2010 International Conference on E-business, Management and Economics
IPEDR vol.3 (2011) © (2011) IACSIT Press, Hong Kong

Transformation of analysis model to design model

Lalji Prasad
Computer Science Engineering
Truba College of Engineering & Technology
Indore, India
lalji.research@gmail.com

Shyam Patidar
Computer Science Engineering
Medi-Caps Institute of Technology
Indore, India
shyam.p8@gmail.com

Narendra Pal Singh Rathore
Computer Science Engineering
SSSIST
Sehore, India
enggnarendra1029@gmail.com

Sarita Bhaduria
Department of Electronics
MITS
Gwalior, India
Saritamits66@yahoo.co.in

***Abstract*— In this paper we try to minimize the complication of how to transform analysis model into design model. We work to transform analysis model into four levels of design detail: the data structure, the system architecture, the interface representation, and the component level detail. During each design activity, we apply basic concepts and principles that lead to high quality of software.**

I. INTRODUCTION

Design is a meaningful engineering representation of something that is to be built. It can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria for "good" design. In the software engineering context, design focuses on four major areas of concern: data, architecture, interfaces, and components Design begins with the requirements model. The specification is composed of the design models that describe data, architecture, interfaces, and components [11]. In each product of the design process, how do I ensure that I've done it right? At each stage, software design work products are reviewed for clarity, correctness, completeness and consistency with the requirements and with one another.

Software design, like engineering design approaches in other disciplines, changes continually as new methods, better analysis, and broader understanding evolve. Software design methodologies lack the depth, flexibility, and quantitative nature that are normally associated with more classical engineering design disciplines. However, Methods for software design do exist, criteria for design quality are available, and design notation can be applied.

The rest of the paper is organized as follows. In Section 2, we present some basic concepts of design that will be used in our paper. In Section 3, we discuss the method. Section 4 concludes the paper.

II. DESIGN

Software design deals with transforming the customer requirements, as described in the SRS document, into a form (a set of documents) that is suitable for implementation in a programming language [10]. A good software design is seldom arrived by using a single step procedure but rather through several iterations through a series of steps [9]. Design activities can be broadly classified into two important parts: Preliminary (high-level) design and Detailed design. The meaning and scope of two design activities (i.e. high-level and detailed design) tend to vary considerably from one methodology to another. High-level design means identification of different modules and the control relationships among them and the definition of the interfaces among these modules. The outcome of high-level design is called the program structure or software architecture. Many different types of notations have been used to represent a high-level design. A popular way is to use a tree-like diagram called the structure chart to represent the control hierarchy in a high-level design [8]. However, other notations such as Jackson diagram or

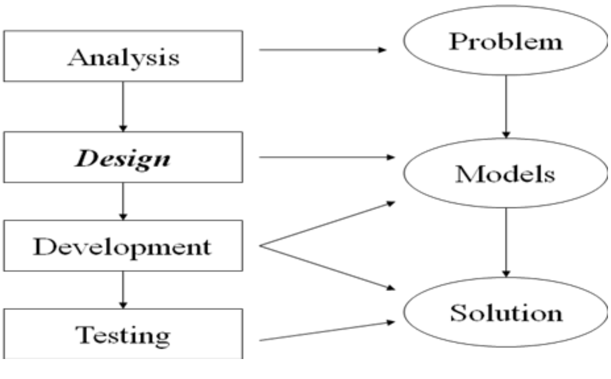

Figure 1. Map Design with System Products

Warnier-Orr [11] diagram can also be used. During detailed design, the data structure and the algorithms of the different modules are designed.

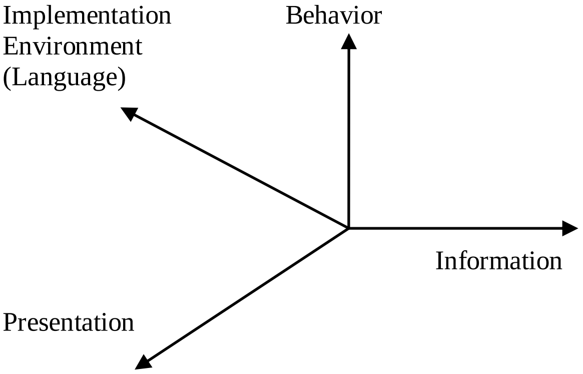

Figure 2. A Design model with a new dimension.

A. Design Model

The analysis model is refined and formalized to get a design model. During design modeling, we try to adapt to the actual implementation environment. In design space, yet another new dimension has been added to the analysis space to include the implementation environment. This is show in fig. 2. This means that we want to adopt our analysis model to fit in the implementation model at the same time as we refine it. But changes cannot be avoided so, we develop a new model.

III. METHOD

Software design sits at the technical kernel of software engineering and is applied regardless of the software process model that is used [9]. Beginning once software requirements have been analyzed and specified, software design is the first of three technical activities—design, code generation, and test—that are required to build and verify the software. Each activity transforms information in a manner that ultimately results in validated computer software. Each of the elements of the analysis model provides information that is necessary to create the four design models required for a complete specification of design. The flow of information during software design is illustrated in Figure 3. Software requirements, manifested by the data, functional, and behavioral models, feed the design task. Using one the design methods. The design task produces a data design, an architectural design, an interface design, and a component design [6]. The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software.

The data objects and relationships defined in the entity relationship diagram and the detailed data content depicted in the data dictionary provide the basis for the data design activity [6]. Part of data design may occur in conjunction with the design of software architecture. More detailed data design occurs as each software component is designed.

The architectural design defines the relationship between major structural elements of the software, the "design patterns" that can be used to achieve the requirements that have been defined for the system, and the constraints that affect the way in which architectural design patterns can be applied [8]. The architectural design representation—the framework of a computer-based system—can be derived from the system specification, the analysis model, and the interaction of subsystems defined within the analysis model. The interface design describes how the software communicates within itself, with systems that interoperate with it, and with humans who use it. An interface implies a flow of information (e.g., data and/or control) and a specific type of behavior. Therefore, data and control flow diagrams provide much of the information required for interface design.

The importance of software design can be stated with a single word—quality [9]. Design is the place where quality is fostered in software engineering. Design provides us with representations of software that can be assessed for quality. Design is the only way that we can accurately translate a customer's requirements into a quality software product or system. Software design serves as the foundation for all the software engineering and software support steps that follow. Without design, we risk building an unstable system—one that will fail when small changes are made; one that may be difficult to test; one whose quality cannot be assessed until late in the software process, when time is short and many dollars have already been spent. Explaining the idea/concept of something usually with graphical diagrams with the intention to build from the explanation. The design is a representation of a product or a system with sufficient detail for implementation. From our understanding of the problem, we start building the software. Translate the analysis model into the design model. Map the information from the analysis model to the design representations - data design, architectural design, interface design, procedural design.

A. The design process should not suffer from tunnel vision

A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job [11].

B. The design should be traceable to the analysis model

Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.

C. The design should minimize the intellectual distance between the software and the problem as it exists in the real world.

That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.

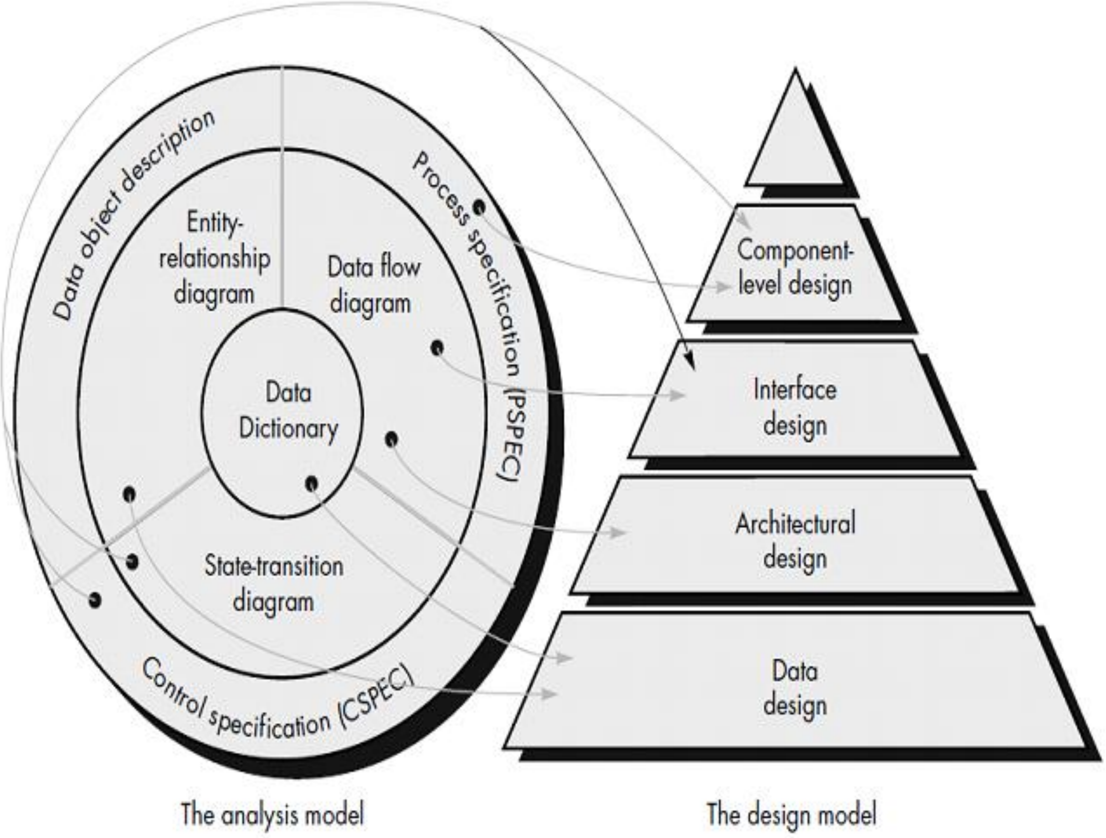

Figure 3. Translating the analysis model into a software design

D. The design should exhibit uniformity and integration

A design is uniform if it appears that one person developed the entire thing. Rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.

E. Design is not coding, coding is not design

Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.

F. The design should be assessed for quality

A variety of design concepts and design measures are available to assist the designer in assessing quality.

G. The design should be reviewed to minimize conceptual errors

There is sometimes a tendency to focus on minutiae when the design is reviewed, missing the forest for the trees. A design team should ensure that major conceptual elements of the design (omissions, ambiguity, and inconsistency) have been addressed before worrying about the syntax of the design model.

In the construction process, we construct the system using both the analysis model and requirements model. We design and implement the system. Firstly, a design model is made where each object will be fully specified. This model will then form an input data for the rest process.

H. When to do this transition?

The transition from the analysis model to the design model should be made when the Consequences of the implementation environment start to show. This is shown in fig. 5 with adaptation of DBMS distributed environment, real-time adaptations etc. then it is fine to be quite formal in the analysis model.

But if these circumstances will strongly affect the system structure, then the transition should be made quite early. The goal is not to redo any work in a later phase that has done in an earlier phase. We try to keep an ideal analysis model of a system during the entire system life cycle.

A design model is a Specialization of the analysis model for a specific implementation environment [1]. Such changes are then easily incorporated because it is the same analysis model that will form should not affect the analysis model as we do not want changes due to design decisions to be illustrated in the analysis model at fig 3.

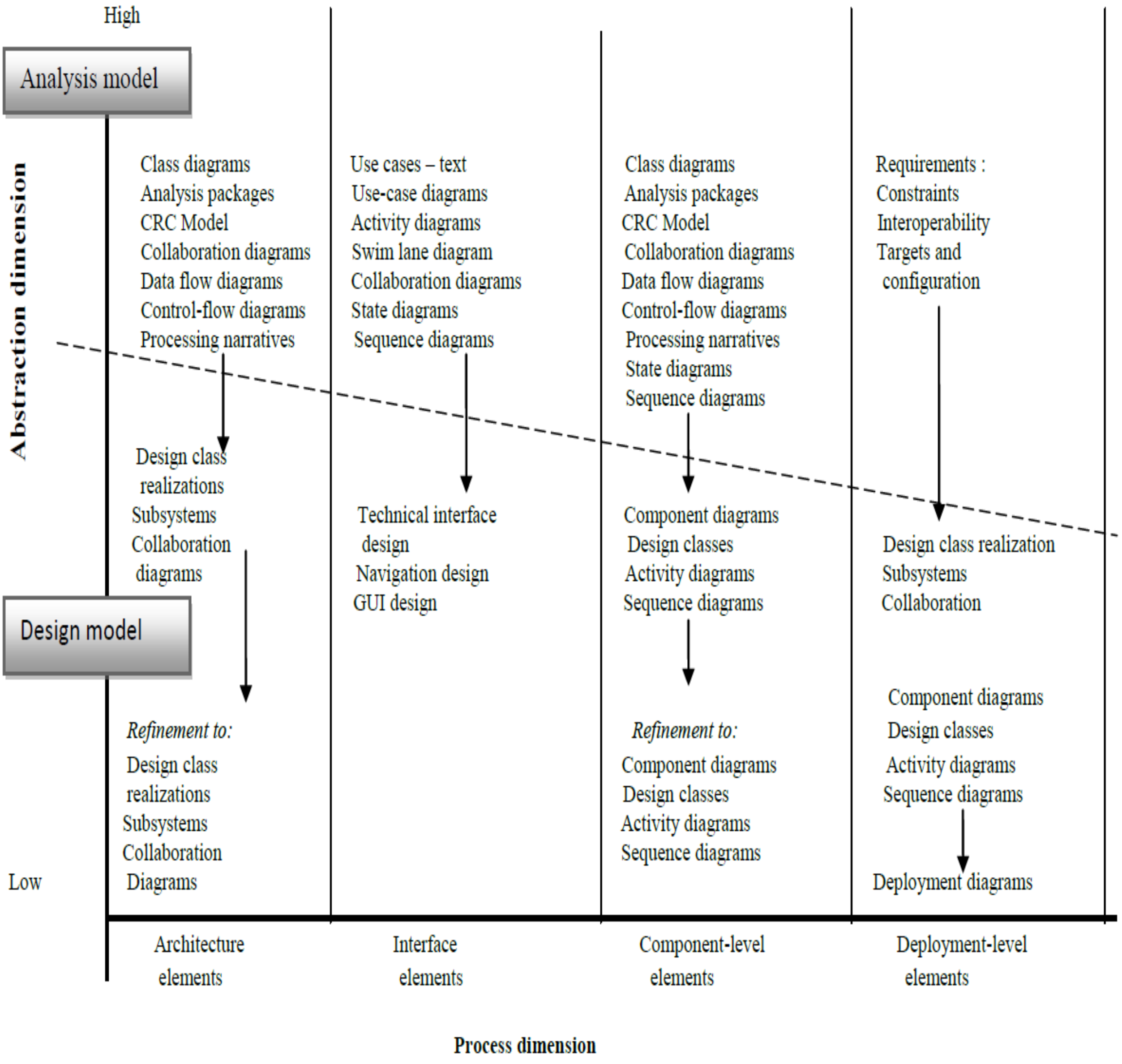

Figure 4. Dimensions of the design model

I. When Changes should be made?

If a change of the design model comes from a logical change in the system, then such changes should also be made in the analysis model.

We use a concept of block now to describe the intention of how the code should be produced. The blocks are the design objects. One block normally tries to implement should not affect the analysis model as we do not want changes due to design decisions to be illustrated in the analysis model at fig 2.

IV. CONCLUSION

Design is the technical kernel of software engineering. During design, progressive refinements of data structure, architecture, interfaces, and procedural detail of software components are developed, reviewed, and documented.

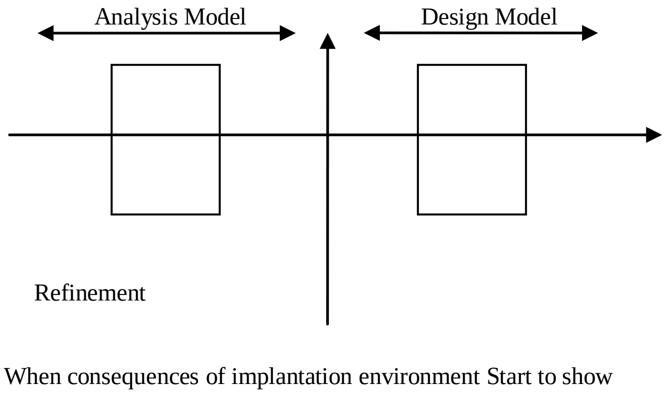

Figure 5. Transition

Design results in representations of software that can be assessed for quality. A number of fundamental software design principles and concepts have been proposed over the past four decades. Design principles guide the software engineer as the design process proceeds. Design concepts provide basic criteria for design quality. Design representations are tied to the others, and all can be traced back to software requirements. It is interesting to note that some programmers continue to design implicitly, conducting component-level design as they code. This is akin to taking the design pyramid and standing it on its point—an extremely unstable design results. The smallest change may cause the pyramid (and the program) to topple. Each method enables the designer to create a stable design that conforms to fundamental concepts that lead to high quality software. We try to solve the problem by rushing through the design process so that enough time will be left at the end of the project to uncover errors that were made because we rushed through the design process.

REFERENCES

[1] Asada, T., et al., "The Quantified Design Space," in *Software Architecture* (Shaw, M. and D. Garlan), Prentice-Hall, 1996, pp. 116 −127.

[2] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice,* Addison-Wesley, 1998.

[3] Dennis, J.B., "Modularity," in *Advanced Course on Software Engineering* (F.L. Bauer, ed.), Springer-Verlag, 1973, pp. 128–182.

[4] Freeman, P., "The Context of Design," in *Software Design Techniques,* 3rd ed.(P. Freeman and A. Wasserman, eds.), IEEE Computer Society Press, 1980, pp. 2–4.

[5] Kazman, R. et al., *The Architectural Tradeoff Analysis Method,* Software Engineering Institute, CMU/SEI-98-TR-008, July 1998.

[6] Wasserman, A., "Principles of Systematic Data Design and Implementation," in *Software Design Techniques* (P. Freeman and A Wasserman, ed.), 3rd ed., IEEE Computer Society Press, 1980.

[7] Myers, G., *Composite Structured Design,* Van Nostrand, 1978.

[8] Somerville, I., *Software Engineering,* 3rd ed., Addison-Wesley, 1989.

[9] Belady, L., Foreword to *Software Design: Methods and Techniques* (L.J. Peters,author), Yourdon Press, 1981.

[10] Rajib Mall, Fundamentals of software engineering, 3rd ed., 2009

[11] Roger S. Pressman, software engineering, "A Practitioner's Approach, 5th ed.