Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

REQUIREMENTS, DESIGN AND BUSINESS PROCESS REENGINEERING AS VITAL PARTS OF ANY SYSTEM DEVELOPMENT METHODOLOGY

by
Alicja Ruszala

This thesis analyzes different aspects of system development life cycle, concentrating on the requirements and design stages. It describes various methodologies, methods and tools that have been developed over the years. It evaluates them and compares them against each other. Finally a conclusion is made that there is a very important stage missing in the system development life cycle, which is the Business Process Reengineering Stage.

REQUIREMENTS, DESIGN AND BUSINESS PROCESS REENGINEERING AS VITAL PARTS OF ANY SYSTEM DEVELOPMENT METHODOLOGY

by
Alicja Ruszala

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January, 1995

APPROVAL PAGE

REQUIREMENTS, DESIGN AND BUSINESS PROCESS REENGINEERING AS VITAL PARTS OF ANY SYSTEM DEVELOPMENT METHODOLOGY

Alicja Ruszala

Dr. B.A. Suresh, Thesis Adviser — Date
Professor of Computer Science, NJIT

Dr. J. McHugh , Committee Member — Date
Professor of Computer Science, NJIT

Dr. Peter Ng, Committee Member — Date
Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Alicja Ruszala

Degree: Master of Science in Computer Science

Date: January, 1995

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology,
  Newark, New Jersey, 1995

- Bachelor of Science in Computer Science, Mathematics and Political Science,
  Rutgers University,
  New Brunswick, New Jersey, 1987

Major: Computer Science

This Thesis is dedicated to my husband, Dariusz Ruszala

ACKNOWLEDGMENT

The author wishes to express her sincere gratitude to her advisors, Doctor James McHugh and Mr. Michael Tress, Coordinator for Student Advisement, for their guidance and moral support.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	11
2 OBJECTIVE	12
3 SYSTEM ENGINEERING	13
3.1 Introduction	13
3.2 Benefits of Good Requirements	14
3.3 Requirements Development	14
3.4 Requirements Engineering and Architectural Design	15
3.5 Fundamentals of Requirements Engineering	15
3.6 Requirements Engineering Practices	16
3.7 Requirements Verification	18
4 STRUCTURED ANALYSIS	19
4.1 Introduction	19
4.2 The Components of Structured Analysis	19
4.3 Object Types	20
4.4 Structured Analysis Requirements Tools	20
4.5 Different Approaches to Structured Analysis	21
4.6 Development Lifecycle Models	22
4.6.1 Baseline Management and Waterfall Models	22
4.6.2 Incremental Development	23
4.6.3 The Transform Model	23

TABLE OF CONTENTS
(Continued)

Chapter Page

4.6.4 Prototyping 23

4.6.5 The Spiral Model 24

4.7 Entity-Relationship Approach to Data Modeling 24

4.8 Spiral Model 25

5 OBJECT - ORIENTED ANALYSIS 27

5.1 Introduction 27

5.2 Object Oriented Analysis 28

5.3 Object Oriented Principles 28

5.4 Characteristics of Object Oriented Design 29

5.4.1 Encapsulation 29

5.4.2 Inheritance 29

5.5 Object Oriented Requirements Analysis 30

5.5.1 Functional Decomposition 30

5.5.2 Data Flow Approach 31

5.5.3 Information Modeling 32

5.6 Object-Oriented Analysis 32

6 KNOWLEDGE-BASED, HYPERTEXT AND HYPERMEDIA ANALYSIS . 35

6.1 Introduction 35

6.2 Knowledge-Based CASE Tools 35

6.3 Evaluation of Knowledge-Based Case Tools 36

TABLE OF CONTENT
(Continued)

Chapter Page

6.4 Domain-Specific Knowledge 37

6.5 The MHEG Standard 39

6.6 MH Object Classes 39

6.7 Basic Objects Representation 40

6.7.1 Content and Projector Classes 40

6.8 Other Multimedia and Hypermedia Standardization Issues 40

6.9 Multimedia 41

6.10 Design Goals and Issues 41

6.11 Group Decision Support 42

7 PROTOTYPING 44

7.1 Introduction 44

7.2 Prototype in Waterfall Life Cycle 44

7.3 Rapid Prototyping 45

7.4 Evolutionary Prototyping 46

7.5 The Throwaway Prototype 47

8 A COMPARISON OF THE MAJOR APPROACHES TO SOFTWARE SPECIFICATION AND DESIGN 48

8.1 Introduction 48

8.2 A Comparison of Techniques for the Specifications 49

8.3 Comparison of Techniques 53

TABLE OF CONTENTS
(Continued)

Chapter Page

8.4 A Comparison of Object-Oriented and Structured Development Methods 54

8.5 Differences Between OOD and SD 54

8.6 Comparing Development Paradigms 55

8.7 Characterizing Object-Oriented Systems 56

8.8 The Operational Versus the Conventional Approach to Software Development 58

8.9 Weaknesses of the Conventional Approach 64

8.10 Weaknesses of the Operational Approach 65

8.11 A Strategy for Comparing Alternative Software Development Life Cycle Models 66

9 VERIFICATION AND VALIDATION 70

9.1 Introduction 70

9.2 Product Reviews 70

9.3 Verifying and Validating Software Requirements and Design Specifications 72

10 BUSINESS PROCESS REENGINEERING 78

10.1 Introduction 78

10.2 Sound Basis for a New System 78

10.3 Business Process Reengineering 78

10.4 Business Process Reengineering Specialist 79

11 CONCLUSION 80

REFERENCES 81

CHAPTER 1

INTRODUCTION

This thesis analyzes different aspects of system development life cycle, concentrating on the requirements and design stages. It describes various methodologies, methods and tools that have been developed over the years. It evaluates them and compares them against each other. Finally a conclusion is made that there is a very important stage missing in the system development life cycle, which is the Business Process Reengineering Stage.

CHAPTER 2

OBJECTIVES

The objective of this thesis is to present different aspects of the requirements and design stages of the system life cycle, compare and analyze the existing methodologies as well as to introduce the business process reengineering stage into the development life cycle.

CHAPTER 3

SYSTEM ENGINEERING

3.1 Introduction

System is build to satisfy set forward objectives, and to process certain functions. The system consists of software, hardware, data and people. The system engineering process is iterative in nature and uses a structured approach to develop the system, keeping in mind fulfilling all of the system objectives. System engineering is a technique to manage both technical and management aspects of the system. The technical aspect involves transforming operational needs into system specifications. The management aspect involves creating the system team consisting of designers, developers, etc., managing the implementation process, monitoring the schedule, cost and risks involved as well as the progress in satisfying the objectives.

During the system engineering process the requirements are gathered and allocated to lower levels. The allocation process assigns parts of the requirements to the lower levels of the system hierarchy. The hierarchy gets established by dividing the system into functional areas at lower levels. In the allocation process requirements change form and become derived requirements. The process transforming the requirements is usually the design phase. During the design the allocation of requirements gets tested and assessed against what can be achieved. The assessment of a system's capability against the specification requirements is called verification. Verification can be done by testing, demonstration, analysis or examination of documentation methods.

3.2 Benefits of Good Requirements

Sound requirements are the basis for meeting the performance and cost goals, and making sure that the project is successful. There are many benefits in developing good requirements: agreement on the objectives of the system, and the acceptance criteria for the delivered system, a good basis for resource estimation, satisfaction with the system's usability and maintainability, as well as the well planned resources. The value of good requirements increased significantly with the size and complexity of the system.

3.3 Requirements Development

The development of requirements based on the specified objectives has many steps and is iterative in nature. The steps involved are: creating the objectives, specifying functions, establishing performance criteria, define the operations, evaluate cost and risks, allocate requirements, specify configuration, document the requirements [7].

There are some system measures that can characterize the system. They are: quantity, quality, coverage, timeliness and availability. Quantity is the capacity of the system, quality is the accuracy of the system, coverage is the functional area covered by the system, timeliness is the time to process the data and finally availability is the open window of the system for processing.

There are also other standards applied in the requirements area. They are: reliability, maintainability, human factors, perts, materials, processes, logistics, and

safety. Those standards are measured based on the already available experience of similar systems.

3.4 Requirements Engineering and Architectural Design

Requirements analysis is answering the "what" question of a problem. It is based on objectives, is implementation-free. The design answers the "how" question, it provides the solutions for meeting the requirements.

Requirements engineering is iterative in nature, therefore the development of the architectural design is a process of altering the requirements and design analysis, with more detail brought out at step. The input to each design stage is the output from the requirements analysis.

3.5 Fundamentals of Requirements Engineering

After all of the top-level requirements have been defined, they are then allocated through the lower levels of the hierarchy. This is done by applying top down analysis. A system is decomposed into a hierarchy of elements. This is done for example by functional or physical decomposition. After the lowest-level elements are defined, they are separately developed and then integrated to form the next-larger elements. The result should be an optimized and balanced system, which in reality does not occur often. Usually, because of different types of constraints incurred during the analysis stages, requirements do not get completely identified and allocated in the first round, and later on some of them have to be reanalyzed or added.

Most of the requirements are allocable, however, there are some which are not. Requirements that are non-allocable specify the environment, operations, standards. Allocable requirements can be allocated directly or indirectly. The directly allocated requirements, are divided among several lower-level elements. The indirectly allocated requirements change their form through a derivation analysis which transforms them in order to test against them.

3.6 Requirements Engineering Practices

There are three levels into which the requirements engineering practices can be divided. The first one is the most general approach which is called a methodology, the next one, more specific one is called a method, and lastly the lowest level, most formalized approach is called a tool. Examples of methodologies are the baseline management, prototyping, incremental development and spiral methodologies. Methods can be divided into four categories: data oriented, control oriented, process oriented and object oriented. Most methods have some of the characteristics of all the categories [20].

Process-oriented methods have to do with the system transformation of inputs into outputs with less emphasis on the data itself and control aspects. Examples of process-oriented methods are: structured analysis (SA), Structured Analysis and Design Technique (SADT), operational/executable models such as PAISLey and Descartes, and formal methods such as Vienna Design Method (VDM) and Z fit into this category.

Data-oriented methods have to do with the data structure of a system. Examples of data-oriented methods are: JSD and entity-relationship modeling. Control-oriented methods have to do with synchronization, deadlock, exclusion, concurrency and process activation and deactivation. Examples of control-oriented methods are: SADT and flowcharting. Object-oriented methods, base analysis on classes of objects and object interaction.

Tools support the methods and in turn also methodologies. Example of a tool which is also a method is Software Requirement Engineering Methodology (SREM). It combines graphics and a requirements language, and is still in development. Examples of tools which are not a method at the same time are: Problem Statement Language/Problem Statement Analyzer (PSL/PSA), Software through Pictures. Fourth generation languages are also an important class of tool. They are used for system interface modeling and database access and reporting systems. Knowledge-based requirements generation, while far from practical application is an important technology for the future.

There are many other tools that assist in defining the requirements. For example: functional block which defines architecture, functional areas, but does not define hierarchy, sequences and database. NxN chart, developed by Lano defines interfaces and relationships but does not define hierarchy and database sequences. The NxN chart can also be functionally decomposed. Then there is data flow diagram which defines data flows, control flows, database but does not control functions, hierarchy and sequences. Finally, functional flow defines functions and sequences but does not define

hierarchy, interfaces and database. Some other tools are data models, process specifications in the forms of structured English, decision trees or decision tables as well as state transition diagrams, showing modes of operation.

3.7 Requirements Verification

There has to be a verification plan designed during the requirements phase in order to test the system completely. This plan should define all verification data to be produced by the programs as well as the availability time. The specified data to be produced, becomes the basis for the design reviews.

CHAPTER 4

STRUCTURED ANALYSIS

4.1 Introduction

In order to conduct Structured Analysis in the system development process, a methodology has to be applied. A methodology helps to develop an information system in a disciplined way. Structured Analysis applies to a number of methodological approaches that can be used by the analyst. A methodology, is a composition of rules and procedures guiding the analytical activity.

A methodology should contain at least four components: a conceptual model of constructs essential to the system, procedures pointing the steps to proceed, guidelines specifying things to be avoided, and finally, a set of criteria for evaluating the quality of the system.

4.2 The Components of Structured Analysis

Systems have three major construct-types of data, activity, and control. Different types of systems place different emphases on the construct types. Data-oriented and activity-oriented systems should have structured top-down analysis applied. Control-oriented systems can have either top-down, leveled, hierarchical approach, or the analysis can begin at the level the system is responsive to the environment, and then be decomposed to the lower levels and eventually creating a hierarchical structure.

The analyst should begin with defining the current physical system, and then creating models of the system without any solution constraints. The current physical system definition should be developed into the logical model. This logical model will help the analyst in developing the new systems requirements. A lot of requirements from the existing system can probably be incorporated into the new model.

4.3 Object Types

There have been six object types identified in structured analysis for the business-oriented analysis: process, data flow, data store, external entity, data group and data element. Process transforms and manipulates data, exchanges data with other processes, stores, and sources. Data flow passes data between sources and destinations, both external and internal. Data store holds data, for a reference or transformation. External entity is an external activity that interacts with processes by means of data. Data group is a cluster of data, and a component of some data flows. Data element is a basic unit of data. Structured analysis should identify the instances of these six object types and show the relationships between the tasks.

4.4 Structured Analysis Requirements Tools

There are four major tools used in traditional structured analysis which help in collecting the requirements of the system: data flow diagram, data dictionary, the primitive process specification and structured walkthrough.

Data flow diagram has a graphic representation of external entities, processes, data flows and data stores to show the progressive transformation of data. Data dictionary contains descriptions of each data object from the data flow diagram. The primitive process specification is a usually structured English specification of the procedure to execute the action of the primitive process. Structured walkthrough is a inspection meeting during which some products of the development effort are presented, analyzed and critiqued, so that requirements problems could be detected early in the cycle and the product could be enhanced and approved.

4.5 Different Approaches to Structured Analysis

There are many different approaches to Structured Analysis. One of the approaches is the Structured Analysis and Design Technique (SADT) proposed by Doug Ross and his colleagues at SofTech, Inc. In SADT, analyst performs top-down design, by progressing from the conceptual abstractions to implementation components. Using the activity diagram, the analyst describes the interaction of data between the activities as well as the mechanisms that will execute each activity [8].

Another technique is Yourdon-DeMarco technique which is a top-down approach. The analyst decomposes the system and its functions through lower levels. Different approach, the Gane and Sarson, approach pays more attention than does the Yourdon-DeMarco approach to the identification of the data components of a system. Gane and Sarson propose the use of a data access diagram to describe the structure and contents

of data stores. The data access diagram shows the different entities in a data store and the access paths between them.

4.6 Development Lifecycle Models

The better-known lifecycle models are: Baseline Management and Waterfall Model, Incremental Development, Transform Model, Prototyping and Spiral Model.

4.6.1 Baseline Management and Waterfall Model

Those models provide high degree of management visibility and control. They are not appropriate, however, in some systems when it is difficult to determine the user's needs without some form of operational system to review. In these models, determination of requirements should be complete, before any implementation begins.

Baseline management differs from the waterfall in that it requires each lifecycle phase to generate defined products which have to pass a review and be placed under configuration control before the next phase begins. The waterfall model provided two primary enhancements to the baseline model: recognition of the feedback between stages and some usage of prototyping. There is also another version of the waterfall model - the risk-management of the waterfall model where in each step there is a validation and verification of the risk and reuse considerations. Waterfall model requires fully documented requirements and design, which sometimes is redundant, specially in development using fourth-generation languages.

4.6.2 Incremental Development

The incremental development lifecycle contains several development projects, each of which delivers an operational product. Each delivered increment provides some needed operational capability. Feedback from users of the operational systems may affect requirements for later increments. Each increment in this model represents a full development cycle, including requirements analysis.

4.6.3 The Transform Model

The transform model works when there is a capability to automatically transform a formal specification of a software product into a program satisfying the specification. The transform model eliminates unstructured code modifications, due to repeated optimizations. It reduces the testing time of the intermediate design, coding and testing.

4.6.4 Prototyping

Prototyping allows to built some system capability to be tested by the users. This helps to determine and verify the requirements. Several successive prototypes will usually be built. After the requirements are finalized with the use of a prototype, they should be developed and then later developed.

4.6.5 The Spiral Model

The Spiral Model allows for combinations of the baseline management, prototyping, and incremental models to be used for various portions of a development. It stresses the risk management, and calls for evaluations of the progress and feasibility of the project.

4.7 Entity-Relationship Approach to Data Modeling

The entity-relationship model is a tool that helps the system analyst and the user communicate during the requirements analysis stage of the development. The ER model depicts graphically the logical database design [5].

The ER model contains entities, and relationships. An entity is an object about which information is to be collected, and is usually depicted as a noun. Relationship on another hand is described using a transitive verb, and it exists between entities. The degree of a relationship is the number of entities associated in the relationship. A recursive relationship occurs if there is a relationship from objects in an entity to other objects in the same entity.

The connectivity of a relationship is the mapping of the associated entity occurrences in the relationship. The values for connectivity are either 'one' or 'many, which defines the cardinality. The basic types of connectivity are: one-to-one, one-to-many, and many-to-many. Each connectivity has upper bound and lower bound. If the lower bound of connectivity is one or many, it is a total or obligatory relationship. When the lower bound is zero, it is a partial relationship.

When an entity is partitioned by different values of a common attribute, a generalization/specialization occurs. A weak entity occurs if the existence of an entity depends on the existence of another entity type. A gerund is a noun converted from a verb, and it corresponds to an entity type converted from a relationship type. An entity attribute is an adjective, an adverb is a relationship attribute.

The entity identifier, which is an attribute identifies the entities. A weak entity is identified by the identifier from the parent entity plus an identifier that uniquely identifies the weak entity. Relationships are identified by utilizing the identifiers of the entities involved in the relationship. A gerund is identified by its own unique identifier, like an entity, plus the identifiers from any associated entities, like a relationship.

4.8 Spiral Model

The spiral model of software development is a risk-driven approach to the software development, rather than document-driven or code-driven process seen in other models. Spiral Model, is an improvement over the other models, because it incorporates their best characteristics. The spiral model can be used for new as well as maintenance development. The spiral model can incorporate any combination of different approaches to software development, specification, prototype, or other stages.

While a software process model determines the order of the software development stages involved as well as steps involved in transitioning from one stage to the next, methodology concentrates on the representation of the products from each phase. A

process model focuses on the tasks that have to be done in each phase and their duration [3].

The spiral model incorporates features of many other models and provides with guidelines for using different combinations of models for a particular software development. A sequence of the same steps is repeated in each cycle of the spiral model. Therefore, each cycle starts with the identification of the objectives of that portion of that state, product implementation alternatives as well as constraints and roadblocks. Then the alternatives are evaluated in terms of the objectives and constraints. Prototyping is applied to help evaluate the risks involved. If many risks are identified, then an evolutionary development step can be applied. Also modified waterfall approach gets applied in order to incorporate incremental development if there are risks involved. Finally a product review ends each cycle in the spiral model.

The advantage of the spiral model is that it incorporates proven characteristics of the existing software development models, and at the same time avoids their negatives or problems. The spiral model is flexible enough to incorporate different approaches such as the reuse of existing software. It easily adapts changes in the software product to the final version of the system, allows for system growth. It assists in eliminating not feasible alternatives early in the process, and it allows for early detection of errors. Also, where it is applicable, it helps point out failure projects in their early stages.

Currently, the Spiral Model is not widely applied. Therefore it does not have a history of use which would speak by itself for its usefulness. There has to be more work done in advertising this model to the development houses, to make sure that more

developers apply it and get convinced on their own of the benefits of the spiral model. The model also needs some further elaboration, Some of the problems of the Spiral Model are the fact there aren't very many software companies using this model. Therefore the spiral model needs to be more widely used, has to be really incorporated into the development process of most of the developmental efforts. Only then, this model will receive full recognition from the professional community.

CHAPTER 5

OBJECT - ORIENTED ANALYSIS

5.1 Introduction

Object Oriented Analysis (OOA) is based on the objects and attributes, classes and members, wholes and parts concepts. There are three basic methods of human organization: object and attributes, classification structure and assembly structure, which OOA applies to requirements specification. This type of analysis concentrates mainly on problem-space understanding. Problem-space understanding means understanding of the process that needs to be automated as well as the environment in which the users operate.

OOA looks at object attributes and services on those attributes as a whole. This is opposite to the separate and incomplete way other analysis methods deal with attributes and services. OOA allows to analyze and specify requirements using minimal dependency between one object and others which is self-contained partitioning. It applies commonality, and consistent representation for analysis and design .

The advantage of using an object in analysis is that it is an abstraction of the real world, and therefore it helps in understanding of the problem-space. During the major steps of OOA, many model layers are produced such as subject, object structure, attribute and service layers. This collection of model layers and be revised, and it is easily manageable. [10].

5.2 Object Oriented Analysis

There are three major difficulties in the software development process, for the systems analysts to control and grasp. They are problem-space understanding, person-to person communication, and continual change.

The analysts need to understand the problem space and be able to extract the problem space and requirements from the users. Ultimately they need to validate their understanding of the problem-space to the user. Software methods assist people communicate such an understanding between each other. The changes of the requirements need to be incorporated into the requirements gathering process

5.3 Object Oriented Principles

Four major OOA principles: abstraction, information hiding, inheritance, and methods of organization are used to manage the requirements and design.

The abstraction principle relies on selecting the most relevant scope rather than concentrating on the whole scope. One type of the abstraction is the procedural abstraction which brakes down the requirements processing into substeps. The data abstraction, on the other hand, is the main principle of the object-oriented analysis, and it is the basis for the primary organization of thinking and specification.

During the OOA, the attributes of objects, and the services which manipulate those attributes are defined. The OOA approach treat the attributes and services as an intrinsic whole, interdependent on each other

5.4 Characteristics of Object Oriented Design

5.4.1 Encapsulation

Encapsulation is a principle used during the system design. It proclaims that each component of a program should support only a single design decision. This principle simplifies the complexity of the system design, and also minimizes the maintenance in developing a new system. Changes in requirements become less of an issue due to the fact that all of the requirements are designed with encapsulation in mind.

5.4.2 Inheritance

Inheritance means that properties of an ancestor, get passed on to the successor, and I is a very powerful technique of propagating commonality. With inheritance common attributes and services along with specialization and extension of those attributes and services into specific cases can be specified once and then used in many places. In addition to receiving attributes and services from an ancestor, the successor, can build on them, extending those properties.

5.5 Object Oriented Requirements Analysis

There are three methods of organization: objects and attributes, assembly structures and classification structures on which the notation and approach of OOA are built. Also, there are four major approaches to requirements analysis: functional decomposition, data flow, information modeling and OOA. During the requirements analysis stage, the needs of the system are documented based on the user's input. During that

requirements gathering, question 'what' is asked, not 'how'. The analyst needs to find out what the system must do to satisfy the user, not how the system should be built. Requirements include functional descriptions, and operational specifications such as reliability, availability, ease of use, performance and maintainability. Requirements also include system interfaces, design constraints and software environment.

5.5.1 Functional Decomposition

Functional decomposition breaks each system business process into function, subfunctions and the functional interfaces. Functional decomposition maps problem space to functions and subfunctions. However, the completeness of this mapping cannot be verified. There are no existing methods right now to accomplish that task.

OOA applies functional decomposition to define services for a specific object. Functional decomposition is very helpful in breaking a complicated service into less complex subservices for simplification reasons. Many techniques can be applied such as a block diagram or data flow diagram in order to depict service requirements. The whole OOA analysis, however, should not be based on the functional decomposition, because it is very difficult to apply this decomposition. Also because there are usually many changes to the functionality, it is very time consuming to keep redesigning the functional structure of services.

5.5.2 Data Flow Approach

Yet another technique of mapping the problem space into a technical representation is by applying the data flow approach. Data flow approach is also referred to as structured analysis. Data flow approach is based on collecting the data and control flows, their transformations, stores as well as terminators. This technique uses data flows and bubbles to depict the problem space. However, this technique does not follow the natural basic methods people use to manage complexity of a problem space.

The transition from analysis to design is a very complicated process, carrying a high risk of missing information and incomplete links between the analysis and design. These difficulties are related to the substantial differences in the representation of the same concepts in the analysis and design stages. While data flows are based on a network representation of bubbles and stores, structure charts, applied during the design are a hierarchical representation of modules.

5.5.3 Information Modeling

The information modeling tools include mainly the entity-relationship diagram. The entity-relationship diagram is a semantic data model which tries to capture the problem-space content. Information modeling involves gathering information about objects, attributes, relationships, supertypes, subtypes, and associative objects. [10]

The older information modeling strategy promotes creating a list of attributes, and then dividing them into object buckets, adding relationships between them, and normalizing the relationships. The newer information modeling strategy differs in the

first step. The initial step finds the objects first and then defines them with attributes.

There are many things missing from the information modeling. Among the missing areas are: services - the processing requirements for each object, encapsulated and treated with the attributes as an intrinsic whole, inheritance - explicit representation of attribute and service commonality, messages - a narrow, well-defined interface between objects, structure - classification structure and assembly structure as fundamental human methods of organization are not central issues but should be [10].

5.6 Object-Oriented Analysis

The object oriented approach incorporates objects including the attributes and exclusive services, and classification and inheritance. Object-oriented analysis builds on the best concepts from information modeling (entity-relationship diagrams) and the best concepts from Object Oriented Programming languages -OOP. Information modeling provides attributes, relationships, structure and an object that represents some number of instances of something in the problem space. OOP languages provide encapsulating of attributes and exclusive services, treating attributes and services as an intrinsic whole, portraying classification structure and explicitly expressing commonality through inheritance. The mapping is direct from the problems space to the model, instead of an indirect mapping from problem space to function/subfunction or problem space to flows and bubbles.

OOA is based on the uniform application of methods of organization, communication with messages - the interaction between the user and the system and the

interaction between instances in the system, behavior classification - the overall framework for identifying services to be provided by each component.

OOA consists of five major steps: identifying objects, identifying structures, defining subjects, defining attributes, defining services and message connections. Once the model is built, it is presented in five major layers: subject layer, object layer, structure layer, attribute layer, service layer.

Subject layer is a mechanism for controlling how much of a model a reader considers at one time. An object layer is an abstraction of data and exclusive processing on those data, reflecting the capabilities of a system to keep information about or interact with something in the real world.

Structure layer - represents complexity in a problem space. Classification structure portrays class-member organization, reflecting generalization-specialization. Assembly structure shows aggregation, reflecting whole and component parts. Structure reflects problem-space complexity, capitalizing on two of the pervading methods of organization used by people. In addition, classification structure provides a basis for subsequent inheritance, giving explicit representation of attribute and service commonality within such a structure.

The attribute layer - an attribute is a data element used to describe an instance of an object or classification structure. Attributes are data elements or logical groupings of data elements. Service Layer - a service is the processing to be performed upon receipt of a message. Services are identified on the diagram and specified on the object repository.

The notation and approach of OOA builds on the three constantly employed methods of organization - objects and attributes, aggregation and classification. OOA is a relatively young method and will continue to evolve [10].

CHAPTER 6

KNOWLEDGE-BASED, HYPERTEXT AND HYPERMEDIA ANALYSIS

6.1 Introduction

Knowledge-based CASE tools play an active part in the design of computer-based systems. Such tools, with in-built domain-specific knowledge, enhance both the performance and the appearance of intelligence. So far, not a lot of work has been completed in this area. Such knowledge may be provided in the form of generic models based on a thesaurus approach, and the technique can be applied to a knowledge-based CASE tool designed to support object-oriented design.

Despite the availability of many methodologies to support differing development approaches, the design of information systems remains largely a knowledge-intensive activity. It begins with an informal set of frequently vague requirements and ends up with a systematically defined formal object. Although contemporary computer-aided software engineering (CASE) tools provide assistance in carrying out many design tasks with improved efficiency, they are largely the results of the automation of established design techniques. In general, the fundamental characteristic of design is not addressed by existing CASE technology [16].

6.2 Knowledge-Based CASE Tools

Artificial intelligence (AI) technology can be used to develop so-called knowledge-based CASE tools (KB-CASE). Rather than validating a model that a human has constructed, KB-CASE tools are able to play an active part during the design process. They are capable of providing intelligent assistance when required in the form of advice, suggesting alternative solutions, helping to investigate the consequences of design decisions, and maintaining the availability of the design knowledge by providing information should a design decision be questioned or require explanation in retrospect. Such tools have an understanding of both the structure and the semantics of the design.

6.3 Evaluation of Knowledge Based Case Tools

A criteria for the evaluation of KB-case tools supporting specification acquisition involves the stage of design covered - which specifies which stage of the chosen design paradigm it attempts to support, user interface employed - the method by which a system receives information from the user. It also involves method used to drive design process - the method used to provide initial input to the system, whether it is directly supplied by the user or provided by some other means, whether continuous user input is required throughout the design process, or whether the process is largely automatic once initial information has been gathered. Also domain-specific knowledge - the use of predefined domain-specific knowledge within a system can enhance the appearance of intelligence and increase the efficiency of a design session are a part of it. Systems are therefore examined in terms of how well such domain knowledge is exploited. The next

criteria are the design technique used - the extent to which the various systems make use of the opportunity to automate appropriate established design techniques, the 'undo' facilities - whether a system provides a facility which allows a user to 'undo' chosen design decisions and investigate new possibilities, the learning ability - the extent to which a system exhibits a learning ability and whether it is capable of making use of any newly gained knowledge in the design process as well as ease of use - a primary function of CASE tools is to increase productivity. Tools are therefore examined as to their difficulty of use and usability.

Areas for further investigation include the transfer of knowledge gained in one design session through to other sessions. Also the area of domain-specific knowledge has to be further developed. Domain specific knowledge and the ability to reason with this knowledge would be of obvious advantage to an intelligent design tool [18].

6.4 Domain-Specific Knowledge

The use of predefined domain-specific knowledge can enhance the appearance of intelligence and increase the efficiency of a tool. Generic models may be used to exploit the similarity of systems by providing templates on which new systems may be based. The tool, having recognized an application domain, could present the generic model as an initial design attempt and customize it to the designer's requirements during the design session. Very little work has taken place in this area to date despite the fact that the use of domain-specific knowledge can potentially yield numerous benefits.

Domain-specific knowledge can improve the overall performance of a KB-case tool in terms of the increased appearance of intelligence - the tool appears to have previous knowledge of the application area, increased efficiency - the user is presented with fewer questions during a design session. It can also improve the overall performance in terms of improved quality of resulting designs - the quality and semantic accuracy of the generic models, and the mechanisms by which the tool interprets the knowledge represented by these models are factors influencing design quality.

It is difficult to foresee widespread acceptance and use of KB-CASE without greater support and acceptability of traditional CASE tools. KB-CASE performance can be improved by the use of domain-specific knowledge in terms of increased appearance of intelligence, increased efficiency and improved quality of resulting designs.

A thesaurus approach to providing domain-specific knowledge for use by KB-CASE tools may be used to construct generic models representing application domains. The effectiveness of the approach depends greatly on the accuracy and completeness of the generic models used, and the extent to which idiosyncrasies within a particular domain may be accommodated when compared to the appropriate generic model [18].

6.5 The MHEG Standard

The MHEG standard aims at defining a common base for many of the multimedia and hypermedia applications which will be developed in the forthcoming years in different fields. This includes training and education, videogames, and advertising; office

information systems, engineering, electronic books, computer-supported multimedia cooperative work, etc.

Object-oriented analysis and methodology have been found to be essential in providing a design for the standard. It is believed to provide the following advantages: data encapsulation, which hides the internal details of an MH object from their client applications, inheritance, which allows abstraction, sharing of common behavior among different kinds of objects, homogeneity of the MH object description, representation of the behavior of autonomous entities in a highly dynamic environment.

6.6 MH Object Classes

The design of MH object classes relies on the analysis of their common behavior and the commonality of properties between object categories. This can lead to a single or multiple inheritance scheme. Implementation is free and even an object-oriented scheme is not required to conform to the standard.

The MHEG standard provides a description of MH objects for each class, a precise definition of the representation of these objects, and a basecoded representation of these objects. Representation of an MH object is specified through the following four steps: informal text description, object-oriented definition - explains the class hierarchy, and the behavior of each class. Then the structure and semantics of each representation attribute of the object is described. A notation for the structure of the representation - The MHEG standard provides a set of equivalent notations for the

formal description of an MH object's structure. Finally, the coded representation is defined by applying encoding rules to the representation.

6.7 Basic Objects Representation

6.7.1 Content and Projector Classes

A basic object is the association of a content object and a projector object. A content object means encoded monomedia data and appropriate information for its decoding and presentation. Projector object means presentation attributes associated to a content object.

For class, audio content class, etc., general attributes are inherited from upper level classes and specific attributes describe the encoding parameters used in the object. The projector classes gather all the presentation parameters which are relevant for each data type. For example, area projector corresponds to parameters such as position of the object into the generic space. Audio projector contains attributes such as volume reference, stereo/mono, balance, direction/speed.

6.8 Other Multimedia and Hypermedia Standardization Issues

MHEG is not the only standardization group dealing with multimedia and hypermedia issues. Most likely in the future multimedia will be dealt with in many groups which will consider it as a natural extension of their current scope. However, the study of the overall framework and model for the development of multimedia and hypermedia has

been assigned to ISO, which will produce helpful guidelines for the design of multimedia services in various areas.

The MHEG standard will offer a generic tool for a broad range of multimedia services or applications which are intended to be used in a communication environment. Because of its specificity, the MHEG standard will provide facilities to represent and encode multimedia synchronization and hyperlinks, while taking into account real-time and interchange requirements [16].

6.9 Multimedia

The reference models for generalized hypertext systems, commonly called hypermedia systems, may be more readily formulated within the object-oriented system paradigm than other approaches currently under consideration for hypertext alone.

6.10 Design Goals and Issues

There are four major goals for the hypertext system: hardware independence, flexible user interface, multi-user support and hardening.

The need for hardware independence, is important, since functional capabilities and performance of hardware platforms present a moving target, software tied to a particular platform is doomed to early obsolescence. Furthermore, if a system is to be useful to a wide base of users without requiring the replacement of currently existing equipment, the software must be as portable as possible.

Issues surrounding user interface design have to do with user preference for a particular style, access for the handicapped and multiple language support. Thus a winning architecture should support user interface adaptability.

The goal of hardening is a set of administrative functions such as provisioning for multiple levels of security, password protection for system access, database integrity, and usage tracking for possible chargeback. A system designer also must consider nontechnical issues, such as the social setting and legal ramifications of media distribution.

Systems intended for use by the public must be simple to use and provide built-in help and tutoring support so that the infrequent or new user can retrieve the desired material.

6.11 Group Decision Support

A Group Decision Support System (GDSS) is an interactive computer-based system that helps facilitate group discussions in order to achieve a solution to a problem. The interactive computer based system removes communication barriers, structures the group interaction and provides analytical tools to assist in decision making. There are two basic types of Group Decision Support System: face-to-face or computer conference.

Computer conferences act as a computer-mediated communication (CMC) system assisting the group in communication. The computer conferences can be used synchronously or synchronously. Different software tools can assist in decision support

and control during the computer-mediated communication. If the decision tools are a part of a computerized conferencing system, then the system is called a Distributed Group Support System [12].

CHAPTER 7

PROTOTYPING

7.1 Introduction

Prototyping may be used to overcome problems with developing the software using the software lifecycle when particularly when the requirements are not clear. There are two basic prototypes: throwaway prototype and evolutionary prototype.

7.2 Prototype in Waterfall Life Cycle

Waterfall life cycle fails to effectively show iterations between phases. A working system becomes available late in the lifecycle, which means that problems may go undetected until the system is almost operational. Software requirements are not properly tested until a working system is available to demonstrate to the end users. Software prototyping may be used to overcome these problems in the development of large-scale software systems.

Users find it very difficult to visualize how the system will function by reading specifications document. They often cannot determine whether the specifications are complete and correct. The requirements analysis and specification errors are frequently not detected until system and acceptance test, or even until production. Of course, the longer the error goes undetected, the costlier it is to correct [20].

The best way to assure that the system will satisfy the user's requirements is to give the user hands-on use of the system. Prototyping is an effective way of providing this kind of an experience.

The advantage that prototyping brings to the requirements specification process is the capability of bridging the communications gap that exists between the system developer and the user because of their different backgrounds.

The prototype must be an actual working system with which one can experiment. It must be comparatively cheap to develop with respect to the total system cost, and it must be developed relatively quickly so that it may be evaluated early in the software lifecycle.

7.3 Rapid Prototyping

When the prototype is introduced, the conventional software lifecycle phases are revised. During the preliminary analysis and specification of user requirements a first attempt is made to analyze the user's needs and to specify a system to satisfy his requirements. During the design and implementation of a prototype - the prototype should emphasize the user interface at the expense of lower-level software that is not visible to the user, it should be developed by a small development team to minimize communication problems, a programming language should be used which will help in the rapid development of the prototype. Emphasis should be on reducing development time and not on the performance of the finished product. Different tools should be applied which would help the rapid development of prototypes. Each user may exercise

the prototype and evaluate how well it performs the tasks he requires. Based on user feedback, changes are made to the prototype. The iterative refinement and experimentation with the prototype continues until it reaches a stage where the benefits of further enhancements to the prototype are outweighed by the time and cost required for these modifications. All user feedback is analyzed and the requirements specification is revised.

Design and Implementation of Production System - the design, coding and testing of the production system proceeds by following the standard software lifecycle. Developing the prototype will provide insights on how the production system should be designed.

7.4 Evolutionary Prototyping

In this prototyping approach, the prototype evolves into the final system. A software development approach that encourages evolutionary prototyping is that of incremental development. The objective is to have a subset of the system working early which is then gradually built on.

The evolutionary prototyping provides a good psychological boost to the team, the incremental versions of the system can be used as prototypes to test certain parts of the system, the prototype can be used to obtain early feedback from users, some performance measurements can be taken to determine the system response to executing a given transaction.

There are many phases of the software lifecycle for the evolutionary prototyping approach. Requirement analysis and specification, architectural design - the system is structured into modules and module interfaces are defined. Incremental module construction - the detailed design of each module to be included in t he system increment is completed. Also coding and unit testing of those modules is done. Incremental system integration - modules to be included in the system increment are integrated and tested to form subsystems. Evolutionary construction/integration - the previous two phases of module construction and system integration are repeated for each system increment. In some cases, the requirements specification and the architectural design may need to be updated. System testing - the whole system or major subsystems are tested to determine conformance with the functional specification. Acceptance testing - performed by the user.

7.5 The Throwaway Prototype

This prototype assists in specifying user requirements, it does not however, reduce the need for a comprehensive analysis of user requirements. However it improves on the completeness and correctness of the specifications.

The evolutionary prototype results from using the incremental development approach and it is an early version of the production system. It improves the quality of the software. It needs to follow the software lifecycle stages. The throwaway prototype can be developed much less formally and therefore it can be developed much more rapidly.

CHAPTER 8

A COMPARISON OF THE MAJOR APPROACHES TO SOFTWARE SPECIFICATION AND DESIGN

8.1 Introduction

A specification is a representation of a proposed or existing computer system. It can serve as the basis for a contract between a developer and a customer to produce the proposed system. The specification is intended to describe all of the required properties of the system while leaving all other properties unconstrained. A specification is generally understood to be simpler, more comprehensible and easier to modify than the actual hardware and software used to implement the specified system.

There are several fundamentally different ways to approach the specification problem, as well as many opportunities for combining the approaches in various ways. There are three pure approaches and there are also combinations of them. The three approaches are: operational specification (execution semantics), mathematical specification (proof semantics) and natural-language specification (informal semantics).

Informal specifications can specify all required properties, impose no inherent bias, and require no special training to read or write. Formal specifications tend to be weak at specifying performance requirements and proofs are used to discover inconsistencies and to derive consequences of the specification. The question of whether an implementation satisfies the specification reduces to the question of whether an assertion is provable in some mathematical system.

An operational specification language has semantics defined in terms of an execution model. It is checked for consistency, and validated by static analysis based on the execution model or by execution. The question of whether an implementation satisfies the specification reduces to the question of whether you can tell the implementation and the specification apart by testing them.

Mathematical specifications present less implementation bias than operational specifications. Also the certainty provided by a proof of a mathematical specification is superior to the confidence provided by testing an operational specification. But, operational specifications are believed to be easier to use than mathematical specifications and provide an easier and more certain path to an implementation.

Informal specifications are often incomprehensible because of their size, ambiguity, incompleteness, and lack of structure. It is also extremely difficult to teach how to write a good specification in English or to evaluate the result.

Although it would be expensive to start using the formal methods, it might be even more expensive not to use them [8].

8.2 A Comparison of Techniques for the Specifications

During the requirements specification phase of the software development life cycle, it is necessary to describe in detail the expected behavior of the system to be built. This behavior is recorded in a document commonly called the Software Requirements Specification (SRS). Most SRSs are written in natural language. However, natural

language is inherently ambiguous, resulting in documents that are ambiguous, inconsistent and incomplete.

Software Engineering is the application of scientific principles to: the orderly transformation of a problem into a working software solution, and the subsequent maintenance of that software through the end of its useful life. Engineered approach usually is a phased approach. Waterfall model was used first to characterize the series of software engineering phases.

In order to reduce the inconsistencies of the natural language, it is best to use formal language, whenever it cannot be afforded to have the requirements misunderstood. There are many techniques for the behavioral requirements specification.

A finite state machine (FSM) is a hypothetical machine that can be in only one of a given number of states at any specific time. In response to an input, the machine generates an output and changes state. There are two notations commonly used to define FSMs: State Transition Diagrams (STD) and State Transition Matrices (STM).

In STD, a circle denotes a state, a directed arc connecting two states denotes the potential to transition between the two indicated states, and the label on the arc denotes the input that triggers the transition and the out with which the system responds. In an STM, a table is drawn with all the possible states labeling the rows and all the possible stimuli labeling the columns.

Decision tables and decision trees are other techniques used for the requirements specification. To construct a decision table, first draw a row for each condition that

will be used in the process of making a decision. Next draw a column for every possible combination of outcomes of those conditions. Fill in the boxes to reflect which actions you want performed for each combination of conditions. A decision tree is graphical rather than tabular. It is a flow chart without loops and without arrows pointing to the same node.

Program Design Language (PDL) is a standard for specifying detailed designs for software modules. PDL, also called structured English and pseudocode, is simply free-form English with special meanings for certain key words. Many people who see PDS in SRSs claim that the requirements writers have overstepped their bounds and fallen into design.

Two extensions of Structured Analysis (SA) were recently proposed by Hatley and Ward. These extensions have been termed Structured Analysis/Real-Time. Those extensions add control diagrams and control specs to their data counterparts. So data flow diagrams and control flow diagrams are added.

Statecharts are extensions to Finite State Machines (FSM), and were proposed by Harel. They make it easier to model complex real-time system behavior without ambiguity. The extensions provide a notation and set of conventions that facilitate the hierarchical decomposition of FSMs and a mechanism for communication between concurrent FSMs. One of those extensions is the superstate. The superstate can be used to aggregate sets of states with common transition.

Requirements Engineering Validation System is a set of tools that analyzes requirements written in the Requirements Statement Language (RSL) developed using

the Software Requirements Engineering Methodology (SREM). The tools, language and methodology were developed by TRW, Inc., for the U.S. Army Ballistic Missile Defense Advanced Technology Center. RSL and its corresponding graphical notation, R-nets represent an extension to conventional FSMs. R-net is a column of the state transition matrix, it is simply an organizational piece of a full FSM.

The Process-oriented, and Interpretable Specification Language (PAISLey) was developed by Pamela Zave. PAISLey is a language for the requirements specification of embedded systems using an operational approach. It is a simple language, with rigor and formality adopted from the disciplines of asynchronous processes and functional programming. When using PAISLey, the requirements write decomposes both the system under specification and its environment into sets of asynchronous interacting processes. Then each process is defined, and the range of possible states which the process can enter is defined.

Petri-nets were first introduced in 1962. Petri-nets are abstract virtual machines with a very well-defined behavior. They are used to specify process synchrony during the design phase of time-critical applications. They are represented as a graph composed of two types of nodes: circles called places and lines called transitions. Arrows interconnect places and transitions. Black dots (called tokens) move from place to place according to the rule, that tokens may pass through a transition only when a clock pulse has arrived, and all the arrows entering that transition are emanating from places that contain tokens. Petri-nets are best used to describe pieces of intended system

behavior where ambiguity cannot be tolerated and precise process synchrony is important.

8.3 Comparison of Techniques

There are several criteria to evaluate requirements specification techniques. Requirements specifications have to be understandable to computer-naive personnel - the ability of a computer-naive customer to understand the technique. There has to be basis for design and test- a technique to make an SRS more useful to designers, and systems testers, its ambiguity level must be lowered and its understandability to the computer-oriented people who design and test must be increased - the resulting SRS should be able to serve effectively as the basis for design and testing. Automated checking has to be used - checking for protocol violations, ambiguity, incompleteness and inconsistency. Also external view, not internal view has to be applied - the technique needs to allow the writer to remain at the requirements level and not proceed into design. Examples of others are SRS Organizational assistance - the technique should help organize the information in the SRS, automatic prototype and test generation - the technique should provide a basis for automated prototype generation and system test generation as well as appropriate applications - the technique should be suitable to the particular application.

8.4 A Comparison of Object-Oriented and Structured Development Methods

Structured techniques are based on a functional view of the system, with the system being partitioned according to its functional aspects. Recently, object-oriented approach for system development has been gaining popularity.

There is no universally agreed upon definition of what constitutes the OOD approach to system modeling. Also the differences between OOD and the structured methods have not been clearly defined. Some authors have suggested that there is a high degree of compatibility between at least some of the structured techniques and OOD. Others disagree with that notion, claiming that the differences in the modeling perspectives preclude any meaningful compatibility between the methods. Some advocates of OOD claim that it involves a more natural way to think than the functional approach. Some proponents of the structured techniques, on the other hand, insist that it is just as "natural" to think about functions as it is to think about objects.

8.5 Differences Between OOD and SD

OOD carries a unique, coherent theory of knowledge for system development. It describes a cognitive process for capturing, organizing and communicating the essential knowledge of the system's problem space, and gives guidance on using specific techniques to map this problem space model to a solution space model.

OOD has gone from being a partial to a full life cycle approach, which means that OOD is much broader than just a method. It promotes a theory of knowledge for

system development, which must give guidance on how to cope with complexity. OOD is an abstraction that makes different assumptions than Structured Development (SD) about what to illuminate and what to suppress, it gives a different perspective on what point of view best guides the thinking of the system modeled.

OOD as a development philosophy has its roots in object-oriented programming, and evolved bottom up, from programming to design to requirements analysis.

8.6 Comparing Development Paradigms

Structured analysis, structured design, and structured programming are collectively known as structured development (SD). The ideas behind these methods are found in the writings of Yourdon and Constantine, Dijkstra, DeMarco, Myers, and many others. All of the version of SD are based on a philosophy of system development that analyzes the system from a functional point of view.

According to Constantine, in the functional paradigm, function and procedure are primary, data are only secondary. Functions and related data are either conceived of as independent, or data are associated with or attached to the functional components.

In the object-oriented paradigm, data are considered primary and procedures are secondary; functions are associated with related data. Problems and applications are looked at as consisting of interrelated classes of real objects characterized by their common attributes, the rules they obey, and the functions or operations defined on them. Software systems consist of structured collections of abstract data structures embodying those object classes that model the interrelated objects of the real-world

problem. The SD techniques are generally associated with a top-down development strategy, whereas OOD is essentially a bottom-up approach.

8.7 Characterizing Object-Oriented Systems

An object is an entity defined by a set of common attributes and the services or operations associated with it. Objects are the major actors, agents and servers in the problem space of the system and can be identified by carefully analyzing this domain.

OOD is interested in how an object appears (an outside view), rather than what an object is (an inside view). The term encapsulation, means viewing the objects from outside and hiding the inside of an object.

Structured techniques build a system-structure model as a hierarchy of functions, which maps to a set of nested subroutines. The object model, is a nonhierarchical topology of objects. This topology forms an abstract view of the problem space, which is meant to map naturally to a nonhierarchic model of the solution space.

In the OOD model, processing takes place inside objects. While an object contains processing capability, it many need to interact with other object, to invoke other services, therefore communication between objects is needed. All communication is accomplished by message sending between the objects.

A framework of classing, subclassing and superclassing, allows individuals within a collection to share common attributes, as needed. This framework is collectively referred to as inheritance.

There are several variations of object-oriented methods, consequently, it is impossible to describe the object-oriented method or to be very specific in discussing exactly what the end product of OOD looks like. However, it is possible to discuss a general object-oriented concept that refers to a way of structuring software systems that is language independent.

Object-oriented software organization refers to a structuring approach that makes it possible to organize a system by object-oriented concepts and implement it as a set of object modules in conventional procedural languages as well as in object-oriented languages.

In order for a method to be object oriented, it has to contain at least three basic ideas: classification of data abstractions, inheritance of common attributes, and encapsulation of attributes, operations and services.

The proponents of OOD usually cite two reasons for their excitement about the approach. One is the claim that the thinking process inherent in OOD is more natural than that of SD, i.e., in building an abstract model of reality it is more natural to think in terms of objects than in terms of functions. The other is that the modeling of the problem space maps more directly to the solution space in OOD than it does in SD. Coad and Yourdon make this later claim, they say that with OOD the mapping is isomorphic, however, there are others who refute that.

Regarding the first notion, Coad and Yourdon state that the object-oriented analysis is based on concepts learned in kindergarten such as objects and attributes, classes and members, wholes and parts. OO approach is then a more natural way of

dealing with systems. Constantine, on the other hand believes that there are no object classes in the physical universe, and that objects no more natural than functions.

There is very little that you can say with much confidence about a most natural way that people think about the realities of their universe. The task ahead is to move the debate to a higher level - not arguing about which is more natural - but exploring how we best take advantage of both approaches.

8.8 The Operational Versus the Conventional Approach to Software Development

Various new ideas for developing software have been emerging besides the conventional software life cycle, but these ideas, such as executable specifications and program transformations, have no place in the conventional approach. They can, however, be organized into an alternative strategy called here the operational approach.

The Conventional Approach - during the requirements phase, a system to solve the problem is formulated and defined, no internal structure is specified. Requirements are almost always written in English, sometimes constrained by structure and supplemented by pictures, tables, formulas.

The design phase determines the internal structure of the software system, usually as a decomposition into modules of code. This is a top-down, hierarchical decomposition such that its modules will produce the required functions, be compatible with the hardware and software resources of the runtime environment, encapsulate information likely to change, meet the required performance constraints, and be implementable within the development environment.

The implementation phase turns the design into code executable in the runtime environment. This entails defining the internal mechanisms by which each module will meet its behavioral specification, and mapping module mechanisms and interface properties into the implementation language.

The Operational Approach - during the specification phase, a system is formulated to solve the problem and specify this system in terms of implementation-independent structures that generate the behavior of the specified system. The operational specification is executable by a suitable interpreter. Thus external behavior is implicit in the specification, while internal structure is explicit. The structures provided by an operational specification language are independent of specific resource configurations or resource allocation strategies. The structures of an operational specification language are independent of implementation-oriented decisions, and also the mechanisms are derived solely from the problem to be solved. They are chosen for modifiability and human comprehension without regard to any implementation characteristics.

During the transformation phase, the specification is subjected to transformations that preserve its external behavior, but alter the mechanisms by which that behavior is produced, so as to yield an implementation-oriented specification of the same system. The goal of all research efforts in this area is to automate the transformations themselves although selection of appropriate transformations will remain in human hands for the foreseeable future.

One type of transformation changes the modifiable and comprehensible mechanisms of the operational specification to equivalent ones lying at different points in the trade-off space balancing performance and the various implementation resources involved.

Other transformations are needed to that specification structures can be mapped straightforwardly and efficiently onto a particular configuration of implementation resources. These transformations may introduce explicit representations of implementation resources, or resource allocation mechanisms, that were not present in the original specification.

During the realization phase, the transformed specification is mapped into the implementation language. The goal of all research efforts in this area is to deal with the challenging problems during specification or transformation so that the realization step is a straightforward one.

Realization may entail making resource-allocation decisions not directly expressible in the specification language. The realization creates a virtual machine on which the transformed specification can run. More routinely, structures of the specification language must be mapped into structures of the implementation language.

Differences - the conventional approach places great emphasis on separating requirements (external behavior) from internal structure, but in the operational approach these are freely interleaved to get the operational specification. This interleaving is necessary to arrive at an executable specification, but it is also an

expression of the inevitable coupling between what is being done and how it is being accomplished.

The operational approach separates the problem-oriented structure of the operational specification from implementation considerations. In the conventional approach the design phase provides the high-level mechanisms that will produce the required behavior, but those structures must also fit the implementation environment and meet the performance constraints. Thus designers must consider both problem-oriented and implementation-oriented issues together.

The conventional approach separates high-level mechanisms, which are determined during design, from low-level mechanisms, which are determined during implementation. each mechanism is tailored for performance and resource consumption at the same time it is chosen to carry out a necessary function. In the operational approach all functional mechanisms have been chosen by the time the operational specification is complete, and optimizations of all types of mechanisms may be interleaved during the transformation phase.

Finally, the operational approach separates mechanisms from their realization in terms of the implementation language, while in the conventional approach intramodule mechanisms are interleaved with implementation-language decisions.

A misconception is that an operational specification is no different from a program in a very high-level language (VHLL). A VHLL is a declarative language in which problems within a well-defined domain can be posed. Such a program is an input to an application generator which then generates a system to solve the particular

problem for a fixed runtime environment. The operational approach is more widely applicable than the VHLL strategy because it requires neither a narrow domain nor a fixed runtime environment.

Comparison of the two approaches - Validation. The conventional approach has the advantage that English requirements can be read and approved directly by customers. On the other hand, informal requirements are notorious for their incompleteness. Operational specifications are formal, and therefore seem to have the opposite characteristics: machine-processable but inaccessible to end users and other non-technical people. Once a formal specification has been obtained, however, it is easy to summarize its properties informally in words or pictures.

Recently the concept of prototyping has added a whole new dimension to the possibilities for user participation in system development. Prototyping is possible under both paradigms, but in substantially different forms. In the operational approach the specification itself can be used as a prototype, since it is executable. This type of prototype can be produced rapidly and will be produced as an integral part of the ordinary development cycle.

In the conventional approach a prototype is produced by iterating the entire development cycle. This can result in a field-worthy prototype, but the conventional approach gives no particular guidance as to how to produce a prototype more rapidly than a product.

Verification. Another major problem is to ensure that the delivered system is faithful to its specification. In the conventional approach, this can only be done through

testing and/or formal proofs of correctness. However program proving can only establish consistency between the design specification and the implementation, since the requirements are not formal. Testing on another hand is an arduous process, however, and can never prove the absence of errors.

The philosophy of transformational implementation seeks to avoid either testing or verification by deriving the implementation from the specification using only transformations and mappings that have themselves been proven correct, i.e., proven to preserve behavioral equivalence.

Automation - the conventional approach has successfully resisted automation, because system representations tend to be informal and because each phase includes decisions about the mechanisms that will generate the required behavior - the automation of which can only be attempted through the techniques of artificial intelligence.

In the operational approach, the specification phase is labor-intensive, but its output is a formal object. The transformation and realization phases of the operational approach are ripe for automation because all of the behavioral requirements have been translated into computational mechanisms by the time the operational specification is written.

Management - The conventional approach is well-suited to managerial and organizational needs. the requirements specification defines the interface between users and developers; the design specification defines the interface among the work of many programmers. By producing an executable version of the system early in development,

the operational approach offers a milestone which is both psychologically satisfying and subject to meaningful evaluation.

8.9 Weaknesses of the Conventional Approach

The conventional approach stresses that all behavioral decisions should be made before any structural ones. This is an unrealistic and even undesirable expectation, since internal structure inevitably affects such external properties as feasibility, capacity, behavior under stress, and interleaving of independent events.

Another serious problem with the conventional approach is its reliance on a strategy of top-down decomposition for design. Basic methodological principles tell us that implicit decisions should be avoided, that if error-prone decisions must be made early then they should be subjected to early checks , and that individual decisions should be as orthogonal to others as possible. top down design leads to decomposition decisions most of whose consequences are implicit, makes the most global decisions earliest yet cannot validate them until the very end, and causes the top-level decisions to affect all properties of the system. It seems that top-down hierarchical decomposition is an excellent way to explain something that is already understood, but a poor way to acquire understanding.

In the operational approach the primary decomposition of complexity is based on problem-oriented vs. implementation-oriented structure rather than hierarchical decomposition. Even within an operational specification, the most prominent structures tend to be discovered by methods other than top-down decomposition. Although it is

true that hierarchical abstraction is often used within an operational specification to defer details, these details must be resolved before the specification phase comes to an end.

8.10 Weaknesses of the Operational Approach

One apparent weakness of the operational approach is the need to reduce external behaviors to internal mechanisms before specifying them. Therefore, the operational specifications are overconstraining and premature. Another potential problem with operational specifications is that they may run too slowly for the kind of testing and demonstration we would like.

The other major weakness of the operational approach is that transformational implementation is a relatively untried approach, and the necessary theoretical supports are only beginning to be developed. A final problem concerns current plans to have human users choose the transformations to be applied. It is not clear that, after several transformations, the specification will still be comprehensible enough to allow human intervention. Only further research and experience will determine whether or not this is a serious problem.

8.11 A Strategy for Comparing Alternative Software Development Life Cycle Models

There are many alternative models of software development such as prototyping, software synthesis, reusable software. Those models of software development differ in terminology, and therefore it is difficult to compare those models. There are different

guidelines to comparing the alternate life cycle models, so that the most appropriate life cycle model could be chosen, the impacts of the life cycle known upfront.

The waterfall model, or its variations are followed in most of he commercial corporations. The requirements stages are called user needs analysis, system analysis or specifications. The preliminary design stage is called high-level design, top-level design, software architectural definition or specifications. The detailed design is called program design, module design, lower-level-design, algorithmic design, etc. For the most part all these methodologies are equivalent.

During the past five to ten years, radically different methodologies have appeared, including rapid throwaway prototypes, incremental development, evolutionary prototypes, reusable software and automated software synthesis.

The rapid throwaway prototyping is to construct a "quick and dirty" partial implementation of the system prior to the requirements stage. The feedback from the users is used to modify the software requirements specification to reflect the user needs, before the development starts.

The incremental development is the process of constructing a partial implementation of a total system and slowly adding increased functionality or performance. This approach reduces the costs incurred before an initial capability is achieved, and also produces an operational system more quickly.

The evolutionary prototyping is a process of constructing by the developers a partial implementation of the system based on the known requirements. Evolutionary

prototyping implies that not all requirements are known at the time, but there is a need to experiment with an operational system in order to learn them.

In summary prototyping reduces development costs through partial implementations. Reusable software reduces development costs by using already developed and proven design and code in new software products. This technique reduces the development time and creates more reliable software. Automated software synthesis is transformation of requirements into operational code. This process is guided by algorithmic or knowledgebased techniques.

There is a paradigm which can be used to compare and contrast each of the above alternative life cycle models. This paradigm applies five measures with which to compare the life cycle models: shortfall - which measures how far the operational system at any time t is from meeting the actual requirements at time t, lateness - measures the time which elapses between the appearance of a new requirement and its satisfaction, adaptability - the rate at which the software solution can adapt to new requirements, longevity - the time a system solution is adaptable to change and remains viable - the time from system creation through the time it is replaced, inappropriateness - the gap between the user needs and the solution.

The Rapid Throwaway Prototypes increase the likelihood that customers and developers will have a better understanding of the real user needs that existed at time to. It increases the functionality provided by the system upon deployment. The length of time during which the product can be efficiently enhanced without replacement is the same as with the conventionally developed products.

The Incremental Development - is built to satisfy fewer requirements initially but is constructed in such a way as to facilitate the incorporation of new requirements. The initial development time is reduced because of the reduced level of functionality and the software can be enhanced more easily and for a longer period of time. The initial development time is less than for the conventional approach, the initial functionality is less than for the conventional approach and there is increased adaptability.

The Evolutionary Prototypes - is an extension of the incremental development. The number and frequency of operational prototypes increases. A solution is evolved in a more continuous fashion instead of by a discrete number of system builds. The evolutionary prototype is far more adaptable than the conventional approach, and it is much more functional.

The Reusable Software - is based on reusing of existing software components, which increases the initial development time for software. The development time is much shorter over the conventional approach.

The Automated Software Synthesis - the requirements are specified in some type of Very High Level Language (VHLL) and the system is automatically synthesized. The development time is greatly reduced, the development costs are reduced, so that it is more advantageous to resynthesize the entire system rather than adapt old systems. The longevity of any version is low.

All five approaches decrease shortfall, lateness and inappropriateness to varying degrees, the area between the user needs and actual system functionality when compared to conventional development is reduced.

CHAPTER 9

VERIFICATION AND VALIDATION

9.1 Introduction

During the requirements phase of critical software development, there should be various product reviews applied together. Walkthroughs can be used to achieve a consensus understanding of key requirements, a technical review can be held to examine requirements for completeness and correctness, a software inspection of the requirements specification can be used to verify and prepare for later stages of development such as design, test, etc.

9.2 Product Reviews

During project planning, available review processes should be mapped to examination needs. A different process, such as walkthroughs, technical review or software inspection might be applied to review test plans than to review the architecture.

The walkthrough is a software engineering review process in which a designer leads members of the development team through a segment of design or code that was written, while other members ask questions and make comments about technique, style, possible errors, and other problems.

The inspection is a formal evaluation technique in which software requirements, design, or code are examined in detail by a group other than the originator to detect

faults, violation of development standards and other problems. A review is a formal meeting at which a product or document is presented to the user for comment and approval.

In the multiprocess examination approach, the walkthroughs process can be used to meet the need for peer approval of individual requirements. The minimum input to the walkthrough process includes: a statement of objectives for the walkthrough, the draft requirements specification document, the requirement to be examined, the standards that are in effect for the development of the software.

The output of the walkthrough should include: the software requirements examined, objectives that were handled during the walkthrough, deficiencies, omissions, contradictions and suggestions, recommendations made by the walkthrough team.

The need for conducting a technical review of software requirements is defined by project planning documents. The minimum input to the technical review process includes: a statement of objectives, the software requirements, related system specifications, customer requirements, plans, standards against which the requirements are to be examined.

The output of the technical review, are the reviewed software requirements, specific inputs to the review, a list of unresolved deficiencies in the requirements, a list of management issues, action item ownership and status, recommendations made by the review team.

The software requirements specification document inspection can be triggered by document availability, schedule compliance of completion of rework required by an earlier inspection. Inputs to the inspection include: the software requirements specification document to be inspected, the approved issue of system requirements or architecture, any applicable inspection checklists, any standards and guidelines against which the document is to be inspected, all necessary inspection reporting forms. Expected output includes a defect listing, summary and process characterization. The listing identifies the location, description and category of each defect found.

Applying a mix of examination processes has the potential of greatly improving product quality and project costs. Defects that are identified at the various evaluation meetings can be categorized by defect type, class and severity. Once the data collection, analysis and reporting program is underway, process decisions will be made based on the applicability and accuracy of data. The consistency of review process application is critical in warding off efficiency deterioration and allowing continuous improvement.

9.3 Verifying and Validating Software Requirements and Design Specifications

The recommendations included in this article, provide a good starting point for identifying and resolving software problems early in the life cycle - when they are still relatively easy to handle.

By investing more up-front effort in verifying and validating the software requirements and design specifications, projects are reaping the benefits of reduced

integration and test costs, higher software reliability and maintainability and more user-responsive software.

Verification is the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase. Validation is the process of evaluating software at the end of the software development process to ensure compliance with software requirements.

There are four basic V&V criteria for requirements and design specifications: completeness, consistency, feasibility and testability. Completeness - a specification is complete to the extent that all of its parts are present and each part is fully developed. There should not be any nonexistent references, no missing specification items, no missing functions, no missing products. Consistency - a specification is consistent to the extent that its provisions do not conflict with each other or with governing specifications and objectives. There has to be an internal consistency - items within the specification do not conflict with each other, external consistency - items in the specification do not conflict with external specifications or entities, traceability - items in the specification have clear antecedents in earlier specifications or statements of system objectives. Feasibility - a specification is feasible to the extent that the life-cycle benefits of the system specified exceed its life-cycle costs. It implies validating that the specified system will be sufficiently maintainable, reliable, and human-engineered to keep a positive life-cycle balance sheet. Human engineering is verifying and validating feasibility from a human engineering standpoint. It involves answering the following questions - will the specified system provide a satisfactory way for users to perform

their operational functions, will the system satisfy human needs at various levels, will the system help people fulfill their human potential. Resource engineering - this involves the following verification and validation questions: can a system be developed that satisfies the specified requirements, will the specified system cost-effectively accommodate expected growth in operational requirements over its life-cycle? Program engineering - addresses the following questions - will it be cost-effective to maintain, will it be cost-effective from a portability standpoint, will it have sufficient accuracy, reliability, and availability to cost-effectively satisfy operational needs over its life cycle.

Simple manual verification and validation techniques are reading, cross-referencing, interviews, checklists, and models. Reading involves having someone other than the originator read the specification to identify potential problems. Manual cross-referencing involves constructing cross-reference tables and various diagrams - for example state transition, data flow, control flow, and data structure diagrams - to clarify interactions among specified entities. Interviews - involve discussing a specification with its originator in order to identify a potential problem. Checklists are specialized lists, based on experience, of significant issues for assuring successful software development can be used effectively with any of the manual methods described above. Manual models - are mathematical formulas which can be used to represent and analyze certain aspects of the system being specified. Simple scenarios - describe how the system will work once it is in operation. Man-computer dialogues are the most common form of simple scenarios, which are very good for clarifying

misunderstandings or mismatches in the specification's human engineering aspects but not for checking completeness and consistency details or for validating performance speed and accuracy.

The automated techniques can be applied to two manual techniques - cross-referencing and simple modeling. Automated cross-referencing - involves the use of a machine-analyzable specification language - for example, SREM-RSL - Software Requirements Engineering Methodology-Requirements Statement Language, PSL/PSA - Problem Statement Language/Problem Statement Analyzer, or PDS - Program Design Language. Once a specification is expressed in such a language, it can be automatically analyzed for consistency, closure properties, or presentation of cross-reference information for manual analysis.

Simple automated models - mathematical formulas implemented in a small computer program provide more powerful representations than manual models for analyzing such life-cycle feasibility issues as accuracy, real-time performance and life-cycle costs. Simple automated models are especially good for risk and sensitivity analysis.

Detailed manual techniques and mathematical proofs are especially effective for clarifying human engineering needs and for verifying finite-mathematics programs, respectively.

Two final techniques - detailed automated models and prototypes - provide the most complete information. Detailed automated models involve large event simulations of the system. While more expensive than simple automated models, they are much

more effective in analyzing such issues as accuracy, dynamic consistency, real-time performance, and life-cycle cost.

Prototypes - the process of building the prototype will expose and eliminate a number of ambiguities, inconsistencies, blind spots, and misunderstandings incorporated in the specification. Prototypes can be expensive and do not shed much light on maintainability, but they are often the only way to resolve the critical feasibility issues.

Several available systems - PSL/PSA, SREM, PDS, Special/HDM provide automated aids to requirements and design verification and validation. The SREM Requirements Statement Language expresses software requirements in terms of processing paths - that is, the sequences of data processing required to operate on an input stimulus to the software and produce an output response. The SREM approach to attaining explicitness throughout a requirement specification is grounded in the use of the Requirements Statement Language - RSL. RSL is a machine-processible, artificial language which overcomes the shortcomings of English in stating requirements. RSL is based on the entity-attribute-relationship model of representing information.

CHAPTER 10

BUSINESS PROCESS REENGINEERING

10.1 Introduction

Currently all of the methodologies begin the system development cycle with the business requirements phase. During this phase, analysts gather the requirements on the current functions, and gather the newly identified needs for the system that have to be developed. However, there is one, very significant stage that is missing and that is Business Process Reengineering.

10.2 Sound Basis for a New System

Regardless of how well the system is designed, documented and developed, if it does not fulfill the business needs, it will be perceived as a failure by the business community. The system will not help the user streamline his business operations. In order to steer the business correctly, managers need to define the information they need in a very precise way. Current operations can be monitored and compared with past operations. Predictions of future operations can be rationally made. New business processes can be devised, and only then new operational systems can be developed to support those new processes.

10.3 Business Process Reengineering

The Business Process Reengineering calls for a very careful look at the existing business processes. Business processes age with time and need to be periodically redesigned. Otherwise, the business owners will follow procedures which will not lead them in the right direction, or will not follow the most optimized path. Business processes can be redesigned using similar techniques to the ones used by Requirements Analysis. The tools to document business requirements will be very appropriate for the redesign of the business processes as well.

Business modeling techniques can be applied to define the structures and processes of the business environment, both internal and external to the enterprise, Informational objects can be created, their structures and processes defined. Functional modules can then be created to validate and transform the business processes.

Technology should be matched to business needs. The systems must match business strategy. It has to be then established what the relevant business objectives are at every level: corporate, business unit, process, function, department. Only that established direction can determine the system and technology strategy.

Business requirements have to be analyzed in the context of what process, organizational, staffing and other changes should be made, and only then determine what kind of demands do they place in the way of information needs and processing power.

10.4 Business Process Reengineering Specialist

There has to be a new support function created, called the Business Process Reengineering Specialist. This Specialist has to have detailed knowledge of the specific business area, have very strong analytical skills as well as knowledge and experience in applying analytical tools. It would be desirable for this specialist to have a combination of business savvy and IS skills, so that he could also function as an interface. He does not however, have to be familiar with the system aspect of development in any great detail. The Business Process Reengineering Specialist should be able to analyze horizontally across departmental functions as well as vertically, understanding the connections between top management's goals and line departments. The cultural and intellectual gap between the world of business and the structured information systems logic needs to be filled by this new function. The Business Process Reengineering Specialist is most likely to be found within the top ranks of application analysts and from a select group of business managers in the operating units.

After compiling the documentation, the Business Process Reengineering Specialist provides the Business Requirements Analyst with documentation and guidance for further stage in the system development life cycle. The Business Analyst can then develop a Business Requirements Document by expanding on the received information. The Business Requirements Analyst has to be very fluent in business aspects so that he could relate to the Business Process Reengineering Documentation as well as to derive from it the next stage - Business Requirements Stage.

CHAPTER 11

CONCLUSION

There have been many methodologies, methods and tools defined and invented to assist the development team in building sound and effective systems. Over the years, new stages get added to the beginning of the life cycle methodology. For example, not that long ago, the business requirements stage became more and more emphasized as a vital part of the system development. I think that now is the time to add yet another initial stage to the product development life cycle, which is the Business Process Reengineering Stage. This stage will make sure that the well designed system, has also well designed business processes to support.

REFERENCES

[1] Alfred Aue and Michael Breu, "Distributed Information Systems: An Advanced Methodology," *IEEE Transactions on Software Engineering*, vol. 20, pp. 594-605, August 1994.

[2] Rodney Bell, "Choosing Tools for Analysis and Design," *IEEE Software*, Vol. 11, no. 2, pp. 121-125, May 1994.

[3] Barry W. Boehm, "A Spiral Model of Software Development and Enhancement," *Software Engineering Project Management*, pp. 128-142, 1987.

[4] Debra Bulkeley, "Andersen Reengineers Big Business," *Systems Integration Business*, vol. 25, no. 8, pp. 22-24, August 1992.

[5] Peter Chen, "Entity-Relationship Approach to Data Modeling," EH0304-6/90/0000/0238 *IEEE System and Software Requirements Engineering*, pp. 238-243, 1990.

[6] Peter Coad, and Edward Yourdon, "Object Oriented Analysis," EH0304-6/90/0000/0272 *IEEE System and Software Requirements Engineering*, pp. 272-289, 1990.

[7] Alan M. Davis, and Pei Hsia, "Giving Voice to Requirements Engineering," *IEEE Software*, vol. 11, no. 2, pp. 12-15, March 1994.

[8] Alan M. Davis, "A Comparison of Techniques for the Specification of External System Behavior," *Communications of the ACM*, vol. 31, no. 9, pp. 1098-1115, Sept. 1988.

[9] Alan M. Davis, Edward H. Bersoff, and Edward R. Comer, "A Strategy for Comparing Alternative Software Development Life Cycle Models," EH0304-6/90/0000/0496 *IEEE, System and Software Requirements Engineering*, pp. 496-504, 1988.

[10] Mohamed E. Fayad, Wei-Tek Tsai, Mark A. Roberts, Louis J. Hawn, and Jay W. Schooley, "Adapting an Object-Oriented Development Method," *IEEE Software*, vol. 11, no. 3, pp. 68-76, May 1994.

[11] Hassan Gomaa, "The Impact of Prototyping on Software System Engineering," EH0304-6/90/0000/0543 *IEEE System and Software Requirements Engineering*, pp. 543-552, 1990.

[12] Starr Roxanne Hiltz, Kenneth Johnson, and Murray Turoff, "Group Decision Support: The Effects of Designated Human Leaders and Statistical Feedback in Computerized Conferences," *Research - Designated Leaders and Statistical Feedback*, pp. 81-106.

[13] Starr Roxanne Hiltz and Murray Turoff, "The Evolution of User Behavior in a Computerized Conferencing System," *Communications of the ACM*, vol 24, no. 11, pp. 739-751, November 1981.

[14] Yogesh H. Kamath, Ruth E. Smilan, and Jean G. Smith, "Reaping Benefits with Object-Oriented Technology," *AT&T Technical Journal*, vol. 72, no. 5, pp. 14-24, Sept/Oct 1993.

[15] Michael C. Kettelhut, "JAD Methodology and Group Dynamics Improving Group Decision Making," *Information Systems Management*, vol. 10, no. 1, pp. 46-53, Winter 1993.

[16] Francis Kretz and Francoise Colaitis, "Standardizing Hypermedia Information Objects," *IEEE Communications Magazine*, vol. 30, no. 5, pp. 60-70, May 1992.

[17] Wayne C. Lim, "Effects of Reuse on Quality, Productivity, and Economics," *IEEE Software*, vol. 11, no. 5, pp. 23-30, Sept. 1994.

[18] Michael Lloyd-Williams, "Knowledge-based CASE Tools: Improving Performance Using Domain-Specific Knowledge," *Software Engineering Journal*, vol. 9, no. 4, pp. 167-172, July 1994.

[19] Colin Potts, Kenji Takahashi, and Annie I. Anton, "Inquiry-Based Requirements Analysis," *IEEE Software*, vol. 11, no. 2, pp. 21-32, March 1994.

[20] Barry F. Rosenberg and Robert M. Zimmerman, "Accelerated Application Engineering - A Cost-Effective Development Approach," *Information Systems Management*, vol. 10, no. 1, pp. 7-14, Winter 1993.

[21] Hugh W. Ryan, "Pursuing an Engineering Discipline - Can an Engineering Approach Work ?," *Information Systems Management*, vol. 10, no. 1, pp. 62-64, Winter 1993.

[22] Jawed Siddiqi, "Challenging Universal Truths of Requirements Engineering," *IEEE Software*, vol. 11, no. 2, pp. 18-19, March 1984.

[23] Kime H. Smith, Jr., "Accessing Multimedia Network Services," *IEEE Communications Magazine*, vol. 30, no. 5, pp. 72-80, May 1992.

[24] Pamela Zave, "A Comparison of the Major Approaches to Software Specification and Design," EH0304-6/90/000/0197 *IEEE System and Software Requirements Engineering*, pp. 197-199, 1990.