Application of Axiomatic Design to Rapid-Prototyping Support for Real-Time Control Software

by

Hiroshi Igata

B.S., Mechanical Engineering (1984)
The University of Tokyo

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

Author ....
Department of Mechanical Engineering
May 29, 1996

Certified by ....
Nam P. Suh
The Cross Professor of Manufacturing and Head
Department of Mechanical Engineering
Thesis Supervisor

Accepted by ....
Ain A. Sonin
Chairman, Department Committee on Graduate Students

Application of Axiomatic Design to Rapid-Prototyping Support for Real-Time Control Software

by

Hiroshi Igata

Submitted to the Department of Mechanical Engineering
on May 29, 1996, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Axiomatic design is applied to design of real-time control software. As a case study of software design for on-board computers of automobiles, the software of the current anti-lock braking system (ABS) is analyzed based on axiomatic design. It is shown that the current design is a coupled design and that it can be decoupled to improve the performance of the anti-lock braking system software.

In developing real-time control systems, it is necessary to do rapid-prototyping. Existing software development methods are not useful as rapid-prototyping support tools. Axiomatic design is found to be the necessary powerful support tool by explicitly showing the inter-dependencies between program modules and by being able to model both the hardware and the software of a given design.

Finally, a new development support system that uses the customer attribute (CA) database and addresses specific problems such as missing rationale is suggested based on axiomatic design.

Thesis Supervisor: Nam P. Suh
Title: The Cross Professor of Manufacturing and Head
Department of Mechanical Engineering

Acknowledgments

First I would like to thank my advisor, Professor Nam P. Suh for his support, guidance, understanding and most importantly his patience through the duration of this research.

I would like to thank Mats Nordlund, Derrick Tate and Vigain Harutunian for their valuable suggestions. The discussion with them always inspired me to try new ideas.

I would also like to thank Professor Yotaro Hatamura of the University of Tokyo for providing me a great opportunity to study at MIT.

Thanks also to the people in Toyota Motor Corporation for their support.

Last but not least, I would like to thank my wife Mariko and my son Yosuke for their support, encouragement and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Modern passenger cars are equipped with 20 to 50 electronic control units and the number is increasing.

These electronically controlled subsystems can be categorized into two groups. One group contains the subsystems that are relatively independent from the dynamics of the mechanical systems of automobiles. The navigation guide systems, the air-bags, and the door lock and power window control systems are classified in this group. The other group comprises the subsystems which are tightly integrated with the mechanical systems of automobiles. This group contains the anti-lock braking systems, the traction control systems, the engine and transmission management systems, the suspension control systems, etc. This group is also characterized by its development process. As the input conditions to the actual vehicles are too diverse to be known beforehand, the development process is heuristic in most cases.

As the electronic controlled subsystems have become ubiquitous, a problem has arisen. The complexity of each subsystem and the combinations of subsystems are spinning out of control especially in the latter group, i.e., the subsystems which are integrated with mechanical systems. Combined with the fact that most engineers in automotive companies are from the mechanical field, this problem tends to be interpreted as a pure software problem.

Though there have been attempts to apply generic software development methods such as structured analysis and structured design (SA/SD) [9] to the software development of the electronic controlled subsystems, they did not produce significant results especially for the systems in the latter group. This is due to the mistakes of considering the problem a pure software issue and ignoring the actual task process of the designers. A design method which is capable of modeling the hardware and the software equally is needed.

Axiomatic design [7] is capable of treating software and hardware problem from a systems perspective. There are many case studies ranging from pure mechanical systems to software problems. In this thesis, axiomatic design is applied not only to the analysis of an existing electronic control subsystem to suggest an improvement on it but also to its design process to address the problem described above.

1.2 Research Objective

This research work is divided into two parts. One is to model a computer controlled subsystem using axiomatic design. In this part, one of the most popular software development method, object modeling technique (OMT) [6] is selected to be compared with axiomatic design. The anti-lock braking system is taken up as an example, its problem is clarified and a solution is suggested. The other part is to model the rapid-prototyping design process of the real-time controlled subsystem. The goal is to present a feasible model of the development process and to suggest an improved process and its support system to help solve the problems caused by the current process.

1.3 Thesis Overview

The second chapter of this thesis outlines the basic concept of axiomatic design. An attempt to use the Internet world wide web (WWW) as an instruction tool, which is accessible literally world wide, is included.

The third chapter describes how software is modeled in terms of axiomatic design. Precedents are examined and a feasible way to model software and hardware interchangeably is presented.

The fourth chapter presents a comparison of axiomatic design and object modeling technique (OMT) focused on the application to rapid-prototyping support for real-time control software.

The fifth chapter shows the structural decomposition of the anti-lock braking system. Problems with the current system are clarified and improvements are suggested.

The sixth chapter presents the analysis of the current development process of the real-time control software.

The seventh chapter is an attempt to model an ideal development process and design a support system for it based on the analysis in the previous chapter.

The eighth chapter contains the conclusions and suggestions for future research.

Chapter 2

Introduction to Axiomatic Design

In this chapter, the key concepts of axiomatic design are reviewed as a preparation for discussions in later chapters. More information is in the book by Suh [7] and the paper by Gebala and Suh [1].

This chapter also includes an attempt to teach the concepts using the Internet world wide web (WWW) which helped the author get some feedback from outside the United States.

2.1 Domains

In axiomatic design, the design process is defined as the mapping between the four domains shown in figure 2-1.

The relationship between the domains is that the domain on the left is "What we want" and the domain on the right is "How we will satisfy what we want" which is shown in figure 2-2.

The design process is not one-way. The design solution evolves iterating through the feedback loop shown in figure 2-3.

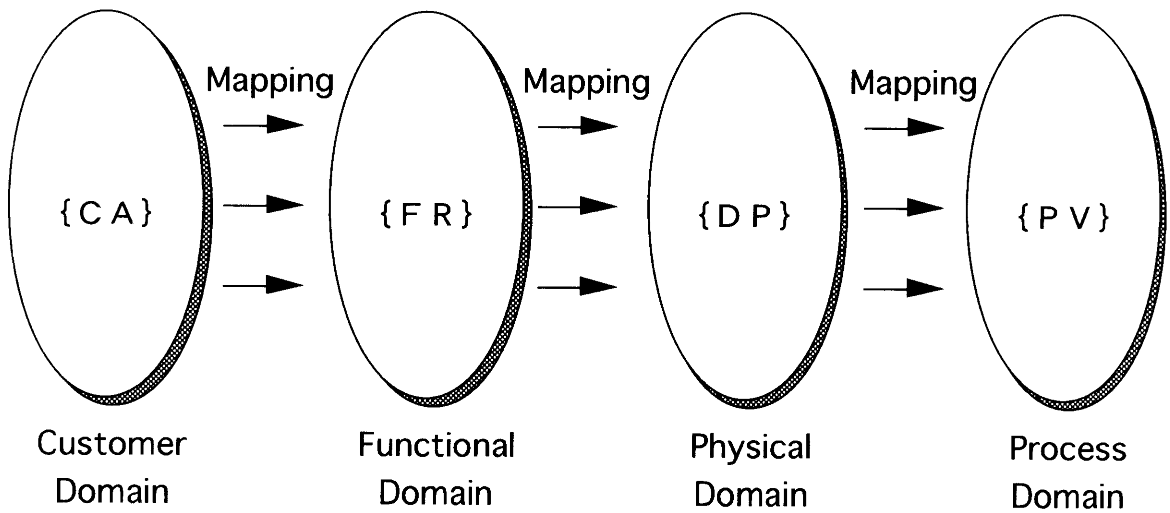

Figure 2-1: Concept of domains and mapping (from Gebala [1])

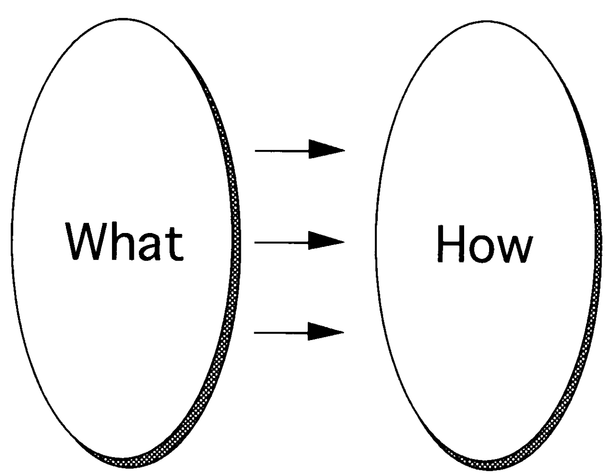

Figure 2-2: What and how (after Kim [3])

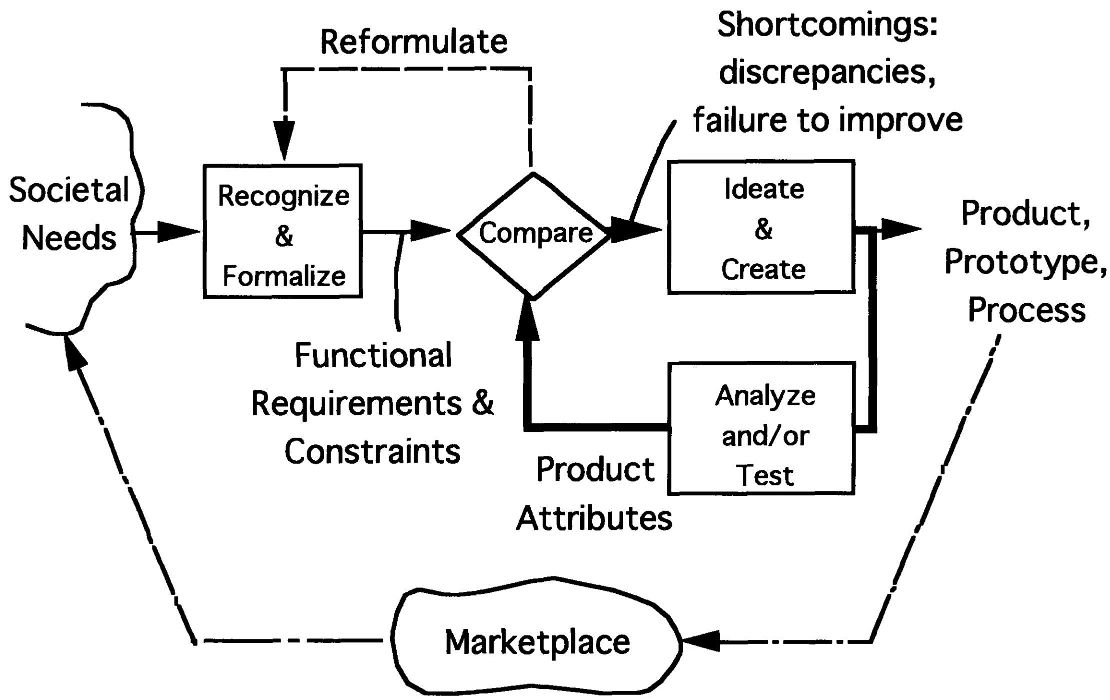

Figure 2-3: Feedback loop of design process (from Suh [7])

2.2 Functional Requirement (FR)

Functional Requirement (**FR**) is defined as *the minimum set of independent requirements of the functional domain that completely characterize the design goals, subject to constraints.* FRs are derived to satisfy customers' needs that characterize the customer domain.

Constraints (**C**s) are defined as *the limiting values or conditions or bounds a proposed design solution must satisfy.* Cs are different from FRs in that Cs do not have to be independent from other Cs or FRs.

DPs are Design Parameters in the physical domain and **PV**s are Process Variables in the process domain. Characteristics of the four domains are shown in table 2.1.

Table 2.1: Characteristics of the four domains (after Gebala [1])

Domains {Vector}	Customer Domain {CA}	Functional Domain {FR}	Physical Domain {DP}	Process Domain {PV}
Manufacturing	Attributes which consumers desire	Functional requirements specified for the product	Physical variables which can satisfy the functional requirements	Process variables that can control design parameters
Material	Desired performance	Required properties	Micro-structure	Processes
Organization	Customer satisfaction	Functions of the organization	Programs or offices	People and other resources that can support the programs
System	Attributes desired of the overall system	Functional requirements of the system	Machines or components, sub-components	Resources (human, financial, materials etc.)

2.3 Axioms

The most important concept in axiomatic design is the existence of the two design axioms, which must be satisfied during the mapping process to come up with acceptable design solutions. The first design axiom is known as the Independence Axiom and the second axiom is known as the Information Axiom.

2.3.1 The Independence Axiom

Axiom 1 (The Independence Axiom)

Maintain the independence of the functional requirements.

In axiomatic design, FRs and DPs are expressed in terms of vectors. The relation between FR and DP vectors is given as the second order tensor **Design Matrix**. The equation of FR vector, DP vector and the Design Matrix is called **Design Equation**. In the following examples, X is a non-zero value, signifying the fact that there is a strong relationship between an FR and the corresponding DP.

The most desirable design is the **Uncoupled Design**. Each DP corresponds to only one FR as shown below.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

The design one should avoid is the **Coupled Design**, where elements on both sides of the diagonal in the design matrix are non-zero. Each DP affects not only the corresponding FR but other FRs, thus making it impossible to satisfy each FR independently.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & X & 0 \\ X & X & X \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

A third case is called **Decoupled Design**. The design matrix is lower or upper triangular. This decoupled design is an acceptable solution as independence can be maintained by adjusting the DPs in a proper order.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

2.3.2 The Information Axiom

Axiom 2 (The Information Axiom)

Minimize the information content.

In figure 2-4, the vertical axis indicates probability density while the horizontal axis indicates the value FR. **Design Range** is defined as the designer's specification of acceptable values for FRs. **System Range** is the range the system is capable of delivering to meet the FRs or DPs. **Common Range** is the overlap of the Design Range and the System Range. The information content in axiomatic design is the ratio of the System Range to the Common Range. The value is converted through logarithm with base 2, so that multiple requirements can be treated as a sum instead of a product.

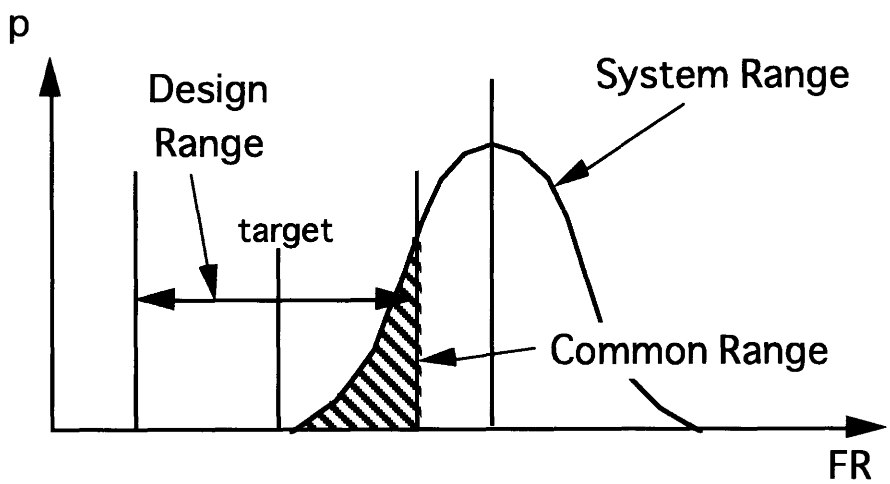

Figure 2-4: Information content (from Suh [7])

$$I = log_2\left(\frac{\text{System Range}}{\text{Common Range}}\right)$$

If the Design Range and the System Range do not overlap, the information content is infinite as the denominator is zero. On the other hand, if the System Range is fully covered by the Design Range, the Common Range equals the System Range, thus the information content is zero. In other words, the information content in axiomatic design is the amount of information necessary to realize FRs by DPs, and consequently, the smaller the better.

2.4 Corollaries and Theorems

All of the corollaries and theorems of axiomatic design are shown in appendix A (from Suh [7]).

2.5 Zigzagging and Hierarchy

FRs and DPs are decomposed into hierarchical structure through zigzagging between the functional domain and the physical domain as shown in figure 2-5. One cannot

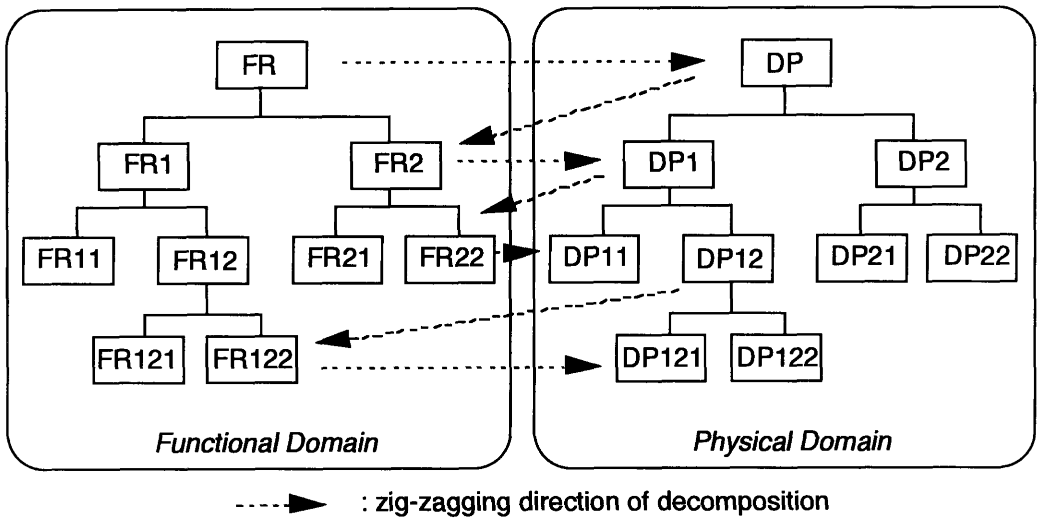

Figure 2-5: Hierarchical tree structures of FRs and DPs (from Kim [3])

proceed to a lower level until one decides the DP which corresponds to the FR at a given level.

2.6 Instruction of Axiomatic Design using the world wide web

Figure 2-6 shows a modified version of this chapter posted on the world wide web to provide the basic knowledge of axiomatic design to whoever is interested in it in the world. The URL (universal resource locator) address is:

http://web.mit.edu/igata/www/axiomatic1_e.html

Providing this kind of information is one of the most efficient uses of the web as one can provide the most up-to-date information with minimum effort. Future enhancements include an interactive guide through case studies. The author also set up a Japanese version of the same contents so that he can get some feedback from Japan (figure 2-7).

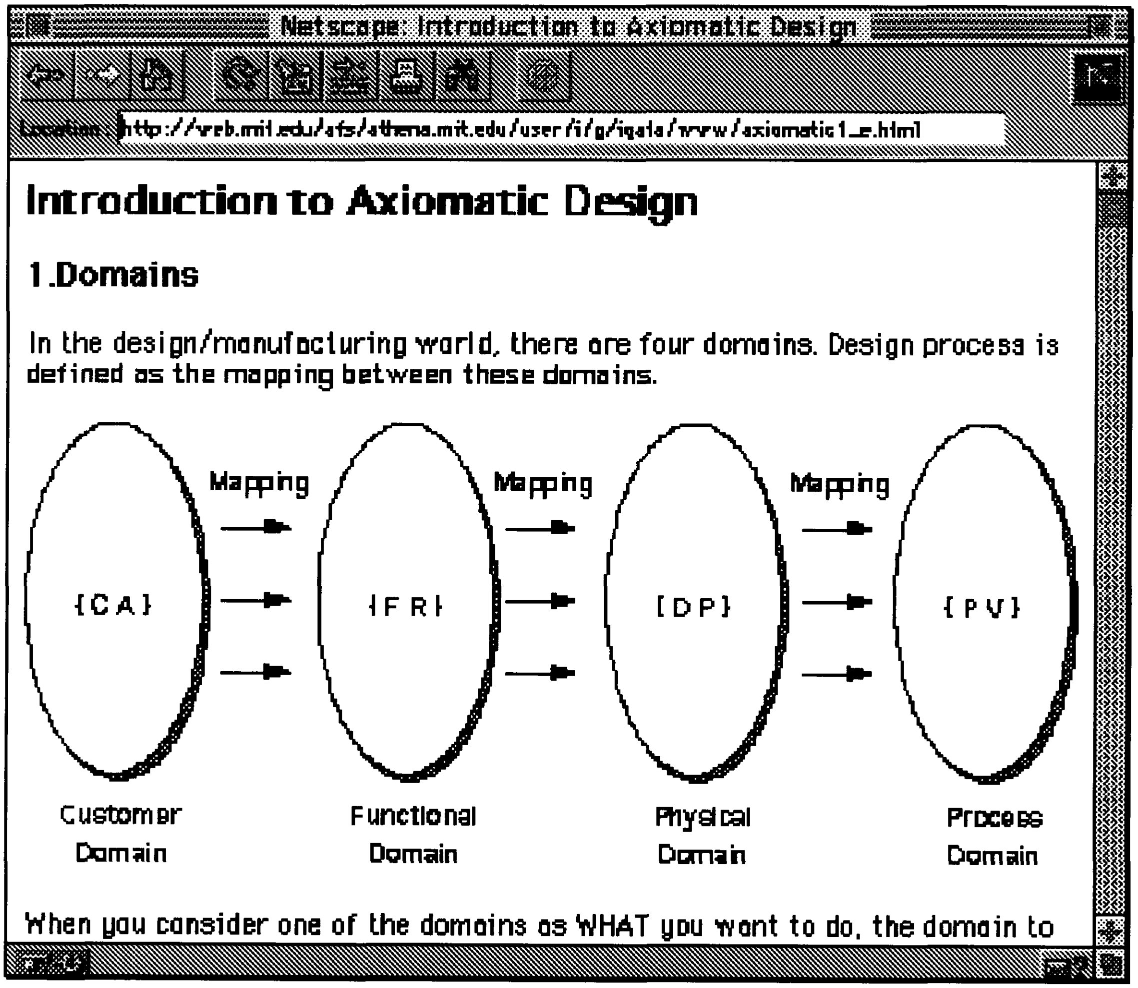

Figure 2-6: Introduction to axiomatic design on the web

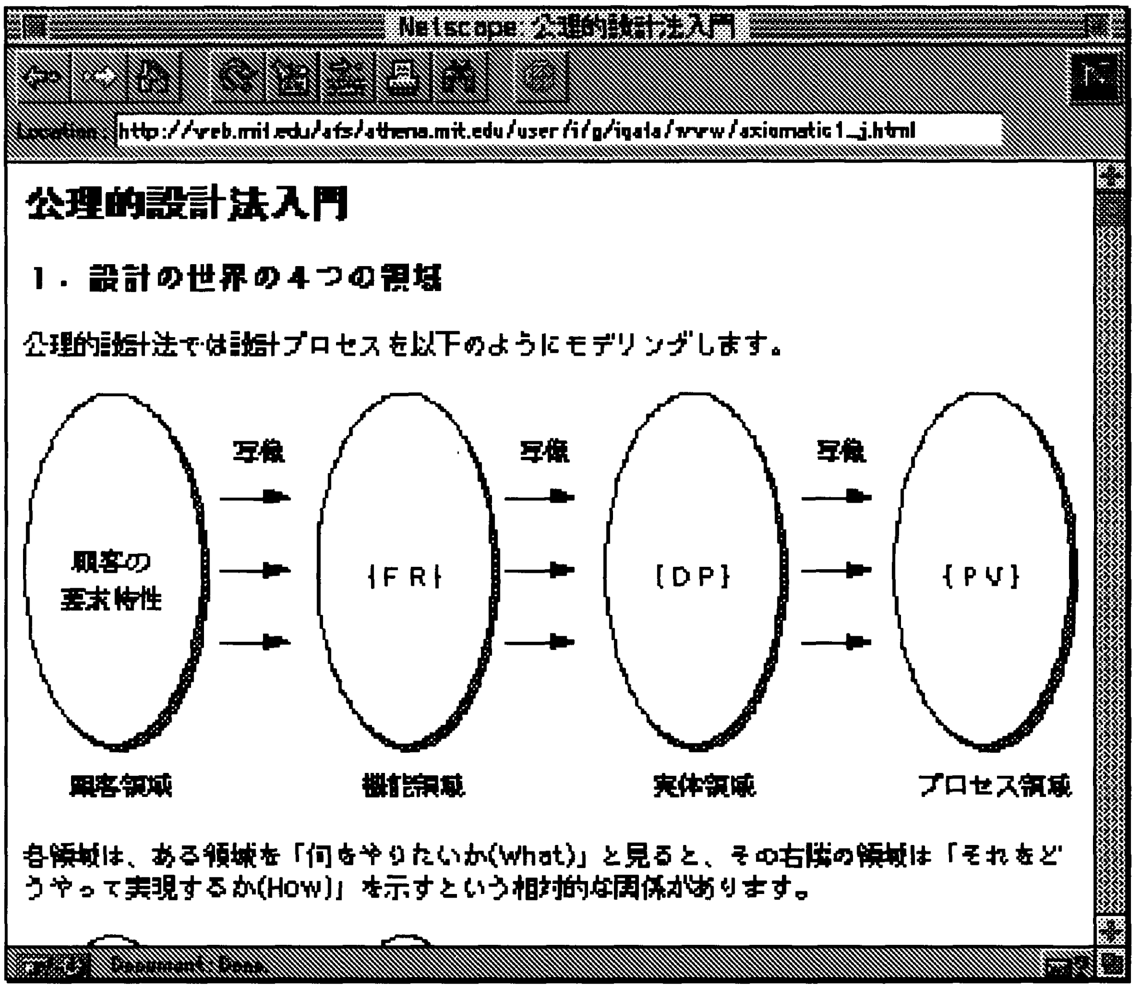

Figure 2-7: Japanese version of the web page

Chapter 3

Modeling of Software using Axiomatic Design

When one has to modify software, it is most important to know the scope that is affected by the alteration. If one can explicitly describe the inter-dependencies between program modules in terms of a design matrix or a design equation, that would make a great tool to help programmers understand this scope of influence. In this chapter, the design equation of software is defined.

3.1 Software Development Approaches

Rumbaugh [6] states that there are basically two approaches to software development, i.e., a life-cycle approach and a rapid-prototyping approach. With the life cycle approach, software design process is one-way, that is, software is fully specified, fully designed, then fully implemented. In contrast, with the rapid-prototyping approach, a small portion of the software is initially developed and evaluated through use. And the software is gradually made robust through incremental improvements to the specification, design and implementation. Although the life-cycle approach is preferred by some software development methodologies, there are also many cases in which some of the system behavior and characteristics are not known a priori, and then the rapid-prototyping approach is much more suitable.

Software for the on-board computers of automobiles is a typical example of the latter. The first prototype is generally very simple. Various sub-modules are added through development to improve performance in particular conditions and to make the system more robust. The rapid-prototyping means going through the "Formalize-Compare-Create-Analyze" loop or the zigzagging path as quickly as possible.

3.2 Precedents of Axiomatic Approach to Software Design

3.2.1 Kim, Suh and Kim

Kim, Suh and Kim [3] was the first attempt to apply the axiomatic approach to design software. In this reference, the following assignments were suggested.

FR : Outputs

DP : Inputs

PV : Subroutines

The software code is considered to be a set of design matrices.

In this thesis, this model is not adopted for the following reason:

Dividing software into relatively small modules is the key concept to treat complicated software problems. If one uses this "divide and conquer" method, it sometimes happens that certain data is calculated in a module and then corrected in another module. In this case, it is hard to setup a design equation as there is only one FR (the output data) and two DPs (two sets of "inputs" for calculation and correction). This is caused by the fact that the description of each module which includes input and output is divided into FR (output) and DP (input).

3.2.2 Case Study at Saab

A case study by Stefan Svensson in software engineering of Saab Military Aircraft [8] uses following model.

CR : Customer Requirements

FR : Input $->$ Output

DP : Algorithm

PV : Routines

The case study says that the algorithms in the design domain (physical domain) answer questions like "How is the input/output relationship solved?" Though this claim is reasonable, there are a couple of problems when one wants to set up a design equation.

First, when one changes a DP, corresponding set of input/output changes quite often, e.g., changing the feedback control type from classical PID control to modern control requires changes not only in algorithm but also in input/output.

Second, when one sets up a design equation, how does one define a relation between a set of input/output and unrelated algorithm? Off diagonal elements in the design matrix do not make sense.

As for PV, the case study redefines Process Domain as Realization Domain and assigns program routines to Realization Variable (RV) to fit software problems. It claims that the routines in the Realization Domain answers questions like "How is the algorithm implemented?" As actual routines inevitably include both input/output and algorithm, the mapping between DP and RV is also hard to define.

3.2.3 Baskin

Baskin [4] suggested following assignments.

CA : Problem Model

FR : Analysis Model

DP : Design Model

PV : Implementation Model

The mapping between each domain is named “realized-by”.

The four models shown here are more of four stages of software development with the life-cycle approach, rather than domains. Therefore, this idea works most effectively with the life-cycle approach but not so much with the rapid-prototyping approach.

James Rumbaugh’s Object Modeling Technique (OMT) uses three kinds of models to describe the FRs (the analysis model):

1) The object model, describing the objects in the system and their relationship.
2) The dynamic model, describing the interactions among objects in the system.
3) The functional model, describing the data transformations of the system.

The fact that three models are used to describe the FRs (the analysis model) makes it hard to model the design process as mapping [6].

On the other hand, in the object design stage of OMT, the analysis model is copied from the analysis domain to design domain, then modified by adding details for implementation and changing structure to make the most of the object oriented technique in the design domain alone. Therefore, the analysis model (FR) and the design model (DP) usually have different structures, thus making it hard to represent it in terms of mapping using matrix expression.

Application of object oriented technology to real-time software is further discussed in chapter 4.

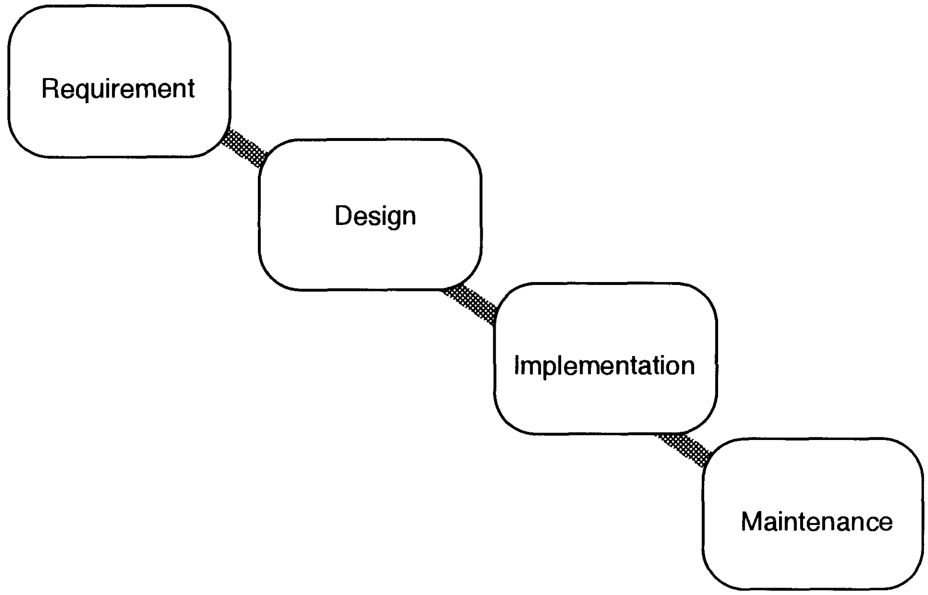

Figure 3-1: Waterfall model (from Yourdon [9])

3.3 Proposed Design Equation of Software

Figure 3-1 shows the traditional waterfall model which is widely used for modeling the software development process in SA/SD (structured analysis / structured design) method [9]. If one follows this convention, it is natural to assign CA, FR, DP and PV for software as follows.

CA : User's Needs
FR : Specifications or Requirements of Software
DP : Program Codes
PV : Machine Codes

With axiomatic approach, software designers should first formalize customer needs into a set of FRs (specifications), then write program codes (DPs) based these. Through numerous tests using simulator or an actual machine, FRs and DPs are repeatedly modified until the system reaches the desired level of performance and robustness. In terms of axiomatic design, the performance means to make sure that the set of FRs chosen is appropriate and the robustness means that the design matrix is at least decoupled.

Note that, as discussed in the previous section, the generic term specification is usually a partial description of the problem. Consequently, specification and program code have different structure making mapping between them difficult. In other words, there is a relationship like following equation.

$$\text{Specification} + \text{Implementation Detail} = \text{Program Code}$$

Therefore, in this thesis, FR is defined as "enhanced specification which includes all the details necessary for implementation" and DP as "corresponding program code".

CA : User's Needs

FR : Detailed Specifications of Software

DP : Program Codes

PV : Machine Codes

Using these definitions, a design equation for software is set up as follows.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & ? & ? \\ ? & X & ? \\ ? & ? & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

The diagonal elements of the design matrix is non-zero by definition. Then, what about non-diagonal elements? Each FR is defined in terms of:

{Inputs}: A set of inputs

{Outputs}: A set of outputs

{Procedures}: The way {Outputs} are created using {Inputs}.

Note that each DP should have the same structure as the FR. If there is one or more common elements in the {Inputs} of FR_n and in the {Outputs} of DP_m, the corresponding element A_{mn} in the design matrix is non-zero.

3.4 Example

Typical ABS (Anti-lock Braking System) software consists of about 50 software modules. One of them, "255 : Quick Apply at the End of Gradual Apply Sequence", is a module which is responsible for cases like vehicle going from icy road to dry asphalt surface during anti-lock brake control. 255 is an identification number assigned by structural decomposition scheme of the current specification book. Therefore, the subscripts in this section do not coincide with those used in other chapters.

The (detailed) specification of this module is as follows.

> The "Quick Apply at the End of Gradual Apply Sequence" is started when the following conditions are satisfied simultaneously:
>
> (1) Gradual Apply Sequence is completed (flag1 checked)
>
> (flag1 is in the Outputs of specification No.252.)
>
> (2) At least one other wheel brake is actuated (flag2 checked)
>
> (flag2 is in the Outputs of specification No.212.)
>
> (3) Vehicle reference speed Vso is higher than the threshold (Vso checked)
>
> (Vso is in the Outputs of specification No.121)

The {Inputs} of specification No.255 includes flag1, flag2 and Vso. Similar consideration between other FRs and DPs brings about a design equation like:

$$\begin{Bmatrix} FR_{121} \\ FR_{212} \\ FR_{252} \\ FR_{255} \end{Bmatrix} = \begin{bmatrix} X & X & 0 & 0 \\ X & X & 0 & X \\ X & X & X & 0 \\ X & X & X & X \end{bmatrix} \begin{Bmatrix} DP_{121} \\ DP_{212} \\ DP_{252} \\ DP_{255} \end{Bmatrix}$$

Unfortunately, this is an example of coupled design. How to decouple this kind of design is discussed in chapter 5. Constraints to this kind of real-time control system software include memory amount, program size, execution speed, etc. Recently, microprocessors have been drastically improved and these constraints are getting insignificant.

3.5 Additional Consideration

In mechanical system design, distinction between functional and physical domain is relatively clear. Coupling of design matrices is partially caused by the peripheral characteristics of DPs other than the one which realizes the corresponding FR. For example, if one chooses to use a rod to transfer force, the rod has some other characteristics than the main function 'transfer force', such as weight which might require a base support and thus possibly affect other FRs.

In software design, an FR 'transfer data' can be realized by a DP 'data transfer module' without any significant side effect. That is, if one can describe all the specifications of the program including process timing, data flow and structure, he has described the program itself. This is endorsed by the fact that small portion of program is often called a 'function'. In order to avoid confusion, the word module is used hereafter to indicate a small portion of software which corresponds to each FR and DP.

The coupling in software design is caused only by the relations between modules, i.e., data and procedure flow. Some program module should be executed after some other module as it uses the data updated by the preceding module. Therefore, design matrices can be used to describe dependencies between modules. Software designers can use the distinction of coupled/decoupled/uncoupled in the same way as the mechanical designers do.

Comparison between design matrices and some of the diagrams used in software development methods, such as data flow diagrams are discussed in chapter 4.

Chapter 4

Comparison of Axiomatic Design and OMT

In this chapter, axiomatic design and Object Modeling Technique (OMT) are compared as design methods for rapid-prototyping development of real-time control software.

4.1 Basics of OMT

OMT is one of the most popular software design methods of today where the object oriented programming languages like C++ are dominant. The key concept of OMT is its emphasis on the *object model* described below. The basic ideas of object oriented design such as abstraction, encapsulation and combining data and behavior are all incorporated into the object model. The abstraction which means to focus on the essential, inherent aspects of an entity, is not new to software engineers but the usage of inheritance and polymorphism is especially encouraged in OMT. The inheritance is to inherit some of the characteristics from parent class objects and the polymorphism is to allow one class object to behave differently according to situations. Both help to establish efficient models.

The basic procedure of OMT is explained in this section. Figure 4-1 to 4-3 and their explanations are derived from [6].

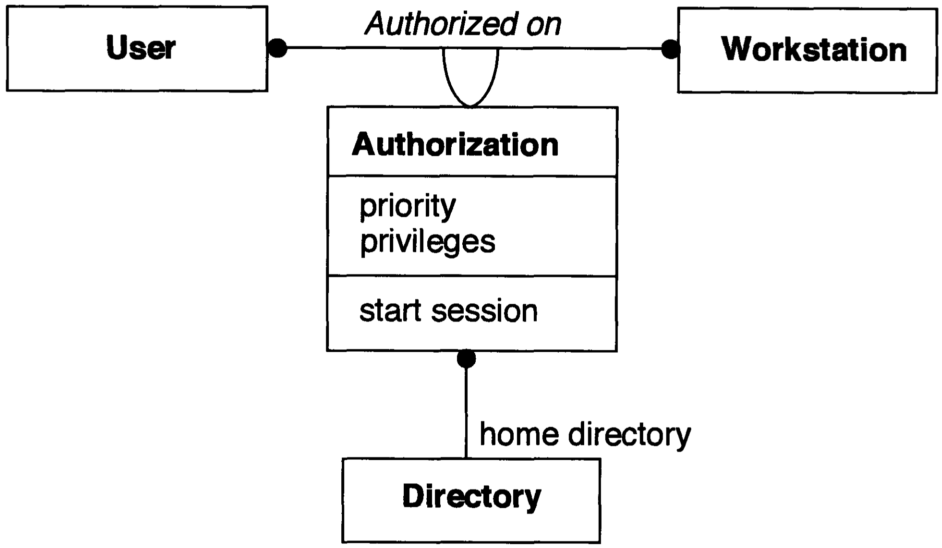

Figure 4-1: An example of object model (from Rumbaugh [6])

4.1.1 Analysis

The first stage of OMT is analysis from three different viewpoints.

Figure 4-1 is the object model which represents the static, structural, "data" aspects of a system. The rectangles indicate classes and the lines between them show relations between them. The black circles mean that the relationship is multiple. For example, "authorized on" relationship between *user* and *workstation* is many-to-many, that is there are many *users* and many *workstations*, and each *user* can be "authorized on" any of the *workstations*. Similarly, a *directory* is related to many *authorizations*, while each *authorization* is related to only one *directory* which is specified by the characteristic "home directory".

Figure 4-2 is the dynamic model which represents the temporal, behavioral, "control" aspects of a system. This diagram is sometimes called the state transition chart. Each rounded-corner rectangle indicates a state of the system. The black circle indicates the initial status. Each arrow in the chart means transition between states which is triggered by the *event* produced elsewhere in the system.

Figure 4-3 is the functional model which represents the transformational, "functional" aspects of a system. This chart is sometimes called the data flow diagram

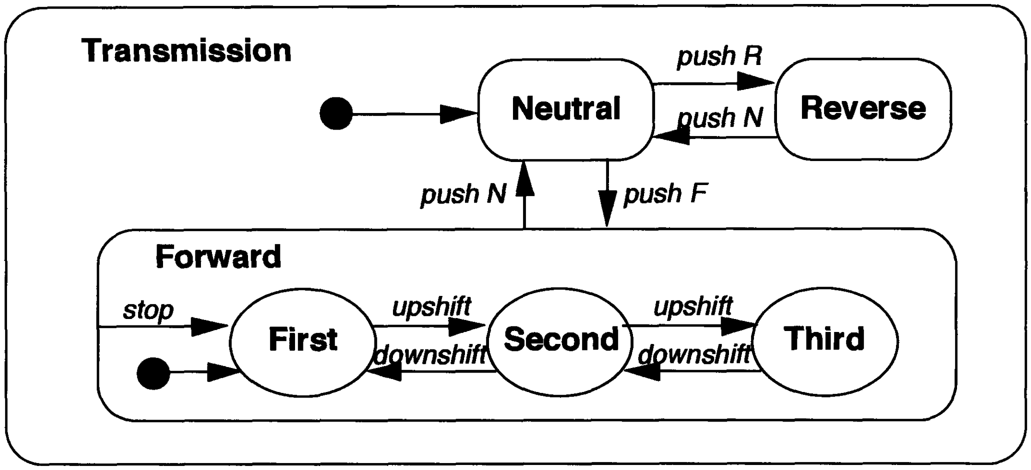

Figure 4-2: An example of dynamic model (from Rumbaugh [6])

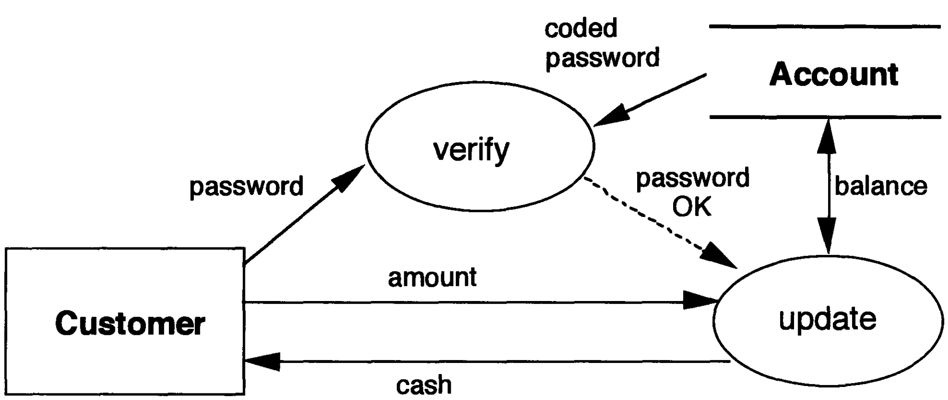

Figure 4-3: An example of functional model (from Rumbaugh [6])

(DFD). The arrows indicate *data flows*, while the dotted arrow indicates a *control flow*. The ellipses indicate *processes* which transform data, while the rectangle means *actor objects* that produce and consume data and the parallel pair of horizontal lines shows the *data store object*.

4.1.2 System Design

Once the analysis is completed, the next stage is the system design where system architecture, or overall structure and style of the system, is decided. This includes organization into subsystems, identifying concurrency, allocation of subsystem to processors and tasks, etc. This is dependent on the structure of the hardware (computer) and the operating system to be implemented on.

4.1.3 Object Design

The next and the most important stage is the object design where the full definitions of classes and their associations, interfaces and algorithms are determined. In this stage, software designers add internal objects to store interim data and revise relationships of objects and optimize data structure. Thus, the object model obtained in the analysis stage is refined into the design model with a lot of details needed for implementation. Note that, therefore, the *analysis model* and the *design model* inevitably have different structures, making simple mapping between them impossible.

4.2 Comparison of Axiomatic Design and OMT

In this section, axiomatic design and OMT are compared from the viewpoint of application to the rapid-prototyping of the real-time control software.

4.2.1 Description of Timing

Unlike other generic software, description of timing is crucial for the real-time control software. In that sense, the conventional flowchart is more convenient than the structured chart created in the structured analysis and design technique (SADT) [5] which is, in a sense, the predecessor of OMT. OMT inherits the incapability of explicitly describing timings. Meanwhile, the design equation in the axiomatic design can be used with the conventional flowchart. For example, once the inter-dependencies between modules are expressed in terms of a design equation, the relations can be automatically projected on to the flowchart. When the design equation is decoupled, the order of execution can be easily checked by using the projection on the flowchart.

4.2.2 Data Structure

OMT, like most of the object oriented methods, is devised to treat the complicated structures of data and procedure such as the payroll system of a large company or the automatic teller machine (ATM) system of banks. As for the real-time software like the anti-lock braking system, the data structure is relatively simple, thus the sophisticated method of OMT to treat complicated data structure is not necessary.

4.2.3 System to be Modeled

The most intuitive image of an object is an icon on Macintosh or Windows computers. For example, an icon of a document has both the data, e.g., the manuscript, and the behavior, e.g., to launch a word processor program when double clicked. And the icon is passive, that is, it will wait forever unless the user does some kind of action on it. Conversely, in the real-time control system such as the anti-lock braking system or the engine management system, the state of the wheels or the engine changes from time to time even if the controller does not send any signal to them at all. It is not a very good idea to model spontaneous behaviors like random input from the road surface as an object.

On the other hand, these control systems are generally combinations of hardware and software. With axiomatic design, hardware and software are treated in the same manner when modeled properly. This is important as the same function can often be realized equally either by hardware or software and designers are often asked to choose between hardware or software to optimize the system design.

4.2.4 Reusability of Software and its Side Effect

One of the merits of the object oriented method is the reusability of software. In order to maintain this reusability, all the program codes in the object oriented programming are allowed to retain codes which may not be used anymore. In other words, if one wants to add a new function to a module, one should not modify the existing function in the module, but always add a new function to it. Consequently, in object oriented methods, the programs almost always have unnecessary codes and require more memory space. This is not allowed in the mass produced real-time control systems such as the anti-lock braking system due to the severe cost constraint.

With axiomatic design, the functional requirements are always defined to be minimum and independent, thus making the software as slim as possible. As to reusability, by controlling the software resources in the past in a database tagged with the functional requirements, similar reusability can be maintained.

4.2.5 Documentation Problem

One needs to maintain three different charts in modeling a system with OMT. The three charts look pretty much alike especially for a novice software engineers, and can lead to a confusion. And as the development process of a real-time control system is often a heuristic process, i.e., repetition of the rapid-prototyping and testing reveals many new characteristics of the controlled system throughout the development period. In other words, it is to continuously modify the analysis models in OMT. It is not realistic to maintain the three complicated charts under the severe time constraint during the development. OMT is most efficient when the analysis model need not be modified often.

With axiomatic design, the design equations are basically the only documentation. And, when needed, the flowcharts can be generated and linked with the design equations automatically.

Either with OMT or axiomatic design, recording and reusing the rationale behind the functional requirements requires additional consideration.

4.2.6 Conclusion of the Comparison

As far as the rapid-prototyping of real time control systems like the anti-lock braking system is concerned, axiomatic design has advantages over OMT, especially in the ability to visualize the inter-dependencies between the program modules and to treat hardware and software interchangeably.

Chapter 5

Structural Decomposition of Anti-lock Braking System

5.1 Functional Requirements in the highest level

The functional requirements at the highest level of an anti-lock braking system are:

FR1=Minimize stopping distance

FR2=Maintain stability and steerability of the vehicle

FR3=Inform driver of the vehicle condition

FR4=Maintain availability (reliability)

FR5=Self diagnosis

There is one constraint:

C=Total system cost

There is a well-known relation between the rubber tire and asphalt road surface as shown in figure 5-1.

The horizontal axis indicates the slip ratio, which is defined as:

$$S = \frac{V_b - V_w}{V_b}$$

where V_w is the wheel speed and V_b is the vehicle body speed. When V_w is zero and V_b is not, the slip ratio S equals to unity. This corresponds to the case such as

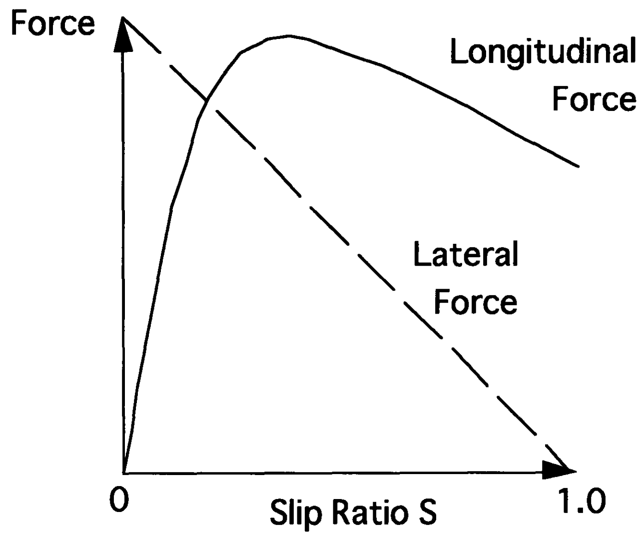

Figure 5-1: Slip ratio and lateral/longitudinal force

when strong braking force is applied to the wheels while a car is driving on ice and the car running at V_b with all the wheels fully locked-up. On the contrary, when V_w equals V_b, which corresponds to the case there is no relative slip between the tire and road surface, S equals to zero. In general, S is somewhere between zero and unity while braking force is applied.

The solid line in the graph indicates the characteristic of the longitudinal force between the tire and road surface. The peak around $S = 0.3$ indicates that the deceleration force is at its maximum there. The broken line shows the lateral force which contributes to the lateral stability and the steerability of the vehicle. The lateral force is at its maximum when $S = 0$ and reduces in inverse proportion to S, and equals to almost zero at $S = 1.0$.

As shown in figure 5-1, the peaks in the longitudinal and lateral force are at different values of S. This means that one cannot optimize

FR1=Minimize stopping distance

and

FR2=Maintain stability and steerability of the vehicle independently as these two require a different optimum value of one parameter (DP). Therefore, FR1 and FR2 should be combined to form a new FR as:

FR1=Minimize stopping distance while maintaining stability and steerability

The new set of FRs is:

FR1=Minimize stopping distance while maintaining stability and steerability
FR2=Inform driver of the vehicle condition
FR3=Maintain availability
FR4=Self diagnosis

And the corresponding set of DPs is chosen as:

DP1=Slip ratio S
DP2=Brake pedal vibration or indicator
DP3=Redundant mechanism
DP4=Self diagnosis module

And the design matrix is:

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 \\ 0 & X & 0 & 0 \\ 0 & 0 & X & 0 \\ 0 & 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \end{Bmatrix}$$

The highest level of the anti-lock braking system is uncoupled.

5.2 Decomposition of FR1

The first FR in the highest level,

FR1=Minimize stopping distance while maintaining stability and steerability

which is realized by the DP,

DP1=Slip ratio S

is decomposed into following FRs.

FR11=Detect wheel speed

FR12=Estimate vehicle speed

FR13=Calculate slip ratio S from the wheel speeds and the vehicle speed

FR14=Determine output signal so that S is controlled to the target value

FR15=Convert output signal to brake force

The corresponding set of DPs is:

DP11=Wheel speed sensor and calculation module

DP12=Vehicle speed estimation module

DP13=Slip ratio calculation module

DP14=Output calculation module

DP15=Brake force actuator

$$\begin{Bmatrix} FR_{11} \\ FR_{12} \\ FR_{13} \\ FR_{14} \\ FR_{15} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 & 0 \\ X & X & X & X & 0 \\ X & X & X & 0 & 0 \\ X & X & 0 & X & 0 \\ 0 & 0 & 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP_{11} \\ DP_{12} \\ DP_{13} \\ DP_{14} \\ DP_{15} \end{Bmatrix}$$

In FR14, the output signal to the brake force actuator is determined using a table of wheel speed/acceleration and (estimated) vehicle speed. FR14 is further decomposed into:

FR141=Estimation of other vehicle condition parameters

FR142=Calculation of output signal

FR143=Correction of output signal by other conditions

In FR141 various conditions other than the vehicle speed are estimated, which include road roughness, whether the vehicle is cornering or not, etc. After the basic output signal is selected in FR142, the signal is corrected according to the other vehicle conditions for improved accuracy in FR143.

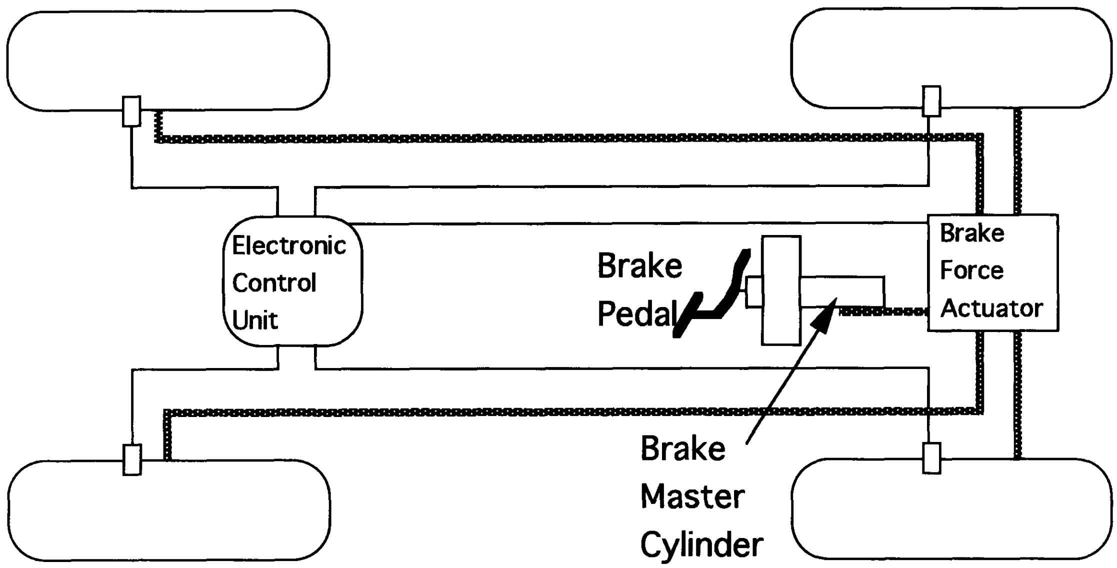

Figure 5-2: ABS configuration

5.2.1 Estimation of Vehicle Speed and Other Condition Parameters

The estimations in FR12 and FR141 are necessary for the following reason. Figure 5-2 shows a typical configuration of ABS.

As it has only four wheel speed sensors at each wheel, the vehicle body speed and other parameters need to be estimated.

Figure 5-3 shows a method used to estimate vehicle speed. In all the graphs in 5-3, the horizontal axis is time and vertical axis indicates tire rotation speed and vehicle body speed.

Figure 5-3 (a) shows the case when a vehicle is running at a constant speed. There is little slip between tire and road surface (S=0). Figure 5-3 (b) shows the case when some brake force is applied and the vehicle stops at t=t1. S is non-zero while braking. Figure 5-3 (c) is the case of excessive braking force. The wheel speed plummets to lock up and the vehicle is still running. Because the lateral force the wheel can provide is minimum, the vehicle is in a very unstable condition and the stopping distance is longer. Figure 5-3 (d) shows the case with ABS. As the brake force is controlled to prevent wheel lock ups, taking the maximum of the four wheel

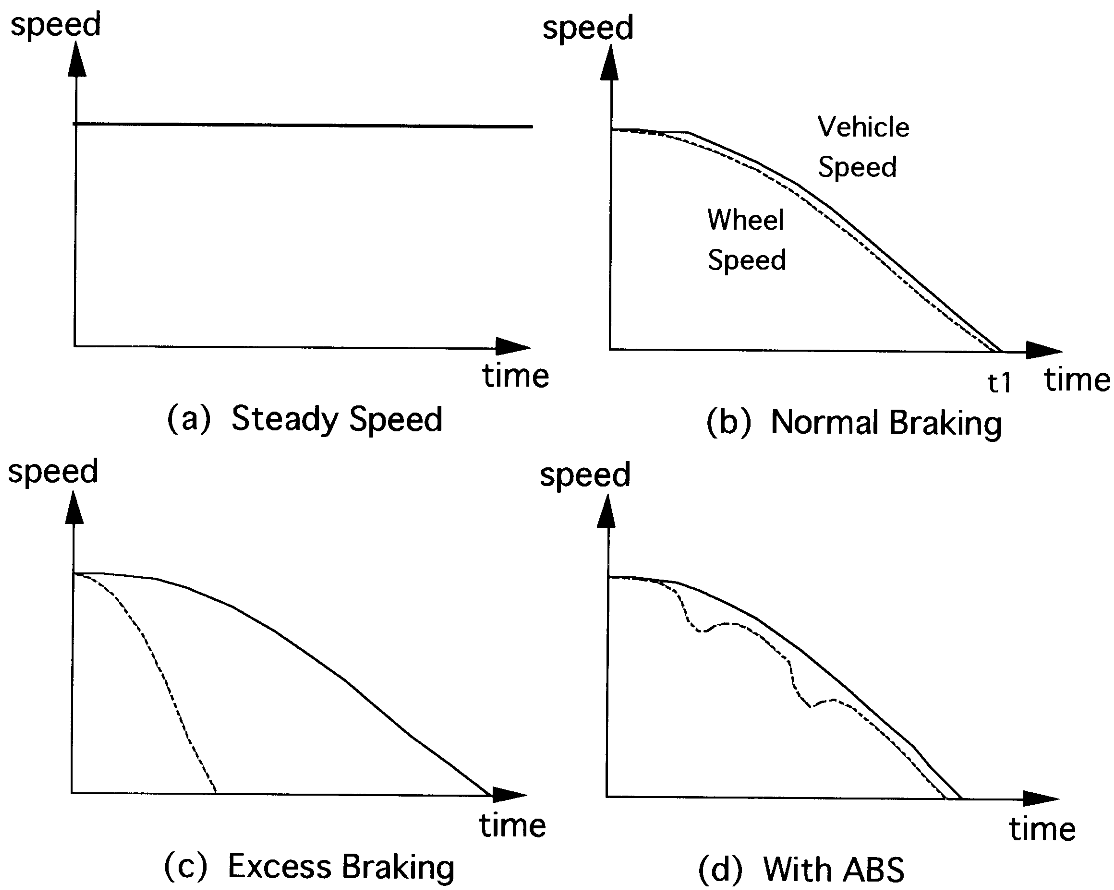

Figure 5-3: Vehicle speed estimation

speed sensor signals provides a good estimate of vehicle speed. In case the four wheel speeds drop synchronously, a guard value is applied to the maximum of the gradient, as the theoretical maximum deceleration on a dry asphalt surface is known to be 1.0G.

Similarly, other vehicle condition parameters such as road roughness, whether or not vehicle is cornering and whether or not one of the wheels has a different tire diameter from others (typically an emergency temporary tire) are taken into consideration.

5.2.2 Selecting DPs for sub-FRs of FR1

For sub-FRs of FR1, corresponding DPs are selected as:

DP11=Wheel speed detection module [hardware/software]
DP12=Vehicle speed estimation module [software]
DP13=Slip ratio calculation module [software]
DP14=Output calculation modules [software]
DP15=Brake force actuator [hardware]

Note that if one follows the model of software design discussed in chapter 3, one can assign hardware or software interchangeably as a DP. The expression [hardware] means that, in a typical configuration, it is realized by hardware, and [hardware/software] indicates that it is partially realized by hardware and partially by software.

The design equation for this level of hierarchy is:

$$\begin{Bmatrix} FR_{11} \\ FR_{12} \\ FR_{13} \\ FR_{14} \\ FR_{15} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 & 0 \\ X & X & X & X & 0 \\ X & X & X & 0 & 0 \\ X & X & 0 & X & 0 \\ 0 & 0 & 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP_{11} \\ DP_{12} \\ DP_{13} \\ DP_{14} \\ DP_{15} \end{Bmatrix}$$

This is a coupled design.

5.3 Analysis of the Coupled Design

The five FRs in section 5.2:

FR11=Detect wheel speed

FR12=Estimate vehicle speed

FR13=Calculate slip ratio S from the wheel speeds and the vehicle speed

FR14=Determine output signal so that S is controlled to the target value

- FR141=Estimation of other vehicle condition parameters
- FR142=Calculation of output signal
- FR143=Correction of output signal by other conditions

FR15=Convert output signal to brake force

were selected to clarify the control process of ABS. Namely:

(1)Detect wheel speeds

(2)Estimate vehicle speed using wheel speeds

(3)Estimate parameters of road roughness, cornering, etc.

(4)Decide output signal using a table of wheel speed and vehicle speed

(5)Correct the output signal using parameters from (3)

(6)Actuate brake force according to the signal

The design equation in the previous section was redundant. FR12 and FR141 can be combined into one FR, "Estimation of vehicle parameters." Similarly, FR142 and FR143 can be combined into an FR, "Create output signal (using wheel speeds and estimated parameters)."

Also, from the design process point of view, hardware design is almost always done before software due to lead time constraints. Therefore, the actuator (hardware) design issue should be moved to the uppermost row. By doing this, one of the couplings is solved.

The new set of FRs is:

FR11=Convert output signal to brake force

FR12=Detect wheel speed

FR13=Estimate vehicle condition parameters

FR14=Create output signal using wheel speeds and estimated parameters

The corresponding set of DP is:

DP11=Brake force actuator

DP12=Wheel speed sensor

DP13=Vehicle condition parameters estimation module

DP14=Output signal calculation module

Thus, the new design equation is:

$$\begin{Bmatrix} FR_{11} \\ FR_{12} \\ FR_{13} \\ FR_{14} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 \\ 0 & X & 0 & 0 \\ 0 & X & X & X \\ X & X & X & X \end{bmatrix} \begin{Bmatrix} DP_{11} \\ DP_{12} \\ DP_{13} \\ DP_{14} \end{Bmatrix}$$

This is still a coupled design.

5.4 Decoupling

The coupling is caused by the fact that output signals are referred to when estimating parameters.

For example, in order to improve accuracy of the estimation, two different values are used for the guard value of the maximum deceleration, which was discussed in section 5.2.1, depending on whether the brake force actuator is activated or not.

If we can use a sensor which is capable of measuring the vehicle body speed directly, it serves as a decoupler.

In practice, most of the anti-lock braking systems for current four wheel drive vehicles are equipped with a deceleration sensor. Because the deceleration signal can be used to calculate vehicle speed by integrating the output signal, this is actually a decoupler. In four wheel drive vehicles, all four wheels are engaged by drive shafts, thus making it easier for the four wheel speeds to perfectly synchronize with each other, especially when braking on slippery roads. The perfect synchronization causes a mis-estimation of the vehicle speed and causes an unstable condition.

In conclusion, the coupling has been solved only for the case where the influence of the coupling is critical. Though axiomatic design suggests adding a decoupler to improve performance of all the ABS for all type of vehicles, it has not been realized due to the cost constraint.

Chapter 6

Problems in Current Software Development Process

In this chapter, the current development process of anti-lock braking system is analyzed, and its problems are clarified as a basis for redesigning the development process and suggesting a new support system.

6.1 Brief History of Anti-lock Braking System

In the first stage, the anti-lock braking systems were realized by a mechanical system using a flywheel or by an electronic actuator and a simple controller which consists of a discrete circuit.

As microprocessors became available at a reasonable price, the latter evolved into the system using a microprocessor and the former disappeared due to low performance. The usage of the microprocessor allowed designers to add various additional signal processes, i.e., detecting a rough road condition and changing the output signal accordingly.

Thus, through several years of evolution, the anti-lock braking system turned out to be a heuristic process of finding new characteristics of the controlled vehicle system, mainly in chassis control, sometimes in engine and powertrain. In other words, the performance of the anti-lock braking systems has been improved by adding

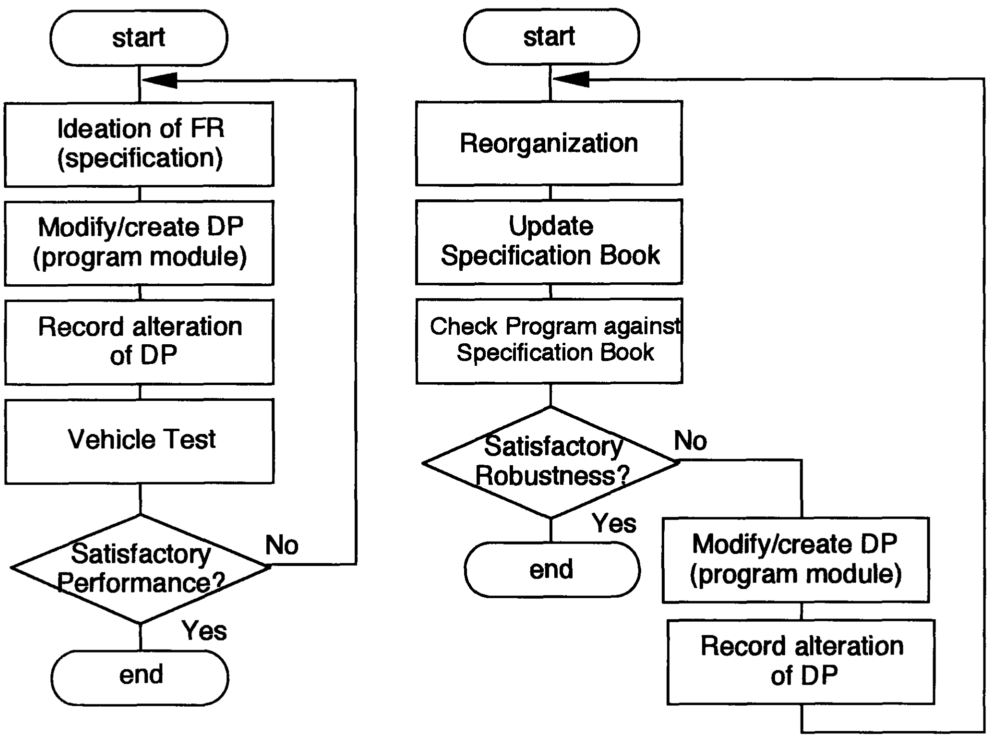

Figure 6-1: Current development process of software

new additional processes each time a new problem was found. Consequently, current anti-lock braking system performs quite well in various conditions. Although, on the other hand, this fact generates a few problems in the design process.

6.2 Analysis of Current Process of Software Development

Figure 6-1 shows the current software development process of the anti-lock braking systems.

The first loop on the left indicates the phase of performance improvement. For example, with anti-lock braking systems, winter field tests are inevitable. Though auto makers do have artificial test courses with very low frictional coefficient, the winter tests cannot be eliminated due to the slight difference between the actual conditions and the artificial conditions in the test course and the compound conditions which are difficult to reproduce in the artificial environment. Systems other than anti-lock braking system also require winter tests, as the performance at a very low temperature and/or on slippery road surface are essential to most vehicle control sub-systems. Winter tests are dependent on natural weather conditions and therefore limited to a very short period of time every year. Consequently, rapid-prototyping is a key factor of development.

The second loop on the right shows the phase of robustness improvement or debugging. First, after numerous alterations in the previous phase, the whole software (and sometimes hardware) design is reviewed/reorganized. The necessity of this process is explained by theorem 5 in axiomatic design (see below).

Theorem 5 (Need for New Design) When a given set of FRs is changed by the addition of a new FR, or substitution of one of the FRs with a new one, or by selection of a completely different set of FRs, the design solution given by the original DPs cannot satisfy the new set of FRs. Consequently, a new design solution must be sought.

Once the review/reorganization of software is done, the specification book is updated. The specification book is a document which is used as a "drawing" of software. As Jackson notes in [2], since this current specification book is at best a partial description of the software which only contains information of surface of the software seen from the environment around, it actually cannot be used as a drawing of the software. If one wants to order software from a new contractor, just handing this specification book does not work. Instead, one needs to hand the same or similar software written by others. The new development support system which will be suggested in this thesis should address this problem.

6.3 Statement of Problems

The analysis in previous section revealed three major problems.

- Missing Rationale
- Unclear Influence Scope of Alterations
- Management of Progress

These are described in more detail below.

6.3.1 Missing Rationale

In the first performance improvement loop, the only information recorded other than the altered program module itself is the notes on the alteration of DPs and there is no process of recording rationale behind each alteration. This is mainly due to the time constraint of the design process. As the priority of recording rationale is considered to be relatively low, it is actually ignored. Then in the second robustness improvement loop, the specification book is updated according only to the information of DP alterations and the personal memory of the software engineers.

Rationale is in short a reason of the changes in software. In chapter 3, the following software model was used.

CA : User's Needs
FR : Detailed Specifications of Software
DP : Program Codes
PV : Machine Codes

In many cases, software engineers are asked to realize relatively vague concepts of other engineers who know little of software or control. These vague concepts can be rephrased to the customer attributes (CAs) or the rationale. Like CAs, the rationale is generally vague and not easy to formalize.

6.3.2 Unclear Influence Scope of Alterations

Another problem which also annoys novice programmers is the difficulty to clarify the scope of influence of each alteration they make. As there is no appropriate tool to provide information of inter-dependencies between the program modules and the whole structure of the program, it is hard to have a good perspective.

Further, there are some implicit relations between program modules. For example, most anti-lock braking systems do not have a dedicated vehicle (not wheel) speed sensor, and the vehicle speed is estimated using four wheel speed sensors as described in chapter 5. If the control output is inappropriate, it affects the four wheel speeds and consequently the estimation of the vehicle speed. On the other hand, if the vehicle speed estimation is not accurate, the control output signal is naturally inappropriate.

There is a strong need to provide a tool which help software designer to know the explicit scope of alterations. Though it still needs some experience to understand the implicit relationship between software modules, isolating it from explicit ones is of great help.

6.3.3 Management of Progress

Though this is commonly the case with generic software engineers, the job process is not clear thus always making it difficult to predict how long it will take a software engineer to finish a new project or how much the delay is to the original schedule and what help a manager can provide to recover the delay. Unlike with mechanical designs, managers tend to lack in capability of understanding the job process of software engineers. It is in partly due to the fact that the complexity of the software and of the modeling the controlled system are not isolated. By analyzing the software design process using axiomatic design, this problem should be solved.

6.3.4 Summary of the Problems

As a summary, the problem in the current development process is that some of the necessary FRs are missing. The FRs of the new development support system to be suggested should include following three.

FR1=Record and Reuse Rationale

FR2=Clarify the Influence Scope of Alterations

FR3=Progress Management

6.4 Problems Solved by the Improvement of Microprocessors

Modern microprocessors are much faster than the mechanical systems to be controlled such as internal combustion engines, vehicle suspension and brake systems. This is also the case for the real-time control software like the one for the anti-lock braking systems.

In this section, some of the problems already solved by the evolution of microprocessors are described in terms of axiomatic design to clarify the reason why real-time control software problems differ from the generic software problems.

When the memory size and the execution speed of microprocessors were severely restricted, various techniques were invented to compensate for these limitations. For example, dividing an 8-bit memory into four flags and a 4-bit size counter, using a flag or an output pattern for plural purposes. Obviously, these are all violations of the independence axiom. Thanks to the availability of lager memory and faster microprocessors, these techniques have become obsolete.

The insufficient performance of old microprocessors forced us to use a low level assembly language. Assembly languages consist of primitive commands, thus leading each software engineer to develop his/her own technique of programming. For example, if one wants to look up a table and interpolate the values from it, there are several ways to realize the same functional requirement. With higher level languages which are available only with recent high performance microprocessors, more powerful commands realize the same function in a fixed way. In other words, much more information was needed with the assembler language to realize the same functional requirement.

Thus, some of the software problems were solved by the evolution of the microprocessor and the problem left unsolved look somewhat different than the generic software problems.

6.5 Model-based Control

The recent trend in the vehicle control field is the transition to model-based control. Primitive control systems like the anti-lock braking system in its early days consists only of simple table look-ups. The model-based approach, on the other hand, first establishes a mathematical model of the controlled system then designs the controller using control theories. Although, the model-based approach has the potential of simplifying the control system, there are a few reasons we cannot adopt it immediately.

First, through several years of improvement, most of the current control systems have reached a high performance level, especially in special conditions where the model-based control still needs a patch to cover. All of the critical problems have been solved, though often by an ad hoc patch.

Second, many systems have been established without using the sensors which are essential with the model-based approach. And under a high pressure to keep down costs, it is not practical to add one unless the sensor cost becomes negligibly low.

Third, vehicles consist of many sub-systems and a lot of effort is required to gather data for modeling all of those which possibly affect the system to be designed. In many cases, testing with an actual vehicle is much more fast, accurate and effective.

Therefore, it is difficult to abandon current software and switch thoroughly to the model-based control. A practical approach is to change the software structure to help incorporate the model-based approach locally. The software modeling discussed in chapter 3 is compatible with this idea. If all the DPs are described in terms of input, output and their relations, one can replace each DP with the one using the model-based approach easily.

Chapter 7

Redesigning the Software Development Process

In this chapter, the development process for a real-time control software is re-designed based on the discussions so far and its support system is suggested taking the anti-lock braking system as an example. Note that this is not to model the software or the anti-lock braking system itself but the development process for them. Therefore the artifact designed here is a development support system.

7.1 Feedback in Design Process

Kim, Suh and Kim [3] related the idea of module-junction structure to axiomatic design. It says summation (S), control (C) and feedback (F) junction corresponds to uncoupled, decoupled, and coupled design, respectively. Then they conclude one should avoid feedback junctions as they are coupled.

Figure 7-1 shows the feedback loop of a typical design process.

Rapid-prototyping is to go through the loop indicated by thick arrows in figure 7-1 as quickly as possible. In order to design a support system for this process, it is inevitable to allow incorporation of some kind of feedback.

One approach is to consider the design equation of the support system to be a open-loop transfer function of the development system. In other words, it is a

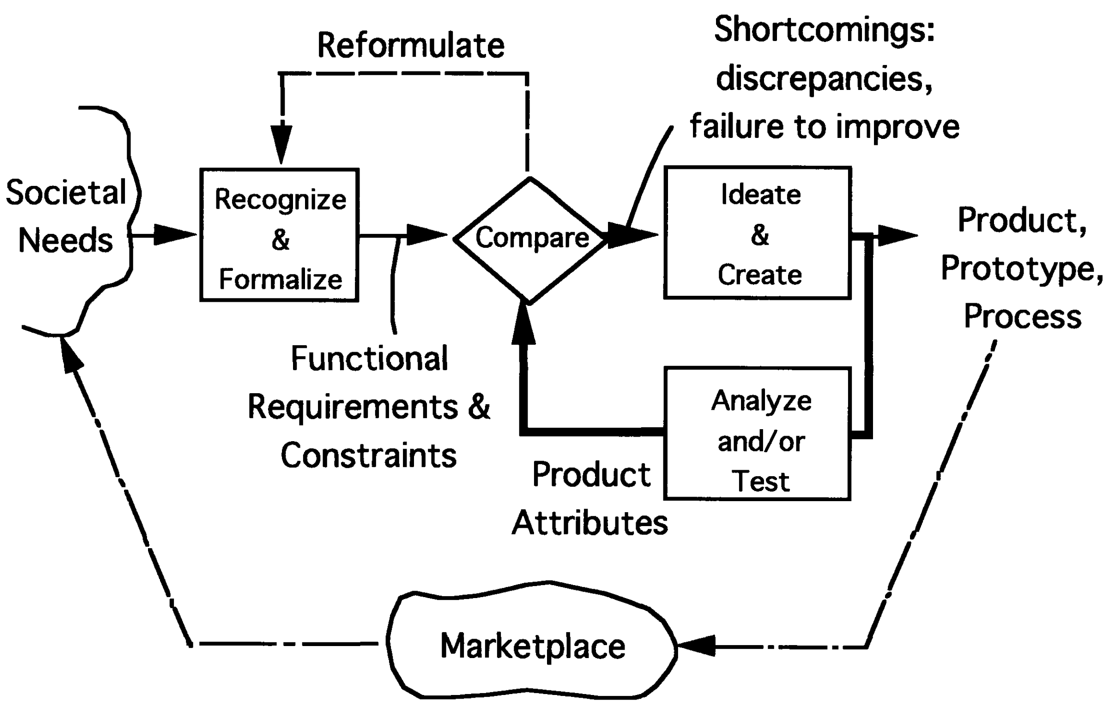

Figure 7-1: Feedback loop of design process (from Suh [7])

snapshot of the design process at a moment. The feedback element which seems to cause coupling is actually a data from the previous cycle and not the coupling in a strict sense. However, feedbacks across different levels of FR hierarchy should be avoided as the program will quickly become unmanageable.

7.2 The Highest Level FRs

First, we set the highest level FRs as follows.

FR1=Accumulate knowledge of the controlled system

FR2=Maintain robustness of the software

Rather than simply improving the performance of the system, the first FR is set as an accumulation of knowledge in order to accommodate the future transition to the model-based control. And by clearly modeling the process in detail, it should help estimating the time needed to go through the process more accurately.

The second FR remains the same as before, but by further decomposing the FR, the problem of documentation will be addressed.

DPs are selected as:

DP1=Customer Attribute (CA) Database and Vehicle Tests

DP2=Customer Attribute (CA) Database and Debugging

As the two FRs need not be satisfied simultaneously, it is acceptable for the corresponding DPs to have some common contents.

$$\begin{Bmatrix} FR_1 \\ FR_2 \end{Bmatrix} = \begin{bmatrix} X & 0 \\ X & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \end{Bmatrix}$$

7.3 Decomposition of FR1

FR1 is decomposed into:

FR11=Acquire test data

FR12=Devise and record CA

FR13=Create FR (Programming)

FR14=Record the CA database activity

Corresponding DPs are:

DP11=Vehicle test

DP12=CA database

DP13=Software version database

DP14=CA database audit system

FR11 means the data acquisition by vehicle test for FR12. FR13 is the actual programming with the help of version control software which is commercially available. Note that, as we defined FRs of software to be "detailed specification", conversion to DP or "Program code" can be automated. FR14 is needed for managerial needs. As all the job done by a software designer in this phase is related to the CA database, a manager can determine the productivity of the engineer by checking the activity or the input/output of the CA database. FR12 is the key functional requirement in this level.

The design equation of this level is:

$$\begin{Bmatrix} FR_{11} \\ FR_{12} \\ FR_{13} \\ FR_{14} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 \\ X & X & 0 & 0 \\ X & X & X & 0 \\ X & X & 0 & X \end{bmatrix} \begin{Bmatrix} DP_{11} \\ DP_{12} \\ DP_{13} \\ DP_{14} \end{Bmatrix}$$

7.3.1 FR12: Devise and record CA

Figure 7-2 is the schematic diagram of the process in which CAs are devised.

In this stage, a software designer takes into consideration the vehicle control data, the sensuous/quantitative evaluations along with the test conditions such as road surface, temperature, tire type, etc.

FR121=Ideate CA

FR122=Check Contradiction

FR123=Record CA

In FR121, the software engineer consults the CA database, which holds the recorded CA data and is capable of searching CAs with keywords, problem patterns and the program alteration history. In each case in the database is tagged to actual alteration in the past.

Once he/she gets the basic idea, he/she checks if the new CA to be implemented causes contradictions with other part of the system (FR122). The designer can specify the affected data by the alteration in mind and check the scope of the change by using the design equation defined for software in chapter 3. If a contradiction is found, he/she should go back to FR121 to modify the CA.

Then, in FR123, before actually implementing the change, the CA is recorded in the CA database. Each data in the CA database is tagged with the actual change in the program, keywords and patterns.

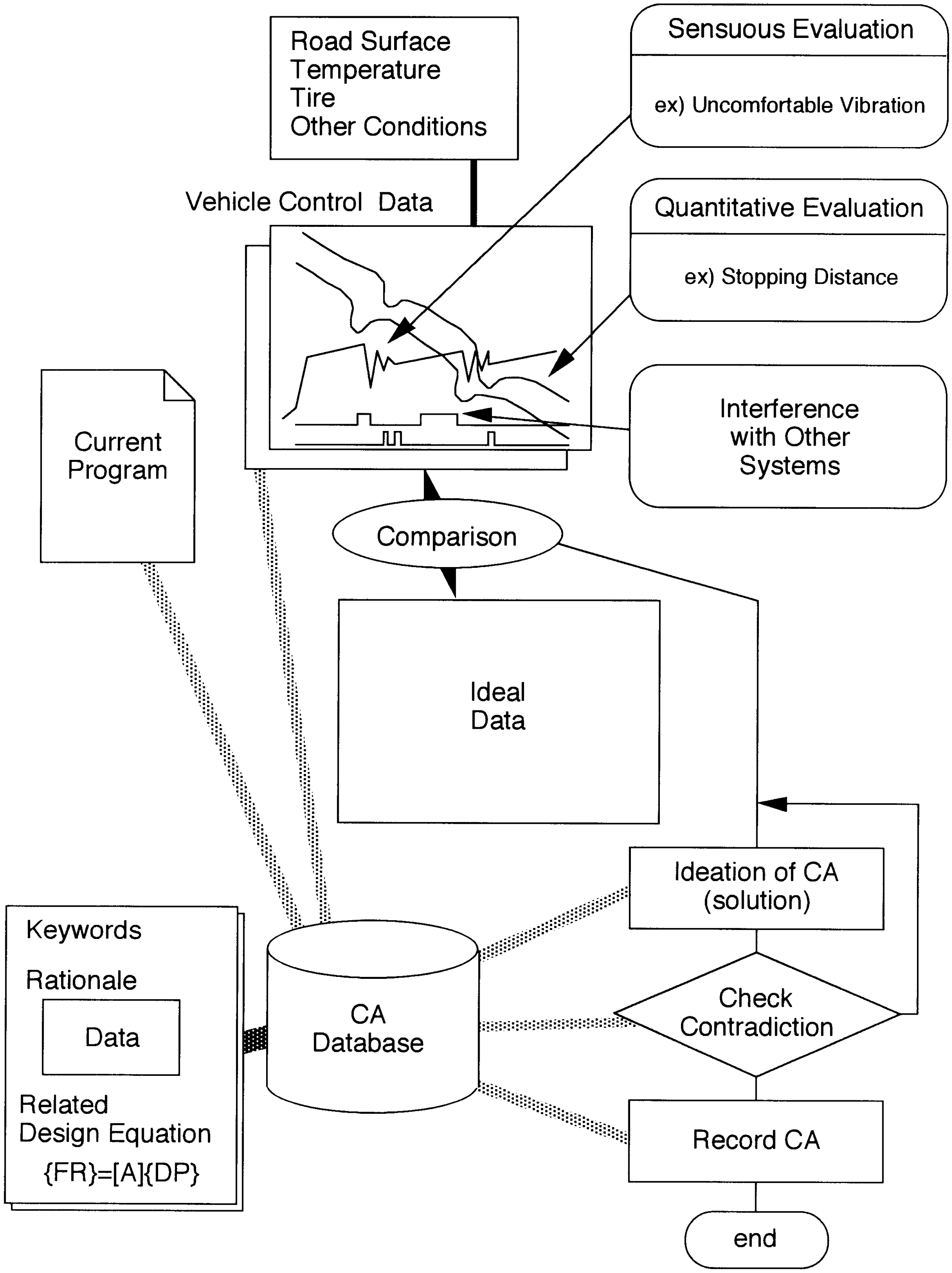

Figure 7-2: Devise process of CA

Corresponding DPs are:

DP121=CA workplace

DP122=CA data browser

DP123=CA data entry

DP121="CA workplace" is an environment where all the necessary information for the software engineer is displayed on a large screen. The information includes the control data, evaluations, interference with other systems, etc. The design equation at this level is as follows.

$$\begin{Bmatrix} FR_{121} \\ FR_{122} \\ FR_{123} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP_{121} \\ DP_{122} \\ DP_{123} \end{Bmatrix}$$

7.3.2 Example

A rough road detection is a commonly used method to improve performance of the anti-lock brake control.

In general, the anti-lock brake systems' only pressure source is the brake master cylinder manipulated by the driver of the vehicle via the brake pedal. On rough roads, the vibration in the wheel speed sensor signals cause unnecessary repetition of application and release of the brake pressure, resulting in the insufficient brake force as the pressure source is limited by the brake pedal effort. On the other hand, the release side is unlimited as it is simply connected to the atmospheric pressure.

Thus, he/she wants to detect the rough road from the vibration pattern of the wheel speed signals and change the output pattern to apply more brake pressure when on rough roads.

CA=Detect rough road

He/she should take into account that a quite similar control data can be obtained from a test on a rough and icy road surface as well as on a dry and rocky road surface. If it were on the former, the extra brake force by the rough road judgment may lead to an excessive braking force for icy roads and may cause unstable vehicle condition. Therefore, the CA must be modified to:

CA=Detect rough and high frictional coefficient road

As one can see, repeated changes of this kind can lead to a large amount of software resources which are almost impossible to reuse if the rationale or CA behind the changes are not recorded. That is why we need a CA database in the support system.

7.3.3 FRs of the CA database

The highest level FRs of the CA database are:

FR1=Record CAs with minimum effort

FR2=Search CAs with various keys

FR1 is needed because these CAs should be recorded in the actual development stages where time is invaluable. Therefore, it is a good idea for the support system to help ideate CAs and automatically record the newly created CAs to the database. It is also necessary to add some keywords for future references. Possible keywords should include contradiction, vibration, noise, cost, synchronization, etc. Each CA may have different structure and needs to be stored in a free style format, especially, adding the actual control data to each CA is important.

FR2 is the key functional requirement for the support system to be of practical use. Users search relevant cases in the past using the keywords added in FR1.

As a basis of the database for the future development, current software should be analyzed in retrospect to select the appropriate keywords and validate the usefulness of the CA database.

7.4 Decomposition of FR2

The second FR of the development process, FR2=Maintain robustness of the software

is realized by the DP,

DP2=CA Database and Debugging.

FR2 is decomposed into:

FR21=Reorganization of the whole structure of software
FR22=Documentation
FR23=Check consistency
FR24=Creation of FR (Programming)

FR21 is the same functional requirement explained in chapter 6 with Theorem 5 (Need for New Design).

FR22 is the process of documentation. As can be seen the discussions so far, the documentation is less important compared to the CA database. The documentation process is still necessary as a compatibility tool with the conventional drawing archive system and a cross check tool of software as it often helps finding inconsistencies between program modules by representing it in natural language rather than in programming language.

Corresponding DPs are:

DP21=CA data browser
DP22=Word processor
DP23=CA data entry
DP24=Software version database

DP21 shows that the optimization of software is done by examining each module with reference to its changes and rationale. DP23 and DP24 mean, if any inconsistencies were found, the CA should be further modified to eliminate it and corresponding FR and DP should be modified, too. The design equation in this level is as follows.

$$\begin{Bmatrix} FR_{21} \\ FR_{22} \\ FR_{23} \\ FR_{24} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 \\ X & X & 0 & 0 \\ X & X & X & 0 \\ X & X & 0 & X \end{bmatrix} \begin{Bmatrix} DP_{21} \\ DP_{22} \\ DP_{23} \\ DP_{24} \end{Bmatrix}$$

Chapter 8

Conclusion

The purpose of this thesis was to identify and solve the problems with the electronic controlled subsystems for automobiles and with the development process of these systems by analyzing them using axiomatic approach.

First, as a preparation for further discussions, the functional requirement (FR) and the design parameter (DP) are defined for software. Using these definitions, the inter-dependencies between program modules can be explicitly expressed in terms of design equations and the software and the hardware can be treated equally and interchangeably.

Then, by comparing axiomatic design and the leading software development method OMT from the viewpoint of the rapid-prototyping support for the real-time control software, the power and the flexibility of axiomatic design were confirmed. Especially, axiomatic design's capability to visualize the inter-dependencies between the program modules in terms of design equations and to treat the hardware and the software interchangeably turned out to be very useful.

The anti-lock braking system was adopted as an example for analysis of the real-time control systems. Though the result turned out to be a coupled design due to a severe cost constraint, a suggestion to decouple it with an additional sensor was suggested. This suggestion can be implemented as soon as the sensor cost drops to a reasonable level.

Next, the development process of the real-time control system was analyzed also using axiomatic design. The characteristics of the development process was the strong need for rapid-prototyping. The analysis revealed that the problem of missing rationale was caused by the fact that current process does not have an FR to record it.

Finally, in order to solve the problems identified, a new development support system was designed using axiomatic design. By changing the highest level FR from "improve performance of the system" to "accumulate knowledge of the controlled system", a new support system which uses the customer attribute (CA) database was designed. With this new support system, as the activities of the software engineers are naturally linked to the CA database, the managerial problems such as getting hold of the project progress will also be solved.

Though these results are not tested in the real world yet, the concepts derived from axiomatic approach have shed a light on the long unsolved problems.

Appendix A

Corollaries and Theorems of Axiomatic Design

A.1 Corollaries of Axiomatic Design

Corollary 1 (Decoupling of Coupled Design) *Decompose or separate parts or aspects of a solution if FRs are coupled or become interdependent in the designs proposed.*

Corollary 2 (Minimization of FRs) *Minimize the number of FRs and constraints.*

Corollary 3 (Integration of Physical Parts) *Integrate design features in a single physical part if FRs can be independently satisfied in the proposed solution.*

Corollary 4 (Use of standardization) *Use standardized or interchangeable parts if the use of these parts is consistent with the FRs and constraints.*

Corollary 5 (Use of Symmetry) *Use symmetrical shapes and/or arrangements if they are consistent with the FRs and constraints.*

Corollary 6 (Largest Tolerance) *Specify the largest allowable tolerance in stating FRs.*

Corollary 7 (Uncoupled Design with Less Information) *Seek an uncoupled design that requires less information than coupled designs in satisfying a set of FRs.*

A.2 Theorems of Axiomatic Design

Theorem 1 (Coupling Due to Insufficient Number of DPs) *When the number of DPs is less than the number of FRs, either a coupled design results, or the FRs cannot be satisfied.*

Theorem 2 (Decoupling of Coupled Design) *When a design is coupled due to the greater number of FRs than DPs (i.e., $m > n$), it may be decoupled by the addition of new DPs so as to make the number of FRs and DPs equal to each other, if a subset of the design matrix containing $n \times n$ elements constitutes a triangular matrix.*

Theorem 3 (Redundant Design) *When there are more DPs than FRs, the design is either a redundant design or a coupled design.*

Theorem 4 (Ideal Design) *In an ideal design, the number of DPs is equal to the number of FRs.*

Theorem 5 (Need for New Design) *When a given set of FRs is changed by the addition of a new FR, or substitution of one of the FRs with a new one, or by selection of a completely different set of FRs, the design solution given by the original DPs cannot satisfy the new set of FRs. Consequently, a new design solution must be sought.*

Theorem 6 (Path Independency of Uncoupled Design) *The information content of an uncoupled design is independent of the sequence by which the DPs are changed to satisfy the given set of FRs.*

Theorem 7 (Path Dependency of Coupled and Decoupled Design) *The information contents of coupled and de coupled designs depend on the sequence by which the DPs are changed and on the specific paths of change of these DPs.*

Theorem 8 (Independence and Tolerance) *A design is an uncoupled design when the designer- specified tolerance is greater than*

$$\left(\sum_{j \neq i, j=1}^{n} (\partial FR_i / \partial DP_j) \Delta DP_j \right)$$

in which case the nondiagonal elements of the design matrix can be neglected from design consideration.

Theorem 9 (Design for Manufacturability) *For a product to be manufacturable, the design matrix for the product, [A] (which relates the FR vector for the product to the DP vector of the product) times the design matrix for the manufactur- ing process, [B] (which relates the DP vector to the PV vector of the manufacturing process) must yield either a diagonal or triangular matrix. Consequently, when any one of these design matrices; that is, either [A] or [B], represents a coupled design, the product cannot be manufactured.*

Theorem 10 ((Modularity of Independence Measures) *Suppose that a design matrix [DM] can be partitioned into square submatrices that are nonzero only along the main diagonal. Then the reangularity and semangularity for [DM] are equal to the products of their corresponding measures for each of the non-zero submatrices.*

Theorem 11 (Invariance) *Reangulanty and semangularity for a design matrix [DM] are invariant under alternative orderings of the FR and DP variables, as long as orderings preserve the association of each FR with its corresponding DP*

Theorem 12 (Sum of Information) *The sum of information for a set of events is also information, provided that proper conditional probabilities are used when the events are not statistically independent.*

Theorem 13 (Information Content of the Total System) *If each DP is probablistically independent of other DPs, the information content of the total system is the sum of information of all individual events associated with the set of FRs that must be satisfied.*

Theorem 14 (Information Content of Coupled versus Uncoupled Designs) *When the state of FRs is changed from one state to another in the functional domain, the information required for the change is greater for a coupled process than for an uncoupled process*

Theorem 15 (Design-Manufacturing Interface) *When the manufacturing system compromises the independence of the FRs of the product, either the design of the product must be modified, or a new manufacturing process must be designed and/or used to maintain the independence of the FRs of the products.*

Theorem 16 (Equality of Information Content) *All information contents that are relevant to the design task are equally important regardless of their physical origin, and no weighting factor should be applied to them.*

Bibliography

[1] D.A. Gebala and N.P. Suh. An application of axiomatic design. *Research in Engineering Design*, 3:149–162, 1992.

[2] M. Jackson. *Software Requirements & Specifications.* Addison-Wesley, 1995.

[3] S.J. Kim, N.P. Suh, and S.G. Kim. Design of software systems based on axiomatic design. *Robotics & Computer-Integrated Manufacturing*, 8(4):243–255, 1991.

[4] S.C. Lu and A.B. Baskin. Two-dimensional decomposition. Personal Communication, 1995.

[5] D.T. Ross. Application and extensions of sadt. *IEEE Computer*, 18(4):25–35, 1985.

[6] R. Rumbaugh. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[7] N.P. Suh. *The Principles of Design.* Oxford University Press, New York, 1990.

[8] S. Svensson. Case study. In Software Engineering Division of Saab Military Aircraft. Personal Communication with Mats Nordlund, 1994.

[9] E. Yourdon and L. Constantine. *Structured Design.* Academic Press, 1975.