time for extracting parameters (mins),
p = number of parameters = (d+f)
d = number of dimensions
f = number of features

Close observation of Equation 4.2 reveals that it takes half of a minute to extract one dimension or feature into DW. Another observation is that the y-intercept is 1. 0 (non-zero value) suggesting that it at least takes 1 minute to go from step I (selecting models) to step III (selecting drawings to the models).

The next step in this stage is to specify the custom properties for the components and assemblies. For the grid products, an average of four custom properties are used for components and three custom properties are used for assemblies. Custom properties include a) Work Order, b) Drawn by, c) Date, d) Num1, and e) Num2. In the HI IRBD project, the relation for finding the time required for specifying the custom properties can be obtained from performing the linear regression analysis on the recorded times that are shown in Table 4.2.

Table 4.2: Recorded times while specifying custom properties

Observation	Custom properties	Time (minutes*)
1	3	1
2	5	1
3	3	1
4	5	1
5	3	1
6	5	1
7	5	1
8	27	5
*Rounded off nearest 1 minutes		

The linear plot is between the number of custom properties and the time in minutes is shown in Figure 4.12.

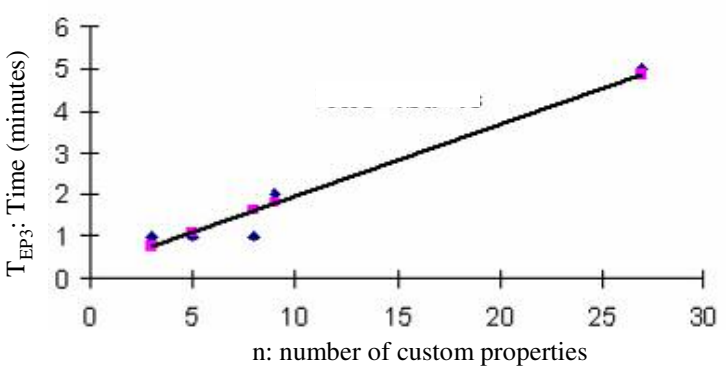


Figure 4.12: Linear plot between custom properties and time

The time required for specifying the custom properties (T_{EP3}) is given as:

$$T_{EP3} = 0.2 * n + 0.3 \Rightarrow 0.2\,(4P+3A) + 0.3 \qquad \textit{Equation 4.3}$$

From Equation 4.3, it can be seen that it takes on average about 0. 2 minutes (12 seconds) to specify one custom property. The y-intercept suggests that, it takes on average about 0. 3 minutes (18 seconds) to move from Step II (extracting dimensions) to Step IV (specifying drawings) without specifying any custom properties.

The next step in this stage is specifying drawings for driven parts and assemblies. In general, drawings need to be created for every model or assembly that is driven by DW. The total number of

drawings ‘D’ is equal to the sum of assemblies ‘A’ and parts ‘P’ that need to be controlled. The time taken for specifying drawings is obtained by doing regression analysis on the recorded observations, which are shown in Table 4.3.

Table 4.3: Recorded times while specifying drawings

Observation	No of Drawings (D=A+P)	Time (minutes*)
1	2	1
2	5	2
3	8	3
4	1	1
5	2	1
6	2	1
7	2	1
8	2	1
9	4	2
*Rounded off to next minute		

The linear plot between number of drawings ‘D’ and the time for specifying drawings (T_{EP4}) is shown in Figure 4.13.

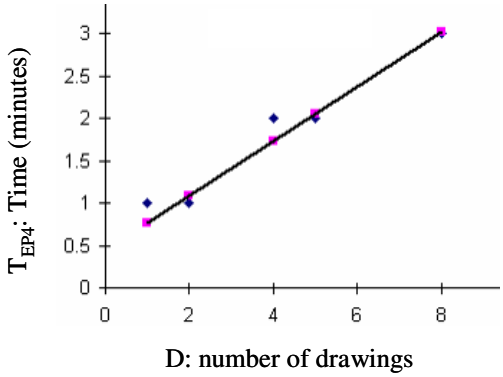


Figure 4.13: Linear plot between drawings and time

The time for specifying the drawings in DW model wizard is given by Equation 4.4 and is valid only for adding drawings that do not require any editing.

$$D = A + P$$
$$T_{EP4} = 0.35 * (A + P) + 0.45 \qquad \textit{Equation 4.4}$$

Where,

T_{EP4} = Time for specifying drawings (mins)

D = number of drawings = (A + P)

A = number of assemblies

P = number of parts

From Equation 4.4, it can be seen that it takes 0. 35 minutes (21 seconds) to specify a drawing for the model. The activities involved during the process are selecting the model for which drawing need to be added and browsing through folder to select this particular drawing. The y –intercept value suggests that it will take at least 0.45 minute (27 sec) to move from step III (specifying custom properties) to step V (specifying instances) without adding drawings.

The next step in this stage is to control the instances of the sub assemblies and parts. An instance in CAD modeling is defined as the occurrence of any entity (feature, part, or assembly) that refers to the base entity. Multiple instances of an entity means that the entity has been loaded into memory more than once. Usually nuts, bolts, and general hardware that are used more than once in an assembly are shown as instances. Figure 4.14 shows the instances of a bolt that is frequently used in 2800 PFinger Elevator Metric type of grid product. The time required to capture the instances would be negligible when compared with capturing dimensions or features in step II.

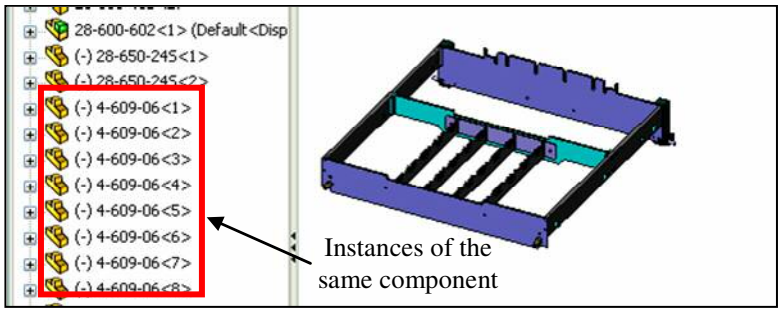


Figure 4.14 Model Tree showing instances of a component

There are some other additional tasks such as specifying alternative file and specifying output documents. These tasks are not generally required for grid products and depend upon the situations like using one component in place of other. Based on the experience from HI IRBD project, one can say that

the programmers will take approximately 15 – 45 minute for selecting components and assemblies (Step I), specifying instances (Step V), and specifying alternative files (Step VI). The time for these additional tasks can be considered as constant C_2 and depends upon the number of parts a grid assembly has. The below values are only valid for HI grid products and may not be suitable for other products.

$$\begin{aligned} C_2 &= 15 \text{ mins for assemblies that have components} < 20 \\ &= 30 \text{ mins for assemblies that have components in range of 20-50} \\ &= 45 \text{ mins for assemblies that have components} > 50 \end{aligned} \qquad \textit{Equation 4.5}$$

In every DW RBD program, the Step 1 of selecting assemblies and components will be present regardless the activities of specifying the instances or alternative files.

Total time calculations for extracting parameters

The total time for extracting parameters is obtained by adding the time taken for individual steps. The summation of Equation 4.2, Equation 4.3, Equation 4.4, and Equation 4.5 gives the estimated time for Stage III of RBD programming process.

$$T_{EP} = T_{EP1}+T_{EP2}+T_{EP3}+T_{EP4}+C_2$$
$$T_{EP} = 0.5 *(d+f)+1.0 + 0.2\ (4P+3A)+0.3 +0.35\ (A+P)+0.45+C_2$$
$$(\because T_{EP1} = 0, \text{ negligible when compared with other phases})$$

The result of summation is called time for extracting parameters T_{EP}, and can be obtained as shown in Equation 4.6 and holds good for HI grid products.

$$T_{EP}=0.95A+1.15P+0.5(d+f)+1.75+C_2 \qquad \textit{Equation 4.6}$$

Where

T_{EP} = time for extracting the parameters (mins),
A = number of assemblies,
P = number of parts,
d = number of dimension,
f = number of features, and
C_2 = extracting time constant

Table 4.4, shows the time calculations for extracting parameters (Stage III) of the RBD programming process for the products that were selected in the previous chapter.

Table 4.4: Total time calculations for extracting parameters

Product	A	P	D	F	C_1	T_{EP} (mins)
2800 HRHM	14	40	155	89	30	211. 4
2800 PFEM	5	26	41	12	30	91. 25
2800 PFEMNB	4	15	20	6	30	64. 15

Chapter Summary

This chapter concentrates on extracting the model parameters that require DW control. It presents a major section on best practices in CAD modeling that helps in extracting parameters. In depth discussion on activities related to Stage III of RBD programming is presented in this chapter. It also presents equations for predicting time estimations for extracting parameters such as assemblies, parts, dimensions, features, and custom properties. The equations are the results of performing linear regression on previously recorded times in extracting parameters for grid products as part of the HI IRBD project.

CHAPTER 5

DRIVEWORKS PROGRAMMING

The previous chapters (Chapter 3 and Chapter 4) discussed gathering the design knowledge and extracting the parameters that need control. This chapter discusses the Stage IV of programming process in the DW Administrator as shown in Figure 5.1.

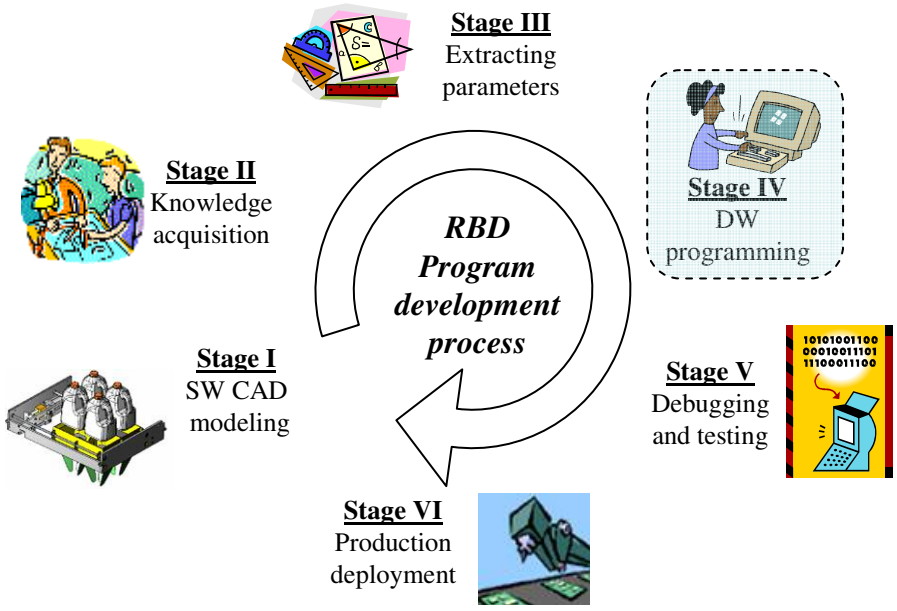


Figure 5.1: DW programming stage in RBD program development process

The DW programming process starts with the creation of a new group in the DW Administrator. The assemblies that are to be controlled are added to these groups as projects and an interface is created between DW and SW. Projects consist of CAD models, user forms, data, rules, and documents. CAD models include assemblies, components, and drawings that change with a given design specification. User forms are the user interfaces by which a user interacts with a particular DW program. Data is the background design information that used for building rules. Rules are the equations that control the model parameters such as dimension or features. Documents may include spreadsheets or word documents that need to be prepared for every new design. They may be quotations, bill of materials, or user manuals.

DW Programming process

The DW programming process consists of three major steps as shown in Figure 5.2. They include creation of a user interface, specifying data and tables that are required for programming, and finally building model rules for the previously extracted parameters. The more detailed discussion and time calculation for these steps are presented in the following sections.

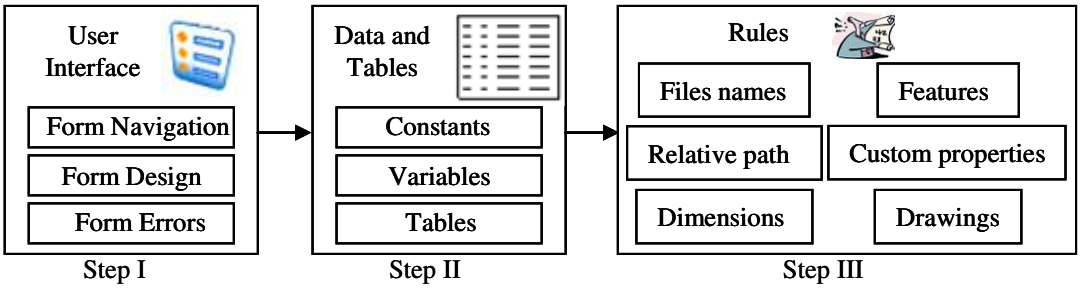


Figure 5.2: Phases in programming in DW Administrator

User Interface

The first step in DW programming is to create a user interface. A user interface acts as a link between the DW program and the user. The steps in building a user interface are given below:

Identify the people who are going to use this RBD system: The most important aspect in building a user interface is identifying the user who is going to use this system. Marketing personnel may use values (in 'mm') that are different from the values used by the design engineer (in 'inches'), even though both of them are referring the same thing. Therefore, it is important to know and use the same technical jargon (terminology, units, and references) on the user interface. If it is required, provide some examples or brief definitions of the terms, which help the users to understand the terms used on interface. For complex information, it is a good idea to explain by showing a picture. Figure 5.3 shows an example of explaining the term *"basket holding holes"*. It also provides the maximum and minimum values that are permitted so that the user can input the value in that range. It also shows a particular note that all dimensions are in inches. This way it instructs the users (designer or marketing personnel) to input the correct values.

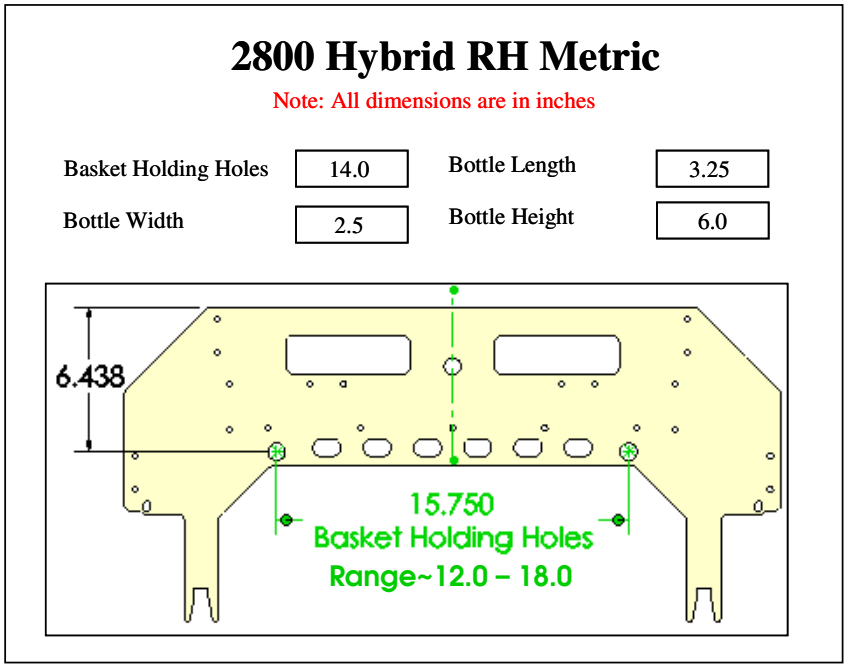


Figure 5.3: Use of pictures for explaining the terms

Make use of existing interfaces: As far as possible, try to replicate the new interface with the existing interface. This method is effective for lower level details of a user-interface such as button controls or menu names [29]. An existing layout will be easier and quicker to implement than building a new interface because many design decisions have already been made for the existing interface. In addition, the use of an existing interface makes it easy and comfortable for users to learn and use since the users are somewhat familiar with the design. Whether to use an existing interface or building a new interface depends upon how often the users will be using the new interfaces when compared to how often they will be using the existing interfaces. Users want to have similar interfaces that they know. If the situation demands a new interface, suitable training should be provided to the users for making them comfortable with new interfaces.

Prepare a prototype: The prototype of intended fields and layout of the user interface should be made on paper. Unnecessary fields which are not used anywhere in the design process should be avoided. A rough description of the user interface should be shown to the concerned person for the getting the format approval. This way the programmer can get a green signal going ahead and building a concrete

system or can get some suggestions for making improvements to the existing layout. At this stage, the user interface may not reflect the entire design but should reflect exactly what fields should be present and their location.

Actual building and testing the user interface: After building the user interface based on the approved format, it should be thoroughly tested for hidden errors. The purpose of testing is not to prove the interface but to improve the interface [29]. Improvement should be made by using additional controls such as pull down menus or option groups instead of using text boxes. Text boxes give the freedom of inputting any values, which are hard to control and should be avoided. Necessary error messages should be incorporated into the user interfaces to avoid taking wrong values such as taking alphabets in place of numerical or taking out of range values.

The programmer should consider the above factors in designing a user interface. In addition, user interface should be pleasing to look at. The succeeding section deals the principles to make professional user interfaces.

Graphic design principles

The programmer has to make lot of decisions to prepare a user interface such as the layout of the screen, where to put things on the screen, the size and font to be used, and what colors to be used. The following are the few graphic design principles, which not only help the programmer in building an attractive user interface but also make the user more comfortable in using the interface [29].

Clustering principle: Group similar fields under one heading and split the screen in separate blocks. Try to group similar fields under one heading, which makes the user interface more consistent. If necessary, differentiate fields by using colors or a bounding box. Another way of differentiating fields is by using font modifications such as bold or italics formatting and by using Times New Roman or Arial font. The application of the clustering principle is shown in Figure 5.4.

Visibility principle: Frequently used fields should be obvious and distinct. On the other hand, hide or grey out the controls that give suggestions. In Figure 5.4, the visibility principle is applied to suggest the dimensions of case information such as width, length, and height. This principle is again applied for "Flap Height" and "Partition Height". When the "Does Case Have Flaps? If Yes, Check box" field is ticked, the "Flap Height" field is shown; otherwise it is hidden. Similarly the visibility control of "Partition Height" depends upon the "Does Case have Partitions? If Yes, Check box" field.

Alignment Principle: All the fields should be aligned to one reference. This makes the user interface more pleasing to look at. This alignment principle is applied to many fields in the Figure 5.4. For example, the "Packer Information" block and the "Case Information" are aligned together.

Color as supplement principle: Assign separate colors for fields that need emphasis or that need the user's attention. For warning messages or informational messages, assign different colors for making them different from the normal fields. A simple rule is to use black and white colors for the form fields. Using minimal colors is a best principle for producing attractive interfaces. Figure 5.4 shows an informational message in red color that attracts the user's attention.

Reduced clutter principle: Do not fill the entire screen with too many fields. If necessary, create a new form and link the two forms together. Same font and styles should be used consistently through out the form. Do not use new styles and fonts for new blocks or many font sizes within/across the blocks. The reduced clutter principle was used in designing the form that is shown in Figure 5.4.

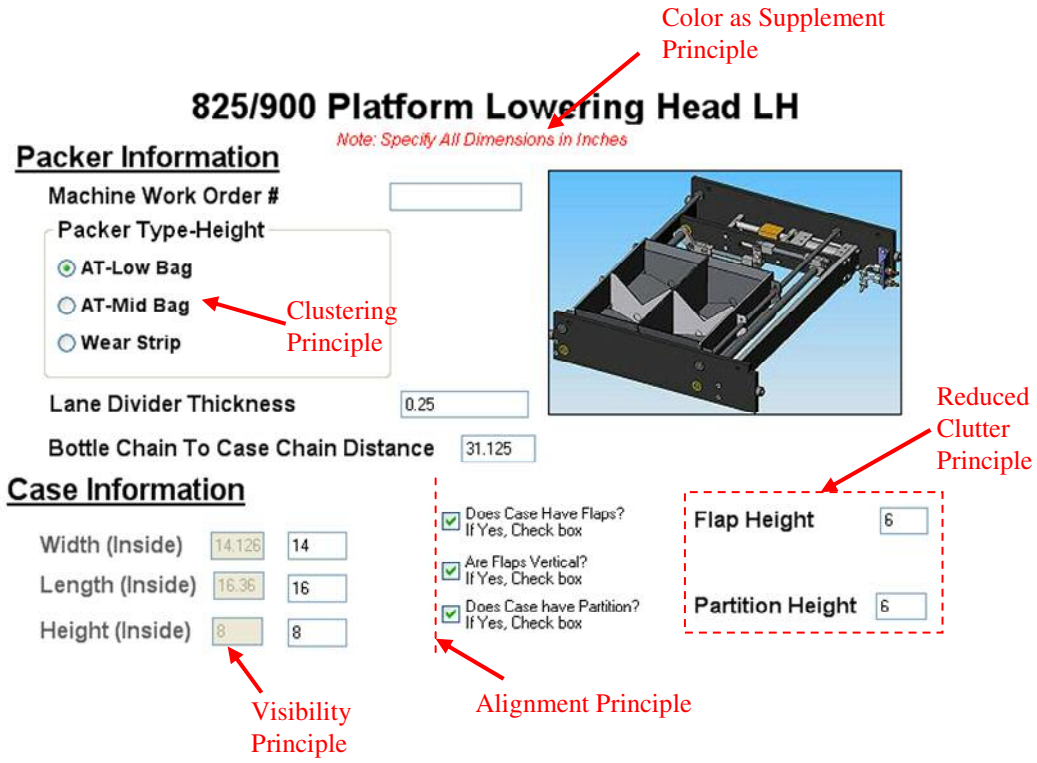


Figure 5.4: Graphic design principles

Time calculations for building a user interface

Building a user interface in DW Administrator consists of: i) Form navigation, ii) Form Design, and iii) Form Errors. Form navigation comes into picture if you have more than one form. It also plays a vital role in deciding which form to use next. Typical form navigation is shown in Figure 5.5. In this, three main forms are used (Packer Input, 825 Form, and 2800 Form) and, based on decisions after the Packer Input, 825 Form or 2800 Form will appear next to the user. The time taken for form creation and form navigation would be less than the other activities like form design or form errors and thus can be neglected.

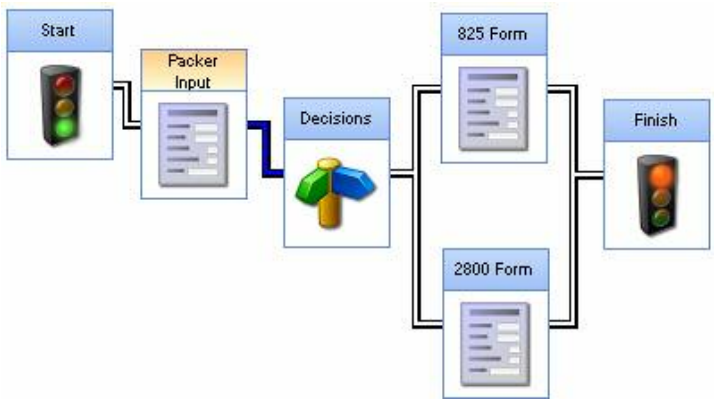


Figure 5.5: Form navigation

The next step after specifying the forms is to design the forms. All the information that is necessary for designing a product should be present on the forms. This information may include design specifications, design suggestions, error or warning messages, and pictures. Many controls are available such as radio buttons, list groups, drop down menus, options button, spin buttons, and text boxes, which help in stopping the user from entering wrong values. Additionally, program specific error messages or warning messages can be added to the form fields. A typical form is shown in Figure 5.4. The time taken for building the forms depends upon the number of fields on the form. The recorded times for building the forms in HI IRBD project are presented in Table 5. 1.

Table 5.1: Time taken for building forms

Sl. no	Name of the form	NFields	Time(min)
1	825 Platform Low RH Form 1	24	55
2	825 Platform Low RH Form 2	29	60
3	2800 Hybrid RH Metric Form 1	13	25
4	2800 Hybrid RH Metric Form 2	45	75
5	2800 Hybrid RH Metric Form 3	7	10
6	2800 Pfinger Elev Metric Form 1	22	30
7	2800 Pfinger Elev Metric Form 2	25	40
	Average number of fields for grid forms	24	43

The regression plot between the time taken for building the forms (T_{Forms}) and the number of fields (N_{Fields}) is shown in Figure 5. 1.

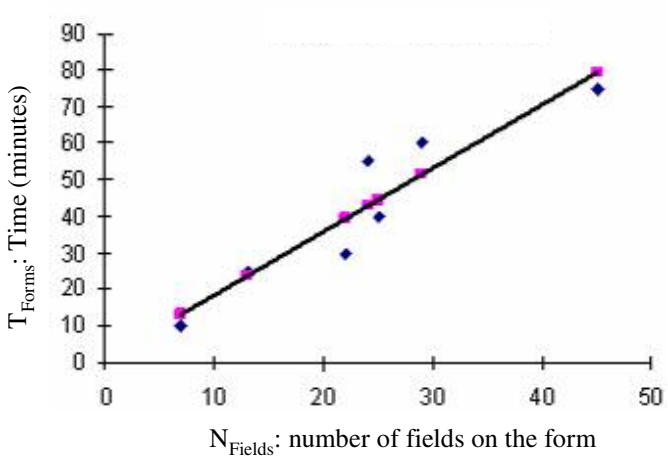


Figure 5.6: Linear plot between number of fields and time taken

The relation between the time (T_{Forms} in minutes) taken and the number of fields (N_{Fields}) is given by Equation 5. 1and is obtained from regression analysis.

$$T_{Form} = 1.75 * N_{Fields} + 1.0 \qquad \textit{Equation 5.1}$$

This equation reveals that it takes on average 1.75 minutes to create a new field. Another observation is that it takes about one minute to specify the forms as the part of form navigation without creating any fields on it.

The next step in the process of building a user interface is to build the mechanisms that will alert the user should they enter wrong values. DW allows adding error messages tied to the form fields. This way DW stops the user from specifying wrong specifications. There is a provision to add error messages for the fields in the form of rules. Typically, this process is done by setting the maximum and minimum values for the fields. The error rule is invoked when the entered value lies outside the maximum and minimum value. The following example shows the application of error rule.

```
IF (OR(Bottle_Width_Return<2 ,Bottle_Width_Return>6)
THEN "Entered value is out of range, please enter value in between 2 and 6"
ELSE "Do nothing"
```

Equation 5.2

Therefore, if the Bottle_Width_Return is in between 2 and 6 inclusive, the program will work fine by releasing specifications. Otherwise it pops up the error message that the value is out of range and

suggests that the new value should be entered between 2 and 6. The time taken for building the error rule depends upon the number of order specifications. The number of order specifications will be approximately equal to form fields. It can be seen from the HI case study that, it approximately takes 1 minute for defining the error rule and another 1 minute for specifying the error rule in DW. Therefore, it approximately takes 2 minutes for specifying the error rule for one field.

$$T_{Error} = 2 * N_{Fields} \qquad \textit{Equation 5.3}$$

The summation of Equation 5. 2 and Equation 5.3 gives the time required for creating a user interface as shown in Equation 5.4 and the time required for creating a user interface ($T_{UserInterface}$) depends upon the number of fields on the form (N_{Fields}). The number of NFields can be taken as the number of design variables (DV), since they change with every new specification of product.

$$T_{UserInterface} = T_{Forms} + T_{Errors}$$

$T_{UserInterface} = 3.75 * N_{Fields} + 1.0$
Replacing N_{Fields} with design variables (DV);
$T_{UserInterface} = 3.75 * DV + 1.0$

$$T_{UserInterface} = 3.75 * DV + 1.0 \qquad \textit{Equation 5.4}$$

The average number of form fields used in HI grid project forms from Table 5. 1 is 24 fields for one form. Since on average two forms were used for the RBD programs, the total number of forms fields can be taken as 24 x 2 = 48. Therefore substituting the value of 48 for design variables 'DV' in Equation 5.4, the time taken on average for creating a user interface can be obtained as 181 minutes for HI grid products.

Data and Tables

Data and tables play an important role in building rules. This information is obtained from the company's legacy documents and from previous designs. The three different types of information that fall into the category of data and tables are constants, variables, and look-up tables.

Constants are the fixed values that are used in writing rules. Frequently used numbers, clearances, and machine dependent values can be declared as constants in DW. Constants help a programmer in better understanding a rule that is already written by another programmer during the process of debugging or upgrading the program. If the same value is used in many rules and needs to be modified later, the programmer has to manually change all the rules with this value. It is time consuming and error prone to change all the rules. On the other hand, the value can be declared as a constant and all the rules can be written by using this constant. Changing the constant with a new value will automatically reflect the change in all the rules. In general, the constants are declared while writing the rules for the variables. Therefore, the time calculations for declaring constants are addressed in the time calculations of variables.

Variables are the sub rules that are frequently used in writing rules for parameter that are extracted in stage III of RBD program development. Constants and the form fields (design variable) are used in creating variables. If the same rule is to be used in different components, the rule should be assigned to a variable and the variable should be used instead of the rule. This way, changing the rule for the variable changes the rule for all components. This method is shown as an example in the following sections.

The following figures show three different components: 28-600-48S (Figure 5.7), 28-600-474 (Figure 5.8), and 8-600-478 (Figure 5.9). In each of the components, the distance between adjacent slots need to be controlled and the distance between the slots remain the same for all the three components. The distance in HI grid terminology is called the "Lane Center." Instead of writing individual rules for 'Lane Center' in each component, a variable by name Lane_Center can be declared and can be used in all the three components. This way the program remains consistent, and changing the rule for the variable changes the "Lane Center" rule for all the components. The rule for the Lane_Center variable is given in

Equation 5 5 and changing the rule for this variable changes all the rules in Equation 5.6, Equation 5.7, and Equation 5.8.

$$\text{Lane\_Center} = \text{Bottle\_Width} + \text{Bottle\_Lane\_Clearance} + \text{Lane\_Divider\_Thickness} \quad \textit{Equation 5 5}$$

$$\text{Lane Center}_{(28\text{-}600\text{-}48S)} = \text{Lane\_Center} \quad \textit{Equation 5.6}$$

$$\text{Lane Center}_{(28\text{-}600\text{-}474)} = \text{Lane\_Center} \quad \textit{Equation 5.7}$$

$$\text{Lane Center}_{(8\text{-}600\text{-}478)} = \text{Lane\_Center} \quad \textit{Equation 5.8}$$

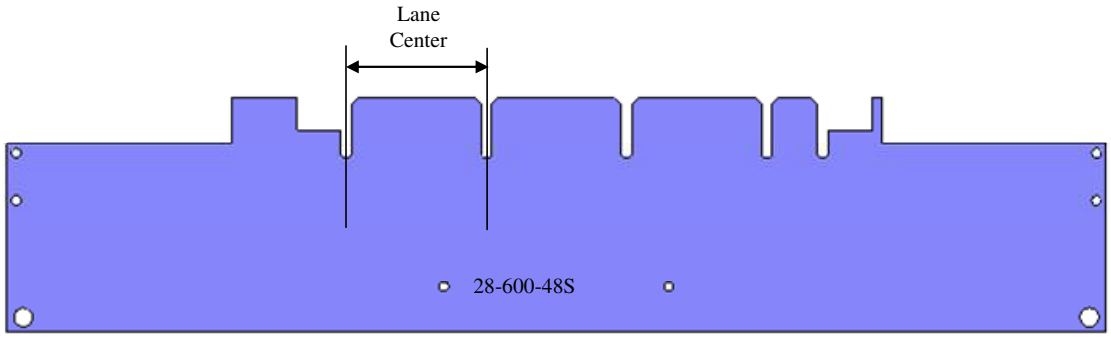


Figure 5.7: Dimension to be controlled in 28-600-48S

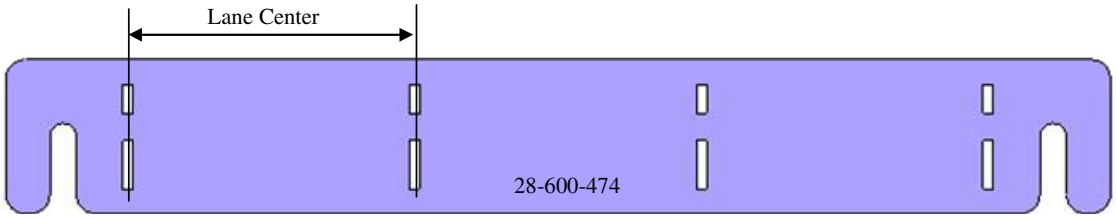


Figure 5.8: Dimension to be controlled in 28-600-474

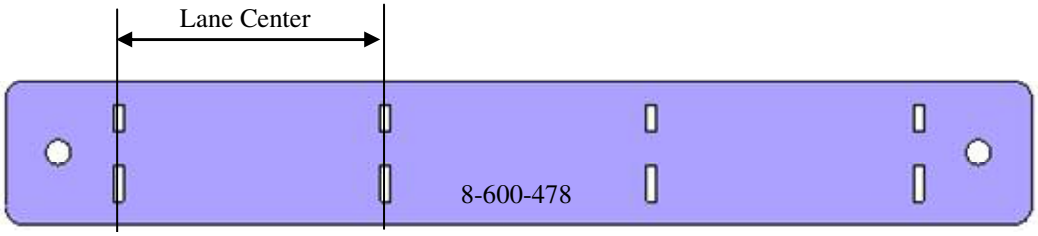


Figure 5.9: Dimension to be controlled in 8-600-478

Look-up tables contain the standard information that cannot be represented by rules. The information like bolt sizes, number of bolts, finger widths, and other company related databases can be represented in the form of look-up tables to write rules in DW Administrator. The example for look-up table used in HI grids is shown in Table 5.2. Look up tables are the easy way of presenting random information that is difficult to represent in the form of rules.

Table 5.2: Look up table for Finger_Width in HI Grids

Sl. No	Bottle_Width	Finger_Width
1	1	0. 97
2	2	0. 97
3	2. 5	1. 47
4	3	1. 97
5	3. 5	2. 47
6	4	2. 97
7	5	3. 97
8	6	4. 97
9	7	6

In calculating the time required for creating data and tables, declaring variables plays a major role. During declaring variables, constants and look-up tables are also created. Therefore, the time calculation of variables takes care of constants and look-up tables. In general, it is difficult to say how many constants, variables, or look-up tables a program will contain without having the design document. The average number of variables can be obtained as "17" from the Table 5.3, which shows the number of constants, variables, and data tables used in various RBD programs of HI grid products.

Table 5.3: Metrics used in HI IRBD project

Sl. No	Name of the product	Constants	Variables	Tables
1	825 Platform Low RH	18	21	4
2	2800 Hybrid LH Inch	4	21	1
3	2800 PFinger Elevator AT	5	15	1
4	825 Laser Platform Low RH	8	22	1
5	825 Platform Elevator LH	11	6	0
	Average numbers used in IRBD	10	17	2

The recorded times for declaring the variables is presented in Table 5.4 and based on this information a regression plot between the number of variables (PV) and the time for declaring them (T_{PV}) can be obtained as shown in Figure 5.10.

Table 5.4: Variables used in HI IRBD project

Sl. No	Name of the product	Variables	Time in 'min'
1	825 Platform Low RH	21	48
2	2800 Hybrid LH Inch	21	55
3	2800 PFinger Elevator AT	15	42
4	825 Laser Platform Low RH	22	52
5	825 Platform Elevator LH	6	14
	Avg for grid products	17	

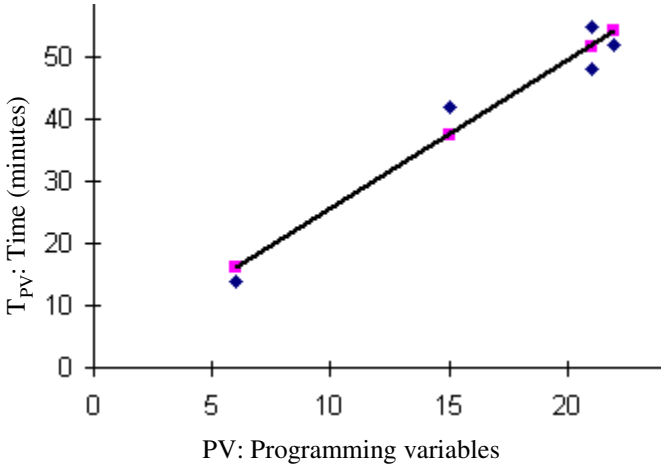


Figure 5.10: Linear plot between programming variables and time

The relation between the variables (PV), programming variables since they are used in program, and the time for creating them (T_{PV}) can be related by using a relation as shown in Equation 5.9 Here T_{PV} is equal to $T_{Data\&Tables}$, because in the HI IRBD project, the constants and tables are created during the process of creating the programming variables.

$$T_{Data\&Tables} = 2.5 * PV + 2.0 \qquad \textit{Equation 5.9}$$

The above equation reveals that it takes on average two and a half minutes to declare variables and miscellaneous activities like browsing through the data and tables section, which consumes on average

about 2 minutes. The exact number of programming variables should be known for using Equation 5.9 to calculate the estimated time, which is not know until the preparation of design document. Since estimation of time for RBD program is done before preparing the design document, the average number of variables used in previous RBD programs can be considered for time calculation of HI grid product.

From the Table 5.4, on average 17 variables are used in HI RBD programs. Therefore, plugging the value of 17 into Equation 5.9, the estimated time for writing rules for variables in DW can be obtained as 45 minutes and is applicable only for HI grid products:

$$T_{Data\&Tables} = 2.5 * PV + 2.0$$
$$\Rightarrow 2.5 * 17 + 2.0 = 44.5 \approx 45 \text{ mins} \qquad \textit{Equation 5.10}$$

Model Rules

Writing rules is an intellectual task and depends upon the problem solving skills of the designer and it differs from programmer to programmer. One programmer may use an equation for building the rules and the other may use look-up tables. It all depends upon the experience and problem solving skills of the programmer. The following sections describe this nature with an example.

In the component that is shown in Figure 5.11, the first finger hole needs to be placed at the center when one bottle is present, a half bottle width away from the center when two bottles are present, and one bottle width away when three bottles are present and so on. Let us call this distance as the first finger offset distance (FFOD). The FFOD can be obtained by using an algebraic equation or by using a look-up table. Equation 5.11, shows the use of an algebraic expression for obtaining for obtaining the FFOD. This equation consists of two parts where, "Part I" is the controlling factor and "Part II" is simply a constant distance. This controlling factor can be obtained by using a look-up as shown in Table 5.5.

$$FFOD = \underbrace{(NOB-1)/2}_{\text{Part I}} \underbrace{* (BW + C)}_{\text{Part II}} \qquad \textit{Equation 5.11}$$

where,

FFOD = First Finger Offset Distance

NOB = Number of bottles

BW = Bottle Width (from user forms)

C = Clearance

$$FFOD = \underbrace{\left(\begin{array}{c}\text{Look in Table 5.5 for NOB}\\ \text{for the corresponding value}\\ \text{from last column}\end{array}\right)}_{\text{Part I}} \underbrace{* (BW + C)}_{\text{Part II}} \qquad \textit{Equation 5.12}$$

Table 5.5 Look up table for finding the first part of equation

Sl No	NOB	Part I
1	1	0
2	2	0. 5
3	3	1
4	4	1. 5
5	5	2

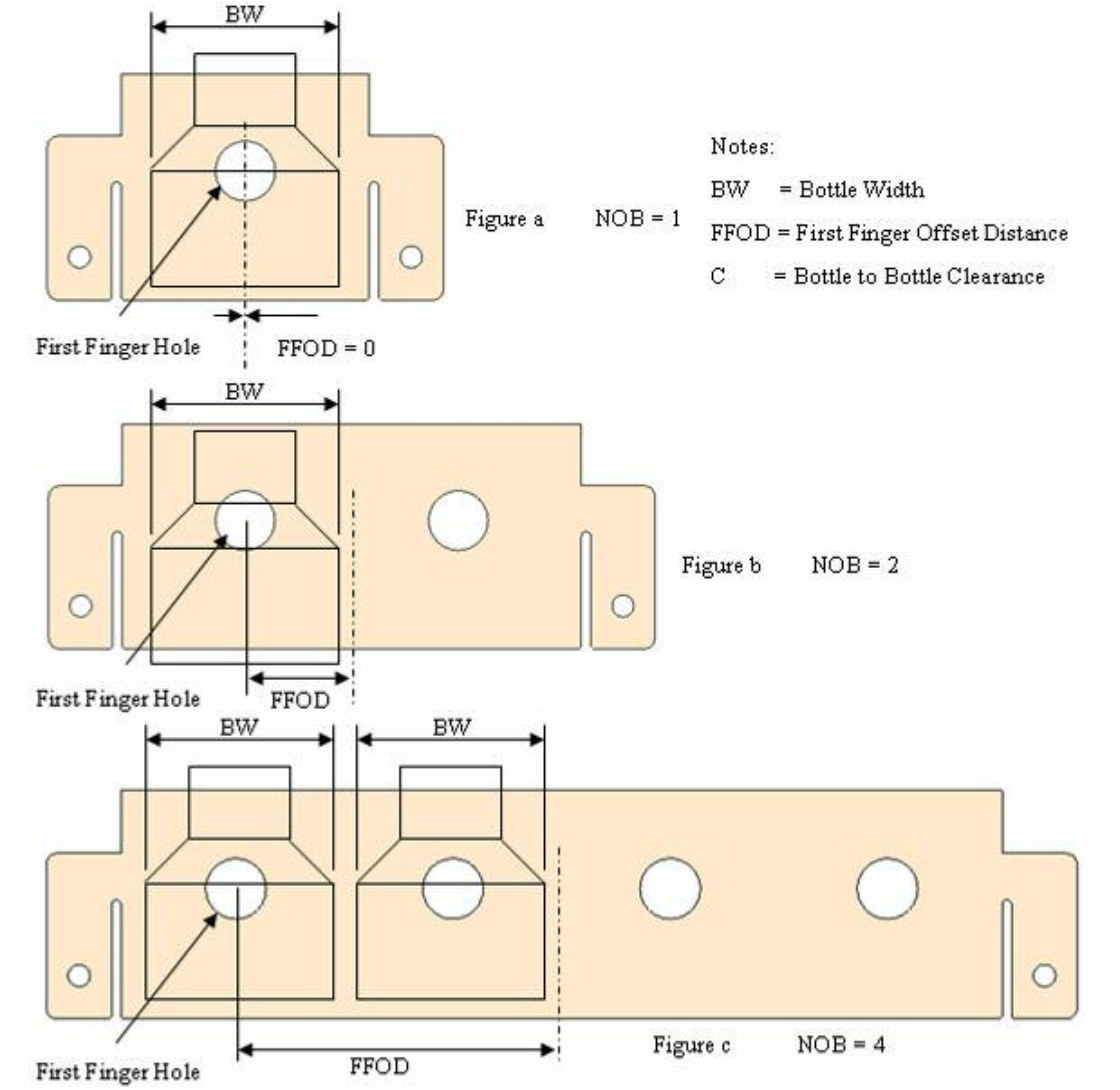


Figure 5.11: Comparison between use of look-up tables and algebraic expression

The time taken for writing these rules in the DW Administrator will depend upon the above factors and from person to person. Writing rules is a two-step process. First, the programmer has to open the locations where constants, variables, and forms are stored. The second step is to locate the required value from the group of values for writing equations or IF-THEN rules. These two steps consume time for writing rules. The time for writing the rules in the DW Administrator are recorded in the HI IRBD project and are shown in Table 5.6.

Table 5.6: Recorded times while writing rules in DW Administrator

Observations	Dimension and Features (d+f)	Time (minutes)
1	15	210
2	103	210
3	1	90
4	3	4
5	14	42
6	5	14
7	30	28
8	8	14
9	24	32
10	48	90
11	4	2
12	6	8
13	15	6
14	15	12
15	28	14
16	3	1
17	6	6
18	11	5
19	15	26
20	8	7
21	2	3
22	14	16
23	1	20
24	1	2
25	2	3
26	6	6
27	4	19
28	4	8

The linear relation between the number of rules to be written and the corresponding time taken is shown in Figure 5.12.

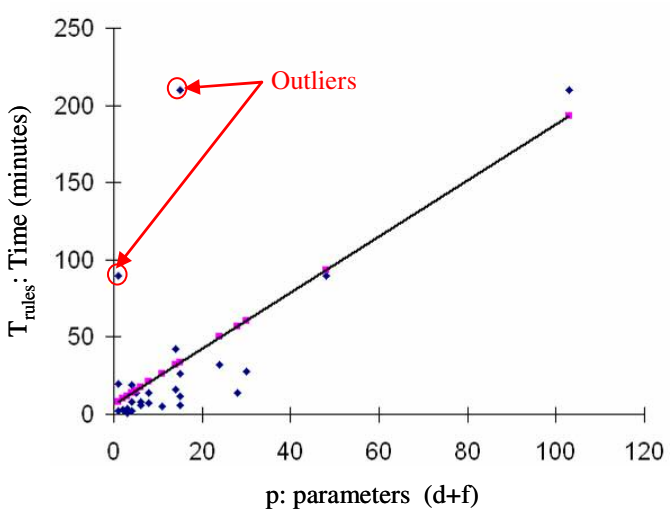


Figure 5.12: Plot between extracted parameters and time for writing rules

Close observation of Figure 5.12 reveals that there are outliers present while recording times for writing rules. These outliers resulted because design document was not present for 2800 Hybrid RH Metric grid product. The other reason is that the program was started by HI grid designer and later one of the Clemson student fixed that program which took lot of time for understanding the already written program.

The plot shown in Figure 5.12 can be represented in the form of a relation as given in Equation 5.13. Some rules may take more time for programming and some may not. When regression analysis was done on recorded times, the value of R^2 was obtained as 0. 45. This low value of R^2 reveals that many factors influence in writing rules. This low value of R^2 can be avoided if the design document is present while writing rules.

$$T_{Rules} = 2.0 * Parameters + 6.5$$
$$\Rightarrow T_{Rules} = 2.0 * (d+f) + 6.5 \qquad \textit{Equation 5.13}$$

Based on the above equation, one can say that it takes on average about two minutes for writing a rule for an extracted parameter of grid products in DW. The y-intercept value reveals that it will take at least of 6.5 minutes to start writing rules in DW. This may include launching the DW software, opening the required group, opening the required project within a group, and locating the parameter for which a rule needs to be written.

Rules for file names, relative paths, drawing names, and custom properties will fall into the category of similar rules. Programming similar rules will take less time than writing rules or extracted parameters. The recorded times for writing rules for similar parameters is shown in Table 5.7.

Table 5.7: Recorded times for writing rules for similar parameters

Observation	# of similar rules	Time (minutes)
1	15	6
2	15	4
3	15	12
4	15	5
5	30	8
6	28	14
7	3	1
8	3	1
9	11	4
10	47	3
11	45	4
12	6	1

Figure 5.13 shows the plot between the number of similar rules and the time required for writing rules for the similar parameters.

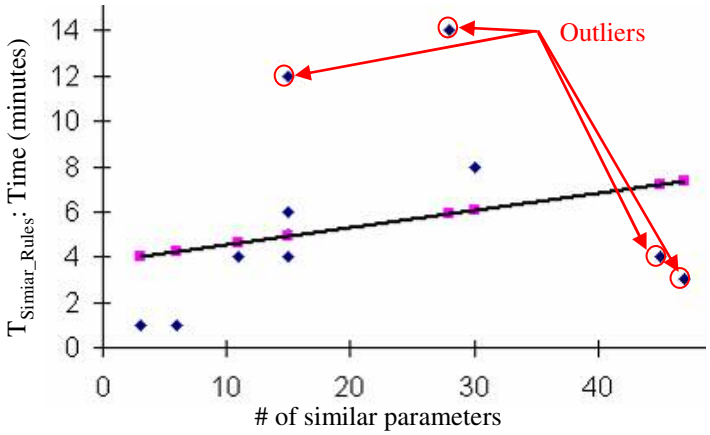


Figure 5.13: Relation between number of similar parameters and times

There are some outliers present in the plot, which tells that, not only files names, relative paths, custom properties, fall into the category of similar parameters but also rules may be similar across the components as shown in Figure 5.7, Figure 5.8 and Figure 5.9.

The relation between the number of similar parameters and the time taken for writing rules for these parameters can be given as:

$$T_{Similar\_Rules} = 0.1 * similar\_parameters + 3.8 \qquad \textit{Equation 5.14}$$

Where,

$T_{Similar\_Rules}$ = time for writing similar rules

similar_parameters = total number of similar parameter within an assembly

Close observation of Equation 5.14 reveals that it takes on average about 0. 1 minutes (6 seconds) to write rules for the similar parameters. On the other hand, the y-intercept value suggests that it takes on average about 3. 4 minutes to group the similar parameters together for writing rules.

DW Administrator has an option of seeing similar parameters together, which makes it easy for writing similar rules. The same rule can be copied and pasted for other similar parameters without actually writing the rules. In general, the HI grid products has 7P+6A similar rules, where P represents the number of parts that need to controlled and A represents the number of assemblies that need control.

The derivation for 7P + 6A is as follows:

$$\#\text{ of file names} = P + A \qquad \textit{Equation 5.15}$$

$$\#\text{ of relative paths} = P + A \qquad \textit{Equation 5.16}$$

$$\#\text{ of drawings names} \sim P + A \qquad \textit{Equation 5.17}$$

$$\#\text{ of custom properties} = 4P + 3A \qquad \textit{Equation 5.18}$$

Adding Equation 5.15, Equation 5.16, Equation 5.17, and Equation 5.18, we get the total number of similar rules for one RBD program as shown in Equation 5.19.

$$similar\_parameters = 7P+6A \qquad \textit{Equation 5.19}$$

$$T_{Similar\_Rules} = 0.1 * similar\_parameters + 3.8$$
$$\Rightarrow T_{Similar\_Rules} = 0.1*(7P + 6A)+ 3.8 \qquad \textit{Equation 5.20}$$

$$T_{Similar\_Rules} = 0.7P + 0.6A + 3.8 \qquad \textit{Equation 5.21}$$

Where,

$T_{Similar\_Rules}$ = Time for writing similar rules

A = Number of assemblies

P = Number of parts

Therefore, the summation of Equation 5.13 and Equation 5.21 gives the estimated time required for writing model rules as shown below:

$$T_{WritingRules} = T_{Rules} + T_{Similar\_Rules}$$
$$\Rightarrow T_{WritingRules} = 2.0*(d+f) + 6.5 + 0.7P + 0.6A + 3.8 \qquad \textit{Equation 5.22}$$
$$\Rightarrow T_{WritingRules} = 0.6A + 0.7P + 2d + 2f + 10.3$$

Where,

$T_{WritingRules}$ = Time taken for writing model rules

A = number of assemblies

P = number of parts

d = number of extracted dimensions

f = number of extracted features

Time calculations for programming in DW Administrator

The total time for completing DW programming stage is given by the sum of the times taken for creating the user interface $T_{UserInterface,}$ for declaring the data and tables $T_{Data\&Tables}$, and the time taken for writing rules $T_{WritingRules.}$ The total time for completing programming T_P , can be obtained by adding Equation 5.4, Equation 5.10, and Equation 5.22 as shown below:

$$T_P = T_{UserInterface} + T_{Data\&Tables} + T_{WritingRules} \qquad \textit{Equation 5.23}$$

$$T_P = T_{UserInterface} + T_{Data\&Tables} + 0.6A + 0.7P + 2\,(d+f) + 10.3 \qquad \textit{Equation 5.24}$$

For HI grid products the time for creating the user interface can be approximated as 180 minutes. Similarly, the time for specifying the data and tables for HI grid products can be approximated as 45 minutes. However, if one is interested in exactly estimating the time for creating a user interface, the value of design variables (DV) needs to be substituted in Equation 5.4. Similarly, for estimating the time for specifying the data and tables can obtained by substituting the number of programming variables (PV) in Equation 5.9. Table 5.8 shows the time calculations for the three products that were selected in Chapter 3.

Table 5.8: Time Calculations for programming

Sl. No	Product	A	P	D	f	$T_{UserInterface}$ (min)	$T_{Data\&Tables}$ (min)	$T_{WritiingRules}$ (min)	T_P (mins)
1	2800 HRHM	14	40	155	89	180	45	524	749
2	2800 PFEM	5	26	41	12	180	45	127	352
3	2800 PFEMNB	4	15	20	6	180	45	65	290

Chapter Summary

This chapter deals with writing rules for the extracted parameters. It presents the important steps in DW RBD programming process, which includes user interface creation, specifying the related information in the form of data and tables, and finally writing rules for the extracted parameters. In addition, this chapter discusses graphic design principles that should be followed during user interface creation. Finally, this chapter ends with presenting the estimated time for the programming process. The next chapter deals with the debugging and testing process that was employed in the HI case study.

CHAPTER 6

DEBUGGING AND TESTING

The next stage that follows the DW programming is the debugging and testing. The program should be thoroughly checked for errors that might crept in during the programming process. Debugging, verification and testing is done for identifying and removing these errors, which are called bugs in software terminology. The most difficult task in the process of debugging is to locate and fix the part of the code that is responsible for violating the known specifications [19]. The Figure 6.1 shows the phases of programming where debugging and testing activities are involved for eliminating bugs in DW rule based programming.

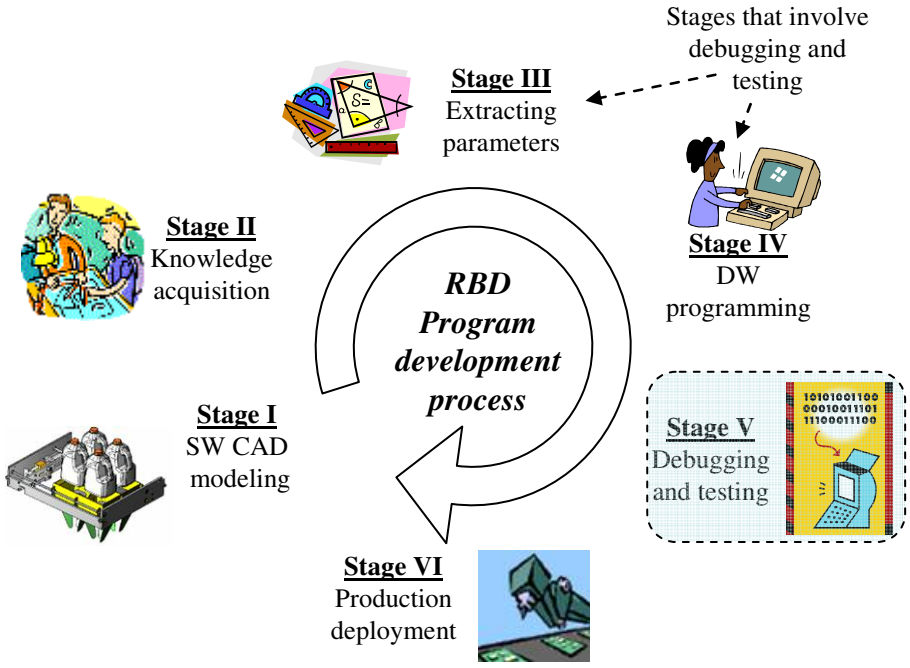


Figure 6.1: Activities that involve testing in rule based program development [19]

Bugs may be present in the program because [46], the user may execute the untested code, the order in execution of statements may be different during the testing and in actual case, the user applied may apply wrong values or untested input values, and the users operating environment may be different from the testing environment.

Method adopted in HI IRBD project

In HI IRBD project, testing process is classified under two major headings. They are "α-*testing"* and *"β-testing."* The Clemson University is responsible for the former and the Hartness International is responsible for the latter. Parts, assemblies, and drawings become the part of DW generated components. The primary focus of α-testing is to check the structural and functional completeness of the program [46]. In DW terminology, a program is said to be structurally complete when all the extracted parameters are driven by one or the other rule. Similarly, a program is said to be functionally complete when all these rules produce physically correct clones, which do not have any rebuild errors that arise from inconsistencies in sketches, features, and mating conditions. It is a good idea to separate programming from testing and is advisable to have different persons for programming and testing. This way, the tester will not be biased during the testing process and there is a larger scope of covering many test cases for unearthing hidden bugs. However, in HI IRBD project, the tester was same as the programmer due constraint on time and resources. As part of □-testing process; unit testing, integration testing, boundary value partitioning technique (BVPT), cross product boundary value partitioning technique (CPBVPT), and typical use scenario testing was done. The Hartness International is responsible for □-testing because it is the production deployment stage and is not covered in this study. All these testing process are classified as shown in Figure 6.2.

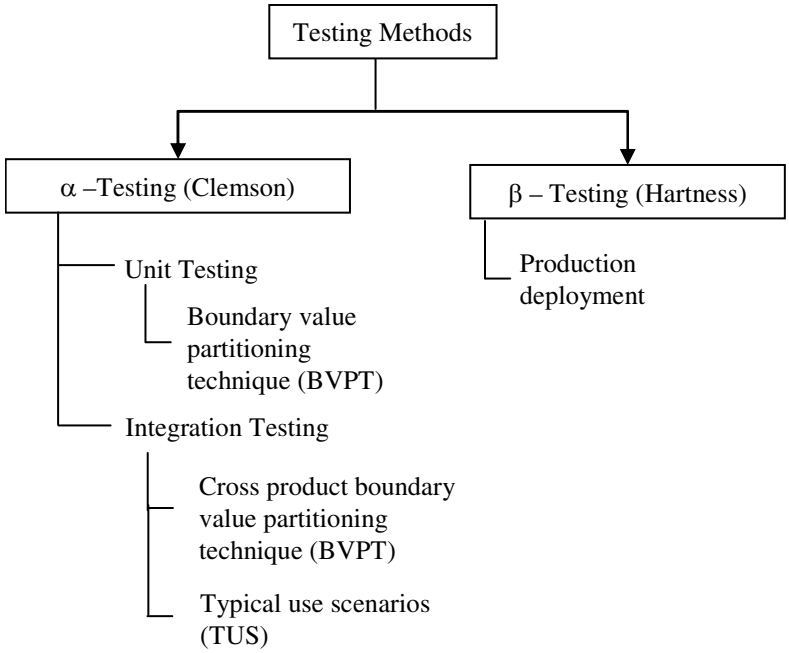


Figure 6.2: Testing methods used in HI IRBD project

Unit testing process starts with testing the user interface forms. As mentioned in the previous chapter, some form fields are created using rules and two different fields are linked together with the help of these rules. For example, in designing grid user forms, the number of transversals (NoT) depends upon the number of bottles per lane (NBPL). The Table 6.1 shows the relations between NBPL and NoT. 'T' indicates True, the possible combination of design for NBPL and NoT. 'F' indicates False, the infeasible combination of NBPL and NoT. The two form fields, NBPL and NoT are linked together to accept only feasible values as shown in shown in Equation 6.1.

IF NBPL = 2,THEN NoT = '0|1',
ELSE IF NBPL = 3, THEN NoT = '0|2',
ELSE IF NBPL = 4, THEN NoT = '0|1|3',
ELSE IF NBPL = 5, THEN NoT = '0|4',
ELSE IF NBPL = 6, THEN NoT = '0|1|2|5',
ELSE IF NBPL = 7, THEN NoT = '0|6',
ELSE IF NBPL = 8, THEN NoT = '0|1|3|7',
ELSE '0'

Equation 6.1

Note: '0 | 1' represents '0' OR '1'.

Testing of user interface was done by inputting all the values for NBPL field to check whether or not, the NoT field is reflecting the corresponding value. Other methods used in testing the user interface for HI IRBD project were boundary value partitioning technique (BVPT) for single fields and cross product boundary value partitioning technique for multiple fields (CPBVPT) [46].

Table 6.1: Truth table for Transversals

Number of bottles per lane NBPL	Number of Transversals NoT									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	T	F	F	F	F	F	F	F	F
2	T	T	F	F	F	F	F	F	F
3	T	F	T	F	F	F	F	F	F
4	T	T	F	T	F	F	F	F	F
5	T	F	F	F	T	F	F	F	F
6	T	T	F	T	F	T	F	F	F
7	T	F	F	F	F	F	T	F	F
8	T	T	F	T	F	F	F	T	F

In BVPT, the maximum and the minimum values for the fields on the forms are tested. While designing HI grid forms, the error messages are embedded for the form fields as given in Equation 5.2. Testing was done by inputting the maximum and the minimum values in addition to the out of range values to see whether the form error rules are working or not. The out of range values lie outside the maximum and minimum boundaries. The values between these boundaries are treated as similar numbers while testing the forms. For example, in Equation 5.2, the maximum and minimum values are 6 and 2 respectively. Therefore, in BVPT for checking a single field, it does not matter whether the tester uses in between values such as 2.5, 4 or 5.75 that does not make any difference to the program under test.

In CPBVPT, the maximum and minimum values are compared with the resultant value of two or more fields in the form. The resultant value refers to the outcome of the mathematical operations between two fields. The mathematical operations for building rules may include addition, subtraction, division, or multiplication. For example, CPBVPT technique was applied in HI IRBD project for constraining the maximum and minimum values for the size of the basket as shown in Figure 6.3.

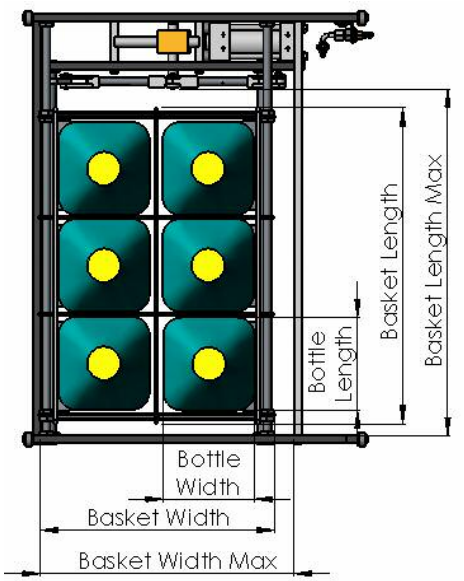


Figure 6.3: Cross Product boundary value partitioning technique

In the above figure, the basket length is obtained by multiplying the bottle length with the number of bottles per lane and then adding transversal thickness with the corresponding bottle clearance. The corresponding rule is given by Equation 6.2.

$$\text{BasketLength} = (\text{NBPL}*\text{BL}) + \text{NoT}*(\text{TT}+\text{TBC}) + \text{EBC} \times 2 \qquad \textit{Equation 6.2}$$

Where,

NBPL = # of bottles per lane
BL = Bottle Length
NoT = # of Transversals
TT = Transversal Thickness
TBC = Transversal Bottle Clearance (0.625 Constant)
EBC = End Bottle Clearance (0.25 Constant)

Basket length is calculated every for new inputs of bottle length, transversal thickness, and number of transversals; and the CPBVPT testing method is applied to compare this calculated basket length to see whether it lies in between the maximum and minimum values. The Table 6.2, shows the type of testing method applied for checking the form fields.

Table 6.2: Testing Techniques used in HI IRBD

Sl. No	Variable Name	Maximum Value	Maximum Value	Testing Technique
1	Bottle Length	4. 5	8. 0	BVPT
2	Bottle Width	2. 25	8. 0	BVPT
3	Transversal Thickness	0. 02	0. 75	BVPT
4	Basket Width	8. 5	14. 5	CPBVPT
5	Basket Length	5. 5	21	CPBVPT

The user interface was tested to see whether it identifies the out of range values for the corresponding fields after mathematical operations between the form fields. The sample UI test log is presented Table 6.3. The last column of this table leaves a note of why the testing condition was passed or failed.

Table 6.3: UI Test log for 825 Platform Lowering LH

Sl. No	Test	BL I	NBPL II	NOT III	TT IV	BasLen R	Test P/F	Method Failed	Remarks
1	1T721	7	2	1	0. 746	12. 0	P	Nill	-
2	2T612	6	1	2	0. 394	7. 6	F	CPBVPT	III!<II
3	3T324	3	2	4	0. 127	5. 5	F	Both	I, R, & III!<II
4	4T532	5	3	2	0. 245	16. 1	P	Nill	-
5	5T531	5	3	1	0. 202	16. 2	F	CPBVPT	III !~II
6	6T332	3	3	2	0. 857	12. 2	F	BVPT	IV
7	7T531	5	3	1	0. 643	16. 1	P	Nill	-
8	8T732	7	3	2	0. 223	22. 6	F	CPBVPT	R
9	9T822	8	2	2	0. 501	16. 2	F	BVPT	III!~II
10	10T871	8	7	1	0. 622	54. 8	F	CPBVPT	R & III!~II
11	11T513	5	1	3	0. 090	7. 0	P	Nill	-
12	12T251	2	5	1	0. 312	11. 0	F	BVPT	I
13	13T731	7	3	1	0. 286	22. 8	F	CPBVPT	III!~II
14	14T842	8	4	2	0. 483	29. 1	F	BVPT	II & III!~II
15	15T621	6	2	1	0. 357	10. 2	P	Nill	-
!< : NOT GREATER THAN; !~ :NOT COMPATIBLE; P : Pass; F : Fail; R: Result									

The interpretation of rows in Table 6.3 is as follows: For example, consider the test case 3T324, by the name itself one can depict that it is the third test that is conducted, and the variables used were BL=3, NBPL=2 and NoT =4. The outcome of this testing resulted an error value for the basket length,

which violates the maximum and minimum criterion as shown in Table 6.2. The remarks column shows why the testing was failed. 'I' in the last column represents that the bottle length was out of range while 'R' represents that the resultant basket length is out of range. In addition, 'IV' represents that the transversal thickness is out of range and 'III! < II' represents that the number of transversals should be always be less than the number of bottles per lane. Finally, III! ~II represents that even though the number of transversal is less than the number of bottles per lane, they are not compatible with each other. For example, one transversal cannot be placed evenly between three bottles. Suitable number of transversals for this case would be either zero or two and can be derived from Equation 6.1.

In HI IRBD project, testing was integrated with programming and was done in phases. After creating the user interface and the required data (constants, variables, and look-ups), main components in the assemblies are selected and rules were written for the parameters that are extracted in stage III of RBD programming. After writing rules for extracted parameters, the components were tested to see whether the written rules are working in building the variant SW Models. There is a provision in DW Administrator called test specifications from which, the programmer can test only a part of the program. This type of testing a particular portion of the program is also referred to as *"unit testing"* in the literature [46]. The programmer can specify the test cases to see how the rules are effecting a particular SW CAD models in producing the clones. The process of producing clones is shown in Figure 6.4. For every run of test specifications, DW Administrator produces model generation information, which is utilized by DW model generator for generating variant SW CAD models, i. e. clones. DW model generator found in DW tool bar within SW, produces model generation reports for every test specification. These model generation reports give an overall outline of what DW could or could not do in producing the clone models. This generation report helps the programmer in debugging the implementation and analyzing the causes of failure in clone models.

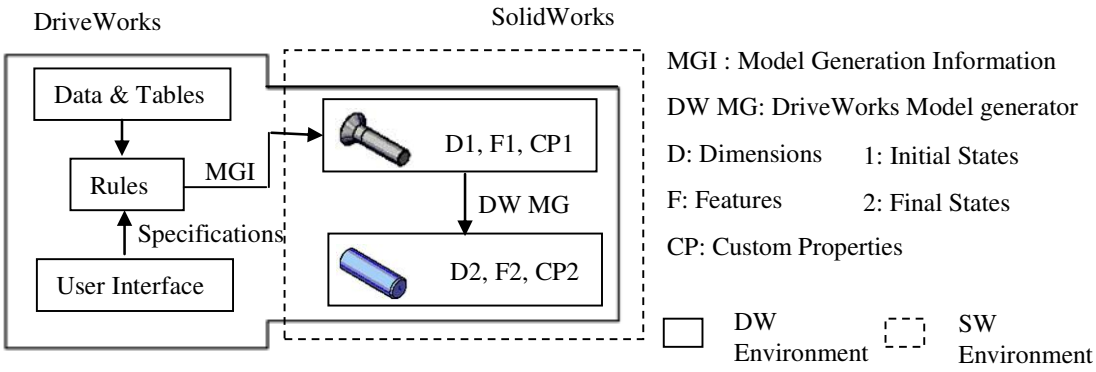


Figure 6.4: Process of producing clones

The process of fixing errors is an iterative process and is completed when the program produce geometrically correct models as shown in Figure 6.5. Once the errors are fixed for a particular component, the program is extended for writing rules for the other components. After writing the rules for all the components, the program is ready for *"integration testing"*. The focus in integration testing lies on checking the interactions between components when tested as a whole. Unit testing of all the components needs to be done before starting the integration testing. . In HI IRBD project, this method of testing was done in releasing the test specifications for producing clones models in SW. CBPBVPT testing also becomes the part of integration testing.

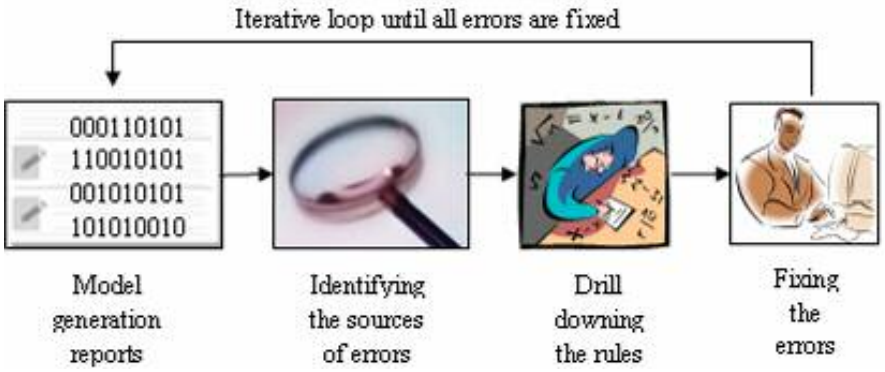


Figure 6.5: Debugging process in DW programming

The other method of testing the RBD program is checking the typical use scenarios [46]. Typical use scenarios are the specifications for which the RBD program is generally used when deployed in production. This way common errors can be detected and removed for obtaining a more consistent

program. The process of testing one test case i. e. releasing one set of specifications is termed as test cycle and it includes implementing the specifications, releasing the specifications, generating the specifications, and checking the generated models. If errors are present in the generated models the program needs to be fixed before starting the new test cycle. If errors are not present, then certain number of test cases need to be specified in which case, step 4 (checking the models) is followed by step 1 (specifying the test case),

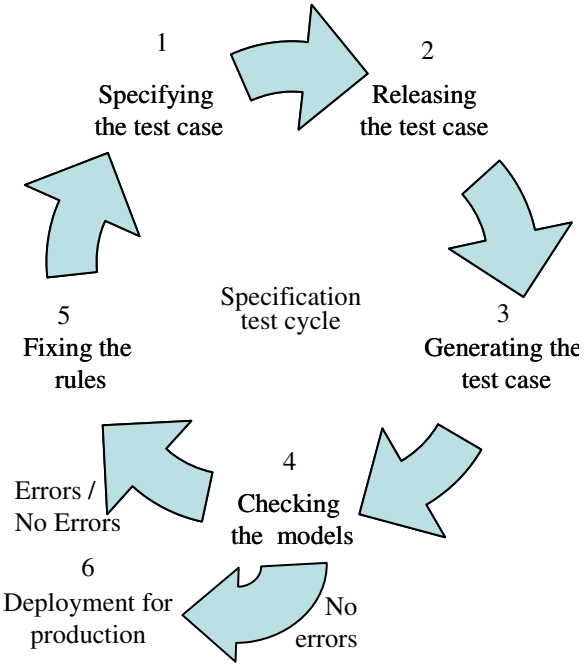


Figure 6.6: Phases in specification test cycle

If the program produces correct models for every run specified test cases, step 6 (deployment for production) follows step 4 (checking the models) as shown in Figure 6.6. The number of specified test cases depends upon the trade off factors like budget, time, and quality.

Time for testing RBD program

Time for testing can be approximated to 35-40%[5] of total product development time [8]. This percentage of time was verified by interviewing an Oracle software tester who is in the field of testing software products for more than four years. This is a rough approximation for time calculations and can be

[5] http://www. ibm. com/developerworks/rational/library/2114. html#author

used for estimating the time required for RBD programming. When an experienced programmer develops a RBD program, the time taken for testing may take less time than this specified percentage. In this case, only specified number of test cases needs to be tested without the requirement of fixing the bugs. There are some automated testing procedures, where specific code can be written in Java platform. Automated testing was not used in HI IRBD project since the Clemson students were not familiar with writing Java Code. The succeeding sections shows the time calculations for testing based on this assumption.

Let us say, the time taken for program development is T_{PD}. This includes the time taken extracting the parameters, and time taken for writing the rules with in the DW environment. Then the time taken for testing would be as follows:

Time taken for program development = T_{PD}
Time taken for testing = T_T
Total Time for programming is $T = T_{PD} + T_T$ ***Equation 6.3***

Assumption T_T =40% of T
$T_T = 0.4\ T$ ***Equation 6.4***

Time for T_{PD} = 60% of T
$T_{PD} = 0.6T$ ***Equation 6.5***

From Equation 6.4 and Equation 6.5, the time for testing (T_T) can be obtained as,

$$T_T = \frac{2}{3} T_{PD} \qquad \textit{Equation 6.6}$$

If testing the program is taking more time than this specified percentage, it means that the program is ill written and was not properly debugged and tested in the unit-testing stages. The other reason for this kind of scenario would be errors in the platform itself. The Table 6.4 show the calculations for the testing times in minutes (T_T) for the grid products that are given in Chapter 3.

Table 6.4: Time taken for program development

Product	T_{EP} (mins)	T_P (mins)	$T_{PD}=T_{EP}+T_P$ (mins)	T_T (mins)
2800 HRHM	211	749. 4	961	640
2800 PFEM	91	352. 2	443	296
2800 PFEMNB	64	289. 9	354	236

Chapter Summary

In this chapter, the most important stage of RBD program development, software testing, is discussed. This chapter starts with identifying the stages in program development where testing is required. It discusses in detail about user interface testing because most of the errors are the result of inputting wrong rules. It describes the boundary value partitioning technique and the cross product boundary value partitioning technique that are used in HI IRBD project. In addition, it describes the "typical use scenarios" that are used in HI IRBD project as part of the $\tilde{\alpha}$-testing process. The chapter discusses in brief about the DW software test cycle and the phases that are involved in this process. Finally, this chapter ends with presenting the time calculations for testing the RBD program, on the assumption that it takes 30 – 40 % of the time for testing any software program as seen in literature.

CHAPTER 7

DEMONSTRATION STUDY: 2800 PLATFORM LOWERING INCH

In this chapter, the use of all the equations for calculating the estimated time for RBD program development within DW is shown with the help of an example. This is done by choosing grid product from Table 1.1 to show the affect of specification variables on the parameters. The stages that are considered for showing the time calculation are knowledge acquisition (stage II), extracting parameters (Stage III), DW programming (Stage IV), and debugging & testing stage (Stage V) as shown in Figure 7.1. The first stage of the programming process, SW CAD modeling, is not considered in these time calculations since CAD models are already present in HI design database. Similarly, the last stage, production deployment, is also not considered because this will be the application stage and will last until there are any major changes in grid design resulting new programs.

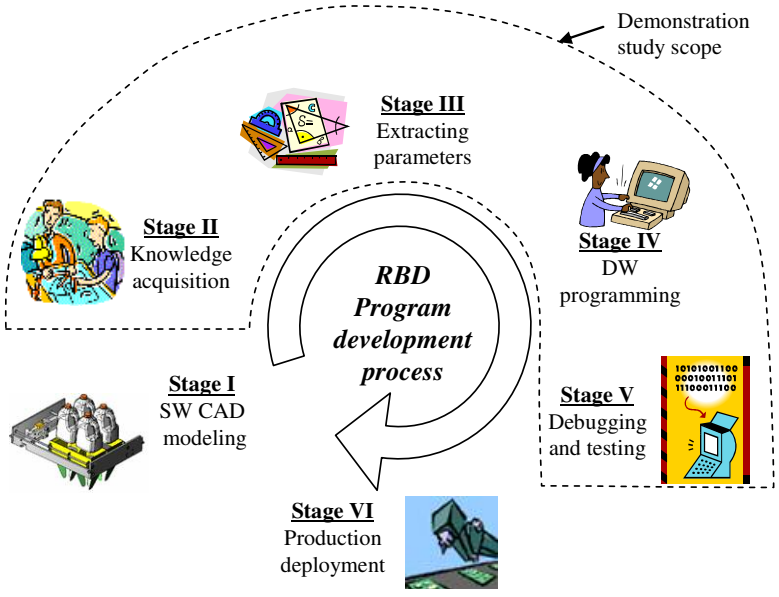


Figure 7.1: Rule based program development process with in DriveWorks

2800 Platform Lowering Inch is selected for estimated calculations from Table 1.1 for the following reasons:

1. This product is similar to the other grid products that were used for deriving the equations for time estimations and helps in removing uncertainties that arise from product knowledge that is required. The time models are developed based on the assumption that, the programmer should be familiar with the products before capturing the design information.

2. The major assemblies like basket assembly and the platform doors assemblies are present, which involves many parameters that need to be controlled.

3. A completely new product that was never programmed before in HI IRBD project.

4. This product is a medium sized assembly compared to other grids that are shown in Table 1.1 and gives an opportunity to study many parts and sub assemblies in terms of parameters to extract and control by writing rules.

5. There is an opportunity to study the affect unique and similar rules in various parts of this assembly.

6. The programming factors such as constants, variables, and user interface can be created and their affect can be studied.

2800 Platform Lowering Inch (2800 PLI) is generally used to handle a wide range of round and non round container shapes with varying sizes that range from 2 ounces to 2 ½ gallons. It is one of the HI's flexible medium speed vertical case packing machine that has integrated high-speed laner option with air transfer mechanism for the packing of a variety of difficult shaped containers such as flasks, chimed cans, and trigger bottles as shown in Figure 7.2.

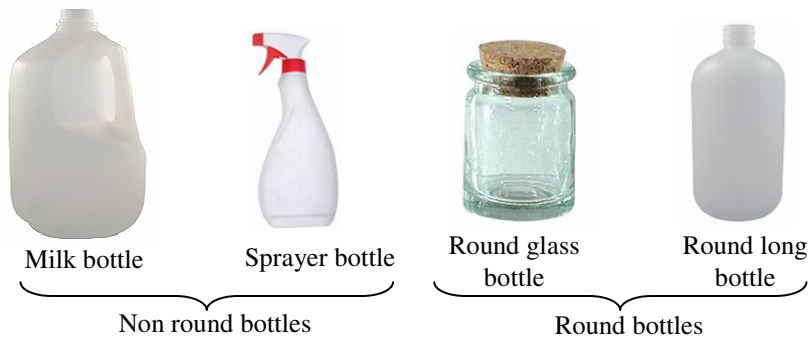


Figure 7.2: Different shaped and sized bottles

Typical CAD assembly model of 2800 PLI is shown in Figure 7.3. Air bags are used to transfer bottles from upper conveyor on to the grid, where the platform doors hold the bottles in place inside the basket assembly. The entire grid assembly moves down, above the case while the platform doors are opened by operating the cylinders by servo-mechanism. Platform doors and fingers guide the bottles to fall into the corresponding case partitions and thus the cases packed with bottles are delivered from the bottom conveyor. The same grid cannot be used for running different types of bottles or cases and the grid assembly should be redesigned for every new bottle/case specifications. The parameters that affect the grid assembly design are dependent upon four categories of information that include bottle information, grid information, case information, and miscellaneous information.

Bottle information includes the design variables such as bottle width, bottle length, bottle height, and contained shape. Grid information includes the design variables such as number of lanes, number of bottles per lane, grid identifier, bottle lane clearance, transversal adder, and information related to basket fingers. Case information includes the case width, case height, case length, case flap height, and case partitions information. Additional information include, the grid work order, designer, and design date that are useful for preparing drawings.

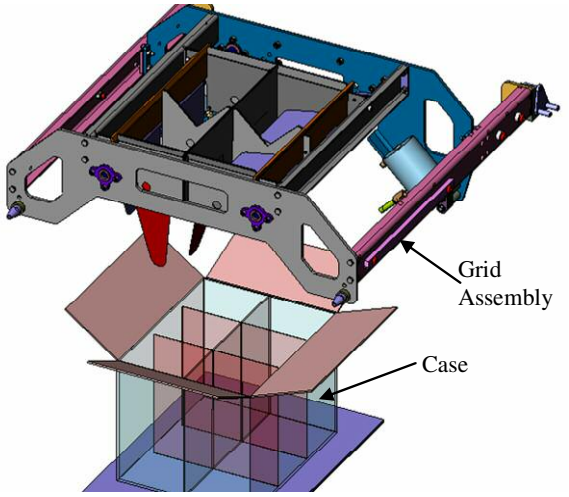


Figure 7.3: CAD assembly model of 2800 Platform Lowering Inch

Redesigning some of the grid components is required because of the change in any of the above design variables and are shown in the Figure 7.4.

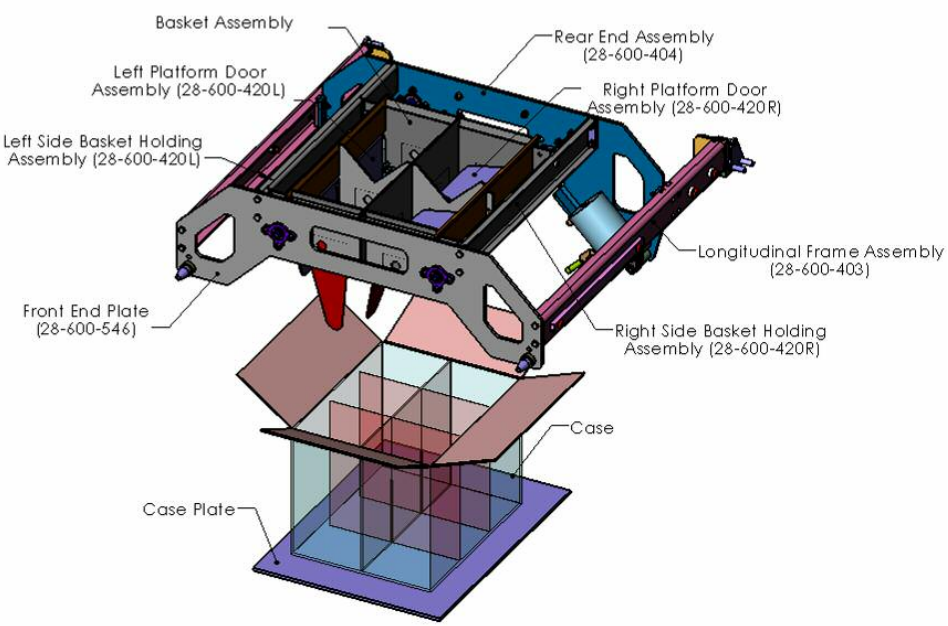


Figure 7.4: CAD assembly model showing various components that needs to be controlled

Method used for verifying and validating the estimated time calculations

Initially, the driving parameters for the above-mentioned assembly are identified. Then the estimated time for RBD program development is obtained by using these parameters in conjunction with

the equations that are presented in earlier chapters. The obtained estimated time is verified by comparing the time that is actually taken for writing the rules for the same assembly in DW Administrator (2800 PLI in this case). The driving parameters and their count are given in Table 7.1. The estimated time calculations would be precise, if the low level information is known which includes number of assemblies, number of parts, number of dimensions, and number of features that needs control rather than having a rough estimation for each of them.

Table 7.1: Design Variables for 2800 Platform Lowering Inch

Sl. No	Parameters	Count
1	Total of Assemblies	24
2	Total number of Parts	258
3	# assemblies that needs control (A)	13
4	# parts that needs control (P)	27
5	# of dimensions (d)	59
6	# of features (f)	22

Time calculation for development of RBD program

Mentioned before, this research is concentrated only on four stages (excluding the first and the last stage) of DW rule based programming process, which are knowledge acquisition, parameters capture, RBD programming, and debugging and testing. The succeeding sections deal with the time calculation estimations for completing these individual stages that are shown in Figure 7.5.

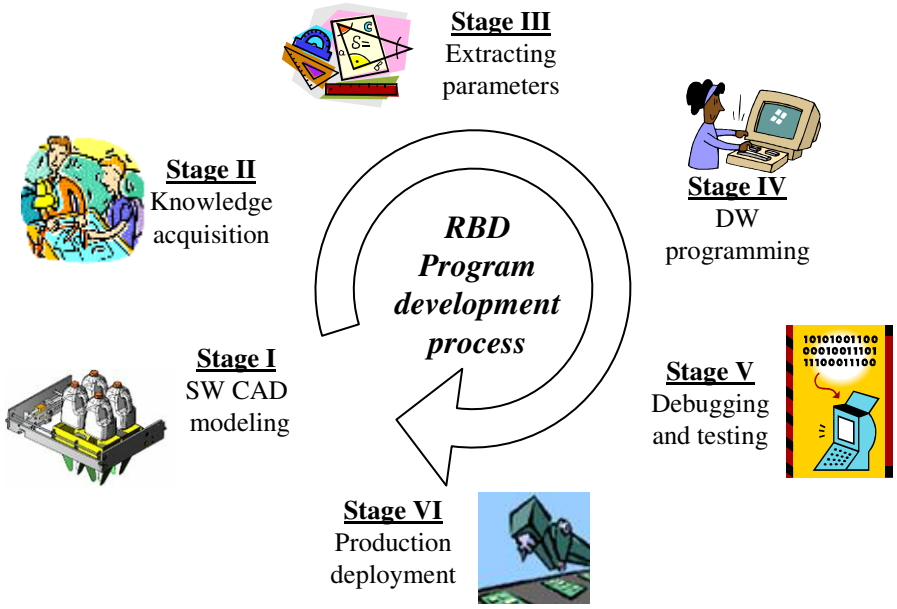


Figure 7.5: RBD program development process

Stage II: Knowledge acquisition stage

The second stage will be the knowledge acquisition stage in which the knowledge engineers capture the information related to the design of the product. This is dependent upon the number of dimensions and features that change with the design specifications for a particular product. Here design specifications are the new bottle/case specifications for which the grid components need to be designed. The time for capturing the information of grid products is dependant upon features and dimensions within these components and can be estimated by using Equation 3.1 as follows:

$$T_I = 3 * (d+f) \qquad \textit{Equation 3.1}$$

'd' is the number of dimensions to control in the 2800 PLI, which is equal to '59' from Table 7.1. Similarly, from the same table 'f' the number of features to control can be obtained as '22'. Plugging these two values in the Equation 3.1, one can obtain the estimated time required for capturing the knowledge about the 2800 PLI as 4 hours as shown below.

$$T_I = 3 * (59+22) = 222$$
$$T_I = 222 /60 = 3.7 \approx 4 \text{ hours} \qquad \textit{Equation 3.1}$$

After capturing the design knowledge, a design document needs to be prepared which shows all the information pertaining to the product that is of interest. From Table 7.1, the number of components in 2800 PLFI can be obtained as '27'. Referring to Equation 3.2, the corresponding value for time to prepare the design document for this number of parts can be taken as 20 hours.

$$C1\ (T_{DD}) = 20 \text{ hrs for assembly with 27 components} \qquad \textit{Equation 3.2}$$

The time taken for knowledge acquisition stage will the summation of time taken for conducting the interviews, the time taken for capturing the design information, plus the time taken for preparing the design document. Adding the above two values, one can get the time taken for knowledge acquisition for 2800 PLI as "24 hours" as shown below:

$$T_{KA} = T_I + T_{DD}$$
$$\Rightarrow T_{KA} = 3.7 + 20 = 23.7 \approx 24 \text{ hours} \qquad \textit{Equation 3.3}$$

Therefore, the time required for knowledge acquisition of 2800 PLI can be estimated as 24 hours. *Equation 7.1*

After creating the design document, it is submitted to the HI for getting the approval for design document and the rules. The HI grid designer will go through this design document thoroughly to check whether this design document reflects back the design intent of the product and approves the document by suggesting any changes. The next stage after getting the document approval is starting the extraction of interested parameters from SW CAD models. This stage of DW RBD programming include, creation of DW group, specifying the assemblies to control in the form of projects and finally extracting all the parameters, dimensions and features, that change with design specifications.

Stage III: Extracting parameters to control

This stage primarily consists of creating an interface with DW and SW software's in terms of parameters. The time taken for extracting the parameters depend upon the number of assemblies (A), parts (P), dimensions (d), features (f), and corresponding value for miscellaneous tasks (C_2). From the Table 7.1, substituting the values of 'A' (= 13), 'P' (= 27), 'd' (= 59), and 'f' (= 22), and the value for 'C_2' (= 30

minutes) (from the Equation 4.5, the corresponding time for number of components (=27)), in the Equation 4.6, the time taken for capturing the parameters can be obtained as "2 hours" as shown below.

$$T_{EP} = 0.95A + 1.15P + 0.5(d+f) + 1.75 + C_2$$
$$T_{EP} = 0.95 * (13) + 1.15 (27) + 0.5(59 + 22) + 1.75 + 30 = 115.65 \quad \textit{Equation 4.6}$$
$$T_{EP} \approx 2\text{hrs}$$

Therefore, the time required for extracting the parameters of 2800 PLI is estimated as 2 hours — ***Equation 7.2***

Stage IV: DW programming

As discussed in Chapter 5, the DW programming consists of preparing the user interface, specifying the data and tables, and finally writing rules for the extracted parameters. The time taken for completing these activities can be estimate by using the Equation 5.24.

$$T_P = T_{UserInterface} + T_{Data\&Tables} + 0.6A + 0.7P + 2\,(d+f) + 10.3 \quad \textit{Equation 5.24}$$

For grid products, the values for $T_{UserInterface}$ and $T_{Data\&tables}$ can be taken as 180 minutes and 45 minutes respectively. . Therefore by substituting the remaining values of 'A' (=13), 'P' (=27), 'd' (=59), and 'f' (=22) in the Equation 5.24, the estimated time for DW programming for 2800 PLI can be obtained as "7 hours" as shown below:

$$T_P = 180 + 45 + 0.6(13) + 0.7(27) + 2\,(59+22) + 10.3 = 424 \text{ mins}$$
$$\Rightarrow T_P = 424/60 = 7.0667 \approx 7 \text{ hrs}$$

Therefore, the time required for DW programming of 2800 PLI can be estimated as 7 hours — ***Equation 7.3***

Stage V: Debugging and Testing

The time required for testing (T_T) the DW RBD program can be obtained by using Equation 6.6, which is given below:

$$T_T = \frac{2}{3} T_{PD} \quad \textit{Equation 6.6}$$

Where, the time required for RBD program development (T_{PD}) can be obtained from Equation 6.3 as shown below:

$$T_{PD} = T_{EP} + T_P \qquad \textit{Equation 6.3}$$

Where,

T_{PD} = Time for program development,

T_{EP} = Time for extracting parameters, and

T_P = Time for DW programming

The values for T_{EP} (= 2 hours) and T_P (= 7 hours) for 2800 PFI can be taken from Equation 7.2 and Equation 7.3 respectively, and estimated time calculations for 2800 PFI RBD program development (T_{PD}) can be obtained as '9 hours' as follows:

$$T_{PD} = 2 + 7 = 9 \text{ hours} \qquad \textit{Equation 7.4}$$

Therefore, substituting the value of T_{PD} (=9 hours) in Equation 6.6, the time that can be allocated for testing (T_T) the 2800 PLI grid DW RBD program can be approximated as "6 hours", which is obtained as shown below:

$$T_T = \frac{2}{3} \times 9 = 6 \text{ hours}$$

Therefore, the time required for testing the DW RBD program of 2800 PLI can be approximated as 6 hours *Equation 7.5*

Total time estimations DW RBD programming process

Therefore, finally the total time ($T_{Estimate}$) required for developing a RBD program is given by the sum of the times of four stages, which include knowledge engineering (Stage II), extracting parameters (Stage III), DW programming (Stage IV), and testing (Stage V) as given below:

$$T_{Estimate} = T_{KA} + T_{EP} + T_P + T_T \qquad \textit{Equation 7.6}$$

The total time (T) required for 2800 PFI can be estimated as '39 hours' by substituting the values of 'T_{KA}' (= 24 hours from Equation 7.1), 'T_{EP}' (= 2 hours from Equation 7.2), 'T_P' (= 7 hours from Equation 7.3), and 'T_T' (= 6 hours from Equation 7.5) in the Equation 7.6 as shown below:

$$T_{Estimate} = 24 + 2 + 7 + 6 = 39 \text{ hours}$$

Therefore, the time required for testing the DW RBD program of 2800 PLI can be approximated as 6 hours

Equation 7.7

Verification of estimated time calculations

This estimated time ($T_{Estimate}$) is verified by actually programming 2800 PLI grid product in DW Administrator. This program was written by a Clemson student with eight months of DW experience. The programmer took approximate four days to write the DW RBD program for this product and detailed breakdown of time for each stage of DW RBD programming is given in Table 7.2.

Table 7.2: Comparison between estimate and actual times

	RBD programming Stage	Activity	Estimated Time (Hrs)	Observed Time (Hrs)
Stage II	Knowledge Capture	Interviews	4	6
		Design Document	20	12
Stage III	Extracting Parameters		2	3. 2
Stage IV	DW Programming	User Interface	3	3. 18
		Data & tables	0. 75	0. 75
		Model Rules	3. 3	2. 7
Stage V	Debugging & Testing		6	4. 5
Total Time in Hours			**39**	**33**

Actual time for DW programming (T_{Actual}) is obtained as 33 hours without considering any allowances. The estimated time ($T_{Estimate}$) obtained from Equation 7.7 is 39 hours.

The percentage error between estimated time and actual time can be calculated as:

$$\% \text{ Error} = \frac{|T_{Estimate} - T_{Actual}|}{T_{Actual}} \times 100$$

$$\% \text{ Error} = \frac{|39 - 33|}{33} \times 100 = 18.2\%$$

$$\therefore \% \, Error = 18.2\%$$

Equation 7.8

From the above, the percentage error can be seen as 18. 2 %. Therefore, with 80% confidence interval, the estimated time calculations fall with in the bounds allowing 20% for variations, proving the accuracy of the model. Thus, the models for time estimate calculations of RBD programming in DW Administrator for HI grid products is validated and can be used for estimating the time for other products of HI grid department.

Chapter Summary

This chapter presents a demonstration study in showing the use of time calculation models for estimation in RBD program development process. A suitable grid product is selected to show the use of these models for time estimation. This is compared with the time that is taken for actually writing RBD program in DW Administrator for the selected grid product. For validating the time models, the percentage error is calculated between the estimated time and actual time by choosing 80% confidence limits. The percentage error was obtained as 18.2% that falls within 20% of each other, which builds confidence in the correctness of the models for grid products.

CHAPTER 8

CONCLUSION AND CLOSURE

This chapter presents the summary of the case study research that was done as part of automating the grid design process of HI. It provides a brief discussion on the significance of developing models that estimate the time required to complete the DW RBD program development process. The limitations of these time estimation models together with the discussion on directions for the immediate future work for refining them is also presented.

Addressing the Research Questions

The research questions that are proposed in Chapter 1 are presented once again to remind the reader about them. They are addressed by conducting a case study research on automating the design process of HI grid products. The first research question is:

RQ 1: What are the stages that are involved in the DW RBD program development process?

This research question is addressed in the Chapter 2, which is the result of experience from the HI IRBD project. The stages that are involved in RBD program development process and they include: CAD modeling (Stage I), knowledge acquisition (Stage II), extracting parameters (Stage III), DW programming (Stage IV), debugging and testing (Stage V), and finally production deployment (Stage VI). Stage I of CAD modeling is not addressed in this research because SW CAD models are already present in HI for RBD programming. The assumption is that, each CAD model is built in a way that it can be driven by DW software using the guidelines that are presented in the Chapter 4. The Stage II of knowledge acquisition is addressed in Chapter 3, where it presents interviewing as the preferred method of capturing the design information about the product. The categories of information needed for this process are also presented. It also presents the requirements for preparing a design document, emphasizing its importance in RBD program development process. The Stage III of extracting parameters is presented in Chapter 4. The modeling guidelines for preparing CAD models are presented in this chapter as one of the primary

outcomes of this research. This chapter discusses the various phases that are involved during this stage for extracting the parameters from SW CAD models that need to be controlled. The Stage IV of DW programming is presented in Chapter 5, where it discusses the three major phases of programming that include : creating a user interface, specifying the product related data, and writing rules for the parameters extracted in the Stage III. The Stage V of testing and debugging is presented in Chapter 5, which discusses the testing methods adopted in HI IRBD project. This chapter explains about the boundary value partitioning technique and the cross product boundary value partitioning technique as part of the □-testing process that needs to be done by the program development team. The other method of typical use scenario testing is also discussed in this chapter. The results of testing are not addressed in this research since there was not an opportunity to study its effect on the programming process. The Stage VI of production deployment is also not addressed in this research since it is the actual use of RBD programs for building CAD models. There was no opportunity in the HI IRBD project to study the results of the effect of these RBD programs when applied in production. This is reserved for future work.

In brief, the thesis elaborates on the activities that are involved in each stage of the RBD program development process and presents the method adopted in the HI IRBD project as part of the case study. These individual stages of the RBD programming process help in identifying the resources for completing each of them, thus providing upper management with a means to allocate various resources to the different projects. By knowing these stages, multiple projects can be scheduled with a minimum of conflicts in the resources.

The second research question is:

RQ 2: What are the programming parameters that affect the stages in the RBD program development?

The second research question was addressed in identifying the parameters that affect the RBD program development process. The parameters are identified as number of assemblies (A), parts (P),

dimensions (d), and features (f). They change for every new design specification and rules should be written for controlling these parameters while building variant CAD models. Therefore, the effect of these parameters and their count, on the time required to complete various stages of RBD program development process are presented in Chapter 3, Chapter 4, Chapter 5, and Chapter 6 in the form of equations.

The third research question is:

RQ 3: How do time estimates for the RBD program development process depend on CAD parameters?

The time estimates for the RBD program development process depend on the number of parameters that need to be controlled for a given product. These parameters are obtained as the result of addressing the second research question (RQ 2). The time required for completing the individual stages of the DW RBD program development increases with the increase in number of parameters that need control. The models for time estimates are the major results of this research. To obtain these models, times taken for completing the activities involved in Stage II thru Stage IV were recorded while automating the grid design process in HI IRBD project. Linear regression analysis was done on these recorded observations to show the affect of the parameters on time estimations. The results of this process are the estimation models that predict times for finishing the individual stages of the DW RBD program development and depend upon the parameters that need to be controlled.

Chapter 3 deals with the estimated time calculations for capturing the knowledge about the product, i. e. knowledge acquisition (Stage II). The time is dependent upon the parameters that change with design specifications. The parameters may include the assemblies, parts, dimensions, and features that change in a given grid product. It also discusses the time that is required for preparing a design document and roughly estimates this in terms of parts that need to be controlled. This approximation will work only for grid products of HI and may not be applicable for other products. From the case study, it can be seen that on average ten parameters, dimensions or features, are to be controlled for grid products. This case might not be true for other products in which more dimensions or features need to be controlled. Building a design document is a tedious task and depends upon the number of dimensions or features that change for

a given design specification rather than the number of parts that need control. Chapter 4 deals with the time required for extracting the parameters that need to be controlled, which is Stage III of the DW RBD program development process. Time estimation models are presented for individual phases that are involved in the Stage III. An equation to estimate time for completing this stage is presented in this chapter, which is dependent upon the number of assemblies, parts, dimensions, features, and custom properties that are to be controlled. Chapter 5 deals with the Stage IV of DW programming of the RBD program development process. It presents the time estimation models for completing individual phases of: creating the user interface, specifying the product related data, and writing rules for the parameters that are extracted in Stage III. This chapter ends by presenting an equation that can predict the time for completing Stage IV of DW RBD program development process and is dependant upon the number of extracted parameters that need DW control. Chapter 6 deals with time estimations for debugging and testing stages (Stage V) of the DW RBD program development process. In estimating the times for this stage, a major assumption from the literature is considered that 35-40% of software development time is spent in testing the program. This assumption can be applied here because the same testing methods of the software programs can be used for the DW RBD programs. In DW RBD terms, the development time is the summation of time taken for extracting parameters (Stage III) and DW programming (Stage IV).

Finally, the estimated total time for DW RBD program development process can be obtained as the summation of time estimations models for completing Stage II thru Stage V. These times are dependent upon the parameters that need to be controlled in a particular grid product and should be an accurate estimate for those parameters. These models for time estimations are demonstrated by actually developing a DW RBD program using other grid product, and are used for validating the accuracy of the models.

Validation

In Chapter 7, a grid product is selected from Table 1.1 in demonstrating the time estimation calculations for developing a DW RBD program. This product was chosen because it is one of the major

grid assemblies, which provides an opportunity to study the effect of many parameters in terms of different assemblies, components, dimensions, features, and custom properties. The other reason for selecting this product is that the time estimation models in this research are obtained with the help of the other grid assemblies in Table 1.1. The selected product is also taken from the same table and was never programmed for DW RBD. Thus, this product is a potential candidate that can be used in validating the time estimation models.

With this product as the subject of demonstration, the time for the DW RBD program development process, Stages II-Stage V, was estimated as 39 hours. This estimated time was compared against actually programming the grid product in DW. The actual time for DW programming was obtained as 36 hours. The percentage error was calculated between the actual and estimated time for building the DW RBD program and was obtained as 18. 2 %. Therefore, with 80 % confidence limits, one can say that the actual programming time falls with in 20% of the estimated programming time and hence proves the accuracy of the proposed time estimation models for the DW RBD program development process of HI grid assemblies.

Observations

Close observations of the individual stages in the DW RBD program development process reveal that the second stage of knowledge acquisition is more time consuming and involves more uncertainties than the remaining stages. One reason for this type of behavior is that the knowledge engineer's work is dependant upon designers' experience for capturing the product information. This knowledge engineering stage can be eliminated by training the designer to use the DW RBD system instead of a programmer or a knowledge engineer. The other reason for taking more time for Stage II, is preparation of the design document. The preparation of the design document is a tedious task since the knowledge engineer has to explain all the assemblies, parts, dimensions, and features that change with design specifications. This process includes preparing CAD drawings and annotating parameters such as dimensions and features in detail. Preparing a design document can be avoided in the DW RBD programming stage, when a designer

produces this document for every new design that is produced in SW. Therefore, if the design documents are present beforehand, the knowledge engineer can capture most of the design knowledge from them without depending upon the designer. This results in a significant amount of time saving in knowledge acquisition.

The knowledge engineer may prepare a rules document rather than a design document. The former document can be considered as the condensed and of the latter, which emphasizes the rules for driving parameters. Preparing a rules document can be considered as an intellectual task in coming up with the constants and variables or sub-rules that might be used in actual programming in DW. The time for preparing the rules document can be minimized by using the template documents. The figures from the design document can also be directly copied into rules document without creating them from scratch, which saves a considerable amount of time.

Getting an approval for the design document is also dependant upon other people, which raises uncertainties and, once again is time consuming. Once the design document is approved, writing rules in DW for controlling the product parameters takes less time than capturing the design information about the product.

Contributions

The significant contributions in conducting this research are presented as follows:

- As part of this study, the SW design processes of 18 out of 32 grids products shown in Table 1.1 were automated by writing DW RBD programs. This helps HI to quickly generate new variant CAD models for these grid orders in the future, which ultimately results in significant savings with respect to time. Since time is money, savings in time results in savings in money [38].

- This study developed a DW RBD method for automating the grid design process of HI. This helps ETO companies to follow the proposed method for automating the design process for their variant products. This research also helped the other student who was part of this project in better understanding

about the RBD programming process. It helped in understanding the importance of each stages in DW RBD programming and the parameters that affect these individual stages.

- Valuable product knowledge of grid assemblies was captured and design documents were prepared as part of HI IRBD project. The design document serves as a reference source for grid products and secures the product knowledge in HI design databases. This design document can be used as a standard template for DW RBD programming for other products of HI.

- This research helps any other programmer who is not familiar with DW RBD programming to follow these steps for automating the design process. This research also helps people to appreciate the importance of parameters that affect the DW RBD program development process, and can be used to follow the method while creating the programs. Simultaneously, they can be asked to record the times for significant activities.

- Best practices for SW CAD models were presented in this thesis, which are the one of the outcomes of this research. These best practices help HI to build SW models to build DW RBD programs for products of other departments.

Limitations of the proposed time models

In this research, proposed time models were developed with several assumptions and are valid for HI grid products. More work is required for refining these models to include uncertainties and allowances in extending these models for other product domains. These models did not take into consideration the effect of the learning curve of the programmer. In the HI IRBD project, the student who was trained for about 4 months on DW had very little chance to record the activities related to DW RBD programming. This is because a lot of time was spent in building the SW CAD models rather than DW programming and ultimately resulted in coming up with best practices in CAD modeling for DW program development. The other reason for not studying the effect of the learning curve is the DW software platform. Since DW is continuously upgrading the software, the effect of the learning curve was not accurately studied in HI the IRBD project.

While developing the time models, the effect of computer hardware configurations such as processing speed or memory was not considered. This results in uncertainties in the time estimation calculations when programmers use different systems for developing RBD programs.

The other limitation is that the models are developed by taking the recorded times of one programmer who developed the programs for the HI IRBD project. It did not take into consideration the recorded times of other programmers. More work needs to be done for generalizing these time estimation models.

The major limitation of these models is that they are dependent upon the low-level parameters for estimating the times for the RBD program development process. In general, the company personnel will not have an accurate estimate of the low-level parameters before preparing a design document. In other words, they know the high-level information about the assemblies and their corresponding components that require RBD programming and are ignorant of the low-level details such as number of dimensions or features. This is a major drawback of these time models. Therefore, more research is required for developing a method for estimating the total parameters, and percentage of parameters that change for a given product in a given type of manufacturing environment. Addressing this issue is left for future research.

Future Work

Refining these models for addressing the above limitations is identified as immediate future work. The specific limitations are as follows:

The time models are based on several assumptions and are applicable for HI grid products. The demonstration study proved that these models could be used for estimating the time for the RBD program development process. In general, by programming one product it is difficult to reach to a conclusion that these time models are accurate. Therefore, it is suggested that more products need to be programmed by the proposed approach and the times should be recorded for the corresponding activities. These recorded

observations can be used in refining the equations by adding this new data to the data that was used previously. This helps in refining and validating the proposed model equations.

This research was done on HI grid products as part of the case study. Therefore, the models may or may not be applicable for other product domains. In answering this hypothesis, a product from another domain needs to be selected and time estimations should be done by using time models presented in this research. Later, it should be programmed for DW RBD automation. The estimated time should be compared with the actual time. More work is needed to see the effect of this comparison to deal with other products that are different from grid products, and the questions that should be asked during this process are:

1. Do the proposed time estimate models work for other types of product?
2. What is difference between the estimated time and actual time and what things need to be addressed in reducing this difference.
3. How can these time models be extended to other product domains? For example, multiplying with a scale factor, or any complex calculations?

The effect of the learning curve on time estimation models should be considered in training a designer to use the RBD systems. From the previous observations, it is identified that the knowledge engineer needs to be eliminated to reduce the time for developing a RBD program. The top-level management personnel cannot use these equations in present form, since it does not take into the effect of learning curve of the designer as part of eliminating the knowledge engineer or programmer.

What are the allowances and uncertainties that need to be addressed in these time estimation models? Major work is needed in addressing the allowances and uncertainties that arise when dealing with different hard ware configuration systems when used by multiple programmers. The same product should be programmed by different programmers and the time obtained can be used for getting the average value.

Different products should be programmed by different programmers and all of the recorded times for their activities should be considered for generalizing the proposed models.

The extension of these models will be identifying the percentage of low-level parameters that change in a particular product together with the total number of parameters for a given product. As mentioned before, there is no idea of these low-level parameters before preparing a design document. Therefore, more research needs to be conducted in identifying the percentage of parameters that change in a given type of product. This idea is presented for the sample ETO products as shown in Table 8.1:

Table 8.1: Percentage of low level of parameters

Sl. No	Type of product	Total parameters (Avg number)	% of parameters	Parameters that are of interest
1	HI Grid products	25	32%	8
2	HI Conveyor Products	175	46%	80
3	Typical book shelves	30	84%	25
4	Office partitions	10	40%	4
	So on for all ETO products.			

Closing thoughts

Intoday's fast moving job markets, with the difficultly of producing quality products RBD comes into the rescue for ETO industries. The RBD process helps in reducing product development time by automating the low-value mundane design activities like editing models, producing drawings, preparing bill of materials, and preparing quotations. It also helps to reduce the dependency of an expert, for designing regular variant products relieving this designer from doing low value design activities. This way the designer will have more time to think in coming up with innovative ideas for designing better products. The major of advantage of RBD systems is that, it helps in preserving the corporate design knowledge, which as the sixth and most important production factor besides people, machines, material, money, and information. Therefore, this thesis suggests that RBD systems should be used for designing variant products.

LIST OF REFERENCES

1. Aihu, W., K.O.C. Bahattin, and N. Rakesh, *Complex assembly variant design in agile manufacturing. Part I: System architecture and assembly modeling methodology.* IIE Transactions, 2005. **37**: p. 1-15.

2. Aihu, W., K.O.C. Bahattin, and N. Rakesh, *Complex Assembly Variant Design in Agile Manufacturing. Part II Assembly Variant Design Methodology.* IIE Transactions, 2005. **37**(1): p. 17-33.

3. Barker, V.E., D.E. O'Connor, J. Bachant, and E. Soloway, *Expert systems for configuration at Digital: XCON and beyond.* Communications of the ACM, 1989. **32**(3): p. 298-318.

4. Boothroyd, G. and P. Dewhurst, *Product Design for Manufacture and Assembly*. 2002: Marcel Dekker.

5. Boucher, T., *The rules are meant to be broken*, in *Product management*. 2005, Time-Compression Techonologies: MA, USA.

6. Brousell, D.R., *Rewriting the rules for ETO manufacturers*. 2005, Managing Automation Research services: MA, USA.

7. Brown, J., *Rules-driven Product Management: Achieving greater value from Customer-Specified Products*. 2005.

8. Buckley, C. (2004) *Eating an Elephant, One Bite at a Time*. Rational.

9. Bullinger, H.J., J. Warschat, and D. Fischer, *Rapid product development\—an overview.* Comput. Ind., 2000. **42**(2-3): p. 99-108.

10. CIM data, *The Value of Rules-Driven Product Management*. 2005.

11. Dixon, J.R. and C. Poli, *Engineering Design and Design for Manufacturing: A Structured Approach.* Field Stone, Conway, MA, 1995.

12. DriveWorks (2007) *DriveWorks Help Manual*.

13. Fowler, J.E., *Variant design for mechanical artifacts: A state-of-the-art survey.* Engineering with Computers, 1996. **12**(1): p. 1-15.

14. Friedman-Hill, E., *Jess in Action: rule-based systems in Java*. 2003: Manning Publications Co.

15. Friedman-Hill, E.J., *Collecting Knowledge for Rule-Based Applications*. 2003.

16. Gillam, A. *A Knowledge-Based Systems Engineering Tool for Global Design Tradeoffs*. in *AI, Simulation, and Planning in High Autonomy Systems, 1993. 'Integrating Virtual Reality and Model-Based Environments'. Proceedings. Fourth Annual Conference*. 1993. Tucson, AZ, USA.

17. Gu, P. and S. Sosale, *Product modularization for life cycle engineering.* Robotics and Computer-Integrated Manufacturing, 1999. **15**(5): p. 387-401.

18. Gupta, S.K., W.C. Regli, D. Das, and D.S. Nau, *Automated manufacturability analysis: A survey.* Research in Engineering Design, 1997. **9**(3): p. 168-190.

19. Hailpern, B. and P. Santhanam, *Software debugging, testing, and verification.* Integration, 2001. **3**(x9): p. x3.

20. Huang, C.-C., *Modularity in Design of Products and Systems*, in *IEEE Transactions on Systems, Man, and Cybernetics*. 1998, IEEE. p. 66-78.

21. Huang, G.Q. and G.Q. Huang, *Design for X: Concurrent Engineering Imperatives*. 1996: Springer.

22. Hvam, L., M. Malis, B. Hansen, and J. Riis, *Reengineering of the Quotation Process: Application of Knowledge Based Systems.* Business Management Process Journal, 2004. **10**(2): p. 200-213.

23. Institute For Defense Analyses Alexandria, V.A., K.J. Richter, K. Blemel, J. Fink, C. Grewe, and A. Hashmi, *The Aries Project for Rule-Based-Design Knowledge Acquisition.* 1995.

24. Jacob, R. and J. Froscher, *A software engineering methodology for rule-based systems*, in *IEEE Transactions*. 1989. p. 173-189.

25. Kingston, J., *RULE-BASED EXPERT SYSTEMS AND BEYOND: AN OVERVIEW.*

26. Koubek, R.J. and W. Karwowski, *Management of complex projects as cooperative task: Manufacturing Agility and Hybrid Automation-I.* 1996: IEA Press.

27. Lee, J.Y. and K. Kim, *Geometric reasoning for knowledge-based parametric design using graph representation.* Computer-Aided Design, 1996. **28**(10): p. 831-841.

28. Lee, J.Y. and K. Kim, *Generating Alternative Interpretations of Machining Features.* The International Journal of Advanced Manufacturing Technology, 1999. **15**(1): p. 38-48.

29. Lewis, C. and J. Rieman, *Task-centered User Interface Design: A Practical Introduction.* 1993.

30. Li, X., Quality time-What's so bad about rule-based programming? Software, IEEE, 1991. 8(5): p. 103-105.

31. Lin, A.C., S.Y. Lin, and S.B. Cheng, *Extraction of manufacturing features from a feature-based design model.* International Journal of Production Research, 1997. **35**: p. 3249-3288.

32. McNamara, C. *General guidelines for conducting interviews*. 1997-2000 [cited 2007 June 15]; Available from: http://www.managementhelp.org/evaluatn/intrview.htm.

33. Monedero, J., *Parametric design: a review and some experiences.* Automation in Construction, 2000. **9**(4): p. 369-377.

34. Myung, S. and S. Han, *Knowledge-based parametric design of mechanical products based on configuration design method.* Expert Systems with Applications, 2001. **21**(2): p. 99-107.

35. O'Driscoll, M., *Design for manufacture.* Journal of Materials Processing Technology, 2002. **122**(2-3): p. 318-321.

36. Pahl, G.G. and W. Beitz, *Engineering Design: A Systematic Approach*. 1996: Springer.

37. Pawlak, Z., *Reasoning about Data-A Rough Set Perspective.* Lecture Notes in Artificial Intelligence, 1998. **1424**: p. 25–34.

38. Payne, J.W., J.R. Bettman, and M.F. Luce, *When time is money: Decision behavior under opportunity-cost time pressure.* Organizational Behavior and Human Decision Processes, 1996. **66**(2): p. 131-152.

39. Prasad, B. and J. Rogers. *A Knowledge-Based System Enigeering Process for Obtaining Engineerig Design Solutions*. in *ASME 2005 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference* 2005. Long Beach, Californica, USA: ASME.

40. Production, A., *Inventory Control Society.(1987).* APICS Dictionary, 6th ed. Alexandria, VA: APICS.

41. Rossignac, J.R., P. Borrel, and L.R. Nackman, *Interactive design with sequences of parameterized transformations*, in *Intelligent CAD systems II: implementational issues*. 1989, Springer-Verlag New York, Inc. </pre> </body> </html>. p. 93-125.

42. Siddique, Z. and K. Boddu, *A CAD Template Approach to Support Web-Based Customer Centric Product Design.* Journal of Computing and Information Science in Engineering, 2005. **5**: p. 381-387.

43. Siddique, Z. and Z. Yanjiang, *Automatic Generation of Product Family Member CAD Models Supported by a Platform Using a Template Approach*, in *ASME Design Engineering Technical Conference and Computer and Information Engineering Conference*. 2002: Montreal, Canada.

44. Vajna, -.I.S., *Approaches of Knowledge-Based Design*, in *Design Engineering Technical Conferences and Computer and Information in Engineerin Conference*. 2003, ASME: Chicago, Illinois, USA.

45. Whitney, D.E., *Physical Limits to Modularity*. 2004, MIT.

46. Whittaker, J.A., *What is software testing? And why is it so hard?* Software, IEEE, 2000. **17**(1): p. 70-79.

47. Xie, S.Q. and Y.L. Tu, *Rapid one-of-a-kind product development.* International Journal of Advance Manufacturing Techonology, 2006. **27**: p. 421-430.

48. Yin, R.K., *Case Study Research: Design and Methods*. 1985: Sage Publications Inc.

49. Zalik, B. and N. Guid, *Combining Parametrically Represented Geometrical Objects using Constraints.* IEEE, 1994.