DESIGN OF INFORMATION SYSTEMS
A SOFTWARE ENGINEERING APPROACH

A thesis submitted
in partial fulfilment of the requirements for the
award of the degree of

MASTER OF ENGINEERING
(Computer Science)

Submitted By:
GURPREET SINGH

Supervised By:
Dr. P.K. BANSAL & Dr. D.P. GOYAL

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA-147 004

2000

CERTIFICATE

I hereby certify that work which is being presented in this thesis entitled "Design of Information Systems - A Software Engineering Approach" in partial fulfilment of the requirements for the award of the degree of Master of Engineering in Computer science, and being submitted in the department of Computer Science & Engineering, Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Dr. P.K. Bansal and Dr. D.P. Goyal.

(Gurpreet Singh)

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

25/5/2000

(Dr. P.K. Bansal)
Professor
Dept of Computer Science & Engg.
Thapar Institute of Engg. & Technology
Patiala.

25/5/2000

(Dr. D.P. Goyal)
Reader
Punjab School of Management Studies
Punjabi University
Patiala.

26.5.2000

Head
Dept of Computer Science & Engg.
Thapar Institute of Engg. & Technology
Patiala.

26.5.2000

Dean Academics
Thapar Institute of Engg & Technology
Patiala.

ACKNOWLEDGEMENT

It is my proud privilege to express my profound gratitude and sincere thanks to Dr. P.K. Bansal, Professor, Department of Computer Science and Engineering Thapar Institute of Engineering & Technology, Patiala and Dr. D.P. Goyal, Reader, Punjab School of Management Studies, Punjabi University Patiala for their valuable guidance, invaluable and timely suggestions and consistent encouragement which has helped me in preparing this thesis report.

I also pay my sincere thanks to all my friends and colleagues who provided me many useful suggestions and all possible help.

Last but not the least, I am highly grateful to my Parents and my wife who have given me emotional support which helped me in making this report.

(Gurpreet Singh)

ACKNOWLEDGEMENT

it is my proud privilege to express my profound gratitude and sincere thanks to Dr. P.K. Bansal, Professor, Department of Computer Science and Engineering Thapar Institute of Engineering & Technology, Patiala and Dr. D.P. Goyal, Reader, Punjab School of Management Studies, Punjabi University Patiala for their valuable guidance, invaluable and timely suggestions and consistent encouragement which has helped me in preparing this thesis report.

I also pay my sincere thanks to all my friends and colleagues who provided me many useful suggestions and all possible help.

Last but not the least, I am highly grateful to my Parents and my wife who have given me emotional support which helped me in making this report.

(Gurpreet Singh)

ABSTRACT

In most of the organizations in India, information system are developed using an unstructured approach, which leads to faulty and ineffective systems. Many systems fail due to the reason that they are not developed using any proper methodology. Moreover Information system development is expensive and is often unreliable.

The goal of software engineering is to face this software crisis. Software Engineering is a discipline that aims at providing methodologies or engineering approaches for development of information system. It is a combination of two things, software and engineering, by combining these two things Software Engineering can be defined as the systematic approach to the development, operation & maintenance of software.

Analysis of various studies reveals that the researches mainly concerns with reliability and testing of software using different techniques and are silent about the techniques used for overall development of information system. Hence a survey of relevant studies establishes a need for applying the software engineering approach for information system development.

In this study emphasis is given on applying software engineering approach for overall development of information system. Various aspects of software engineering like complexity, correctness, maintability and reusability have been introduced and information system of University has been designed by applying various software engineering approaches. An analysis of

different system development process models that are used to guide the analysis, design, development and maintenance of information system have been undertaken in this work, and a new model for the development of system has been proposed.

On the basis of the above study it has been recommended that the software engineering approach should be applied while developing information system for the University.

CONTENTS

CHAPTER 1

PROBLEM DEFINITION

INTRODUCTION

In this chapter software engineering approach alongwith scope & methodology of the system to be developed has been discussed. A brief information about the organization under study has also been given.

1.1 SOFTWARE ENGINEERING: AN INTRODUCTION

In most of the organisations including India, information system are developed by using an unstructured approach, which leads to faulty and ineffective systems. Many systems fail as they are developed without following any proper methodology.

Software Engineering approach to system development is a well organised methodology which is combination of two things:

Software

Software is not just a set of computer programs but comprises programs, data and documentation necessary for operation & maintenance.

Engineering

It is the application of science for the control and use of power, especially by means of machines. In other words it is said that engineering is about the systematic application of scientific knowledge in creating and building cost-effective solutions to practical problems.

Software Engineering

If we combine the above two sets of definitions we can come up with a working definition for software engineering

"Software engineering is the systematic approach to the development, operation and maintenance of software"

Another definition of software engineering given by Boehm is that it is the application of science and mathematics by which the capabilities of

computer equipment are made useful to man via computer programs, procedures and associated documentation.

The use of terms "systematic approach" or "mathematics and science" for development of software means that software engineering is to provide methodologies for developing information system that are closer to the scientific method as possible. The basic goal of software engineering is to produce high quality software at low cost.

The main phases of software development in software engineering are Requirement Analysis, System Design, Coding & Testing. Figure 1.1 portrays different phases of software engineering for information system development.

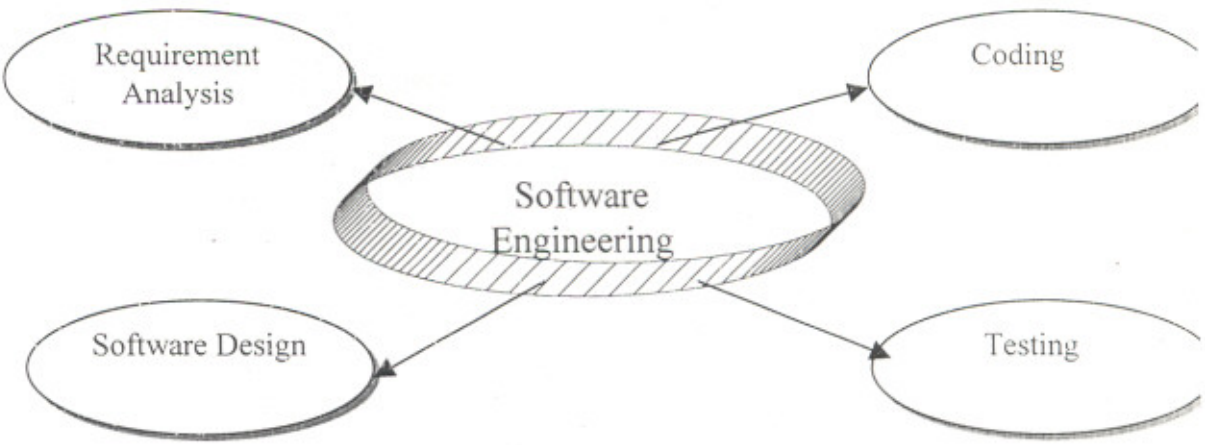


Figure 1.1 Different Phases of Software Engineering

Requirement Analysis

Requirement Analysis is the starting step for development activities. The main goal of requirement specification is to produce a document of the information system's requirement and forms the basis for further

development. Proper project planning which is an important ingredient for a successful project. Without proper planning a large information system development project is doomed.

System Design

The System Design phase plans for a solution such that if plan is implemented, the implemented system will satisfy the requirements of the system. The main emphasis of design phase is to find best possible design for the information system. The design activity is a two level process. The first level produces the system design which defines the components needed for the system, and how components interact with each other and the second level refines the system design, by providing more description of the processing logic of components and data structures, so that design can be easily implemented.

Coding

In the coding phase, the design of a system is translated into code that can be compiled and executed. Although the coding phase does not affect the structure of the system, it has great impact on the internal structure of modules, which affects the testability and understandability of the system. So the main goal of coding phase is to produce simple & clear programs. The aim is not to reduce the coding effort, but to program in a manner such that the testing and maintenance costs are reduced. Programs should not be constructed so that they are easy to write; they should be easy to read and understand.

Testing

Testing plays a critical role in quality assurance for information system. As design and requirement faults also appear in the code, testing is used for

detecting these errors, in addition to detecting the errors introduced during the coding phase. In testing phase system to be tested is actually executed and the behaviour of the system is observed. Due to this, testing observes the failures of the system, from which the presence of faults can be deduced and different levels of testing is applied to identify the faults.

Need of Software Engineering

Software Engineering is used keeping in view the various aspects of software like complexity of software; maintability of software i.e the software should be easily maintained; correctness i.e. software created should be correct and reusability i.e. the software should be easily reusable. So in software engineering main emphasis is on correct and good software. A good software has number of features like the software should be correct & easy to correct, software should be easy to test, easy to maintain and extend, easily portable and reusable and also main emphasis is on efficiency of software and user friendliness of software (for details see chapter 3). The above mentioned features of a good software can be easily achieved by adopting software engineering methodology for software development.

The above discussion clearly states the importance of the software engineering to be followed in developing information systems in general and critical information systems in particular. There is a need to develop information system following software engineering approach. A survey of literature (see chapter 2) reveals that there exist a gap in research in this important area. Keeping in mind the importance and research gaps, the following project was undertaken.

"Design of Information Systems- A Software Engineering Approach"

Examination system of the University has been selected for the project because of the size, complexity and sensitivity involved in developing such a system.

1.2 OBJECTIVES OF THE STUDY

The project has the following objectives :

- to study the existing examination system at Punjabi University Patiala.
- to design a module of the examination system following Software Engineering approach.

1.3 SCOPE OF THE STUDY

The project has been confined to the design of only one module of the entire examination system. However for the system to be developed at the University level, high integration of all the modules is required.

1.4 METHODOLOGY FOR THE SYSTEM

For the study of this project software engineering approach has been applied (see chapter 3).

1.4.1 Preliminary Steps to Mobilise the Work

The very first step was to study about the organisation and the broad examination system of the organisation. The next and immediate step was to take into confidence higher officers and briefly explain them about the advantages of computer concepts as in the beginning there were difference of opinion among the officers. However, once they were convinced, they agreed to provide the necessary information for the required work.

1.4.2 Information Gathering Methods

To study about the organisation and about the system many interviews were conducted with officers and staff and also material was collected from the

University handbooks, examination forms and Syllabus guidelines issued by the University. Figure 1.2 portrays information gathering method.

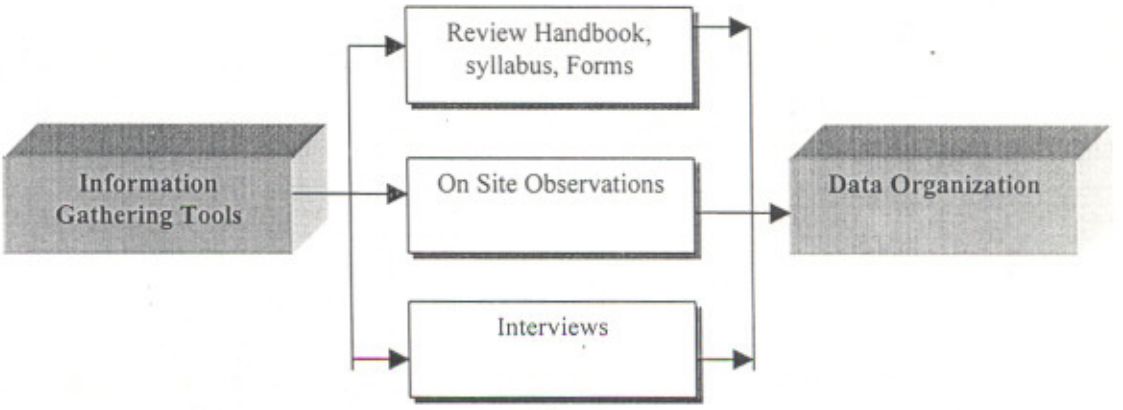


Figure 1.2 Information Gathering Methods

1.4.3 Design of Proposed System

On the basis of information gathered, design of one module has been proposed following software engineering approach.

1.5 THE ORGANISATION A PROFILE

The Punjabi University is located in city Patiala in the east of the Punjab. It was established on April 30, 1962. After the Hebrew University of Israel, this is the only University to be named after a language. The initial task before the University was to develop and promote the language of the Punjabi People and it was established as a residential and teaching University which mark the new trend in educational thinking after independence. Initially the jurisdictional area of the University was fixed as the 10-mile radius from the University office in Baradari Gardens Patiala. There were six postgraduate teaching departments two in the languages:

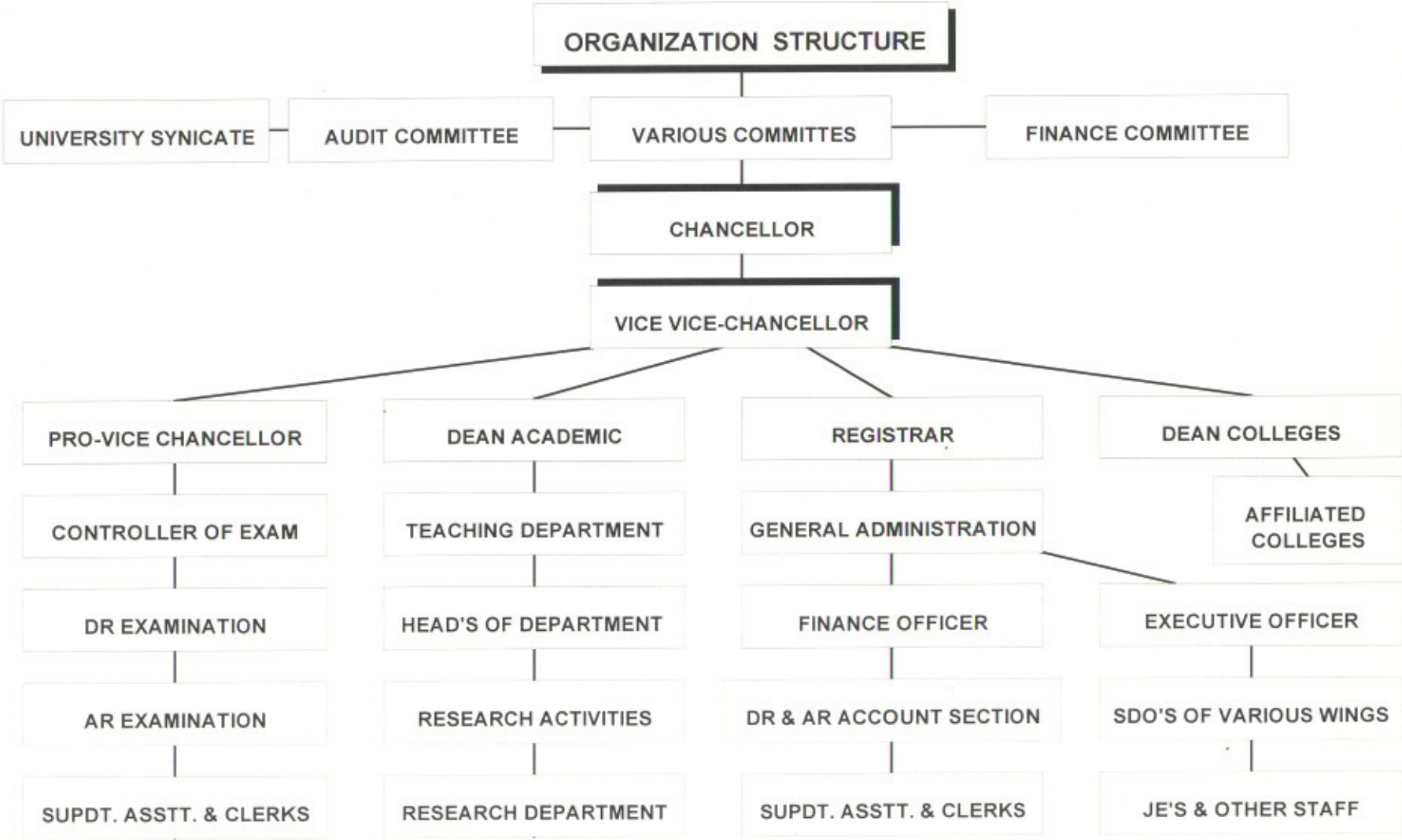


Figure 1.3 ORGANIZATIONAL STRUCTURE

Punjabi and English, one in social sciences: Economics, and three in sciences: Chemistry, Physics and Maths.

There were nine colleges, which falls in the jurisdiction of the University.

The University has a modern, well-planned campus located about 320 acres in pollution free environment. The University has also three regional centres namely The Guru Kanshi regional Centre at Bathinda, The Guru Kashi Campus at Talwandi Sabo and the Sher Mahommad Khan Institute of advanced studies in Urdu, Persian and Arabic at Malerkotla.

The Main University Library stocks more than 300,000 volumes and subscribes to several hundred journals. Library is kept open for 360 days of the year. The Computer Centre has established a local area network and all university departments enjoy the facilities of Internet. The University also has a large gymnasium and a hall for indoor games. A well-equipped health centre with qualified physicians and paramedical staff is at the service round the clock. The transport department of the University runs a fleet of buses from campus to Patiala. There are four residential Hostels with 800 rooms for boys and three residential Hostels for girls.

There are sixty-three colleges with over 2500 teachers affiliated with Punjabi University so University also caters for the needs of sixty-three Affiliated colleges in its jurisdictional area in Punjab.

University is running various graduate post-graduate & diploma courses at University Campus in different department and Regional Centres. The different departments in the university are Computer science & Engineering, Business Management, Pharmaceutical Sciences, Law, Physics, Chemistry, Botany, Zoology, Punjabi, Hindi, Sanskrit, English, History, Economics, Political Science, Education, Math, Correspondence

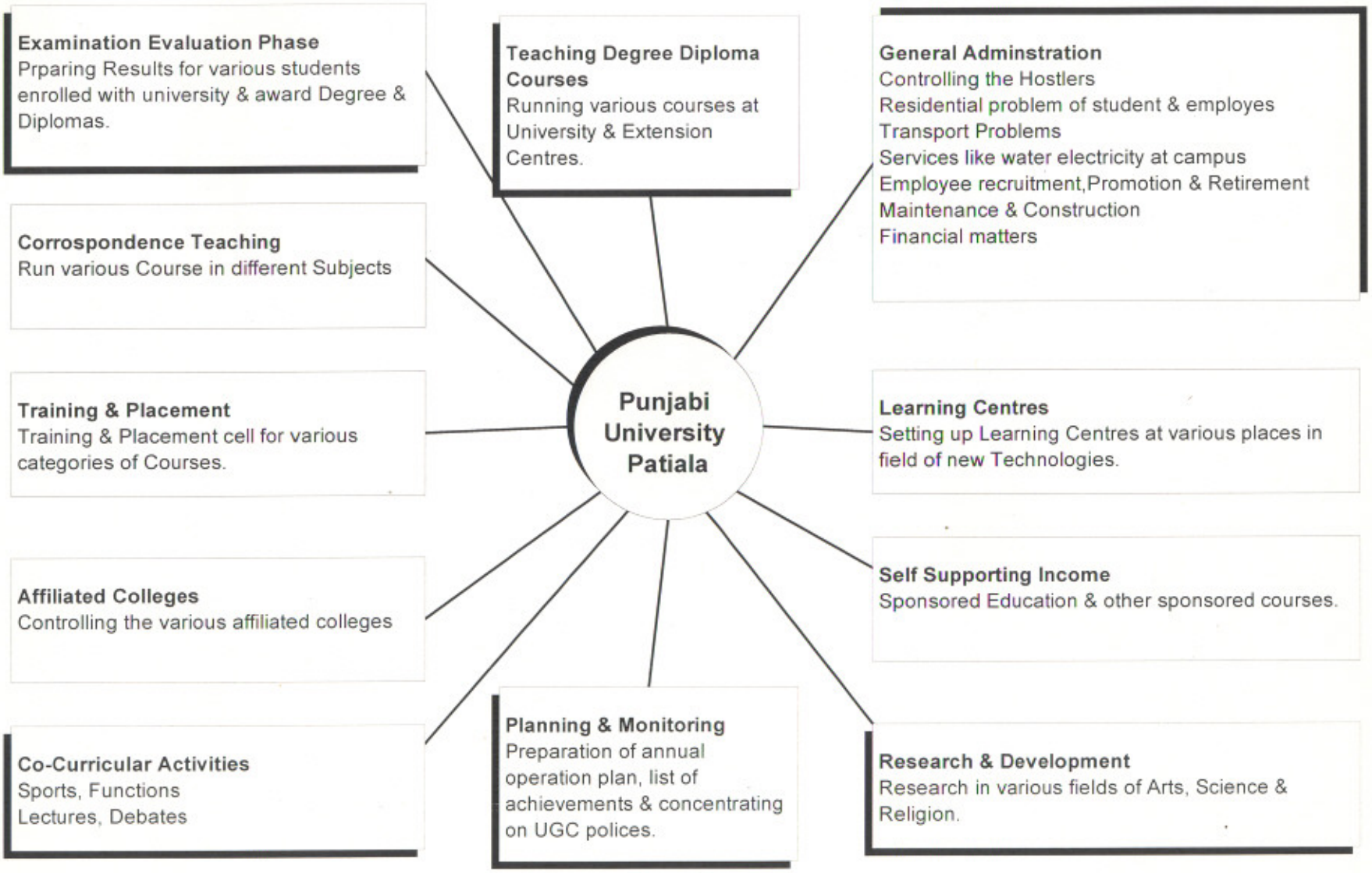


Figure 1.4 ACTIVITIES OF THE ORGANIZATION

Courses etc. Fig 1.3 portrays the organisation Structure and Fig 1.4 portrays main activities of the organisation.

1.6 COMPUTERISATION IN THE ORGANISATION

A local area Network has been established in the organisation and all the departments are connected through thicknet network backbone. The local area network is connected to Internet through VSAT connection by the ministry of information technology.

Various departments that are computerised are Salary Section in which salary is generated of all the employees every month, Budget Section which maintain the annual budget of organisation, PF section which maintain the provident fund and other savings of employees and some parts of library are also computerised.

1.7 ORGANIZATION OF THE THESIS

Chapter 1

In the first chapter brief description about the software engineering approach is given and also scope & methodology of the system to be developed is clearly defined. Finally information about the organization is given.

Chapter 2

Second chapter entitled "Literature Survey" explains the research carried out in the area of software engineering and identifies gaps therein.

Chapter 3

Third chapter entitled "Development Methodology" discusses the methodology used for information system development. Explanation about software engineering approach and reasons for following this approach are also explained.

Chapter 4

Chapter 4 explains about the solution strategy used in system design. Also various software development life cycles has been analysed and finally a new software model is proposed by combining some basic aspects of different software development models.

Chapter 5

Chapter 5 explains about the system design methodologies that are used in conjunction with new proposed model to finally develop the system.

Chapter 6

Finally implementation of system is done by using the discussed strategies. Starting from the analysis of the problem the system has been designed using software engineering approach.

At the end conclusion from the above study are given which includes the experiences gained during present study and finally concludes strategy of software engineering.

CHAPTER 2

LITERATURE SURVEY

INTRODUCTION

This chapter discusses some of the relevant studies conducted on the current status of the use of Software Engineering Tools for Software Development.

2.1 INFORMATION GATHERING & STUDY

For studying the actual work done in the field of software engineering the sources which were used are portrayed in figure 2.1.

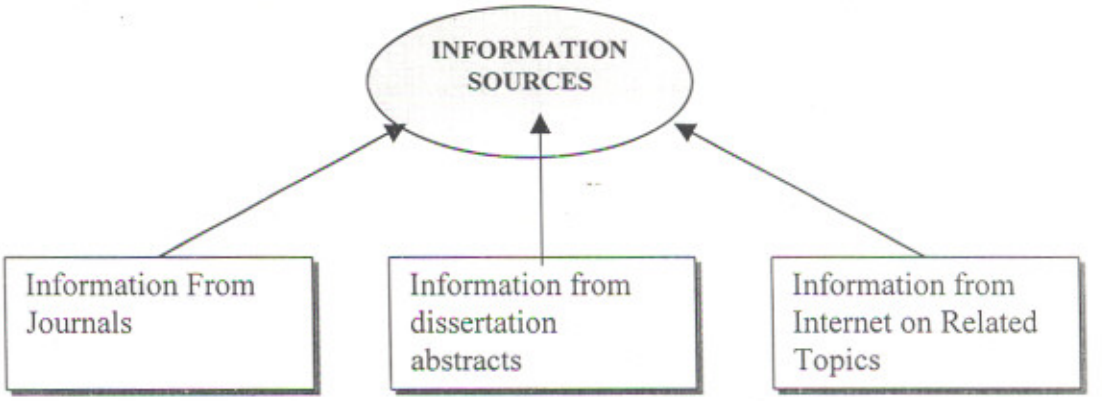


Figure 2.1 Information Sources

Gokhale, Swapna Sudhir in his thesis on Analysis of Software reliability and performance **[13]** stated that with the steadily growing power & reliability of hardware, software has been identified as a major stumbling block in achieving desired levels of system dependability. Conventional approaches for the assessment of software reliability suffer from several limitations and are based on various stringent and unrealistic assumptions. In this dissertation Gokhale emphasised on relaxation on some of stringent and unrealistic assumptions underlying the software reliability models, and discussed some of their limitations and proposed new approaches to overcome these limitations. The focus is mainly on three objectives:

It is an old programming adage that a relatively small number of modules in a software system contain disproportionately large number of errors. Software complexity metrics has been shown to be closely related to the number of faults in the software modules. A number of predictive relationships have been developed to classify the software modules into fault-prone and non-fault prone categories based on the software complexity metrics.

Author has proposed some Techniques to predict the actual number of errors in a software module based on the complexity metrics. He proposed the application of a regression tree modelling technique to predict the number of faults in a software complexity metrics, so that testing and validation efforts can be channelled in an effective direction. Data analysis using the regression tree model demonstrates that this technique enjoys better performance than the commonly used fault density approach.

The predictions obtained using the existing software reliability growth models are optimistic due to several reasons, namely, the saturation effect of testing, inaccuracies in the operational profile, and the assumption of instantaneous and perfect fault removal. He focused on the enhancement of software reliability growth models to incorporate code coverage and finite fault removal time which can help alleviate the problem of optimistic predictions. A methodology to compute the failure rate of the software based on the model with finite fault removal time is described. In addition, an economic model which accounts for finite fault removal has also been presented, which can well determine optimal software release criterion based on cost constraints.

Prevalent black-box based approaches to software reliability modelling are inappropriate to model the modern heterogeneous systems, where components having different failure behaviours and workloads interact. He outline the specification and solution methods for architecture based software reliability and performance prediction and present an exhaustive discussion of the Markov chain based approaches. He then presented a hybrid approach to architecture-based reliability prediction, where we parameterise the analytic model of an application using trace information obtained from the testing of the application. To facilitate this, he used ATAC(Automatic Test Analyser in C) which is a part of Software Understanding and Diagnosis System developed at Bellcore, which has been used to solved stochastic models of reliability, performance and performability.

Temporality in Object database management systems.

Goralwalla, Iqbal Abduirahim, from. University of Alberta (Canada) in his Ph.D. thesis, **[14]** describe the temporality in object database management systems as:

Conventional databases represent the state of an enterprise at one particular point in time. That is, they contain only current data. As a database changes, out-of-date information, representing past states of the enterprise, is discarded. However, temporal support is a requirement posed by many database applications, such as office information systems, engineering databases, and multimedia systems. Most of the research on modelling time has concentrated on the definition of a particular temporal model and its incorporation into a (relational or object-oriented) database management system. This research has led to the definition and design of a multitude of

temporal models. Many of these assume a specific set of temporal features and therefore do not incorporate sufficient functionality or extensibility to meet the varying temporal requirements of today's application. Instead, similar functionality is re-engineered every time a temporal model is created for a new application. This suggests a need for combining the diverse features of time under a single infrastructure that allows design reuse. In this thesis, an object-oriented framework that provides such a unified infrastructure is presented. An object-oriented approach allows one to capture the complex semantics of time by representing it as a basic entity. Furthermore, the typing and inheritance mechanisms of object-oriented systems directly enable the various notions of time to be reflected in a single framework. The framework can then be tailored to accommodate the temporal needs of different applications, and existing temporal models can be derived by making a series of design decisions through subclass specialisation. The framework can also be used to derive a series of more general temporal models that meet the needs of a growing number of emerging applications. Furthermore, the framework can be used to compare and analyse different temporal object models with respect to the design dimension. This helps identify the strengths and weaknesses of the different models according to the temporal features they support.

The framework is then used to instantiate a single temporal object model, which has multiple facets of time. There have been many temporal object model proposals {RS91,SC91,WD92,KS92, CITB92, BFG97}. These models differ in the functionality that they offer, however as in relational systems, they assume a set of fixed notions of time. The temporal object model developed in this thesis consists of an extensible set of primitive time

types with a rich set of behaviours to consistently and uniformly model the diverse features of time. The model is one possible implementation of the temporal framework and provides concrete and consistent semantics for the different temporal features, which is necessary for their coexistence.

The establishment of a temporal object model provides a foundation from which investigations are carried out on using temporality to model other database features such as schema evolution. The issues of schema evolution and temporal object models have traditionally been considered to be orthogonal and handled independently. This is unrealistic because to properly model applications that need incremental design and experimentation (such as CAD, softward design process) the evolutionary histories of the schema objects should be traceable. In this thesis, a method for managing schema changes by exploiting the functionality of the temporal object model is proposed. The result is a uniform treatment of schema evolution and temporal support for many object database management systems applications that require both.

Input validation testing: A System-level, early lifecycle technique.

A study on Input validation testing using a System-level, early lifecycle technique by Hayes, Jane Huffman, from George Mason University, 1999. **[15]** describe that

Syntax-directed software is an application that accepts inputs from the user, constructed and arranged properly, that control the flow of the application. Input validation testing is defined as techniques that choose test data that attempt to show the presence or absence of specific faults pertaining to input tolerance. A large amount of syntax-directed software currently exists and will continue to be developed that should be subjected to input

validation testing. System level testing techniques that currently address this are not well developed or formalised. There is a lack of system level testing formal research and accordingly a lack of formal, standard criteria, general-purpose techniques, and tools. Systems are expansive (size and domain) so unit testing techniques have had limited applicability. Input validation testing techniques have not been developed or automated to assist in static input syntax evaluation and test case generation. This dissertation seeks to address the problem of statically analysing input command syntax and then generating test cases for input validation testing, early in the life cycle. The IVT technique was developed and a proof-of-concept tool was implemented. Validation results show that the IVT method, as implemented in the MICASA tool, found more specification defects that senior testers, generated test cases with higher syntactic coverage that senior testers, generated test cases that took less time to execute, generated test cases that took less time to identify a defect than senior testers and found defects that went undetected by senior testers.

An assessment model to determine test process maturity.

Homyen, Ariya, in his study on Test process maturity **[16]** developed a Testing Maturity Model (TMM) which define the attributes of a mature testing process, assist organisations in understanding, evaluating and improving their software process, and contribute to the body of knowledge in the area of software process engineering. The TMM is a complement to the Capability Maturity Model (SW-CMM) and will be used alongside the CMM as a tool for overall process improvement. In this research an Assessment Model for the TMM has been developed. The research satisfies the needs of test process assessment using the TMM as a reference model,

and supports testing process evaluation and improvement programs in the software industry.

A set of test process assessment principles is defined which serves as the framework for development of the TMM-AM. Based on the principles, an assessment model that includes an assessment instrument, an assessment procedure, assessment team-related items, project selection criteria, a ranking algorithm, and a procedure for action planning, is defined and developed which is specifically applicable to the software testing process. A trial evaluation of the TMM assessment instrument has been carried out on 3 projects, and results look promising. Tools and templates have been developed to support further evaluation in industry.

Effort estimation model for component-based software development.

Smith, Randy Keith, **[22]** in his study on component based software development concluded that:

The modern approach to software application development involves moulding new work products and previously developed work products with commercial tools and Commercial-Off-The-Shelf (COTS) components. All of these ingredients are combined to meet and often extend user requirements. These many and varied ingredients require a software effort estimation model that captures cost factors at the component level. By examining cost factors at the component level, practitioners can take a fine-grained approach to isolating and capturing the primary parameters to develop an aggregation model for effort estimation.

During the construction phase of the software development lifecycle, it is possible to concurrently assign various components to different sets of programmers. A principal goal in making such assignments should be to

minimise the development effort for each component, which will ultimately minimise the cumulative development effort for the entire project. Cost estimation models predict development effort, but help little in making these scheduling decisions. Existing models, such as COCOMO, fail to consider component scheduling factors in their prediction of software development effort. This dissertation identifies five work assignment factors: *team size, concurrency, intensity, fragmentation, and component experience,* which capture the dynamic nature of contemporary software development.

These five factors are shown to improve the predictive ability of a known effort estimation model, COCOMO. When augmented with the work assignment factors, COCOMO estimates are improved by over 25%. A parsimonious effort estimation model is derived that utilises a subset of the work assignment factors and a size estimate. This parsimonious model is shown to have a predictive ability that is 75% more accurate than the COCOMO model yet utilises fewer cost factors.

Interactive query formulation techniques for databases.

Study on Interactive query formulation techniques for databases by Zhang, Guogen, **[25]** shows that:

Building a query tool for end-users to query a database bas been an important aspect of the development of database technologies. To advance the state of the art of this technology, a set of techniques for interactive database query formulation is developed in this dissertation and a prototype system is implemented. The techniques include the following: (1) High-level query formulation for simple queries; (2) Incremental query answering for complex queries; (3) Associative query answering for relevant

information; ;and (4) Semantic query guidance for incremental query sessions. This set of techniques is aimed to make formulation of ad hoc queries, including both navigational and aggregational queries, easier for end-users.

Our approach to query formulation is based on a semantic graph model, which is a semantic representation of the date in the database augmented with a probabilistic information measure. This graph model can be semi-automatically generated from the database schema. The high-level query formulator allows users to formulate their queries by specifying simple requests and constraints, without having to use the database query language. The formulator completes queries based on user input, ranks candidate queries according to probability, and uses English-like query descriptions for resolving ambiguity during the formulation process.

Incremental query answering allows users to obtain the necessary information by constructing a series of simple queries rather than being burdened with query syntax and structures for complex queries. Furthermore, associative query answering provides additional relevant information for queries; query guidance provides suggestions for subsequent queries at each step in an incremental query session. All the techniques provide users with useful assistance in finding their desired information. Case-based and probabilistic reasoning techniques are used for associative query answering and query guidance. The query formulation techniques can be incorporated with a GUI or with a natural language interface for voice input.

Architecture-based specification-time software evolution.

Medvidovic, Nenad, **[19]** in his thesis (from University of California) on Architecture-based specification-time software evolution.

Motivates, presents, and validates a methodology for software evolution at architecture specification-time. The methodology consists of a collection of techniques that, individually and in concert, support flexible, systematic evolution of software architectures in a manner that preserves the desired architectural relationships and properties. The methodology is comprehensive in its scope: it addresses the evolution of individual architectural building blocks-components and connectors-as well as entire architectures; it also supports the transfer of (evolved) architecture-level decisions into implemented systems. The unique aspects of the methodology are: component *evolution* via heterogeneous subtyping, well suited to a wide range of design and reuse circumstances; *connector evolution*, facilitated by evolvable interfaces and heterogeneous communication protocols, *architecture evolution*, facilitated by minimal component interdependencies and heterogeneous, flexible connectors; *analysis* of architectures for consistency, where the architect possesses the authority to override the analysis tool; off-the-shelf component and connector *reuse*, necessary for economic viability in large-scale software development; and *implementation generation*, aided by a well-bounded implementation space and accomplished via a component-based, evolvable environment.

The dissertation is validated empirically, by constructing a series of demonstration applications, and analytically, by evaluating the manner and degree to which the applications validate the claims of the dissertation. The dissertation is concluded by examining its impact on the tension between

flexibility and formality, which characterises current software architecture research.

Producing dependable object-oriented software using testing and design patterns.

A study on Producing dependable object-oriented software using testing and design patterns by Daniel's, Fonda Jonette **[12]** is concerned with two issues:

(1) Development of fault-tolerant design patterns, and

(2) Development of object-oriented testing techniques.

Designs that provide fault tolerance can be used to increase the reliability of object -oriented systems. Design patterns represent the common abstraction that experienced object-oriented designers can use to solve design problems in a way that can be understood and reused by other designers. A new design pattern called the Reliable Hybrid Pattern is presented, that provides a generalised framework that supports development of applications based on "classical" fault-tolerant software strategies such as N-Version Programming and Recovery Block, as well as advanced hybrid fault-tolerant techniques such as Acceptance Voting and Consensus Recovery Block.

Unfortunately, as practical object-oriented systems become more complex, the design and implementation constructs cannot usually be proven correct. When testing object-oriented software, a tester has to consider four levels of abstraction: (1) Routine or method level, (2) Class (intra-class) level, (3) Cluster (inter-class) level, and (4) System level. In this research the focus is inter-class and intra-class testing, which are forms of integration testing. A new inter-class test order strategy is presented along with several inter-class

properties. The approach develops an inter-class test order structure based on topological ordering of inheritance, aggregation and association relations. The classes are designed, integrated and tested according to the derived structure.

A new approach to intra-class testing is also developed that involves deriving method sequences from sequencing constraints derived from method pre and post-conditions. Method sequences are generated based on method sequencing coverage criteria developed as part of this work. The basic approach can be augmented through post-condition checking, or a test oracle or a combination of both. Code mutation and simulations were used to evaluate the effectiveness of this approach. Results show that generation of method sequences from sequencing constraints is an effective approach for assessing program correctness. Basic sequencing detected between 21-49 % of the seeded faults, while post-condition checking found 68-87 %. Addition of a test oracle further improves the effectiveness of the approach.

Modelling Software reliability during non-operational testing.

In his study on Modelling Software reliability during non-operational testing **[21]**. Rivers, Anthony Thyron discusses:

The problem of software testing in (real) production environments where *resources are constrained*, and the testing is often driven by "*business model*" considerations (such as cost and time to market minimisation) rather than software engineering models. In such environments the testing process is often analogous to a "*sampling without replacement*" of a finite (and sometimes very limited) number of pre-determined structures, functions, and environments. The principal goal is to verify required product functions to a market-acceptable level, while minimising re-execution of previously

tested functions. As a result, commercial testing is often a process that does not grow in efficiency, or grows very slowly over a number of releases.

To study this issue, this work presents a suite of software reliability engineering *model* specifically developed to study the above resource-constrained non-operational testing process. The models are based on the *hyper-geometric* sampling function, and are formulated and validated using information from both academic experiments and real industrial project data collected as part of current work. *Simulations* developed for this study show that "sampling without replacement" can be a very efficient way of testing provided test cases are error-sensitive. Modelling and analyses confirm the "*constant*" *testing efficiency* assumption for the available case-study data. It is conjectured that, in commercial situations, the "business model" may "prompt" this effect, and that this may be a principal factor that blocks an organisation from advancing on the Capability Maturity scale by discouraging within-phase feedback that might result in within-phase testing efficiency growth. However, *feed-forward* improvements in the testing efficiency can still take place in consecutive testing phases (e.g. from one release of the product to the next) even under such "business model" constraints. The data collected as part of current work confirm this. The same data also indicated that the field quality of a product does appear to be positively correlated with the increase in the average testing efficiency. This may be an *incentive* to adopt "business practices" that allows for within-phase "learning".

Requirements-based software reliability model.

Sova, Donald Willian, Jr. in his study on Software reliability model **[23]** discusses many issues with the current state of software reliability models.

Software reliability models that make the assumption that past failure history data is the best indicator of future behaviour still dominate the discipline. Because of this assumption, software reliability models treat the process and the product as a black box and therefore cannot explicitly be used for assessing the software process or product. Software reliability models that currently predict reliability early in the life cycle lack a sound statistical foundation. Finally, software reliability models should contribute to understanding and removing faults to improve reliability.

The thesis of this dissertation is that it is possible to address these issues using a "systems engineering" approach to software reliability assessment and prediction. A requirements based software reliability modelling approach provides the structure and organisation to integrate several engineering concepts into software reliability modelling. This model shifts the focus of software reliability modelling from modelling failure history to focusing upon the requirements. Reliability assessment is based upon validation of requirements, root cause analysis of their failure modes, and structural analysis of the functional architecture of the software. Failure modes and fault trees are used to develop a component-based reliability model. The component-based approach to reliability assessment allows for early assessment of reliability. Bayesian inference provides the statistical basis for quantifying the software reliability at the component level. A framework for the model is developed and illustrated through a simple experiment. The reliability data sources are explored and means of incorporating different sources of data examined. The dissertation demonstrates how the component nature of the model allows the model to

adjust its assessment of reliability based upon the changing requirements, without loss of significant reliability data.

The firewall concept for regression testing and impact analysis of object-oriented systems.

A study on the firewall concept for regression testing and impact analysis of object-oriented systems. Abdullah, Khalil Abdullah **[11]** concludes that: Software maintenance is concerned with providing a systematic process for dealing with change. When the process of software maintenance is applied to implement a change, we must ensure that unmodified functionalities are not inadvertently affected. When such functionalities are affected, this is identified as a *regression error*. Regression testing is a part of the maintenance process intended to detect regression errors.

In 1990, Leung and White introduced the firewall concept in order to deal with regression testing of a procedural system at the integration level. The firewall concept has been demonstrated to be a continuing viable technique for this purpose. Given one or more modules that have been changed, the firewall encloses the set of modules that must be resettled.

In establishing the firewall it was assumed that both units an integration testing involved were reliable. The effects of unreliable integration testing on the firewall for procedural systems are established, showing how the firewall construction must subsequently change. It is further shown how unreliable unit testing will lead to no firewall construction being possible.

An analysis is performed where the concept of a firewall is applied to object-oriented software with the same objective in mind. The analysis involves an investigation of the type of change of the component; specification change or code change. The effects of that change on the

component itself and its interaction with other parts of the system were thoroughly investigated. Then the parts of the system that need to be resettled using the firewall concept were identified. Finally, test suites are reused, or a new one designed when needed to reveal the defects for use with CTW. Results show that each checkpointing algorithm developed for CTW occupies a different point in the spectrum of possible trade-offs between memory usage and execution time.

We also present a dynamic load balancing algorithm developed for Clustered Time Warp which focuses on distributing the load of the simulation evenly among the processors and then tries to reduce inter-processor communications. We make use of a triggering technique based on the throughput of the simulation system. Performance results show that by dynamically balancing the load, the throughput of the simulation system could be improved by more than a 100 %. No substantial improvement was observed on the overall simulation time when trying to minimise inter-processor communications, suggesting that loan distribution is the most important factor to be taken into consideration in speeding up the simulation of digital circuits.

Furthermore, we examine the impact of partitioning and mapping on the performance and behaviour of the Clustered Time Warp algorithm. We show that partitioning algorithms, which try to minimise the number of cutsets between the partitions, do not necessarily succeed in minimising inter-processor communications. We also show that in our environment, load imbalance has a stronger effect than rollback overhead.

Finally, we study the problem of scalability encountered when using optimistic technique. We show that the performance of Time Warp can

greatly suffer from rollback explosions or when the *dog chasing its tail* phenomenon is observed. We also show that Clustered Time Warp is less sensitive to this phenomenon and as such, is more scaleable than Time Warp.

An empirical justification and methodology for reusable built-in test features in object-oriented software.

In his thesis Michael Edward from Misissippi State University **[20]** on an empirical justification and methodology for reusable built-in test features in object-oriented software represent that:

The objective of testing object-oriented systems does not differ from traditional or structured systems, i.e., one wishes to expose the maximum number of product defects with the minimal amount of effort. However, due to the inherent design differences in object-oriented systems, the techniques and strategies for testing will differ. One approach to testing object-oriented systems is to carry the object-oriented paradigm into the testing arena in the same manner in which the concept was adapted from hardware to software for designs. Although hardware designs utilising BIT (Built-In-Test) result in some performance challenges, the advantage has clearly been more effective and efficient testing. This research is concerned with the process of increasing the reusability of software by not only reusing the software component itself, but some of the testing effort associated with the component.

Three research questions are addressed in this work. First, BIT is empirically evaluated in terms of its effectiveness and efficiency in testing object-oriented programs. The results of a study of programmers are presented. Second, the potential for reuse of the testing effort is evaluated

through the investigation of potentially reoccurring designs within programs equipped with BIT methods. Lastly, BIT is applied to a larger scale application to determine the usefulness of the method in the context of class interactions. From the research, we suggest a methodology for incorporating BIT into a software development life cycle.

Our results suggest that BIT provide an effective and efficient means of testing classes. Additionally, the research presents some heuristic for applying BIT at the class and system level for testing object-oriented programs. Future work suggested by this research include the refinement of heuristics into software patterns in order to increase reusability of not just software components, but the testing of those components as well.

Impact of knowledge and data distribution on software performance and reliability

A study on Impact of knowledge and data distribution on software performance and reliability.Hilford, Victoria **[17]** describe that:

The generation of *vast amounts of data* raises a *performance* issue, how to deal with ever increasing amounts of data at ever faster processing speeds. The need for *dependable computing systems*, where dependability means trustworthiness such that reliance can justifiably be placed on the service they deliver, raises a software *reliability* issue for these ever larger and ever more complex systems.

To address the *performance* issue, we propose a novel distributed data structure EH, a variation of Extendible Hashing. It consists of buckets of data that are spread across multiple servers and autonomous clients that can access these buckets in parallel. EH is scaleable in the sense that it grows gracefully, one bucket at the time, to a large number of servers. The

communication overhead is relatively independent of the number of servers and clients in the system. The directory element from the classical Extendible Hashing algorithm is replaced by a Cache Tables data structure that is kept by both servers and clients. Lazy updates are used to bring obsolete Cache Tables up-to-date. We also introduced several versions of EH: a loan balanced version that deals with non-uniform distribution of the data, a fault-tolerant version that deals with server failures, and another version that allows data deletion.

The software development process is known to be fault-prone. The *complexity and size explosion* of software programs are important factors that affect the number of faults introduced during software development phases and the number of faults that remain undetected. We propose the *pipeline method* of developing software. By using diversity, the goad is to find as many faults in the very early stages of the development process before any testing is done, since finding faults during testing is more time consuming. It was evaluated on a real, automatic aeroplane-landing problem. Compared to existing methods, the pipeline method did not detect significantly more faults. Its advantage, however, lies in the fact that these faults were detected with significantly less effort since the process took place before any testing was done. We also conclude that automating the development process will decrease the number of faults introduced or left undetected by the software engineers.

Software vulnerability analysis.

Krsul, Ivan Victor, in his topic 'Software vulnerability analysis' **[18]** describe Software vulnerabilities as the consequences of a class of system

failures, violate security policies. They can cause the loss of information and reduce the value Or usefulness of the system.

An increased understanding of the nature of vulnerabilities, their manifestations, and the mechanisms that can be used to eliminate and prevent them can be achieved by the development of a unified definition of software vulnerabilities, the development of a framework for the creation of taxonomies for vulnerabilities, and the application of learning, visualisation, and statistical tools on a representative collection of software vulnerabilities.

This study provides a unifying definition of software vulnerability based on the notion that it is security policies that define what is allowable or desirable in a system. It also includes a framework for the development of classifications and taxonomies for software vulnerabilities.

This dissertation presents a classification of software vulnerabilities that focuses on the assumptions that programmers make regarding the environment in which their application will be executed and that frequently do not hold during the execution of the program.

This dissertation concluded by showing that the unifying definition of software vulnerability, the framework for the development of classifications and the applications of learning and visualisation tools can be used to improve security.

Techniques for software renovation.

Siff, Michael Benjamin, in his studies on Techniques for software renovation. **[24]** describe that:

Software renovation is the process of introducing new features- including polymorphism, objects and encapsulation - into existing software systems

while preserving the original functionality of the system. The goal of software renovation is to improve the efficiency of development, maintenance, and comprehension. The research described in this thesis focuses on three software-renovation techniques: (1) *Generalisation*. The identification and subsequent transformation of program components that operate on a particular type of input into polymorphich program components that operate on a wide array of inputs. (2) *Modularization*. The clustering of associated data types and functions with the intent of encapsulating the types and function into distinct classes or modules. (3) *Physical subtyping*. The identification of relationships among data types based on the representation of the types in memory with the intent of generating inheritance hierarchies.

The techniques described in the thesis are aimed particularly at the problem of transforming legacy C programs into C++ programs that make use of C++'s advanced features- most notably classes, templates, inheritance, and virtual functions. Some aspects of this work apply specifically to the C-to-C++ problem; however, most aspects apply to almost any language.

2.2 CONCLUSIONS

Analysis of various studies reveals that the research mainly concerns with reliability & testing of software using different techniques and are silent about the technique used for overall development. The main topics covered in the analysis are like analysis of reliability of software, statically analysing the input and for generating test cases for validation and testing processes, cost estimation for development of projects, Query formation in database, Defining effective architecture of software system, development of object oriented testing techniques, software testing using business model,

improving reliability using system engineering approach, how data distribution help in effective performance and reliability and analysis of software security etc.

From above it can be concluded that researchers have concentrated on checking reliability and effectiveness of software using different techniques but a little research has been done showing how a reliable cost effective & correct software can be developed using the complete software engineering techniques. What techniques or what steps exactly one should follow for development of software. Hence a survey of relevant studies establishes a need for applying the software engineering approach for information system development.

CHAPTER 3

DEVELOPMENT METHODOLOGY

INTRODUCTION

As their are many issues involved in the development of the software, we have to follow a particular good technique or steps to solve the existing problem. In this chapter important features of Software Engineering have been discussed to clearly understand the approach of Software Engineering.

3.1 DEVELOPMENT METHODOLOGY

Main points that are in our mind while problem solving are:

- What is the problem to be solved?
- What are characteristics of software that is used to solve the problem?
- How will the Software be realised?
- How will Software be constructed?
- What approach will be used to overcome the errors that were made in the design and construction of the Software?
- How will the Software be supported over the long term when user of system requests the corrections Adaptations & enhancements?

To follow the above steps we have to fix our development strategy and we have adopted the software Engineering strategy to solve the defined problem.

Here starting with the definition of Software Engineering we will also discuss why we use Software Engineering.

3.2 WHAT IS SOFTWARE ENGINEERING?

3.2.1 Software

Software are programs etc. for computer, or other interchangeable material for performing operations.

The critical point in the above definition is the 'programs etc'. Software includes the computer code, which is the actual implementation of the

program. However it also includes many other aspects; documentation (specification documents, design documents, testing and maintenance documents, user manuals, reference manuals, installation guides, keyboard overlays or templates), maintenance contracts, software licenses, software updates etc. All these elements are not physical components of the computer, but contribute to the development or use of the system. They are all included in the more broad definition of 'software'

3.2.2 Engineering

It is the application of science for the control and use of power, especially by means of machines.

"The use of Engineering method rather than the use of reason is mankind's most equitable divided endowment. By the engineering method, we mean the strategy for causing the best change in a poorly understood or uncertain situation with the available resources; by reason, I mean the 'ability to distinguish between the true and the false' or what Descartes has called 'good sense."

The above two definitions show that "engineering is about the systematic application of scientific knowledge in creating and building cost-effective solutions to practical problems"

3.2.3 Software Engineering

If we combine the above two sets of definitions we can come up with a working definition for software engineering

"Software engineering:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as described above in (1)."

In other words, software engineering is about the application of engineering methods to the development of software. Rather than simply creating software, we need to be able to design it. We need to be able to ensure that our design is appropriate. We need to understand the problem we are trying to solve. We need to be able to measure the appropriateness of our solution. The main phases of software development in software engineering are Requirement Analysis, Software Design, Coding & Testing.

3.3 WHY USE SOFTWARE ENGINEERING

Why is software engineering poorly understood?

One of the greatest problems that many people encounter when first learning software engineering is understanding why it is required. This problem arises from a number of areas.

Firstly, most people have not encountered (and are unlikely to encounter until much later) large complex software systems. The software they generally encounter is relatively simple and straightforward. In the same way that a circuit to make a light flash does not really require engineering skills to design, a program to find the first 100 prime numbers will not require software engineering. Nevertheless, as soon as the problems become non-trivial then engineering skills quickly become very valuable.

Secondly, software is deceptive. Single software instructions are relatively easy to grasp. Their use is typically very logical and straightforward. This makes writing software seem easy - so why bother having to go through a complex and laborious development process? The reason is that although single instructions are simple, the combinations of hundreds or thousands of instructions can be extremely complex. Software is NOT simple.

Thirdly, the concepts and tools used in software engineering are often difficult to grasp. This takes the focus away from solving the problem, and places it on using the tools - a situation, which is not at all appropriate. This can be overcome by either using simpler tools or improving the understanding of the tools available.

Finally, an understanding of the various aspects of software is typically missing or inadequate. How many students are aware that various estimates place software maintenance as representing up to eighty percent of the cost of software, and actual coding as low as three percent? The implications of this sort of information are usually not well understood.

Why do we need Software Engineering?

Having explained why students (or many other people) do not understand why we need software engineering, it should be explained why we do need it. It can be explained very simply. Computers are very complex machines (undoubtedly the most complex machines mankind has yet created). Part of these machines is the software. As engineers we are involved in designing machines to be used for specific applications. We use appropriate techniques - engineering methods - in this design.

Many people will recognise an inadequacy in this explanation; the phrase "appropriate techniques". Just what techniques are appropriate for software? In order to answer this question we need to consider various aspects of software. (Figure 3.1 portrays main aspects of Software) Some of the more salient issues are:

3.3.1 Complexity

As mentioned above software often appears deceptively simple. On a micro scale, individual instructions (or even blocks of instructions) can appear

quite simple - their operation is very logical and straightforward. On a macroscopic

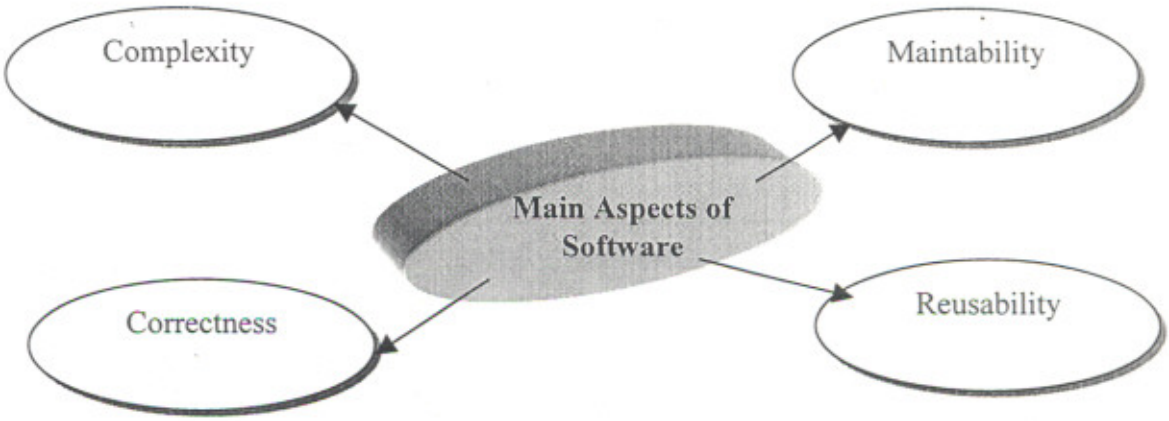


Figure 3.1 Main Aspects of Software

level, however, the software very rapidly becomes incredibly complex. We need to develop and use methods for coping with this complexity. Using an ad hoc approach and diving straight into coding without the use of any software engineering techniques will simply not work.

3.3.2 Correctness

Although writing single lines of code is simple, and even small programs are typically quite straightforward, this does not release us from the requirement of ensuring that our software solves the desired problem. We need to be able to assess that the software is correct. This is typically not a simple issue, and an assurance that "but I know that I wrote the code correctly!" is not sufficient. Everyone makes mistakes, and we need to have appropriate (and formal) methods of finding these.

3.3.3 Maintainability

As mentioned above. Estimates for the costs of maintaining software

(keeping it up to date, fixing errors as they are found, moving the software to newer computer systems etc.) vary from 60 to 80 percent of the total cost of software. It is important to use correct methodologies in the design of software in order to improve the ease with which software can be maintained. If writing code became twice as easy, we might save five percent of software costs. If maintaining code became twice as easy, we could save up to forty percent of the cost of producing software.

3.3.4 Reusability

The current situation with software is typically to redesign each piece of software from scratch. This is extremely wasteful. A more appropriate approach would allow us to write code in such a way that makes it easy to re-use (Lego software?) and also to recognise when re-using existing software is preferable.

3.4 ASPECTS OF GOOD SOFTWARE DEVELOPMENT

Correct software as distinct from good software. As we have gathered from the above discussion, there can be a big difference between 'correct' software and 'good' software. If you were given as a problem "Design a new type of electronic blackboard eraser", then an electric sander would be a correct solution, but not a particularly good solution. The issues that how to make software good, as opposed to correct place different (and often contradictory) requirements on software and will be discussed in detail with some further issues. It is a difficult process finding the optimal balance, and will always depend upon the intended use of the software. Figure 3.2 portrays features of good software.

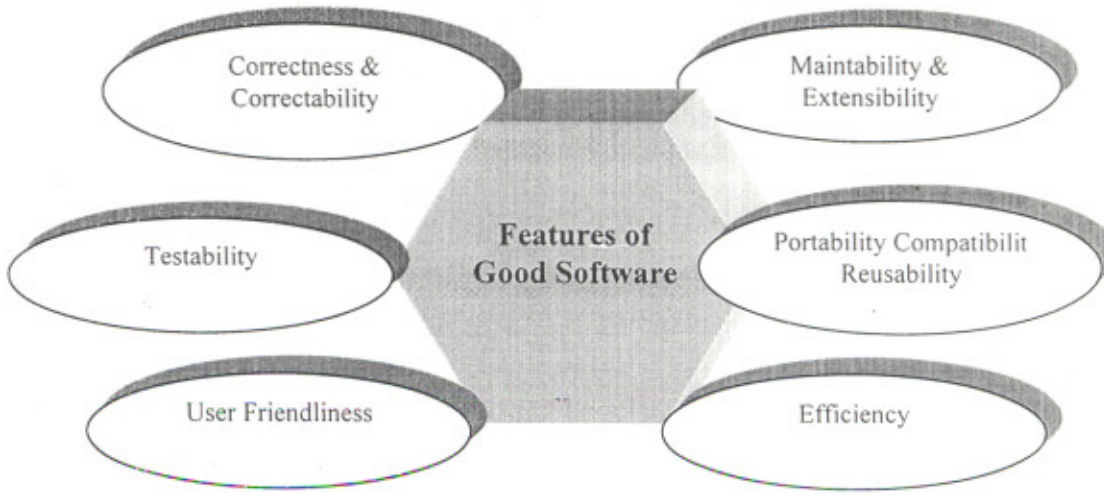


Figure 3.2 Features of Good Software

3.4.1 Correctness and Correctability

Two related concepts are correctness and correctability. As software becomes more complex, it becomes progressively more and more difficult to ensure that the software is completely correct. Obviously good software should be 'correct' as well, but this is not a black and white issue. It is possible to have various degrees of problems.

Firstly, the frequency of errors can vary. It may be acceptable for the software to fail once a year, but not once a minute. This will depend greatly on the application for which the software is being used. If the software is controlling an aircraft or a life-support system then any errors at all will be totally unacceptable. If the software is a game for home use, then having one error a year may be acceptable (especially if the cost of finding and fixing the error would have tripled the price).

Secondly, the type of error can vary. In an automatic teller, an error which sets the withdrawal limit too low on a leap day (Feb 29th, once each four years) is not going to be as critical as an error which gives every customer

twice as much money as they requested. Some errors can be tolerated; the expense of fixing the error may be greater than the inconvenience the error causes.

Thirdly, the manner in which the error is handled varies. The software can simply crash destroying all data, etc. Or it can detect the error and attempt to fix the problem, or at least notify the user that the error has occurred.

Having accepted that errors will occur, and some may even be tolerable, we can identify a related aspect of 'good' software correctability. Good software may have errors, but the software has been developed in such a way that allows the errors to be readily fixed once they have been located; i.e. the software is correctable (as opposed to correct). Poorly developed software may have fewer errors, but the errors that do exist will be extremely difficult to remove.

3.4.2 Testability

Connected to the concept of correctness and correctability is that of testability. We need to have some method of ascertaining whether the software is correct, and if not, locating the errors. It is possible to write software in such a way that makes it either easy or difficult to test. If the testing fails, then we need to determine why it has failed. Good software should facilitate this process.

3.4.3 User-Friendliness / Ergonomics

Software may be developed in such a way that it performs the job required, and is 100 percent correct. Yet if it is difficult to use then it will still be poor software. This may arise from the user-interaction being unnecessarily complex, frustratingly slow, inconsistent, difficult to understand, having poorly presented results etc. All of these factors make the software

significantly less useful than it should be, despite being completely correct in terms of its functionality.

3.4.4 Maintainability and Extensibility

Software needs to be maintained. The reasons for this are three-fold. Firstly, as the software is used, errors are found. These errors may be programming errors, design errors, or even errors in the original specification. Secondly, as the software is used, the users often discover that what they specified, is not in actual fact what they wanted. Thirdly, over time requirements change. The environment in which the software is being used changes or evolves, and the software needs to be modified to suit the new requirements. For example, a bank may need to maintain their software because they have introduced a new type of account. It is important to recognise that software is not static, but will continue to evolve and change after it has been 'completed'.

Maintainability is a measure of how easy software is to maintain. Can changes be readily made without spending an unnecessary amount of effort understanding the software? Extensibility is a measure of how easy the software is to extend and expand beyond its original scope.

3.4.5 Portability, Compatibility and Reusability

Portability is a measure of how simple it is to transfer software from one computer system to another. Software, which is non-standard or makes heavy use of particular features of a computer and its resources, will be quite difficult to port to a different system. Software that uses only standard languages and interfaces etc will be much simpler to port, and therefore better software (at least in this one respect).

Compatibility measures the ease with which software can be combined with other software. For example, a word processing program may be compatible with a drawing program, and be able to include pictures in the document being edited. It is usually important to develop the software in such a way that ensures that future compatibility can be easily maintained.

Reusability is the ease with which software can be re-used for purposes other than those intended. For example, software developed for auto-pilots in aircraft (or at least sections of the software) may be re-used in a flight simulator game. This will obviously reduce the cost of developing software, if existing software can be partially re-used or adapted.

3.4.6 Efficiency

The efficiency of software is often an important consideration. It is possible to produce a correct solution to a software problem, which is nevertheless very poor software because it wastes the computer resources. The software may run very slowly (using the CPU resource) or use a lot of memory (both data and program size). It may waste space on the screen, use a lot of disk space, or waste network bandwidth. It may clutter the printer output unnecessarily.

All these factors relate to the efficiency of the software. Often these are a trade-off between each other. For example, it may be possible to make the program run faster, but this may require more memory etc. The best compromise will depend upon the application and the computer system being used.

3.5 DEVELOPMENT STRATEGY

After getting quite a good idea what software is, what it can be used for, what makes good software and what makes bad software. So next thing is

that how do we actually go about developing software? It is NOT simply matter of sitting down at the computer and writing a program (except for maybe the simplest problem). There are a series of development phases, which form the development process.

Since software goes through various phases (i.e. it is developed, then commissioned, then used and maintained, then grows 'old', and is finally discarded) we refer to life cycle of software. The life cycle is simply the entire existence of a software product. Another way of looking at the life cycle is to consider it as the process model; i.e. a model for the development and use of software.

A process model for Software Engineering is chosen based on the nature of the project and application, the methods and tools to be used and control deliverables that are required.

All software development can be characterised as a problem-solving loop in which four distinct stages are encountered:

Status Quo: Represents current state of affairs.

Problem Definition: Identify specific problem to be solved.

Technical Development: Solve the problem through the application of some technology.

Solution Integration: Delivers the result to those who requested the solution in the first.

There are a number of different process models for software. Each gives a different method for developing software. We will discuss all this in next chapter.

CHAPTER 4

SOLUTION STRATEGY

INTRODUCTION

In this chapter an overview of the more common system development Process Models, used to guide the analysis, design, development, and maintenance of information systems has been discussed. There are many different methods and techniques used to direct the life cycle of a software development project and most real-world models are customized adaptations of the generic models. While each is designed for a specific purpose or reason, most have similar goals and share many common tasks. The similarities and differences among these various models have also been explored and chapter also discuss how different approaches are chosen and combined to address practical situations.

4.1 TYPICAL TASKS IN THE DEVELOPMENT PROCESS LIFE CYCLE

Professional system developers and the customers they serve share a common goal of building information systems that effectively support business process objectives. In order to ensure that cost-effective, quality systems are developed which address an organization's needs, developers employ some kind of system development Process Model to direct the project's life cycle. Typical activities performed include the following:

- System conceptualization
- System requirements and benefits analysis
- Project adoption and project scoping
- System design
- Specification of software requirements
- Architectural design
- Detailed design
- Unit development

- Software integration & testing
- System integration & testing
- Installation at site
- Site testing and acceptance
- Training and documentation
- Implementation
- Maintenance

4.2 ANALYSIS OF EXISTING MODELS

While nearly all system development efforts engage in some combination of the above tasks, they can be differentiated by the feedback and control methods employed during development and the timing of activities. Most system development Process Models in use today have evolved from three primary approaches:

- Ad-hoc Development
- Waterfall Model
- Iterative Process

4.2.1 Ad-hoc Development

Early systems development often took place in a rather chaotic and haphazard manner, relying entirely on the skills and experience of the individual staff members performing the work. Today, many organizations still practice Ad-hoc Development either entirely or for a certain subset of their development (e.g. small projects).

Draw Back of the Process:

Problem With this process is that with Ad-hoc Process Models, "process capability is unpredictable because the software process is constantly

changed or modified as the work progresses. Schedules, budgets, functionality, and product quality are generally (inconsistent). Performance depends on the capabilities of individuals and varies with their innate skills, knowledge, and motivations. There are few stable software processes in evidence, and performance can be predicted only by individual rather than organizational capability."

We cannot implement adhoc development because it is suitable for small projects only, in our organization as our problem is vast problem and it takes time to complete and by that time some of team members may left the organization.

In the absence of an organization-wide software process, repeating results depends entirely on having the same individuals available for the next project. Success that rests solely on the availability of specific individuals provides no basis for long-term productivity and quality improvement throughout an organization.

4.2.2 The Waterfall Model

The Waterfall Model is the earliest method of structured system development. Although it has come under attack in recent years for being too rigid and unrealistic when it comes to quickly meeting customer's needs, the Waterfall Model is still widely used. It is attributed with providing the theoretical basis for other Process Models, because it most closely resembles a "generic" model for software development.

Figure 4.1 portrays the different steps involved in waterfall model.

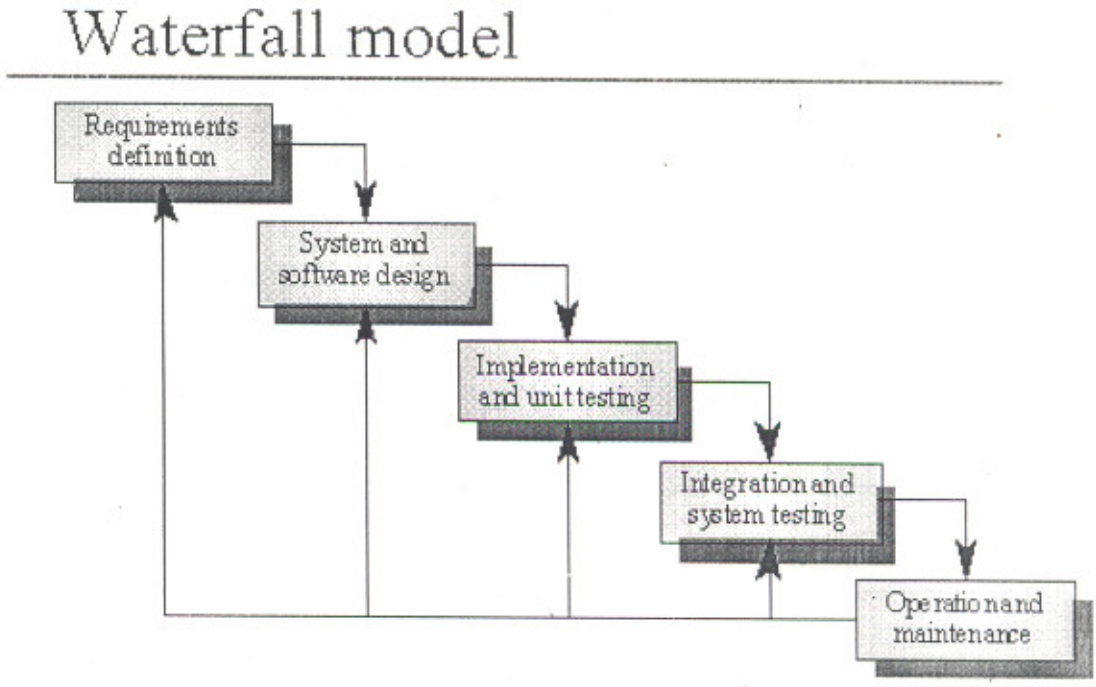


Figure 4.1 Waterfall Model

The Waterfall Model consists of the following steps:

System Conceptualization

System Conceptualization refers to the consideration of all aspects of the targeted business function or process, with the goals of determining how each of those aspects relates with one another, and which aspects will be incorporated into the system.

Systems Analysis

This step refers to the gathering of system requirements, with the goal of determining how these requirements will be accommodated in the system. Extensive communication between the customer and the developer is essential.

System Design

Once the requirements have been collected and analyzed, it is necessary to identify in detail how the system will be constructed to perform necessary tasks. More specifically, the System Design phase is focused on the data requirements (what information will be processed in the system?), the software construction (how will the application be constructed?), and the interface construction (what will the system look like? What standards will be followed?).

Coding

Also known as programming, this step involves the creation of the system software. Requirements and systems specifications from the System Design step are translated into machine readable computer code.

Testing

As the software is created and added to the developing system, testing is performed to ensure that it is working correctly and efficiently. Testing is generally focused on two areas: internal efficiency and external effectiveness. The goal of external effectiveness testing is to verify that the software is functioning according to system design, and that it is performing all necessary functions or sub-functions. The goal of internal testing is to make sure that the computer code is efficient, standardized, and well documented. Testing can be a labor-intensive process, due to its iterative nature.

Problems/Challenges associated with the Waterfall Model

Although the Waterfall Model has been used extensively over the years in the production of many quality systems, it is not without its problems. In recent years it has come under attack, due to its rigid design and inflexible

procedure. We cannot fully implement this model in our organization in our problem. Criticisms fall into the following categories:

- It is impossible to follow the sequential flow that the model proposes.
- At the beginning of project there is a great deal of uncertainty about requirements and goals, and it is therefore difficult for me to identify these criteria on a detailed level. The model does not accommodate this natural uncertainty very well.
- Developing a vast examination system using the Waterfall Model can be a long, process that does not yield a working version of the system until late in the process.
- This model stipulates that the requirements of the system should be completely specified before rest of development can proceed. But in our examination system it must be desirable to develop a part of system completely and then later enhance the system in phases.
- Freezing the requirements usually requires choosing the hardware but in a large project like us it may take few years to complete, if hardware is selected early the technology may change by the time of its completion.

4.2.3 Iterative Development

With Iterative Development, the project is divided into small parts. This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users. Often, each iteration is actually a mini-Waterfall process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, the software products, which are produced at the end of each

step (or series of steps), can go into production immediately as incremental releases. Figure 4.2 portrays iterative enhancement model.

Problems/Challenges associated with the Iterative Model

While the Iterative Model addresses many of the problems associated with the Waterfall Model, it does present new challenges.

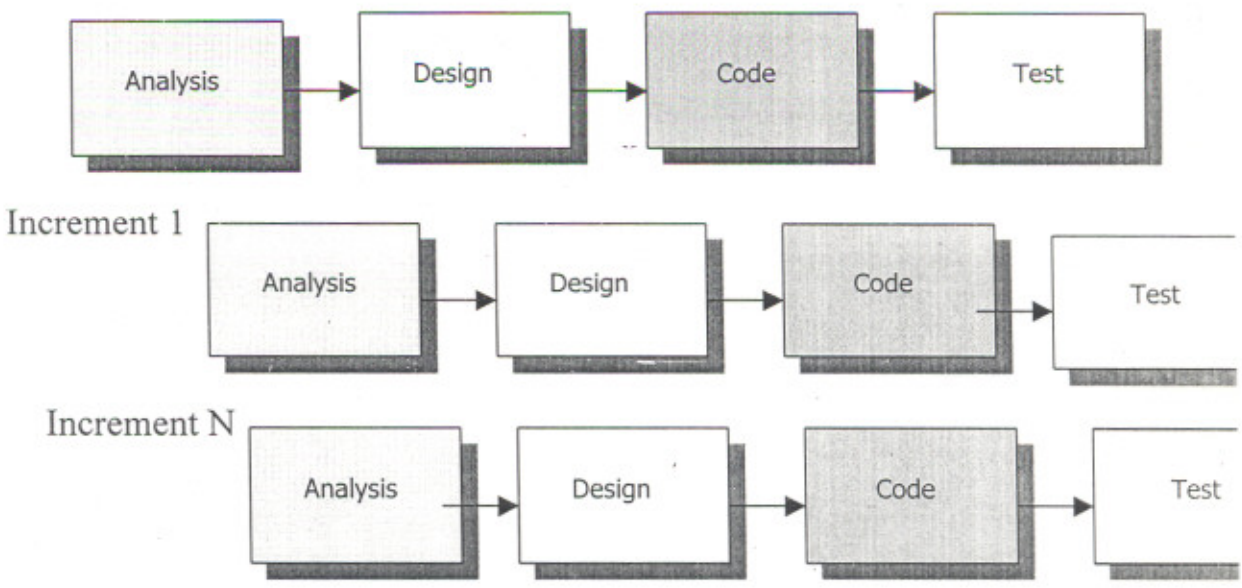


Figure 4.2 Iterative Enhancement Model

- The user community needs to be actively involved throughout the project. While this involvement is a positive for the project, it is demanding on the time of the staff and can add project delay.
- Communication and coordination skills take center stage in project development.
- Informal requests for improvement after each phase may lead to confusion a controlled mechanism for handling substantive requests

needs to be developed.

Variations on Iterative Development

A number of Process Models have evolved from the Iterative approach. All of these methods produce some demonstrable software product early on in the process in order to obtain valuable feedback from system users or other members of the project team. Several of these methods are described below.

4.2.4 Prototyping

The Prototyping Model was developed on the assumption that it is often difficult to know all of your requirements at the beginning of a project. Typically, users know many of the objectives that they wish to address with a system, but they do not know all the nuances of the data, nor do they know the details of the system features and capabilities. The Prototyping Model allows for these conditions, and offers developments approach that yields results without first requiring all information up-front.

When using the Prototyping Model, the developer builds a simplified version of the proposed system and presents it to the customer for consideration as part of the development process. The customer in turn provides feedback to the developer, who goes back to refine the system requirements to incorporate the additional information. Often, the prototype code is thrown away and entirely new programs are developed once requirements are identified.

There are a few different approaches that may be followed when using the Prototyping Model:

- creation of the major user interfaces without any substantive coding in the background in order to give the users a "feel" for what the system

will look like, development of an abbreviated version of the system that performs a limited subset of functions; development of a paper system (depicting proposed screens, reports, relationships etc.), or

- Use of an existing system or system components to demonstrate some functions that will be included in the developed system.

Prototyping is comprised of the following steps:

Requirements Definition/Collection

Similar to the Conceptualization phase of the Waterfall Model, but not as comprehensive. The information collected is usually limited to a subset of the complete system requirements.

Design

Once the initial layer of requirement information is collected, or new information is gathered, it is rapidly integrated into a new or existing design so that it may be folded into the prototype.

Prototype Creation/Modification

The information from the design is rapidly rolled into a prototype. This may mean the creation/modification of paper information, new coding, or modifications to existing coding.

Assessment

The prototype is presented to the customer for review. Comments and suggestions are collected from the customer.

Prototype Refinement

Information collected from the customer is digested and the prototype is refined. The developer revises the prototype to make it more effective and efficient.

System Implementation

In most cases, the system is rewritten once requirements are understood. Sometimes, the Iterative process eventually produces a working system that can be the cornerstone for the fully functional system.

Problems/Challenges associated with the Prototyping Model

Prototype is an idea for those large systems that do not have manual existing system, which is not the case in the system, we are developing, and criticisms of the Prototyping Model generally fall into the following categories:

Prototyping can lead to false expectations

Prototyping often creates a situation where the customer mistakenly believes that the system is "finished" when in fact it is not. More specifically, when using the Prototyping Model, the pre-implementation versions of a system are really nothing more than one-dimensional structures. The necessary, behind-the- scenes work such as database normalization, documentation, testing, and reviews for efficiency have not been done. Thus the necessary underpinnings for the system are not in place.

Prototyping can lead to poorly designed systems

Because the primary goal of Prototyping is rapid development, the design of the system can sometimes suffer because the system is built in a series of "layers" without a global consideration of the integration of all other components. While initial software development is often built to be a "throwaway, " attempting to retroactively produce a solid system design can sometimes be problematic.

4.2.5 The Exploratory Model

In some situations it is very difficult, if not impossible, to identify any of the requirements for a system at the beginning of the project. Theoretical areas such as Artificial Intelligence are candidates for using the Exploratory Model, because much of the research in these areas is based on guesswork, estimation, and hypothesis. In these cases, an assumption is made as to how the system might work and then rapid iterations are used to quickly incorporate suggested changes and build a usable system. A distinguishing characteristic of the Exploratory Model is the absence of precise specifications. Validation is based on adequacy of the end result and not on its adherence to pre-conceived requirements.

The Exploratory Model is extremely simple in its construction; it is composed of the following steps:

Initial Specification Development

Using whatever information is immediately available, a brief System Specification is created to provide a rudimentary starting point.

System Construction/Modification

A system is created and/or modified according to whatever information is available.

System Test

The system is tested to see what it does, what can be learned from it, and how it may be improved.

System Implementation

After many iterations of the previous two steps produce satisfactory results, the system is dubbed as "finished" and implemented.

Problems/Challenges associated with the Exploratory Model

As our system is not an assumption or guesswork we cannot implement this model in our organization. There are numerous criticisms of the Exploratory Model:

- It is limited to use with very high-level languages that allow for rapid development, such as LISP.
- It is difficult to measure or predict its cost-effectiveness.

4.2.6 The Spiral Model

The Spiral Model was designed to include the best features from the Waterfall and Prototyping Models, and introduces a new component - risk-assessment. The term "spiral" is used to describe the process that is followed as the development of the system takes place. Similar to the Prototyping Model, an initial version of the system is developed, and then repetitively modified based on input received from customer evaluations. Unlike the Prototyping Model, however, the development of each version of the system is carefully designed using the steps involved in the Waterfall Model. With each iteration around the spiral (beginning at the center and working outward), progressively more complete versions of the system are built.

Figure 4.3 clearly portrays spiral model. Risk assessment is included as a step in the development process as a means of evaluating each version of the system to determine whether or not development should continue. If the customer decides that any identified risks are too great, the project may be halted. For example, if a substantial increase in cost or project completion time is identified during one phase of risk assessment, the customer or the

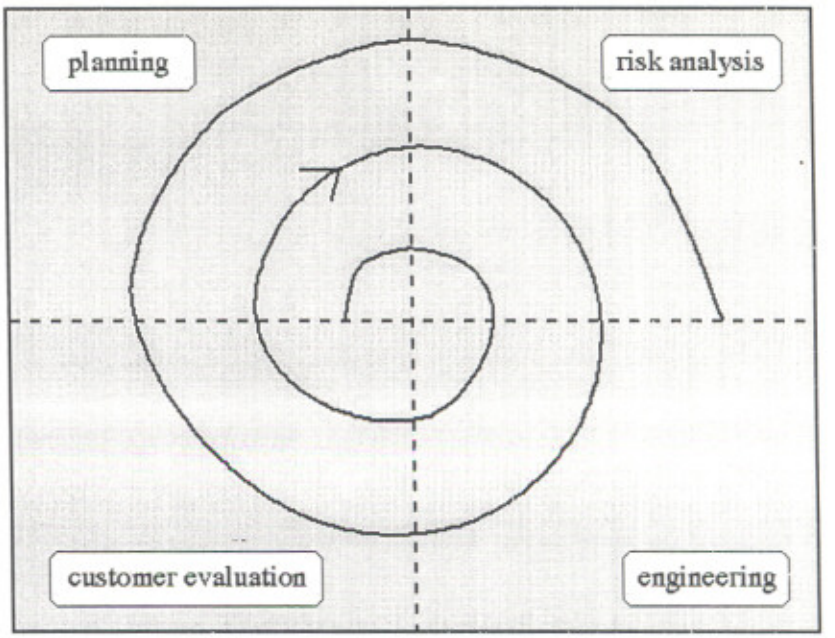


Figure 4.3 The Spiral Model

developer may decide that it does not make sense to continue with theproject, since the increased cost or lengthened timeframe may make continuation of the project impractical or unfeasible.

The Spiral Model is made up of the following steps:

Project Objectives

Similar to the system conception phase of the Waterfall Model. Objectives are determined, possible obstacles are identified and alternative approaches are weighed.

Risk Assessment.

Possible alternatives are examined by the developer, and associated risks/problems are identified. Resolutions of the risks are evaluated and weighed in the consideration of project continuation. Sometimes prototyping is used to clarify needs.

Engineering & Production

Detailed requirements are determined and the software piece is developed.

Planning and Management

The customer is given an opportunity to analyze the results of the version created in the Engineering step and to offer feedback to the developer.

Problems/Challenges associated with the Spiral Model

Due to the relative newness of the Spiral Model, it is difficult to assess its strengths and weaknesses. However, the risk assessment component of the Spiral Model provides both developers and customers with a measuring tool that earlier Process Models do not have. The measurement of risk is a feature that occurs everyday in real-life situations, but (unfortunately) not as often in the system development industry. The practical nature of this tool helps to make the Spiral Model a more realistic Process Model than some of its predecessors.

4.2.7 The Reuse Model

The basic premise behind the Reuse Model is that systems should be built using existing components, as opposed to custom-building new components. The Reuse Model is clearly suited to Object-Oriented computing environments, which have become one of the premiere technologies in today's system development industry.

Within the Reuse Model, libraries of software modules are maintained that can be copied for use in any system. These components are of two types: procedural modules and database modules.

When building a new system, the developer will "borrow" a copy of a module from the system library and then plug it into a function or procedure. If the needed module is not available, the developer will build it, and store a copy in the system library for future usage. If the modules are well engineered, the developer with minimal changes can implement them.

The Reuse Model consists of the following steps:

Definition of Requirements

Initial system requirements are collected. These requirements are usually a subset of complete system requirements.

Definition of Objects

The objects, which can support the necessary system components, are identified.

Collection of Objects

The system libraries are scanned to determine whether or not the needed objects are available. Copies of the needed objects are downloaded from the system.

Creation of Customized Objects

Objects that have been identified as needed, but that are not available in the library are created.

Prototype Assembly

A prototype version of the system is created and/or modified using the necessary objects.

Prototype Evaluation

The prototype is evaluated to determine if it adequately addresses customer needs and requirements.

Requirements Refinement

Requirements are further refined as a more detailed version of the prototype is created.

Objects Refinement

Objects are refined to reflect the changes in the requirements.

Problems/Challenges Associated with the Reuse Model

A general criticism of the Reuse Model is that it is limited for use in object-oriented development environments. Although this environment is rapidly growing in popularity, it is currently used in only a minority of system development applications.

4.3 CREATING NEW MODEL BY COMBINING MODELS

We will integrate parts and procedures from various Process Models to support system development. This occurs because most models were designed to provide a framework for achieving success only under a certain set of circumstances. When the circumstances change beyond the limits of the model, the results from using it are no longer predictable. When this situation occurs it is sometimes necessary to alter the existing model to accommodate the change in circumstances, or adopt or combine different models to accommodate the new circumstances.

The selection of an appropriate Process Model hinges primarily on two *factors:* organizational environment and the nature of the application.

Suitable approaches to system analysis, design, development, and implementation be based on the relationship between the information system and its organizational environment. Four categories of relationships are identified:

The Unchanging Environment

Information requirements are unchanging for the lifetime of the system (e.g. those depending on scientific algorithms). Requirements can be stated unambiguously and comprehensively. A high degree of accuracy is essential.

The Turbulent Environment

The organization is undergoing constant change and system requirements

are always changing. A system developed on the basis of the conventional Waterfall Model would be, in part; already obsolete by the time it is implemented. Many business systems fall into this category. Successful methods would include those, which incorporate rapid development.

The Uncertain Environment

The requirements of the system are unknown or uncertain. It is not possible to define requirements accurately ahead of time because the situation is new or the system being employed is highly innovative. Here, the development methods must emphasize learning. Experimental Process Models, which take advantage of prototyping and rapid development, are most appropriate.

The Adaptive Environment

The environment may change in reaction to the system being developed, thus initiating a changed set of requirements. Teaching systems and expert systems fall into this category. For these systems, adaptation is key, and the methodology must allow for a straightforward introduction of new rules.

4.4 NEED FOR DEVELOPMENT OF NEW PROCESS MODELS

As customers demanded faster results, more involvement in the development process, and the inclusion of measures to determine risks and effectiveness, the methods for developing systems changed. In addition, the software and hardware tools used in the industry changed (and continue to change) substantially. Faster networks and hardware supported the use of smarter and faster operating systems that paved the way for new languages and databases, and applications that were far more powerful than any predecessors.

These rapid and numerous changes in the system development environment simultaneously spawned the development of more practical new Process Models and the demise of older models that were no longer useful.

4.5 PROPOSED MODEL

From the above discussion we conclude that no model can be effectively applied in our organization. So we have to use suitable features from different studied models along with some assumptions to present a new model suitable to our problem with some specific assumptions. Mostly we will follow the some features of Waterfall Model, Iterative Model & Reuse model. Figure 4.4 portrays proposed model for software development.

In our organization the information & system requirement may be slightly changing and also complete computerization of examination is a long process.

We will follow the feature of iterative development to divide the whole project into small modules and each module will follow partial guidelines from Waterfall Model. From Reuse Model we will follow the Object Oriented feature in which libraries of Software modules are maintained and can be copied for use in any module at later stage.

The Proposed Model consists of the following steps:

Analysis of the System

In this step we will completely analyze the system. The complete study of requirements, information resources is done at the beginning of the software developing process.

Module Breakup based on analysis

We will divide the whole system into small modules by fixing the boundaries of modules completely based upon the analysis of system. All

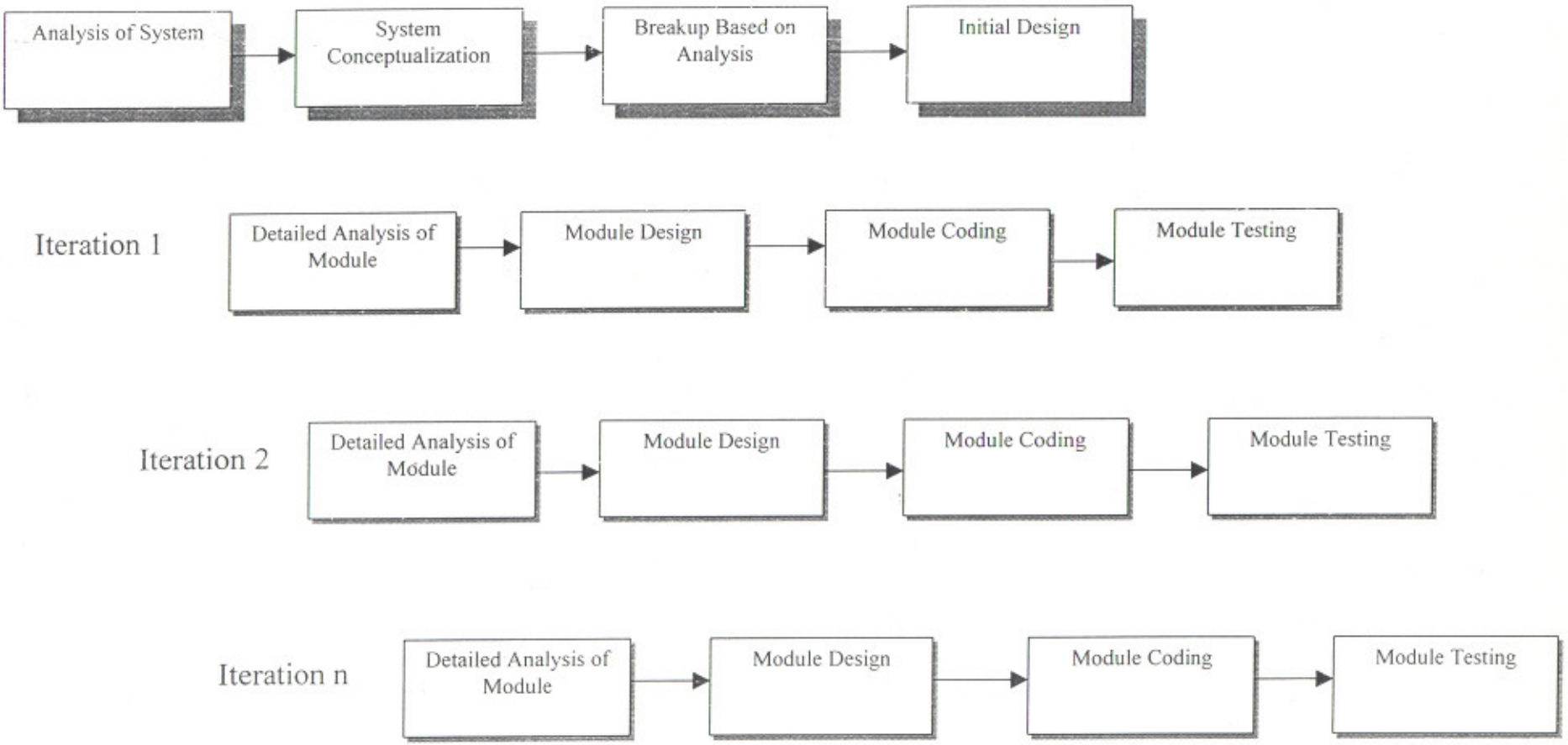


Figure 4.4 Proposed Model For Software Development

the information regarding each module needs to be fixed in the beginning to avoid wastage of time at the end stage.

System Conceptualization

System conceptualization refers to the consideration of all aspects of system that is how the system will physically look like. What are the major aspects like Hardware, Networking, Database approach etc. Here we will fix type of hardware needed but will only purchase that is required for current module so that the complete hardware may not obsolete by the time of project completion.

Initial System Design

The next stage is partial design the whole system at the beginning because our modules are interdependent & related so we have to fix the initial design of each module so that we should find out common things in each module so that we could take advantage of concept of reuse model (OOP's)

Detailed Module Design

Next stage is detailed system design of modules. This we will do module wise firstly we will design & implement a module completely before jumping to second module.

Module Wise Coding

In coding we will follow the iterative model technique to firstly code the one module before jumping to next module. So that we can if possible use some code of previous module.

Module Wise Testing

Next phase is Testing i.e. after the module is coded it must be tested with some dummy data for different conditions before actually implementing it.

CHAPTER 5

SYSTEM DESIGN METHODOLOGY

INTRODUCTION

This chapter presents the outlay of system design and the methodologies used in the current system design. The design phase focus on the detailed implementation of the system recommended in the analysis of system. Emphasis is on translating performance specifications into design specifications. Certain methodologies to be followed in this case study for system design has been explained in this chapter.

5.1 DESIGN PHASES

The design phase is a transition from a user-oriented document to a document oriented to the programmers or data base personnel. Basically

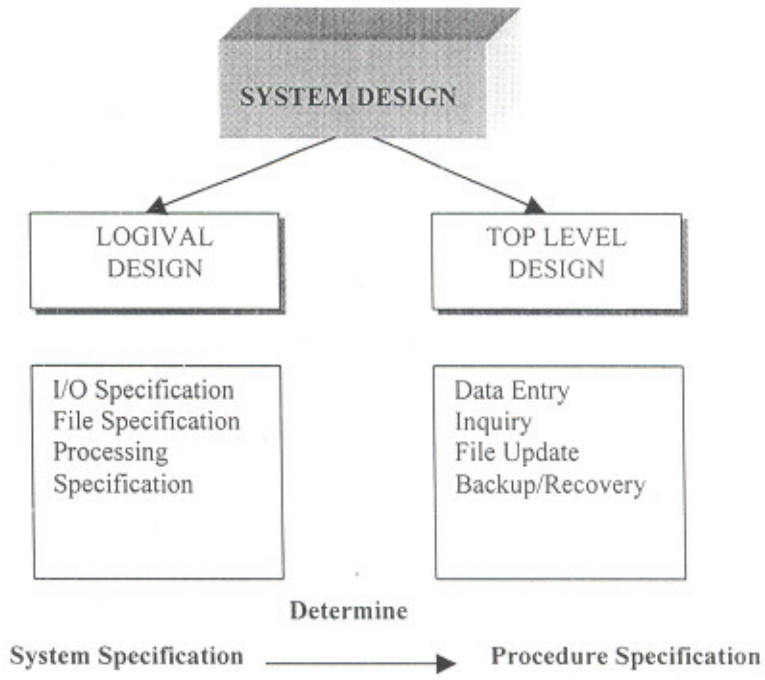


Figure 5.1 System Design Phases

Design process has two levels. One is called system design or top level. design other is called detailed design or logic design. Figure 5.1 portrays system design phases.

5.1.1 Top Level Design

At the first level the focus is on deciding which modules are needed for the system, the specification of these modules and how the modules should be interconnected to produce desired results. This type of design is called system design.

5.1.2 Logical Design

In this phase internal design of modules, or how specifications of the module can be satisfied. It expands the system design to contain a more detailed description of the processing logic and data structures.

5.2 DESIGN METHODOLOGY

To develop any system that meet user's requirement some new effective techniques for developing the system are used called design methodology. The main uses of methodology are:

- Improve productivity of Analyst & Programmer.
- Improve documentation & subsequent maintenance.
- Improve communication among user, analyst, programmer & designer.
- Standardize the approach of analysis & design.
- Simplify design by segmentation.

In the study it has been found that most suitable methodology in our problem is Structured Design. So the structured methodology has been used to develop the system. Salient features of methodology are:

5.2.1 Structured Analysis

Structured analysis is a set of techniques and graphical tools that helps the analyst to develop a new kind of specifications that are easily understandable to user. Some of the structured analysis tools used in my study are given below.

5.2.1.1 Data Flow Diagram

The purpose of data flow diagram is to clarify system requirements and identifying major transformations that will come across programs in system design in graphical manner.

5.2.1.2 Data Dictionary

Data dictionary is a structured repository of data about data. It is a set of definitions of all data flow diagrams data elements and data structures. Data dictionary is a valuable reference and it improves user/analyst communication and it is an important step in building a database.

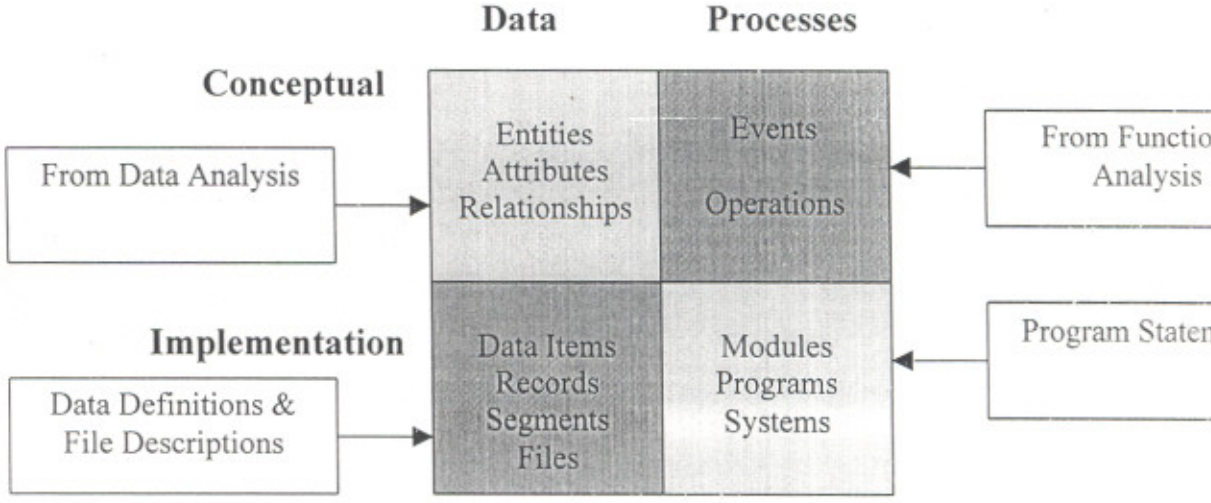
The British Computer Society set up a working party to suggest a standard for Data Dictionary System (DDS) which suggests that DDS should provide two sets of facilities:

To record and analyze data requirements

To record Design decision in terms of data base or file structure implemented and programs which access them.

These two sets are referred as Conceptual data Model & Implementation data structure.

Figure 5.2 Four Quadrants Representation of Data Dictionary System



Conceptual model shows a model of organization i.e. the entities their attributes etc. It can also include the details of events & operations that occur in the organization. Figure 5.2 portrays four quadrant of data dictionary and figure 5.3 portrays users of data dictionary system at different levels..

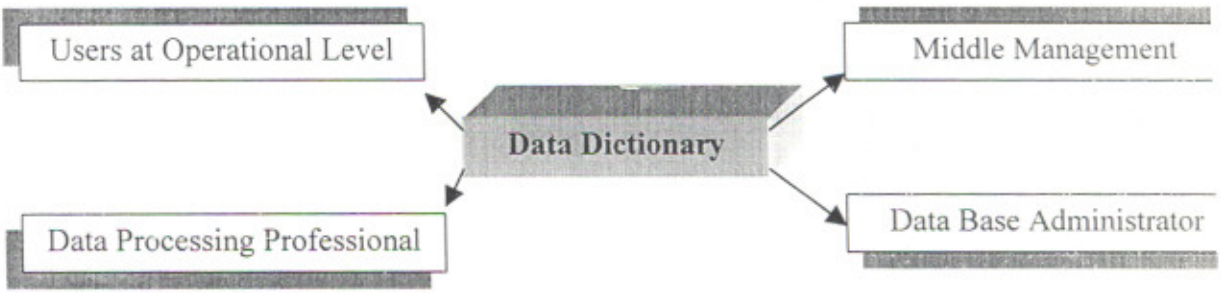


Figure 5.3 Users of Data Dictionary System

The Implementation view gives information about the data processing Smallest Unit of Data applications in computing terms. The Process are described as programs, subprograms and systems and data is described in information management in the form of files records & fields (figure 5.4).

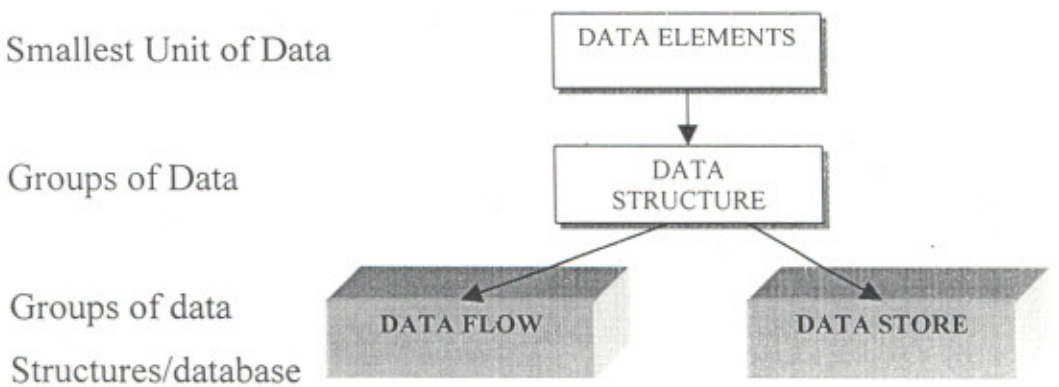


Figure 5.4 Logical Data description Hierarchy

Data dictionary contain much of information that is useful for all levels of project members.

5.2.1.3 Decision Tree

Decision Tree is used to portray the logic of the system. It simply sketches the logical structure based on the stated policy. It is easy to construct, easy to read and easy to update. It show only the skeletal aspects of the process and it does not lend itself to calculations or show logic as a set of instructions for action.

5.2.1.4 Decision Table

Decision table is a table of contingencies for defining a problem and the actions to be taken. Decision table consists of two parts: Stub and Entry. Stub part is divided into upper quadrant called Condition Stub and lower

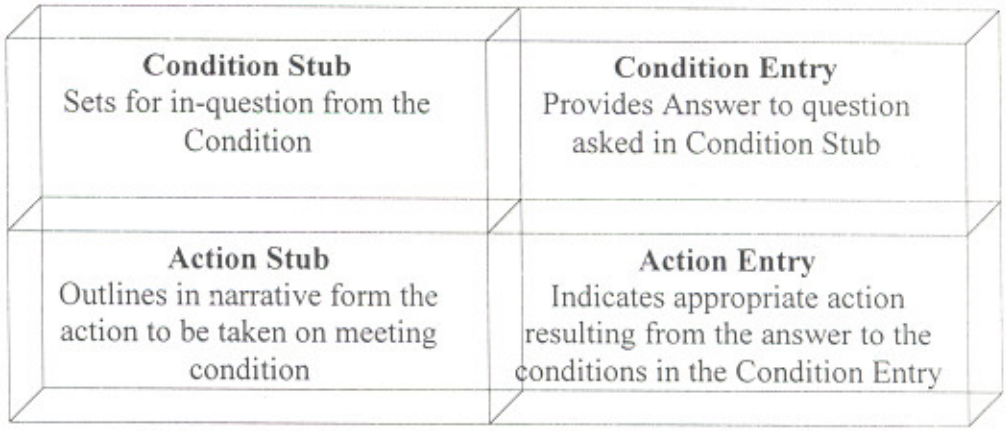


Figure 5.5 Four Quadrants of a Decision Table

Quadrant called Action Stub

The Entry part is also divided into upper quadrant called Condition Entry and Lower Quadrant called Action Entry. Figure 5.5 portrays four quadrants of a decision table.

5.2.2 Structured Design

In whole system study Structured design methodology has been used

which begins with system specification that identify input and output and describes the functional aspects of the system. System specifications are used for graphic representation i.e. Data Flow Diagram. From Data flow Diagram next step is definition of modules and their relationship with one

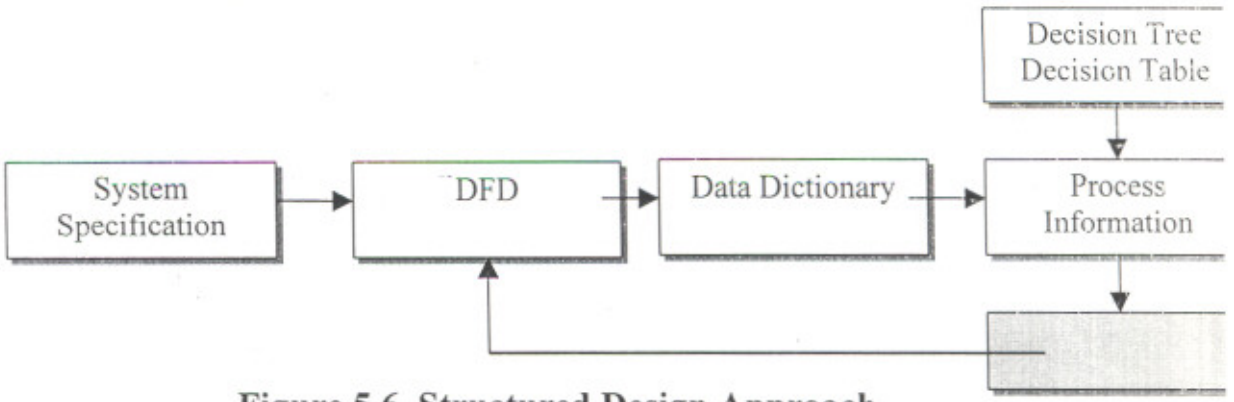


Figure 5.6 Structured Design Approach

another called structure chart using data dictionary and other tools. By structure design program has been divided into smaller parts and then they are organized following the Bottom Up technique which is defined as under. Figure 5.6 portrays structured design approach.

5.2.2.1 Bottom Up Technique

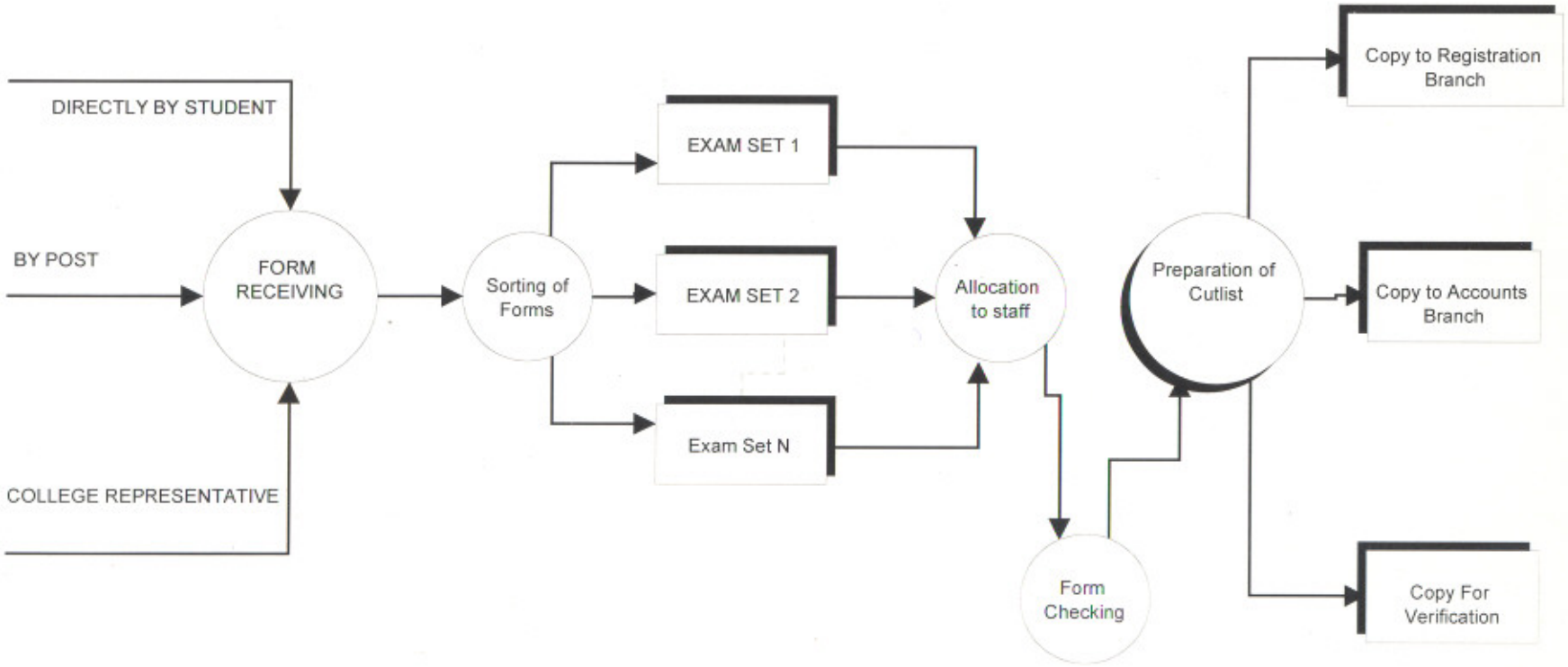
Bottom up approach starts with lowest level component of the hierarchy and proceeds through progressively higher level to the top-level component. Bottom up approach starts with the most basic components and proceeds to higher level components that use these lower level components.

5.3 ANALYSIS OF EXISTING SYSTEM

After talking with Assistant Registrar, Superintendents and Assistants it was found that the existing system has some serious flaws. The salient feature of existing system; as revealed during present study are as given below:

- As per present trends the organization is doing all the examination work manually starting from the beginning. University has made particulars sets for each class. For conducting the examination different centers are made in the jurisdiction area of Punjabi University, these centers can decrease or increase according to requirement. Set is of two persons and is allotted few centers, and they have to take care of all the examination activity for those centers.
- Starting from the beginning, students has to fill the long form for taking the examination and in filling such a long form they even do certain un-recoverable mistakes and University authorities has to correspond to them for knowing the facts waiting too much time.
- After the students submit the forms each form is checked individually for eligibility consideration i.e. weather the candidate is eligible to take the exam or not. This is a very long and time-consuming process.
- It can be said that in the very first step of submission of forms chances of corruption are there. For example students are submitting the forms with the help of internal staff even after the closing date by taking manual receipts in back date.
- After the eligibility checking the forms are sorted according to different centers opted by the student, rollnumbers are allotted to forms.
- After the allotment of centers a cutlist is made for reference at examination centers, examination sets and registration Branch which includes the candidate Roll No., Name, Fname paper opted by candidate.
- In the next step the cutlist is verified by the registration branch and a duplicate copy of records is kept their for Keeping the whole record of

Figure 5.7 DATA FLLOW DIAGRAM BEFORE EXAM FOR OLD SYSTEM



student at one place so that he may not appears for two main examination at one time so in this case duplicate entry has to be made.

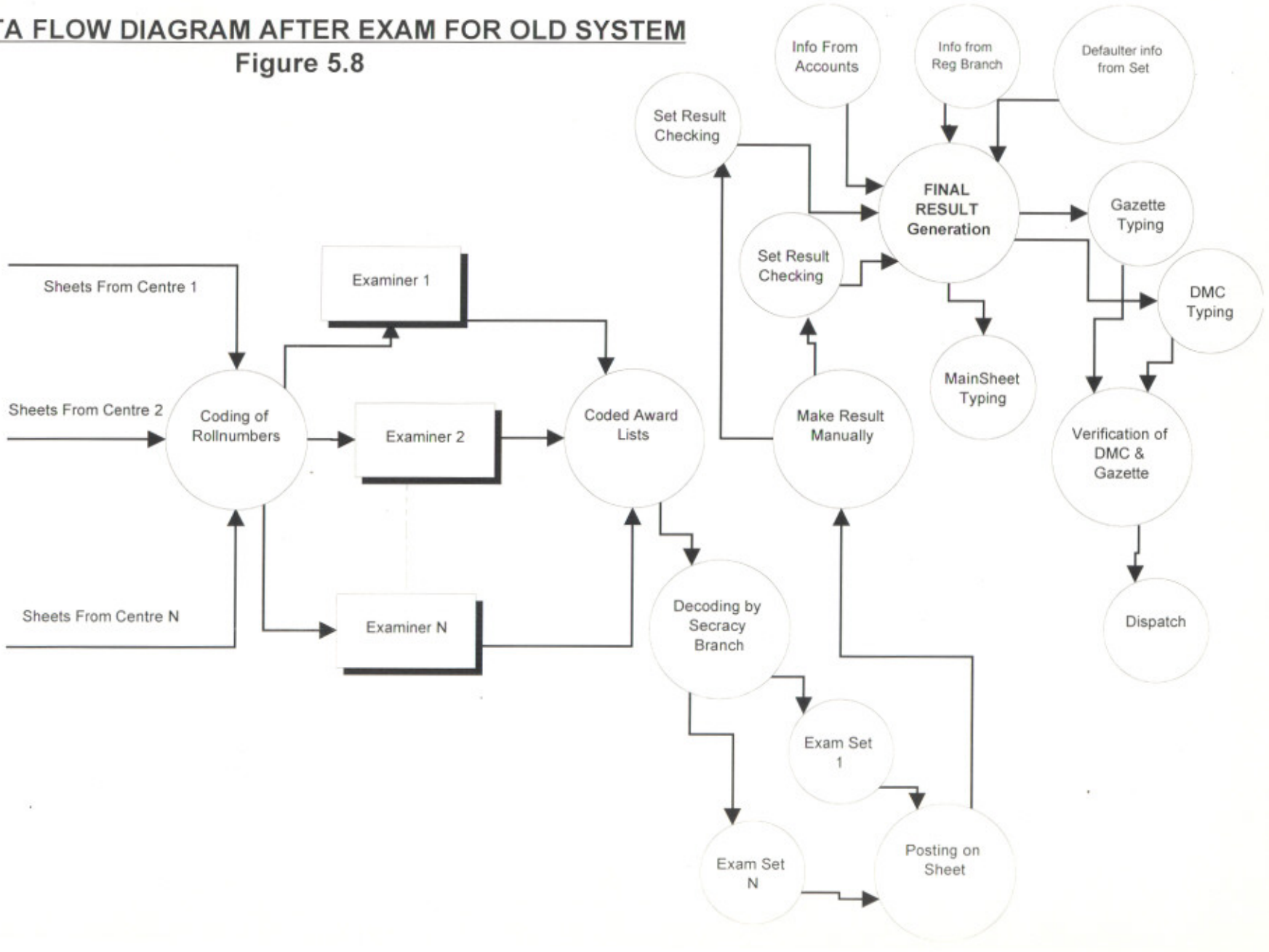
- Finally after the verification of cutlist for name registration no. etc again a final cutlist is made to be used for reference at examination centers and at examination sets. So again staff has to do labor for typing the cutlist.
- In the next step Roll Numbers are dispatched to students.
- As their are thousands of students appearing in the examination and their are hundreds of combinations of subjects and also their are about hundred of examination centers at different places in Punjab and outside Punjab, its very difficult so conduct exams without knowing that which set of papers and how many papers are to be sent to which particular center.
- To overcome the above problem a central statement is made by considering the papers opted by the students in a particulars center and number of students in each subjects at different centers. This center statement making is really a very difficult task, as staff has to watch the individual student record, one by one very carefully to note the subject opted by the student.

Finally the examination starts and first phase of examination is over. Now comes the second phase that is the phase after the examination.

- During the examination phase an absentee statement is made at each center, which includes the Roll Numbers of students absent in the examination on a particular date in a particular paper. It also contains the number of student's present subject wise.

DATA FLOW DIAGRAM AFTER EXAM FOR OLD SYSTEM

Figure 5.8



- In the post examination phase answer sheets are evaluated by the examiners and award lists are prepared.
- In the next step staff members are again duplicating the cutlist information on the result sheet so that awards corresponding to particular Roll Number can be posted on the result sheet for future reference. In this phase absentee statement is referred side by side for marking absent on the result sheet.
- Finally when all the awards and absents are marked on the result sheet an expert, who suppose to know all the rules and regulation of examination system for Pass, Fail and Grace rules examine the result sheet and make calculation like totaling, allotting required grace etc for individual students. In this case more interesting thing is that expert's decision is final and he marks the students fail or pass on the sheet and result is prepared.
- Repetition work starts again in the next step as gazette's have to be made for distribution in different colleges and markets. Again the same information (i.e. Name, Father's Name, Roll Number etc.) is typed on papers for the gazette preparation.
- Not to end here, staff has to repeat again the same typing procedure for typing same information (i.e. Roll Number, Name, Father's Name, Subject etc.) for Detail Marks Card preparation.
- And finally Detail Marks Cards are dispatched to students.

Figures 5.7 & 5.8 portrays data flow diagram of the system.

5.3.1 Need For the Study

- As the University is running so many courses and it has to maintain records of about one lakh of students and has to declare the results of all

the classes within a scheduled time. There is a need to computerise the existing examination system to bring efficiency and reliability in the system.

- Their are thousands of papers and each has different passing conditions and some time their are serious errors in calculating results like some failed students are declared passed and passed students are failed by mistake. This problem may also be tackled with the help of computerisation.

Besides existing system has the following problems.

- Information is not centralised e.g. if a user has to modify his name Firstly he has to move to the corresponding set. (e.g. BA or M.A. set) and then he has to move to Registration Branch for change of name etc.
- It is difficult to remember the rules of result calculation and even result calculations i.e. totally giving grace according to fixed rules is slow manual process.
- There are errors in result calculation.
- It is not easy to maintain huge records. Even we can not find the required information very quickly.
- It is not easy to search any old information as the record are moved to store for future use.

5.4 SYSTEM CONCEPTUALIZATION

5.4.1 Network Approach

Here centralize network approach has been proposed in which server is placed at the central place and all other terminals are connected to it through structured cabling using CAT5 cable and Hubs. All the data is stored at the central terminal and computers are provided at each set so that they can

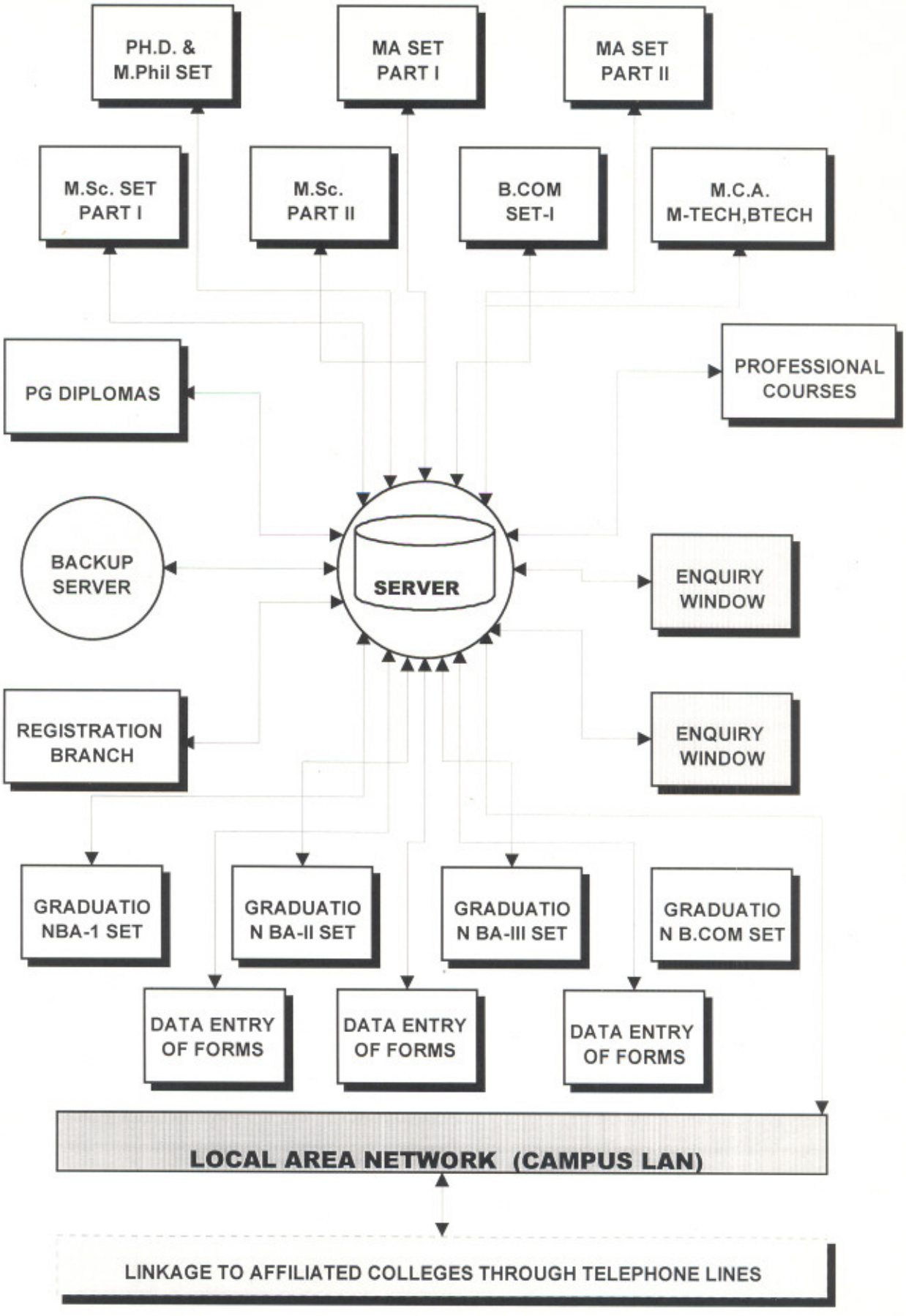


Figure 5.9 OUTLINE OF PROPOSED NETWORK

access the particular data from the server. Server is also connected to local area network of organization and also there is proposal to connect the server to affiliated colleges through telephone lines and can be given read only permission. Figure 5.9 portrays outline of proposed Network.

5.4.2 Database Approach

In the current problem integrated data base approach of database has been proposed due to its advantages that suits the design. Main advantages due to which this approach has been chosen are:

- Sharing of data
- Control of Redundancy
- Data Consistency,
- Better data security
- Better data accessibility
- Better backup and recover procedures.

5.5 SYSTEM ANALYSIS & BREAKUP

5.5.1 Proposed System Breakup

The approach of structured design is started with system specifications that identifies inputs and outputs and describes the functional aspects of the system. Following the methodologies of structured design the large system is partitioned into smaller phases and continue further till we reach a stage where components are small enough to be designed separately, but each phases is related to another. The design consists a hierarchy of modules, with each module having a single entry and single exit.

Clearly by following the bottom up technique the entire examination system is divided in to many modules depending upon the suitability differentiability like Graduation (BA & B.Com & B.Sc.) modules, PG (MA

& Msc.) Modules, Professional Courses Module and Ph.D. & M.Phil. module.

5.5.2 Module Division

Further each module can be divided into three phases. Starting from very beginning by just collecting forms and progressively proceed through higher levels of the result calculations and printing of gazette by following the data available from each bottom level. Each phase of examination is dealt almost separately and separate programs are made for each phase.

i Pre Examination Phase

ii Examination Phase

iii Post Examination Phase

5.5.2.1 Pre Examination Phase

The pre-examination phase is again divided into different smaller phases.

i Form Collection Phase

ii Data Entry of Form & Eligibility Analysis

iii Cutlist Preparation & Center allotment

iv Preparation of Center Statement

In the form collection phase forms are received from the students and are sorted according to the center opted by the students for taking the examination.

In the second phase of data entry of form data entry of forms is done in the computers in a database file and then the eligibility of candidates is checked with the help of computer & candidate found not eligible is marked in the remarks column as not eligible and a letter is send to not eligible candidate so that he can supply the required documents.

Finally the center and rollnumbers are allotted to students according to choice given by the students. This point is to be noted that rollnumbers are

not issued to eligible students but are allotted to the student, this is done to avoid inconvenience to students because if a student clears its eligibility and at that time the center of his choice is filled and then that center cannot be allotted to the student.

A central statement is made by considering the papers opted by the students in a particulars center and number of students in each subjects at different centers. The statement is very useful as their are thousands of students appearing in the examination and their are hundreds of combinations of subjects and also their are about Hundred of examination centers at different places in Punjab and outside Punjab, its very difficult so conduct exams without knowing that which set of papers and how many papers are to be sent to which particulars centers.

5.5.2.2 Examination Phase

i Conduct of Examination

ii Making Absentee Statements

iii Manual Evaluation Phase

In the examination phase more part is to be done manually. As different superintendents are allotted different centers and a copy of cutlist (i.e. the name rollnumber and paper opted by students in that center) is given to the center superintendent so that their is no trouble in the conduct of examination. From the cutlist it is clear that what papers a particular student has to take.

Center superintendent has to make clearly the absentee statements manually i.e. he has to make clear details of rollnumbers for which candidate is absent or present paper wise. This absentee statement is very essential at the time of calculations to tally conflicting rollnumbers.

Finally after the exams the papers are sent for evaluation or marking and paper wise lists are made for awards against each rollnumber and are sent to computer center for data entry.

5.5.2.3 Post Examination Phase

i Entry of Awards

ii Entry of Absentee Statements

iii Calculations & Checking Phase

iv Printing Phase

Post Examination phase starts with the entry of awards for different papers. As soon as a paper is complete a hardcopy of entered awards is taken for rechecking and finding the typing mistakes.

Similarly absent rollnmbers are entered for different papers and hardcopy of entered rollnumbers is taken for rechecking and finding the typing mistakes.

Finally comes the most crucial phase of calculations. It's a very long process and different things that are need to be done are:

- Firstly the data entry of awards is to be completed and the rollnumbers are checked for uniqueness. Also absent rollnumbers are entered and awards for absent rollnumbers are assumed to be -3.
- These files of corresponding papers (i.e. paper A or B) are combined and checked for duplicate. If there are duplicate rollnumbers then there is a problem i.e. the candidate having marks is also marked absent. This duplicacy has to be removed by considering the papers.
- All the papers (A,B, and C) are combined in this way by marking paper an as absent or Paper B as absent.
- Then all the completed files are mapped with cutlist file by considering rollnumber as primary key and also paper option is also checked while mapping these files. If the corresponding options of paper matches then

that rollnumber is mapped otherwise there is a problem and has to be removed.

- Finally when all the discrepancies are resolved the main program of result calculation is started which marks the result and give particular grace where needed according to rules.

In the next phase when all the process is complete printing phase starts and gazettes, resultsheets and detail marks card are printed.

5.5.3 ANALYSIS OF GENERAL RULES OF THE SYSTEM

5.5.3.1 General Grace Rules:

- Total grace that can be given to candidate is 1% of Total Marks. e.g. if total marks are 800 then grace 8 numbers can be given.
- Grace can only be given to those candidates that are going to pass by obtaining the grace or their result is going to be re-appear by obtaining the grace. Grace cannot be given to Passed and failing candidates.
- Grace can be given by division to more than one paper if the candidate is failed in more than one paper.
- Only minimum grace can be given e.g. if a candidate is failed in two papers in first paper he needs 5 marks and in second paper he needs 4 marks to pass. Now the candidate is fail in two papers and total grace needed is 9 that exceed maximum grace(8). Then in this case we have to allot minimum grace i.e. 4 so that the result of candidate should be reappear.
- If the candidate is failed in two papers in both the first papers he needs 5 marks to pass. Now the candidate is fail in two papers and total grace needed is 10 that exceed maximum grace (8). Then in this case we have to allot grace to paper falling first in the fixed coded sequence of papers.

5.5.3.2 General Abbreviations

- Certain abbreviations are also used as

C.C.	Candidature Cancelled
N.E.	Not Eligible
RLF	Result late due to fee dispute
RLE	Result Late due to Eligibility
RLR	Result late due to Registration

In all above cases what ever may be the reason the result in gazette is set to C.C.

5.6 CODING TECHNIQUES

In this phase of Software development design of a system is translated into code that can be compiled and executed. Coding has great impact on internal structure, which effects the testability and understandability of the system. The main points that are taken into consideration while coding the system are:

Simplicity of Programs

Emphasis is on producing simple and clear programs. The aim was not to reduce the coding effort but to program in a manner such that testing and maintenance time is saved.

Structured Programming

Structured programming approach is used to enhance the readability of programs and the programs are in a sequence of single entry single exit statements and control flow during execution is generalized.

Bottom up Technique

Bottom up technique is used for coding that is most basic components are coded first i.e. the components that lies on bottom of hierarchy.

Documentation

*I*nternal documentation is used to make program more readable. Usually comments are written for blocks of code.

5.7 TESTING STRATEGIES

In this Section the plan for testing the software is described and also some software testing strategies have been fixed that will be used in complete development of the system.

5.7.1 Software Testing

Software testing is the process of executing Software in a controlled manner, in order to answer the question " Does the Software behave as specified". Software testing is done to obtain the objectives:

- It is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an error as yet discovered
- A successful test is one that uncovers an as yet discovered error.

Testing should systematically un cover different classes of errors in a minimum amount of time and with minimum effort. A secondary benefit of testing is that the software appears to be working as stated in the specifications.

In testing the Software many other things that were used are Validation and Verification.

Verification is the checking or testing of items including software for conformance and consultancy with an associated specification.

Validation is the process of checking that what has been specified is what the user actually wanted.

Validation: Are we doing the right job?

Verification: Are we doing the job right?

Another things associated with testing are:

Debugging

It is the process of analyzing and locating bugs when software does not behave as expected. Debugging is therefor an activity which support testing but cannot replace testing.

Static Analysis

In this phase the investigation is done on source code of software. The code for the problem is look at for gathering metrics without actually executing the source code.

Dynamic Analysis

In this phase behavior of software is looked at while it is executing, to provide information such as traces, timing profiles and tests coverage information.

5.7.2 Strategic Approach For Software Testing

Figure 5.10 portrays different levels of testing required. The activities or strategies that have been planned in advance for software testing and conducted systematically are:

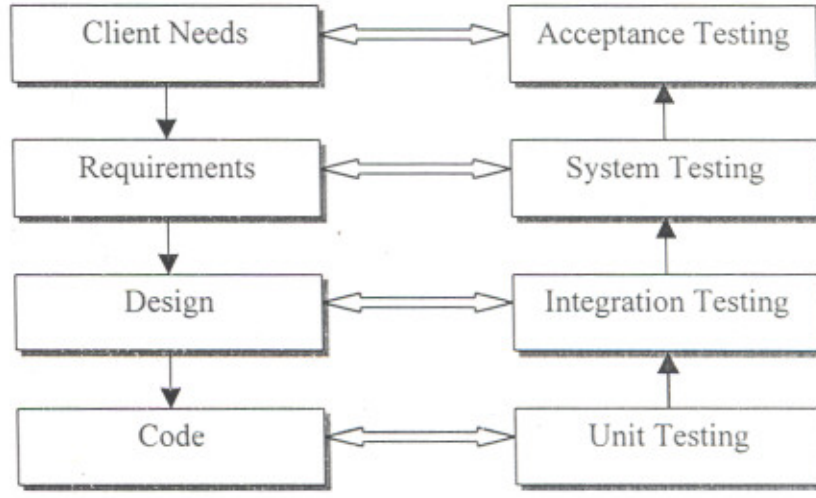


Figure 5.10 Levels of Testing

5.7.2.1 Unit Test

In this test each unit of the software is tested to verify that detailed design for the unit has been correctly implemented. In this we use white box testing techniques at each step.

5.7.2.2 Integration Testing

In this many tested modules are combined into sub systems and then tested, the goal here is to see is system can be integrated properly.

5.7.2.3 System Testing

Here the entire software system is tested and the goal is to see if software meets its requirements.

5.7.2.4 Acceptance Testing

Acceptance testing is done with some realistic data of the client to demonstrate that software is working satisfactorily focus is here on external behavior of system.

5.7.3 Test Case Design Techniques

These can be broadly split into two categories as also shown in figure 5.11.

5.7.3.1 Black Box Techniques (Functional)

Black box techniques use the interface to a unit and a description of functionality, but do not need to know how the inside of a unit is built.

5.7.3.2 White Box Techniques (Structural)

White box make use of information about how the inside of a unit works and uses control structure of the procedural design to derive test cases.

These further can be divided as under:

Specification Derived Tests

Test cases are designed by walking through the relevant specifications. Each test case should test one or more statements of specification.

Equivalence Partitioning

Equivalence partitioning is a much more formalized method of test case

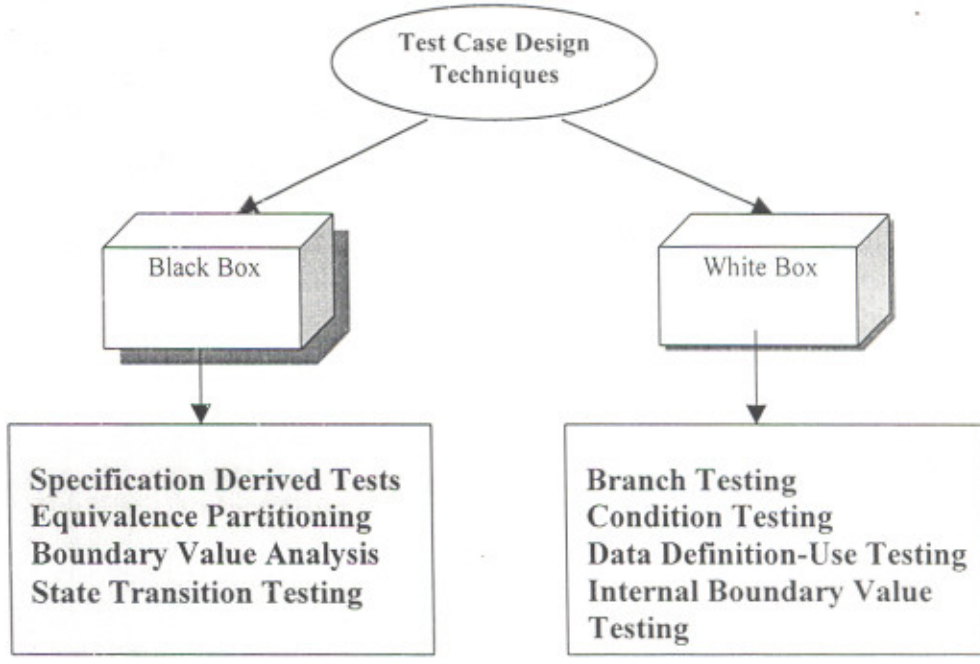


Figure 5.11 Test Case Design Techniques

design. It is based upon splitting the inputs and outputs of the software under test into a number of partitions, where the behavior of the software is equivalent for any value within a particular partition.

Boundary Value Analysis

Boundary value analysis uses the same analysis of partitions as equivalence partitioning. However, boundary value analysis assumes that errors are most likely to exist at the boundaries between partitions. Boundary value analysis consequently incorporates a degree of negative testing into the test design, by anticipating that errors will occur at or near the partition boundaries. Test

cases are designed to exercise the software on and at either side of boundary values.

State-Transition Testing

State transition testing is particularly useful where either the software has been designed as a state machine or the software implements a requirement that has been modeled as a state machine. Test cases are designed to test the transitions between states by creating the events, which lead to transitions.

Branch Testing

In branch testing, test cases are designed to exercise control flow branches or decision points in a unit. This is usually aimed at achieving a target level of Decision Coverage.

Condition Testing

There are ranges of test case design techniques, which fall under the general title of condition testing, all of which endeavor to mitigate the weaknesses of branch testing when complex logical conditions are encountered. The object of condition testing is to design test cases to show that the individual components of logical conditions and combinations of the individual components are correct.

Data Definition-Use Testing

Data definition-use testing design test cases to test pairs of data definitions and uses. A data definition is anywhere that the value of a data item is set, and a data use is anywhere that a data item is read or used. The objective is to create test cases, which will drive execution through paths between specific definitions and uses.

Internal Boundary Value Testing

In many cases, partitions and their boundaries can be identified from a

functional specification for a unit, as described under equivalence partitioning and boundary value analysis above. However, a unit may **also** have internal boundary values, which can only be identified from a structural specification.

5.8 CONCLUSION OF TESTING

From the above study of testing strategies it is concluded that the most suitable techniques that are suitable keeping in view the current problem are Unit testing based on white box testing branch coverage criteria, system testing will be based on black box testing and focus is on invalid and valid cases, special cases and boundary values. Also acceptance testing will be implemented. Moreover bottom up strategy is used for testing and test oracles that are used

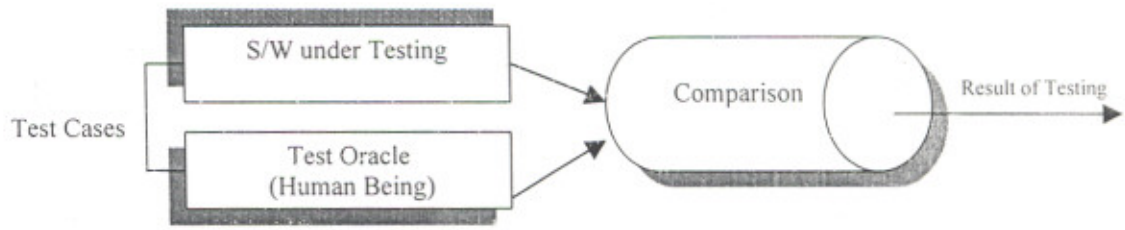


Figure 5.12 Testing & Test Oracles

in the study are human beings and we calculate by hand what should be the output of the program and check that against test result. Figure 5.12 portrays test oracles used.

Mainly the point covered in the testing process are as shown in figure 5.13.

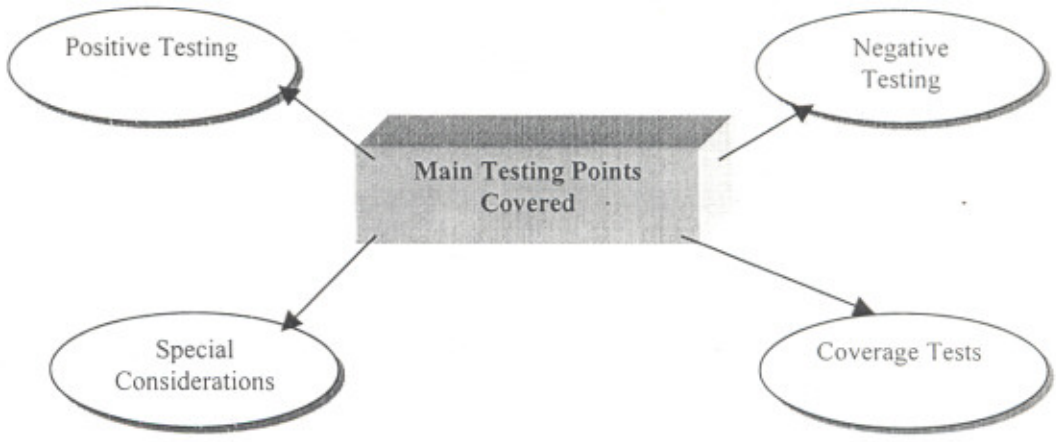


Figure 5.13 Main testing Points Covered

5.9 TESTING COVERAGE USED

Coverage measures the amount of testing done of a certain type. The various types of testing coverage's used in the system under study are:

- *Line coverage.* Test every line of code (Or *Statement coverage:* test every statement).
- *Branch coverage.* Test every line, and every branch on multi-branch lines.
- *N-length sub-path coverage.* Test every sub-path through the program of length N. for example, in a 10,000 line program, test every possible 10-line sequence of execution.
- *Path coverage.* Test every path through the program, from entry to exit. The number of paths is impossibly large to test.
- *Multicondition or predicate coverage.* Force every logical operand to take every possible value. Two different conditions within the same test may result in the same branch, and so branch coverage would only require the testing of one of them.

- *Trigger every assertion check in the program.* Use impossible data if necessary.
- *Loop coverage.* "Detect bugs that exhibit themselves only when a loop is executed more than once."
- *Every module, object, component, tool, subsystem, etc.* This seems obvious until you realize that many programs rely on off-the-shelf components. The programming staff doesn't have the source code to these components, so measuring line coverage is impossible. At a minimum (which is what is measured here), you need a list of all these components and test cases that exercise each one at least once.
- *Fuzzy decision coverage.* As the program makes heuristically-based or similarity-based decisions, and uses comparison rules or data sets that evolve over time, check every rule several times over the course of training.
- *Relational coverage.* "Checks whether the subsystem has been exercised in a way that tends to detect off-by-one errors" such as errors caused by using < instead of <=. This coverage includes:
- *Every boundary on every input variable.*
- *Every boundary on every output variable.*
- *Every boundary on every variable used in intermediate calculations.*
- *Data coverage.* At least one test case for each data item / variable / field in the program.
- *Each appearance of a variable.* Suppose that you can enter a value for X on three different data entry screens, the value of X is displayed on another two screens, and it is printed in five reports. Change X at each data entry screen and check the effect everywhere else X appears.

- *Handling of every potential data conflict.* For example, in an appointment calendaring program, what happens if the user tries to schedule two appointments at the same date and time?
- *Every opportunity for file / record / field locking.*
- *Every dependency on the locked (or unlocked) state of a file records or field.*
- *Performance of every module / task / object.* Test the performance of a module then retest it during the next cycle of testing. If the performance has changed significantly, you are either looking at the effect of a performance-significant redesign or at a symptom of a new bug.
- *Function equivalence.* For each mathematical function, check the output against a known good implementation of the function in a different program. Complete coverage involves equivalence testing of all testable functions across all possible input values.
- *Zero handling.* For each mathematical function, test when every input value, intermediate variable, or output variable is zero or near-zero. Look for severe rounding errors or divide-by-zero errors.
- *Accuracy of every graph,* across the full range of graphable values. Include values that force shifts in the scale.
- *Accuracy of every report.* Look at the correctness of every value, the formatting of every page, and the correctness of the selection of records used in each report.
- *Accuracy of every message.*
- *Accuracy of every screen.*
- *Check for every type of virus / worm that could ship with the program.*
- *Usability tests of:*
- *Every feature / function of the program.*

- *Every part of the manual.*
- *Every error message.*
- *Every on-line help topic.*
- *Every graph or report provided by the program.*
- *Every date, number and measure in the program.*

CHAPTER 6

IMPLEMENTATION

INTRODUCTION

This chapter presents the implementation of the rules, models or hypothesis that has been laid down in previous chapters, on a module of examination system. Clearly the approaches defined & studied during the study of system has been followed. According to proposed model the very first step is to completely analyze the system and laid down the strategies, next step is to decide software & hardware. In the next phase complete system designing, coding and testing is done.

6.1 COMPLETE ANALYSIS OF MODULE

6.1.1 Problem Analysis

In the current problem one unit of examination has to be computerized i.e. Graduation programme as it was discussed that examination system can be divided into three phases Pre-Examination, Examination & Post_Examination each phase has its own functions. In the next step complete system study for the module has been done.

Figure 6.1 portrays the top level data flow diagram of the system.

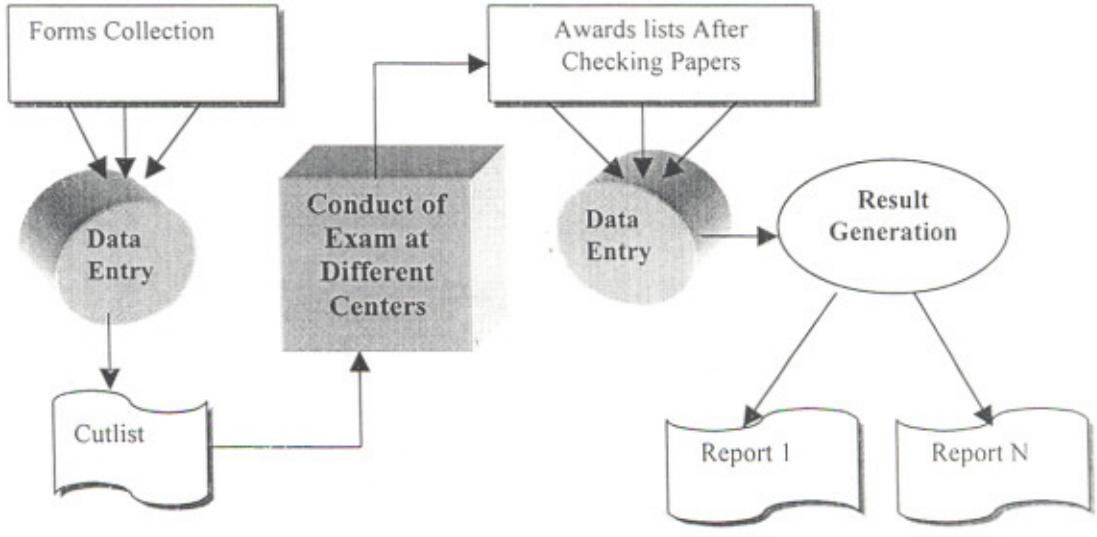


Figure 6.1 Top Level Data Flow Diagram

6.1.2 Paper Options

As revealed during the present study of current examination system after speaking with the Assistant Registrar, Superintendent and Assistants that their are about forty three subjects, out of which two subject are compulsory i.e. English and Punjabi (however in place of Punjabi non domicile of Punjab can take Punjab History and Culture) and student has to choose three optional subjects from different combinations.

Student has to choose one subject from the below given groups of subjects:

Group I

Economics	Defense Studies	Sociology
Fine Arts	Education	

Group II

History	Mathematics	Psychology
Folk Art & Culture		

Group III

Political Science	Philosophy	Statistics
Tabla	Indian Classical Dance	

Group IV

Punjabi (Literature)	Hindi(Literature)	English(Literature)
Persian	Urdu	Sanskrit

Group V

Public Administration	Religion	Home Science
Physical Education	History of Art	Music(Instrumental)

Group VI

Geography	Business and Office practice
Home Management	Rural Development
Linguistics	Music(Vocal)

6.1.3 Passing Conditions For Different Papers & Grouping

As revealed during the study it was found that their are forty-three subjects in total and most of the papers have different passing conditions. Moreover some papers are in two parts i.e. Part one as paper A and Part two as Paper B, even some papers have Part C and Part D as their Practical Papers. The detail of each paper's passing condition was noted and it was found that some papers have same passing conditions for paper A and for paper B.

So keeping in view the above same passing conditions of different papers, the subjects has been divided into different groups by keeping the same passing condition papers in a separate group.

Moreover this will help in reducing many source programs in length, as program has to be written once for same group of passing conditions in spite of writing conditions for individual papers.

Different groups and their passing conditions are as under:

GROUP I

Papers in the Group:

English Punjabi

Only One Paper is there in this group and to pass in the paper student has to obtain 35 marks.

Maximum Marks =100

Passing Marks = 35

GROUP II

Papers in the Group:

Mathematics	Statistics
Philosophy	Religion
Economics	Journalism & Mass Communication
History	Political Science

Education	Linguistics
Sanskrit	Social Work
English Literature	Punjabi Literature
Business Office & Practice	Hindi
Public Administration	History of Art
Sociology	Rural Development
French	

Paper is in Two Parts i.e. Paper A & Paper B. Both the papers are theory Papers. To Pass in the subject student has to avail 70 marks in Total and individual paper marks (i.e. Marks of Paper A and Marks of Paper B) does not matter. Student may fail in one subject but he must have 70 marks in total to pass in the subject.

Paper A (Theory)	Max Marks =100	Passing Marks = 35
Paper B (Theory)	Max Marks =100	Passing Marks = 35

GROUP III

Papers in the Group:

Russian	German
Urdu	Persian
English Communication	

Passing condition is same as in Group II but it is taken as third group due to some technical reasons.

Paper is in Two Parts i.e. Paper A & Paper B. both the papers are Theory Papers. To Pass in the subject student has to avail 70 marks in Total and individual paper marks (i.e. Marks of Paper A and Marks of Paper B) does not matter. Student may fail in one subject but he must have 70 marks in total to pass in the subject.

Paper A (Theory)	Max Marks =100	Passing Marks = 35

Paper B (Theory) Max Marks =100 Passing Marks = 35

GROUP IV

Papers in the Group:

Music Vocal Music Instrumental

Dramatic Art Dance

Tabla

In this group also Paper is in Two Parts i.e. Paper A & Paper B. But in this group Paper A is Theory Paper and Paper B is Practical Paper. To Pass in the subject student has to avail minimum 35 marks in theory and practical papers separately.

Paper A (Theory) Max Marks =100 Passing Marks = 35

Paper B (Practical) Max Marks =100 Passing Marks = 35

GROUP V

Papers in the Group:

Psychology Folk Art

In this group their are two theory Papers i.e. Paper A & Paper B and one practical paper i.e. Paper C. To Pass in the subject student has to avail minimum 49 marks in theory papers i.e. combindly in paper A & B (their is no passing condition in each paper separately) and 21 marks in practical paper. Thus a candidate is pass only when he avails 49 marks in theory papers and 21 marks in practical paper separately.

Paper A (Theory) Max Marks =70 Passing Marks =A+B=49

Paper B (Theory) Max Marks =70

Paper C (Practical) Max Marks = 60 Passing Marks = 21

GROUP VI

Papers in the Group:

Fine Art

In this group their is one theory Paper i.e. Paper A and one practical paper i.e. Paper B. To Pass in the subject student has to avail minimum 21 marks in theory paper and 49 marks in practical paper separately.

Paper A (Theory)	Max Marks =60	Passing Marks =21
Paper B (Practical)	Max Marks = 140	Passing Marks = 49

GROUP VII

Papers in the Group:

Defense Studies

In this group there are different rules for regular and private candidates. For regular candidates a practical paper is their along with two theory papers but in case of private candidates only two theory papers are their.

Regular Students

To pass in the subject regular student has to avail minimum 56 marks in theory paper and 14 marks in practical paper separately.

Paper A (Theory)	Max Marks =80	Passing Marks =A+B=56
Paper B (Theory)	Max Marks = 80	
Paper C (Practical)	Max Marks = 40	Passing Marks = 14

Private Students

In case of private students no practical paper is their and also theory papers A and B are of 80 marks But during result calculations their marks are raised proportionately from 100 and to pass in the subject private student has to avail minimum 70 marks in total of theory papers.

Paper A (Theory)	Max Marks =80(100)	Passing Marks=28(35)
Paper B (Theory)	Max Marks = 80(100)	Passing marks=28(35)

In the above case bracted marks are raised marks.

GROUP VIII

Papers in the Group:

Home Science Home Management

In this group their are two theory Papers i.e. Paper A & Paper B and one practical paper i.e. Paper C. To Pass in the subject student has to avail minimum 59 marks in theory papers i.e. combindly in paper A & B (their is no passing condition in each paper separately) and 11 marks in practical paper. Thus a candidate is pass only when he avails 59 marks in theory papers and 11 marks in practical paper separately.

Paper A (Theory)	Max Marks =85	Passing Marks =A+B=59
Paper B (Theory)	Max Marks =85	
Paper C (Practical)	Max Marks = 30	Passing Marks = 11

GROUP IX

Papers in the Group:

Physical Education

In this group there are two theory Papers i.e. Paper A & Paper B, one practical paper and internal assessment. To Pass in the subject student has to avail minimum 42 marks in theory papers i.e. combindly in paper A & B (their is no passing condition in each paper separately) and 28 marks in practical paper and internal assessment combindly. Thus a candidate is pass only when he avails 42 marks in theory papers and 28 marks in practical paper + internal assessment separately.

Paper A (Theory)	Max Marks =60	Passing Marks =A+B=42
Paper B (Theory)	Max Marks =60	
Paper C (Practical)	Max Marks = 60	
Internal Assessment	Maxmarks = 20	Passing Marks=C+Asses=28

GROUP X

Papers in the Group:

Geography

In this group there are two theory Papers i.e. Paper A & Paper B, one practical paper and internal assessment. To Pass in the subject student has to avail minimum 49 marks in theory papers i.e. combindly in paper A & B (their is no passing condition in each paper separately) and 21 marks in practical paper and internal assessment combindly. Thus a candidate is pass only when he avails 49 marks in theory papers and 21 marks in practical paper + internal assessment separately.

Paper A (Theory)	Max Marks =70	Passing Marks =A+B=49
Paper B (Theory)	Max Marks =70	
Paper C (Practical)	Max Marks = 40	
Internal Assessment	Maxmarks = 20	Passing Marks=C+Asses=21

6.1.4 Coding Scheme:

It is clear that many fields at the time of data input have to coded to avoid any type of spelling mistake and moreover it will ease the work of data entry purposes as code is a concise representation and it will reduce the data entry time. For example in spite of entering full name of subject English just code 1 is entered. Some other codes are also used in the system to identify the person, examination etc.

For example student is uniquely represented by its Roll Number or Registration Number as their may be more than one student having the same name and same father's name and we have to look for a particular name in that case and thus assigning unique Rollnumber and registration numbers to them will remove this type of ambiguity. Thus each coded thing has unique

identification. However there are another technical reasons to code the subjects in a particular order as specified:

Code	Paper	Code	Paper
1	English	23	Music Vocal
2	Punjabi	24	Music Instrumental
3	Math	25	Geography
4	Statistics	26	Dance
5	Philosophy	27	Tabla
6	Religion	28	Public Administration
7	Economics	29	Folk Art
8	Dramatic Art	30	Fine Art
9	Psychology	31	History of Art
10	Journa & Mass Comm	32	Sociology
11	History	33	Home Management
12	Political Science	34	Home Science
13	Defense Studies	35	Punjab History &Culture
14	Education	36	Rural Development
15	Physical Education	37	French
16	Linguistic	38	Russian
17	Sanskrit	39	German
18	Social Work	40	Urdu
19	English Literature	41	Persian
20	Punjabi Literature		
21	Business Office & Pract		
22	Hindi		

Technical Reason for Coding the Subjects:

- As it was discussed in the grace rules section that University has the rule of giving grace marks (1% of total marks) to students in that case only if the student is going to pass after getting the grace marks or his result is changed to Re-appear after the grace.

  If the following type of situation occurs:

- Students is failed in two papers
- Grace needed to pass in individual paper is same
- Total Grace needed exceeds the maximum grace

Computer is restricted to give grace to one paper only

In the above situation it will be difficult to decide to which paper to allot grace marks as a similarity should be their for all such type of cases. So in this case the code number of paper is checked and grace is allotted to paper having smaller code number, as the papers has been coded according to rules in sequence to tackle such type of situation.

- The second technical reason for coding is that at the time of combining the files during execution phase the papers should be aligned in a fixed format in the main database file.
- Another reason is that any subject can be filtered by simply referring its code in spite of its name that may contain typing mistakes.

6.2 SYSTEM DESIGN OF MODULE

6.2.1 Structured Design

As the structured design approach has been used, the very first step of structured design method is data flow diagram. As it was discussed in the design strategies that examination system is divided into three parts, Before Examination phase, examination phase & after exam phase.

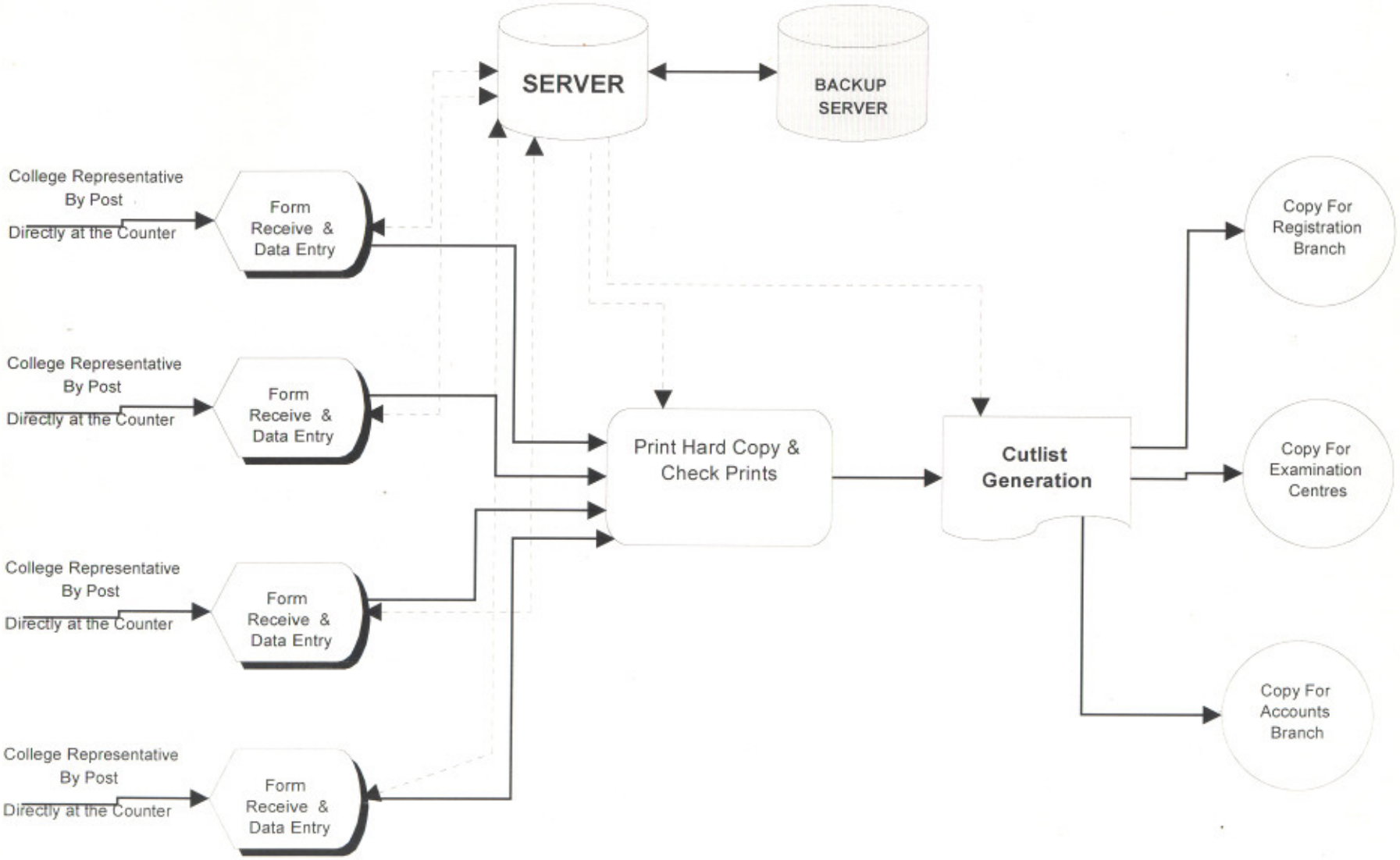


Figure 6.2 DATA FLOW DIAGRAM BEFORE EXAM FOR COMPUTERIZED SYSTEM

6.2.2 Data Flow Diagram

From the data flow diagram of before examination (fig 6.2) it is clear that the data has to be entered after sorting the forms received from many resources and cutlist is generated and sent to centers for conduct of exam.

In the examination phase (fig 6.3) exams are conducted and from there information about the absent and present students in each paper is obtained.

In the after examination phase (fig 6.4) award entry is done and then files are combined to produce the final result after considering information from various resources and final reports are generated.

6.2.3 Data Definitions:

The major data structure or fields used in data flow diagram and the structure chart are specified here (hence called data dictionary):

Table Name

STU_CUT

Data Elem Description	Abbreviation	Picture
Roll of Candidate	ROLLNUM	Numeric (6)
Prev Sess Rollnum	PROLL	Numeric (6)
Registration Number	REGNUM	Character (16)
Session	SESS	Character (4)
Reappear or Full	REAP	Numeric (1)
If Reap Paper Code	REPC	Character (8)
Sex of Candidate	SEX	Character (1)
Exam Center	CENTRE	Numeric (2)
Name of Candidate	NAME1	Character (25)
Father's Name	FNAME1	Character (25)
	CCP1	Numeric (2)
	CCP2	Numeric (2)

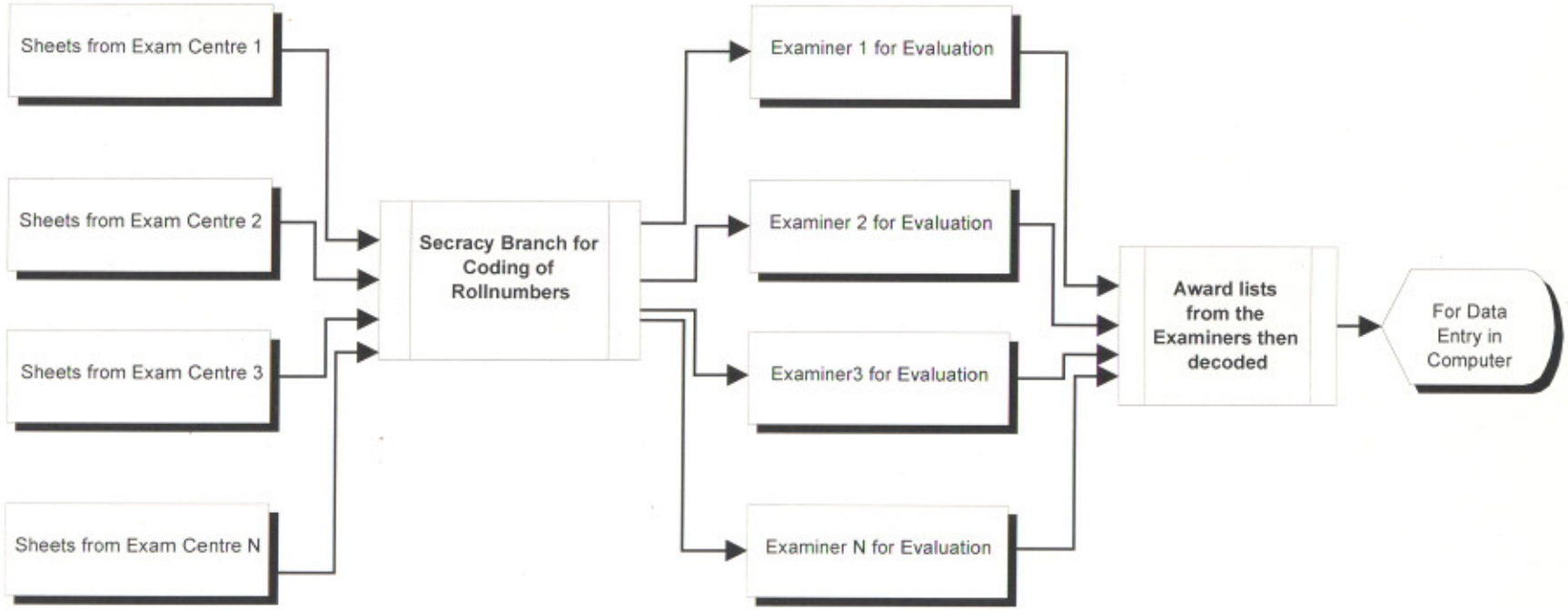
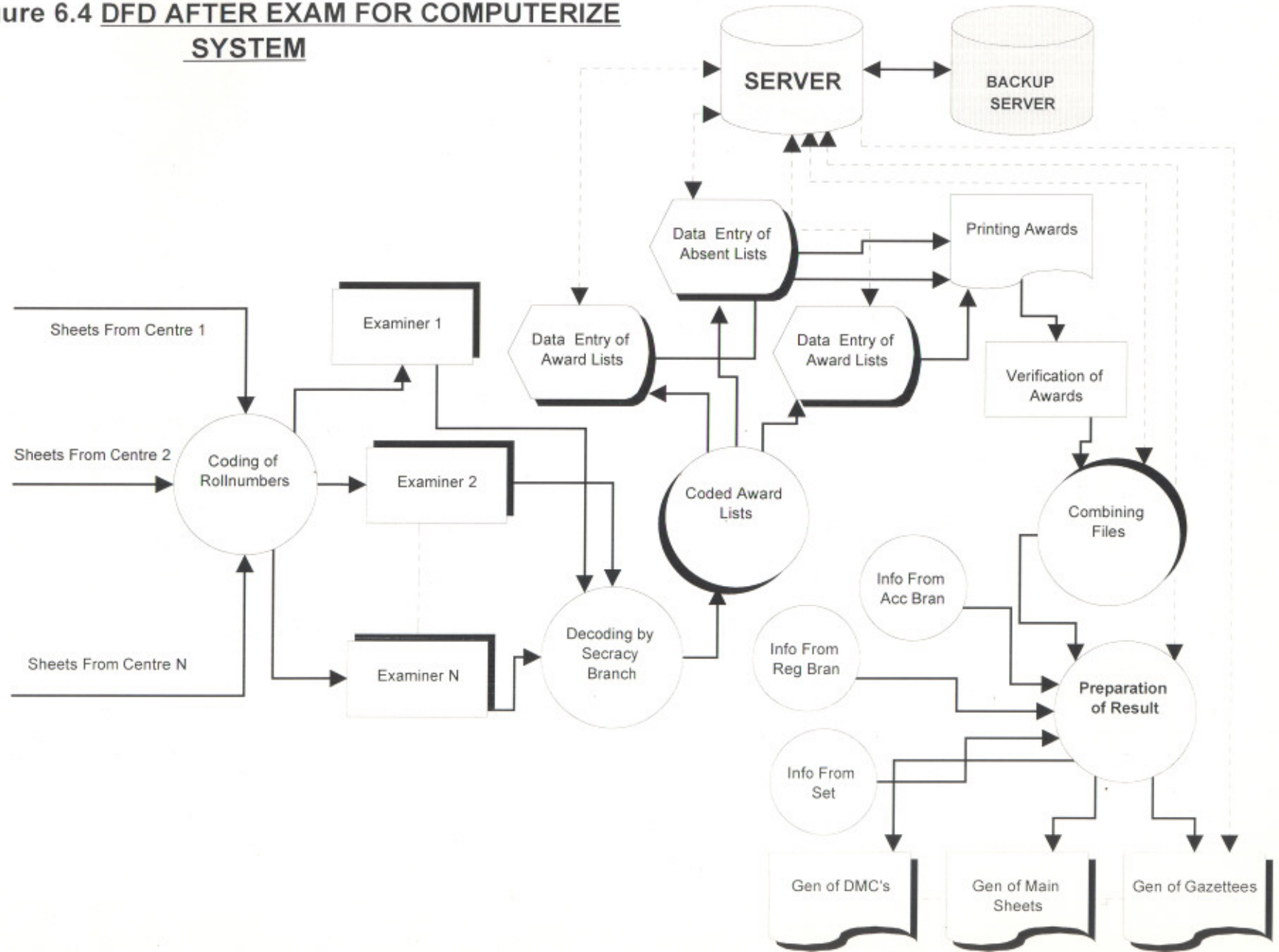


Figure 6.3 INOFRMATION FLOW DURING EXAMINATION PHASE

Figure 6.4 DFD AFTER EXAM FOR COMPUTERIZE SYSTEM



Option 1	CCP3	Numeric (2)
Option 2	CCP4	Numeric (2)
Option 3	CCP5	Numeric (2)
Remarks if Any	REMARKS	Character (3)

Table Name

STU_MAIN

Data Elem Description	**Abbreviation**	**Picture**
Roll number of Candidate	ROLLNUM	Numeric (6)
Reg Number of Cand.	REGNUM	Character (16)
Grace Left	GLEFT	Numeric (1)
Grace Availed	GAVAIL	Numeric (1)
Category(Giani, Prabh)	CAT	Numeric (1)
Last Chance (Y/N)	CHANCE	Logical
Paper 1 Description	P1	Character (8)
Paper 1 Code	CP1	Numeric (2)
Paper 1 Marks	MP1	Numeric (3)
Paper 1 Grace Avail	GP1	Numeric (1)
Paper 1 Total	TOTP1	Numeric (3)
Paper 2 Description	P2	Character (8)
Paper 2 Code	CP2	Numeric (2)
Paper 2 Marks	MP2	Numeric (3)
Paper 2 Grace Avail	GP2	Numeric (1)
Paper 2 Total	TOTP2	Numeric (3)
Paper 3 Description	P3	Character (8)
Paper 2 Code	CP3	Numeric (2)
Paper 3 A Marks	MP3A	Numeric (3)
Paper 3 B Marks	MP3B	Numeric (3)

Paper 3 Total (A+B)	TP3	Numeric (3)
Paper 3 Grace Avail	GP3	Numeric (1)
Paper 3 Marks Pract	PP3	Numeric (3)
Paper 3 Total Pract	TPP3	Numeric (3)
Paper 3 Grace Pract	GPP3	Numeric (1)
Paper 3 Total	TOTP3	Numeric (3)
Paper 4 Description	P4	Character (8)
Paper 4 Paper Code	CP4	Numeric (2)
Paper 4 A Marks	MP4A	Numeric (3)
Paper 4 B Marks	MP4B	Numeric (3)
Paper 4 Total (A+B)	TP4	Numeric (3)
Paper 4 Grace Avail	GP4	Numeric (1)
Paper 4 Marks Pract	PP4	Numeric (3)
Paper 4 Marks Inter	PP4I	Numeric (3)
Paper 4 Total Pract	TPP4	Numeric (3)
Paper 4 Grace Pract	GPP4	Numeric (1)
Paper 4 Total	TOTP4	Numeric (3)
Paper 5 Description	P5	Character (8)
Paper 5 Code	CP5	Numeric (2)
Paper 5 A Marks	MP5A	Numeric (3)
Paper 5 B Marks	MP5B	Numeric (3)
Paper 5 Total (A+B)	TP5	Numeric (3)
Paper 5 Grace Avail	GP5	Numeric (1)
Paper 5 Marks Pract	PP5	Numeric (3)
Paper 5 Marks Int	PP5I	Numeric (3)
Paper 5 Total Pract	TPP5	Numeric (3)
Paper 5 Grace Pract	GPP5	Numeric (1)

Paper 5 Total	TOTP5	Numeric (3)
Grand Total	GTOTAL	Numeric (3)
Result (P, F, R)	RESULT	Character (8)
Reapp Paper Code	R	Numeric (2)
DMC Number	DMCNO	Numeric (6)
Remarks if Any	REM1	Character (3)
Remarks if Any	REM2	Character (3)
Remarks if Any	REM2	Character (3)

*** Fields of STU_CUT table are also used in the main table

Table Name

STU_AWD

Data Elem Description	**Abbreviation**	**Picture**
Rollnumber of Cand	ROLLNUM	Numeric (6)
Paper Code	CP1	Numeric (2)
Weather A, B, P	A_B_P	Character(1)
Paper Description	P1	Character (8)
Marks Obtained	MARKS	Numeric (3)

Table Name

CEN_CODE

Data Elem Description	**Abbreviation**	**Picture**
Center Code	CENTRE	Numeric (2)
Center Name	CEN	Character (60)

Table Name

P3_CODE

Data Elem Description	**Abbreviation**	**Picture**
Paper Code	CP3	Numeric (2)

Paper Name	P3	Character (8)

Table Name

P4_CODE

Data Elem Description	**Abbreviation**	**Picture**
Paper Code	CP4	Numeric (2)
Paper Name	P4	Character (8)

Table Name

P5_CODE

Data Elem Description	**Abbreviation**	**Picture**
Paper Code	CP5	Numeric (2)
Paper Name	P5	Character (8)

Table Name

Data Elem Description	**Abbreviation**	**Picture**
REM_CODE	ROLLNUM	Numeric (6)
Remarks if Any	REM	Character (4)

TP3=MP3A+MP3B+GP3

TPP3=PP3+GPP3

TOTP4=TP3+TPP3

TP4=MP4A+MP4B+GP4

TPP4=PP4+PP4I+GPP4

TOTP4=TP4+TPP4

TP5=MP5A+MP5B+GP5

TPP5=PP5+PP5I+GPP5

TOTP5=TP5+TPP5

GTOTAL=TOTP1+TOTP2+TOTP3+TOTP4+TOTP5

6.2.4 Decision Tables

Decision tables are made as under for most crucial decisions that we have to take in the designing process of the system.

Decision Table for Overall Passing Conditions

Condition Stub	Condition Entry					
If pass in <= three Papers	Y	Y	Y			
If pass in four Papers				Y	Y	
If pass in five Papers						Y
If Giani Candidate	Y					
If Prabhakar Candidate			Y	Y		
PASSED	X			X		X
FAILED		X				
REAPPEAR			X		X	
Action Stub	**Action Entry**					

Decision Table For Passing Conditions in Each Paper

Here groups are as defined above in the system study. A group means paper that refers to that particular group.

Condition Stub	Condition Entry									
Case Papers of Group1	Y									
Case Papers of Group 2		Y								
Case Papers of Group 3			Y							
Case Papers of Group 4				Y						
Case Papers of Group 5					Y					
Case Papers of Group 6						Y				
Case Papers of Group 7							Y			
Case Papers of Group 8								Y		
Case Papers of Group 9									Y	
Case Papers of Group 10										Y
Marks A>=35	X									
Marks A+B>=70		X								
Marks A+B>=70			X							
Marks A >=35 B>=35 separately				X						
Marks A +B>=49 C>=21 separ					X					
Marks A >=21 B>=49 separately						X				
Marks A +B>=56 C>=14 separ							X			
Marks A +B>=59 C>=11 separ								X		
Marks A+B>=42 C+asses>=28 separ									X	
Marks A+B>=49 C+D>=21 separ										X

Action Stub **Action Entry**

6.2.5 Module Specification:

Main_Module:

Inputs : File Names

Outputs : None

Subordinates : Ent_Menu

Mod_Menu

Proces_M

Print_Me

Ppost_Me

Ent_Menu

Inputs : Student Forms, Awards, Reports

Ouputs : Cutlist, Reports

Subordinates : En_Cut, En_Reports, En_Award, En_Pract

Purpose : Validate the database & entering student records to database. Entering reports created by clerical staff.

Ppost_Me :

Inputs : Previous Year Database

Ouputs : Ppost_File

Subordinates : Ppost_Sh, Ppost_Ex

Purpose : Consider the previous year database for reappear students and extracting the previous records of students for further consideration.

Process_M :

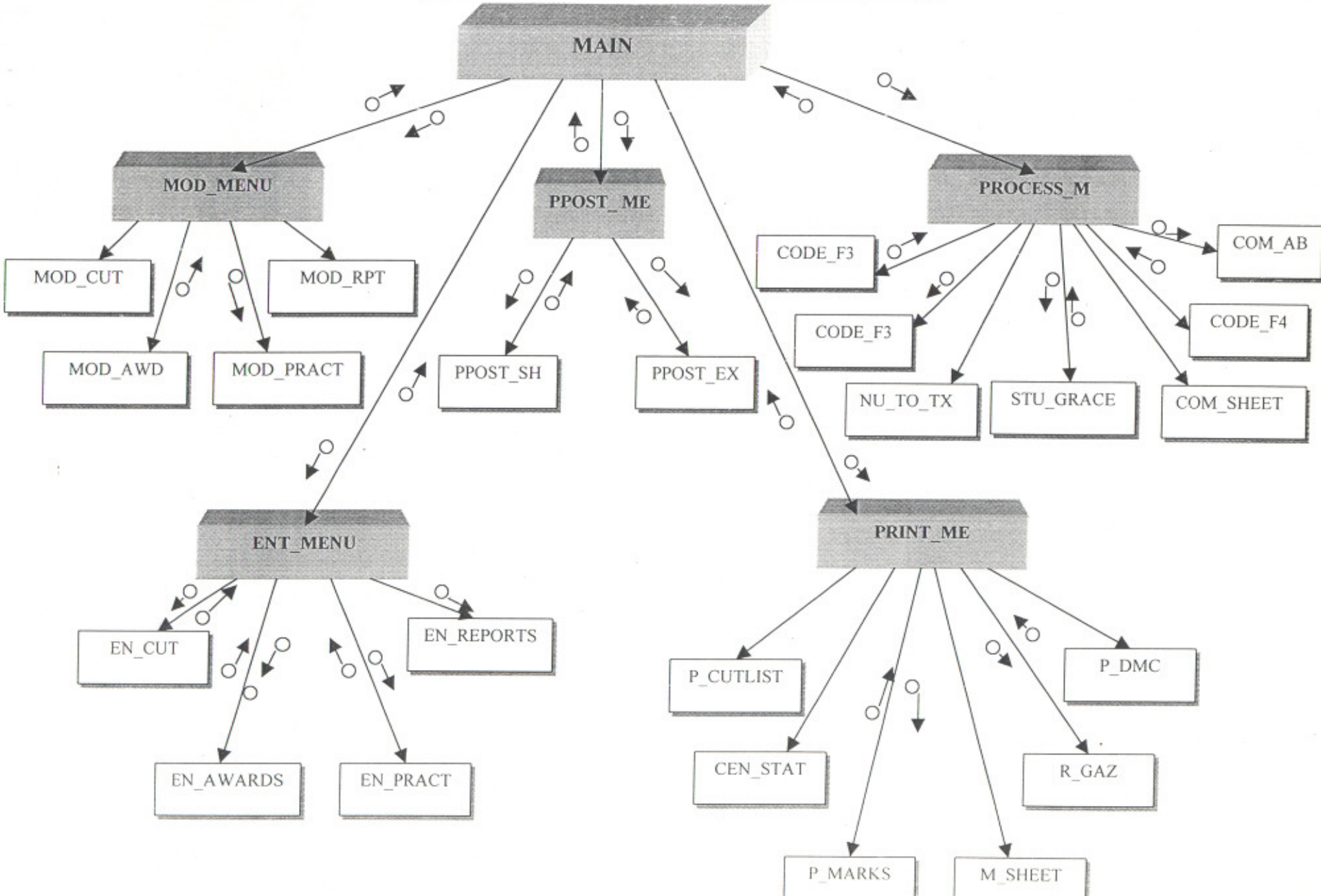
Inputs : Database of Students, Awards, Reports

Ouputs : Calculation of Complete Result

Subordinates : Dup_rec, Com_AB, Com_Sheet, Com_Main Code_F3, Code_F4, Code_F5, Stu_Grace, Nu_to_tx

Purpose : To combine different tables to corresponding tables and then to main database and finally generating the complete result by assigning proper grace according to rules.

Figure 6.5 STRUCTURE CHART FOR THE SYSTEM



Print_Me :

Inputs : Database Records, Paper Awards, Reports

Ouputs : Cutlist, Centre Statement, Award Proofs, Report Proofs
Main Sheet, Result Gazette, Result Cards, Pass Percentage Sheet
Press Statement.

Subordinates : P_Cutltist, Cen_Stat, P_Marks, M_Sheet, R_Gaz , P_DMC

Purpose : To Print cutlist, generate centre statement, printing proof for awards & reports for proof reading and finally printing gazette, main sheets and detail marks cards of students.

Mod_Menu :

Inputs : Database

Ouputs : None

Subordinates : Mod_Cut, Mod_Rpt, Mod_Awd

Purpose : Validate the database by making changes to wrong entries in the Database.

Figure 6.5 portrays structure chart of the system. Detailed logic of one of major module of calculations is described in section 6.6 by using the progress design language that is used to specify complete design system as well as logic design up to maximum degree of details.

6.2.6 Design Decisions

- Their are five papers for General category from which two are compulsory and three are optional i.e. candidates has to choose from a list of about forty papers.
- For candidates of category GYANI "CAT" field of database is set to one(1) and in this case candidate has to appear & pass in three(3) papers.

- For candidates of category PRABHAKER "CAT" field of database is set to one(2) and in this case candidate has to appear & pass in four(4) papers.
- If field REAP in data base file is set to one then that candidate is considered as Re-appear candidate by all the programs of the package.
- To pass in the examination candidate has to Pass in all the required number of papers and if he is fail in more than one paper then his result is Fail and if he is fail in one paper his result is Re-appear.
- To pass in the examination candidate has to obtain 35% marks in each paper.
- For Re-appear candidates if it is the last chance to appear in the examination (by checking "CHANCE" field of database) and candidate is fail in that paper then his result is Fail otherwise if it is not last chance then result is Re-appear.
- Before the main calculations of result a new file GRACE is created with one Field as G : Numeric(1) and file must be indexed on field "G"
- Paper marks is set to -3 if candidate is Absent in particular paper.
- In case of Award wanting the corresponding award of paper are set to -2 and for zero marks corresponding field is set to -1.
- For paper code=13 that is Defense study paper Practical award of private & correspondence students should be set to -9 and awards should be raised from out of 80 to 100.
- Before executing the main program all fields like R, result DMCNO etc should be blank or zero.
- Certain abbreviations are also used as

C.C. Candidature Cancelled

N.E.	Not Eligible
RLF	Result late due to fee dispute
RLE	Result Late due to Eligibility
RLR	Result late due to Registration

In all above cases what ever may be the reason the result in gazette is set to C.C.

6.3 CODING TECHNIQUES USED

The programs are written in Dbase on Pentium based system the size of code is about 6000 lines and there are 30 modules in the code and clearly the bottom up technique structured approach has been followed for coding.

The complete logic of one of important module of calculations is described in section 6.6 to show the procedure used in coding.

6.4 TESTING STRATEGIES APPLIED

6.4.1 Test Units

In the module we have done three levels of testing- Unit Testing, System testing & Acceptance testing as system is small so no need of integration testing is their. Main stress of unit testing is on testing modules stu_grace, cen_stat, m_sheet, P_dmc and R_gaz. Testing for above units is done independently. Other modules are also tested at same level.

6.4.2 Testing Approach

Unit testing is done based on branch coverage criteria of white Box testing techniques. System testing is done based on Black box techniques and main stress points are boundary values, special cases and positive & negative testing.

6.4.3 Unit Test Report

Report for only one unit has been presented, the goal of unit testing is to cover 100% branch coverage. The unit chosen here is most important unit

used for final calculations (Stu_Grace).

The testcases used for testing are kept in temporary files and is called by using the above program. The module Process_M is used for calculating total result of students after giving suitable grace according to rules.

Test cases are generated until each branch is covered and four different input files were used for testing and 100% branch coverage is achieved. Errors detected during unit testing are corrected.

6.4.4 Test case Specification

Main testcases that were generated for the purpose of testing are based on the below given conditions:

- Upper & lower limit of Rollnumber, Proll
- Proper functioning of calling programs so that they call respective programs only
- Check for total gace
- Check for grace availed
- Check for grace left
- Complete checking for Group I Papers for their each passing condition i.e. one paper only >=35 for pass
- Complete checking for Group II Papers for their each passing condition i.e. Two papers but total >=70 for pass
- Complete checking for Group III Papers for their each passing condition i.e. Two papers but total >=70 for pass
- Complete checking for Group IV Papers for their each passing condition i.e. Two papers but for A >=35, B>=35 for pass separately
- Complete checking for Group V Papers for their each passing condition i.e. Three papers but for A +B>=49, C>=21 for pass separately

- Complete checking for Group VI Papers for their each passing condition i.e. Three papers but for A >=21, B>=49 for pass separately
- Complete checking for Group VII Papers for their each passing condition i.e. Three papers but for A +B>=58, C>=14 for pass separately
- Complete checking for Group VIII Papers for their each passing condition i.e. Three papers but for A +B>=59, C>=11 for pass separately
- Complete checking for Group IX Papers for their each passing condition i.e. Three papers+Asses but for A +B>=42, C+Asses >=28 for pass separately
- Complete checking for Group VIII Papers for their each passing condition i.e. Four papers but for A +B>=59, C+D>=21 for pass separately
- Check the passing reappear and fail conditions i.e. if pass in 1 paper, if pass in 2 papers, if pass in 3 papers, if pass in 4 papers, if pass in 5 papers,
- Special check for Giani Category results
- Special check for prabhakar Category results
- Special check for private students opting Defense Studies.

In addition to above test cases were generated for many other testing points. Acceptance Testing is done using the actual data of previous year and is found correct.

6.5 DESIGN OF MODULE PROCESS_M (PSEUDOCODE)

PROC Calculate_grace

select MASTER Table in First Area

Select Temporary grace File in Second Area

Do while (End of Master File Comes)

Initialize Local Variables here

For Group ix & Group x var

ggp5i=0,ggp4i=0

pass=0, fail=0

Variables for Total of papers without grace

t1=0, t2=0, t3=0, t4=0, t5=0

Variable for grace requirement evaluated in papers..

gg1=0, gg2=0, gg3=0, gg4=0, gg5=0

Variable for grace requirement evaluated in practicals.

ggp3=0, ggp4=0, ggp5=0

Field of Temporary File

g=0

gg=0

#####Contains Grace left in Main file

gr=0

*****m1 show Paper-1 theory,Practicals total with grace

m1=0

*****m2 show Paper-2 theory,Practicals total with grace

m2=0

m3=0 & m4=0 show Paper-3 theory,Practicals total with grace

m5=0 & m6=0 show Paper-4 theory,Practicals total with grace

m7=0 & m8=0 show Paper-5 theory,Practicals total with grace

Grace given as per rules in Papers, Practicals

g1=0 Shows Grace given as per rules in Paper-1.

g2=0 Shows Grace given as per rules in Paper-2.

Grace given as per rules in all other Papers-& its practicals.

g3=0, g4=0, g5=0, g6=0, g7=0, g8=0

HERE CALCULATING GRACE IF REQUIRED IN EACH PAPER

FOR PAPER I II & GROUP I

<Papers in this group are pass if total =35

gg1=Calculate Grace Given according to rule

FOR (first optional) PAPER 3

While {all the groups are checked for grace}

Case For Group II

<Papers of this case is passed,if total marks of paper 'A' and 'B' is greater than or equal to 70>

Calculate t3

Calculate Grace Given gg3=70-t3, ggp3=0

Case For Group III

<Papers of this case is passed,if marks of paper & Prac both should be minimum 35 separately>

Calculate t3 according to rule

Calculate gg3=35-t3, ggp3=35-pp3

Case For Group IV

Calculate t3

Calculate gg3=21-t3, ggp3=49-pp3 according to formula

Case For Group V

<Papers of this case is passed,if marks of theory papers & Practical should be minimum 49 and 21 respectively>

Calculate t3 according to condition

Calculate gg3=49-t3, ggp3=21-pp3 as specified

Case For Group VI

calculate t3 according to condition

<Passing condition i.e. if practical not taken (pp3=-9) than total of A & B paper should be minimum 70 and if practical taken (pp3<>-9) than total of paper 'A' & 'B' should be minimum 56 and practical should be minimum 14.>

Check Condition For Practical

If (practical Not taken)

calculate gg3=70-t3, ggp3=0

else

Calculate gg3=56-t3

ggp3=14-pp3

end of if

Case For Group VII

<Passing condition i.e.total of paper 'A' & 'B' should be minimum 60 and practical should be minimum 11>

calculate t3 according to rule

Calculate gg3=59-t3, ggp3=11-pp3

Case For Group VIII

<Passing condition i.e.total of paper 'A' & 'B' should be minimum 61 and practical should be minimum 9>

calculate t3 according to rule

Calculate gg3=61-t3, ggp3=9-pp3

End of Group Condition

End of while condition

FOR (Optional Paper II) PAPER 4

Here In this paper Passing conditions are same as in case of Group of PAPER-3, the only change is to Introduce Two More Groups.

While {all the groups are checked for grace}

{Calculate t4, gg4 & ggp4 according to rules as defined in Paper III for groups I to VII Except for Groups IX & X}

case For Group IX

<Passing Condition i.e. total of 'A' & 'B' papers should be minimum 42 and Practical+Int. Assessment should be minimum 28 marks.>

Calculate t4 according to rule

calculate gg4=52-t4, ggp4=18-pp4 as defined

Case For Group X

<Passing Condition i.e. total of 'A' & 'B' papers should be minimum 49 and Practical+Int. Assessment should be minimum 21 marks>

calculate t4 by checking Condition

calculate gg4=49-t4, ggp4=21-ggp4i

End of Group Condition

End of While Condition

FOR (Optional Paper III) PAPER 5

Passing conditions are same as in case of PAPER-3, for paper code 15 and

25 of Phy. Edu. and Geography respectively, the passing condition are same as defined in Paper-4.

While {all the groups are checked for grace}

Start of Group Loop

{Calculate t5, gg5 & ggp5 according to rules as defined in Paper III for groups I to VII Except for Groups IX & X}

End of Group Loop

End of While Loop

****

GIVING MINIMUM GRACE ACCORDING TO RULES & BALANCE GRACE

****

Select The Temp File In Select Area

Add the Values of gg1,gg2,gg3, ggp3, ggp4,gg5,ggp5 to this file

<Initialize the file here>

gr=Grace Left

sum to g11 {all the values of temp file}

<Initialize the file here>

TEMP FILE LOOP STARTS HERE

while {end of file Comes}

if g11=0 < means no grace needed>

Exit from loop

end if

Do { loop to check different conditions}

case grace needed (g) > 0 .and. grace left (gr) >= g

m=Record Number <Represent Paper Number>

Do { loop to check different parts of different Papers}

Condition m=1 <means paper 1)

Add grace to m1

Assign to grace of paper 1 g1=g

Initialize grace_left=grace_left - grace_given

Condition m=2 <means paper 2)

Add grace to m2

Assign to grace of paper 1 g2=g

Initialize grace_left=grace_left - grace_given

Condition m=3 <means paper 3 Theory)

Add grace to m3

Assign to grace of paper 1 g3=g

Initialize grace_left=grace_left - grace_given

Condition m=4 <means paper 3 Practical)

Add grace to m4 after checking condition of (-9)

Assign to grace of paper 1 g4=g

Initialize grace_left=grace_left - grace_given

Condition m=5 <means paper 4 Theory)

Add grace to m5

Assign to grace of paper 1 g5=g

Initialize grace_left=grace_left - grace_given

Condition m=6 <means paper 4 Practical)

Add grace to m6 with internal assesment condition

Assign to grace of paper 1 g6=g

Initialize grace_left=grace_left - grace_given

Condition m=7 <means paper 5 Theory)

Add grace to m7

Assign to grace of paper 1 g7=g

Initialize grace_left=grace_left - grace_given

Condition m=8 <means paper 5 practical)

Add grace to m8 with internal assessment condition

Assign to grace of paper 1 g8=g

Initialize grace_left=grace_left - grace_given

End of Do Loop

Case grace not needed i.e. g<=0 .or. gr<g

m=recno()

Do { loop to allocate values depending upon value of m}

for m=1 m1=mp1

for m=2 m2= mp2

for m=3 m3=t3

for m=4 m4=pp3 (or m4=0) after check

for m=5 m5=t4

for m=6 m6=gg4pi (m6=0)(m6=pp4)

for m=7 m7=t5

for m=8 m8= ggp5i (m8=0) (m8=pp5)

End of Do Loop

End of Do Loop

END TEMP FILE LOOP

Function <Delete all the records of Temporary file >

****

* CALCULATING RESULT *

****

<Select the master File>

Check for Paper I ,II Group I

Condition (m1 or m2>=35)

Increment pass Variable

else

fail =1 or 2

End Condition

Check For (Optional I) PAPER 3

Do {condition for different cases)

Case For Group II

Condition (m3> =70)

Increment pass Variable

else

fail = 3

End Condition

Case For Group III

Condition (m3> =35 AND m4>=35)

Increment pass Variable

else

fail = 3

End Condition

Case For Group IV

Condition (m3> =21 AND m4>=49)

Increment pass Variable

else

fail = 3

End Condition

Case For Group V

Condition (m3> =49 AND m4>=21)

Increment pass Variable

else

fail = 3

End Condition

Case For Group VI

Condition (m3> =59 AND m4>=11)

Increment pass Variable

else

fail = 3

End Condition

Case For Group VII

Condition (m3> =56 AND m4>=14) and practical option

Increment pass Variable

else

fail = 3

End Condition

Case For Group VIII

Condition (m3> =61 AND m4>=9)

Increment pass Variable

else

fail = 3

End Condition

End case Condition

Check For (Optional II) PAPER 4

Do {Check for Different Groups for Paper 4}

<All the groups are checked for variable m5 & m6 for different Group Conditions as in Paper 3 and fail and pass variable are calculated except two new Groups>

Condition For Different groups from I to X

Condition (m5>=49 and m6>=21)

Increment pass Variable

else

fail=4

End Condition

End Condition

Check For (Optional III) PAPER 5

Do {Check for Different Groups for Paper 5}

<All the groups are checked for variable m7 & m8 for different Group Conditions as in Paper 4 and fail and pass variable are calculated except two new Groups>

Condition For Different groups from I to X

Condition (m7>=49 and m8>=21)

Increment pass Variable

else

fail=5

End Condition

End Condition

****

* **UPDATING RESULT IN MASTER FILE***

****

Do {check for different passing conditions}

<For Gyani student only 3 papers are required to pass, For Prabhakar (cat=2) student only 4 paper are required to pass & for other (cat=0) 5 paper are required to pass.>

****

Condition {Check for Different Pass Conditions of different categories}

<Update the different fields with corresponding variable >

Variables to be used are

gleft-gr, gr,

m1, m2, m3, m4, m5, m6, m7, m8

g1, g2, g3, g4, g5, g6, g7, g8

Update result with 'PASS'

Update Grand Total field

****

Condition {Check for different Reappear Conditions}

repl gavail with gleft-gr

```
repl gleft with gr
<Update  the different fields with corresponding  variable >
Variables to be used are
gleft-gr, gr,
m1, m2, m3, m4, m5, m6, m7, m8
g1, g2, g3, g4, g5, g6, g7, g8
Do { Condition Check for Reappear Paper}
Condition Fail=1
Update  result with 'Paper I (p1)'
Condition Fail=2
Update  result with 'Paper II (p2)'
Condition Fail=3
Update  result with 'Paper III (p3)'
Condition Fail=4
Update  result with 'Paper IV(p4)'
Condition Fail=5
Update  result with 'Paper V (p5)'
End do Condition
**************************************************
Condition {Check for different Fail Conditions for different Categories)
<Update  the different fields with corresponding  variable >
Variables to be used are
gleft-gr, gr,
m1, m2, m3, m4, m5, m6, m7, m8
g1, g2, g3, g4, g5, g6, g7, g8
Update  result with 'FAIL'
End do Condition
```

```
Condition {if all papers are absent}
repl result with "ABSENT"
End Condition

Condition { if any of paper is Award Wanting Then}
repl result with "AWD"
End Condition
End Condition of Master File LOOP
END LOOP MASTER FILE
END OF PROCEDURE
```

CONCLUSIONS

CONCLUSIONS

The goals of Software Engineering is to focus on the development of software that supports the needs of the user and helps the maintainer to adapt the software so that it will continue to support the evolving needs of the user. While designing the information system the main Goals that are emphasized are:

i) Clarity

Clarity is a measure of software's ability to be understood by those who maintain it. That is software or information system that we design must be easily understood by the future users of the system.

ii) Extensibility & Maintability

Extensibility is a measure of how easily software can have its scope modified or expanded with minimal (or no) effect on the software's existing external interface. That is software must be easily extendable when there is a need to do so.

iii) Efficiency

Efficiency is a measure of software's ability to perform its operations in an expedient manner i.e. software should be correct and produce good and correct results.

iv) Reliability & Correctness

Reliability is a measure of software's ability to respond logically to unexpected conditions and to respond predictably under normal conditions.

Correctness means that software should respond correctly and produce good and error free results.

v) Portability & Compatibility

The software should be easily installable on any type of computer and should be compatible to all types or brands of computers.

The main concern of software engineering is to design the information system keeping in view the above mentioned aspects. To achieve these goals, the main principle that are to be kept in mind are:

a) Abstraction

Abstraction is how software engineers represent or model the problem domain in code. Abstraction is the process where by we identify the important aspects of a phenomenon and ignore its details. I.e. after analyzing the requirement of the system we build a model of the proposed application in any language ignoring its details.

b) Modularity

Modularity is how software engineers decompose the problem domain into manageable chunks. The portion of a solution that addresses a chunk of the problem domain is called a module. So the system can be divided into simpler pieces called modules.

c) Encapsulation

Encapsulation is how software engineers hide implementation specific details to prevent unnecessary dependencies between modules i.e. the program hides the various details to avoid the unnecessary complexity.

d) Completeness

Completeness is how software engineers insure modules will support both portability and reusability. I.e. the software can be easily installed on all type of systems and we can reuse that software after making necessary changes.

e) Testability

Testability is how software engineers design modules to support standalone testing of the components. I.e. how various and what techniques can be applied on the information system.

f) Localization

Localization is how software engineers select the contents of modules to insure the closeness of related elements.

Our study on the Software Engineering approach to information system development in the university provided the following advantages:

- Complete software requirement specifications (SRS) were fixed in the beginning of the analysis phase of the information system development. It did not create any confusion either in the beginning, or during the process or at the end of the project. Otherwise common problem of communication gap between developer and client is overcome because of the analysis tools used in this phase.
- Keeping in mind the requirements of the system complete system was designed which proved to be accurate and complete. Only a few changes were required to be made in the design of the system. Also the system has been developed using structured design approach which is modular in nature . It can be easily upgraded as per the future requirements. Software engineering tools like structured chart, E-R

diagrams, decision tables pseudocode etc. helped in clearly designing the system.

- Structured programming using bottom up technique used in coding has enhanced the reliability and clarity of the programs to a greater extent. This has made the code easy to understand by any programmer.
- Various type of testing strategies have been used in system development using software engineering approach which has helped in analyzing the correctness of the output of the system. Moreover the system has been tested in steps keeping in view that each line of code be tested properly. This has increased the reliability of the system.
- Using software engineering concepts implementation of the system has become easy and trouble free. Moreover, the system has already been tested from all aspects using dummy data.

Thus on the basis of above advantages it is recommended that while developing software in the University, only the software engineering approach should be applied because the approach ensures software which is simple, user friendly, easy to maintain, easily reusable, easily portable, and reliable.

Further for following the software engineering approach in software development the following points are suggested to be considered in software development.

i) Sound Project Management

Project management is an ongoing effort of monitoring the progress and anticipating the needs of a project. Anticipation of conflicts and the

development of contingencies are crucial to keeping the project moving toward a successful completion. Hence for developing a software, sound project management principles must be implemented.

ii) Architecture Centric

Architecture centric means that the project is built on an application architecture that is developed early in the project. The application architecture is typically an area of uncertainty and risk. By addressing architecture early, prototyping the options, and identifying a known, working, technical solution the risk is mitigated, and thus software development should be architecture centric.

iii) Iterative and Incremental Approach

The iterative and incremental approach is necessary to reduce the large software development projects of today into manageable pieces. Each piece should represent quantifiable progress toward the completion of the overall development effort. As the development teams progress through each increment, the lessons learned are also noted and the process refined. This results in more successful information systems. Thus while developing information system iterative & incremental approach to system development is suggested.

Future Scope of Work

In this study, only one module of the examination system has been designed following Software Engineering approach. Similarly other modules of the system should be designed and developed to achieve a high integration of various sub systems. Further, other systems of the university needs to be investigated and developed following the same approach. A high level

integration with other universities using networking facilities is another potential area for future research.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Pressman Roger S.: Software Engineering A Practical Approach Mcgraw-Hill, Fourth Edition-1997.
2. Fairley, Richard E.: Software Engineering Concepts (McGraw-Hill)
3. Elias M. Awad: Systems Analysis & Design Galgotia Publication-Edition 1990
4. Jalota Pankaj: An Integrated Approach to Software Engineering Narosa Publishing House
5. Date C.J.: An Introduction to Data Base System AWL Publications, Edition 1994
6. Basandra S.K., Jaiswal S.: Local Area Network
7. Ghezzy, Jazayeri, Mandrioli: Fundamentals of Software Engineering Prentice-Hall of India, Edition-1994
8. www.badsoftware.com
9. Kanner C.: Testing Computer Software
10. http://osiris.sunderland.ac.uk/rif/linda_spence/HTML/contents.html
11. Abdullah, Khalil Abdullah (No. DA9904004) Dissertation Abstracts Int. Feb 99, Vol-59, No-8, Page-4231-B
12. Daniels, Fonda Jonette (No. DA9909462) Dissertation Abstracts Int. Apr 99, Vol-59,No-10, Page-5437-B
13. Gokhale, Swapna Sudhir (No: DA9839454) Dissertation Abstracts Int., Jan 99,Vol-59, No-7, Page-3548-B
14. Goralwalla, Iqbal Abdulrahim (No: DANQ2904) Dissertation Abstracts Int. Jan 99, Vol-59, No-7, Page-3548-B
15. Hayes, Jane Huffman (No: DA9908079) Dissertation Abstracts Int. Mar 99, Vol-59, No-9, Page-4922-B

16. Homyen, Ariya (No: DA9907449)
Dissertation Abstracts Int. Mar 99, Vol-59, No-9, Page-4922-B

17. Hilford, Victoria (No. DA9902929)
Dissertation Abstracts Int. Feb 99, Vol-59, No-8, Page-4244-B

18. Krusal Ivan Victor (No. DA9900214)
Dissertation Abstracts Int. Feb 99, Vol-59, No-8, Page-4248-B

19. Medvidovic, Nand (No. DA9912025)
Dissertation Abstract Int. May 99, Vol-59, No-11, Page-5937-B

20. Michael Edward (No. DA9903498)
Dissertation Abstracts Int. Feb 99, Vol-59, No-8, Page-4234-B

21. Rivers, Anthony Thyron (No. DA9909496)
Dissertation Abstracts Int. Apr 99, Vol-59,No-10, Page-5448-B

22. Smith, Randy Keith (No: DA9907035)
Dissertation Abstracts Int. Mar 99, Vol-59, No-9, Page-4935-B

23. Sova, Donald William, Jr. (No. DA9909013)
Dissertation Abstracts Int. Apr 99, Vol-59, No-10, Page-5449-B

24. Stiff, Michael Benjamin (No. DA9837050)
Dissertation Abstracts Int. Feb 99, Vol-59, No-8, Page-4257-B

25. Zhang, Guogen (No. DA9913076)
Dissertation Abstracts Int. May 99, Vol-59, No-11, Page-5946-B