Prototyping processes for requirements analysis.

Arathi. Ranganathan

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/3215
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

by

Arathi Ranganathan

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada
1994
The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L’auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L’auteur conserve la propriété du droit d’auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
Abstract

It is widely recognized that requirements analysis is an extremely important phase of the software development life cycle. It is often stated that requirements analysis is the most important phase.

Inaccurate and incomplete specification of requirements increases the need for extensive and expensive maintenance. Inaccurate specifications are often caused by lack of proper communication between the systems analyst and the user, or by an incomplete understanding that the user has of the real requirements.

Many analysts are beginning to use prototypes to assist them in identifying requirements. A prototype models aspects of the final system as envisaged by the user, but not the efficiency or robustness of the final product. Claims have been made that prototypes help users identify real requirements and improve communication with system analysts.

Despite the reported successes of prototyping, the use of prototypes is not as wide as it might be. One reason for this is that the use of prototypes in requirements analysis has not been analyzed with sufficiently well defined notation or metrics. Hence, it is difficult to get a good understanding of costs and benefits from the very informal description of case studies. Although industry is often receptive to new methodologies and changes, it is apprehensive when the new techniques, and benefits of using them, have not been adequately defined.

The thesis that is defended in this report is that:

a. Well-defined graphical notations can be used to describe various prototyping processes for requirements analysis.
b. It is possible to identify well-defined metrics by which prototyping processes can be analyzed.

c. Prototyping processes used in requirements analysis are amenable to empirical study.

Establishment of this thesis is a step towards a better understanding of the value of prototyping in requirements analysis. Not only does the thesis demonstrate that prototyping processes are amenable to measurement, the work undertaken to support the thesis also provides some techniques for conducting such measurement.

- A new notation has been developed for describing prototyping processes. It has been tested on various processes and on a number of people.
- A set of well-defined metrics has been identified which can be viably used to measure aspects of the prototyping process.
- There is an indication that some widely held assumptions regarding prototyping processes require further investigation.

The ability to clearly define prototyping processes and to provide measurements related to specific case studies has many uses:

- Analysts will be able to refer to case studies of prototyping and obtain a better understanding of the costs and value involved in the use of a particular process.
- Prototyping processes themselves can be analyzed and improved over a period of time as a database, or experience library, of well-defined case studies is developed.
To my parents, Datta periappa, and Sara periamma.
Acknowledgments

I take this opportunity to thank my supervisor Dr. Richard Frost for his invaluable help and his guidance to successfully complete my degree. Without his active and patient supervision it would have been impossible to complete my thesis within this time frame. I thank Dr. Subir Bandyopadhyay for his active support throughout my master's program. I am thankful to Dr. Diana Kao for her valuable suggestions regarding my thesis. I am grateful to Dr. Joan Morrissey and Dr. Henry Toews for valuable suggestions and comments on this thesis. I thank the department secretaries Mary and Margaret for their help. Special thanks to Ms. Aruna Cidambi and Ms. Sukanya Pillay, for their help and support this past year. I wish to thank my friends and colleagues in the grad. lab., especially Mrs. Arunita Jaekel. I would also like to thank Mr. Rajesh Shenoy. I thank all the respondents who completed the survey, especially the ones whose arms I had to twist. Finally, I would like to thank Mr. Walid Mnaymneh for all help throughout the various stages of this work.
# TABLE OF CONTENTS

Abstract .......................................................... iv

Acknowledgments .................................................. vii

List of Figures .................................................. xi

List of Tables .................................................... xiv

CHAPTER 1: INTRODUCTION ....................................... 1

1.1: Background and Thesis statement ......................... 1

1.2: Importance of the thesis .................................... 11

1.3: Work undertaken to support the thesis .................... 20

CHAPTER 2: FORMALIZING THE PROTOTYPING PROCESS FOR

REQUIREMENTS ANALYSIS ....................................... 22

2.1: Why is it important that the prototyping process be represented
unambiguously ? .................................................. 22

2.2: A New notation for describing prototyping processes ........ 23

2.3: Examples of the use of the new notation .................... 25

2.4: How the new notation was developed ? ...................... 37

2.5: Feedback from potential users of the new notation ....... 43

CHAPTER 3: WELL-DEFINED METRICS FOR ANALYSING

PROTOTYPING PROCESSES .................................. 45

3.1: What is a metric ? .......................................... 45

3.2: Some "metrics" that are often referred to in texts on software
analysis .......................................................... 47
3.3: How an appropriate set of metrics for analyzing prototyping projects was identified .................. 49

3.4: Metrics that were identified .................................. 51

3.5: Reasons why the metrics chosen may be useful .................. 56

CHAPTER 4: A CASE STUDY ........................................... 58

4.1: Introduction .............................................. 58

4.2: Experimental sessions with the prototypes .................. 61

4.3: Analysis of the prototypes .................................. 66

4.4: Findings during the case study .............................. 74

CHAPTER 5: CONCLUSION ........................................... 80

5.1: Analysis of findings ...................................... 80

5.2: Additional findings ...................................... 84

5.3: Future work ............................................. 87

APPENDIX : A: Using natural language for systems specification ......... 88

A.1: The process to be studied .................................. 88

APPENDIX : B: Questionnaire for study on notation .................. 98

B.1: A study of a notation for representing prototyping processes ... 98

APPENDIX : C: Summary of the responses to the survey ............... 111

C.1: Frequency table: Study I ................................. 111

C.2: Frequency table: Study II ................................. 113

APPENDIX : D: Guidelines for building experience libraries ............ 114

APPENDIX : E: Form for data collection during a prototyping project ... 115

E.1: Template ............................................... 115
APPENDIX : F: Survey report — Computer Assisted Analysis and Design of
Database Systems ............................................. 122
  F.1: Introduction to the survey ................................ 122
  F.2: Processes used in the development of database systems . 126
  F.3: Conventional tools and methods .......................... 136
  F.4: Prototyping .............................................. 153
  F.5: Computer Aided Software Engineering (CASE) systems . 159
  F.6: Future trends ............................................. 171
  F.7: Concluding comments .................................... 171
APPENDIX : G: Program Listing .................................. 174
  G.1: SQL/DS ................................................. 174
  G.2: C ..................................................... 183
  G.3: C++ .................................................. 197
Bibliography .................................................. 204
APPENDIX : H: VITA AUCTORES .................................. 216
List of Figures

Figure 1  Categorization of software tools by Pressman .............. 5
Figure 2  Exploratory programming and prototyping ................. 7
Figure 3  Prototyping taxonomy — overview ......................... 9
Figure 4  Prototyping in the software process ..................... 10
Figure 5  The prototyping process as described by Luqi and
          Berzins .............................................. 16
Figure 6  Notations used in the Formalisms ......................... 23
Figure 7  Prototyping process I ......................................... 26
Figure 8  Prototyping process II ......................................... 29
Figure 9  Prototyping process III A ................................. 32
Figure 10  Prototyping process III B ................................. 34
Figure 11  Prototyping process IV ......................................... 36
Figure 12  An example use of the first notation that was developed . 37
Figure 13  An example use of the refined notations .................... 38
Figure 14  An example use of the second refined version of the
          notation ............................................. 39
Figure 15  An example use of the third refinement to the notations  . 40
Figure 16  An example use of the fourth refinement of the notation . 41
Figure 17  An example use of the fifth refinement of the notations  . 42
Figure 18  An example use of the sixth refinement of the notation . 43
Figure 19  Form 1 .............................................. 101
Figure 20  Formalism .............................................. 102
Figure 21  Form 2 .................................................. 104
Figure 22  Notations used in the Formalisms .................. 105
Figure 23  Form 3 .................................................. 106
Figure 24  Form 4 .................................................. 107
Figure 25  Illustration of the underlying differences between the
traditional and automated methods ......................... 124
Figure 26  Traditional Systems Development Life-Cycle ...... 127
Figure 27  Systems Development Lite-Cycle: an extensive look . 132
Figure 28  Systems Development Life-Cycle: Waterfall Model . 134
Figure 29  Systems Development Life-Cycle: Spiral Model . 135
Figure 30  Set of symbols used in the representation of the flow
chart ................................................................. 139
Figure 31  An example of a flow chart .......................... 140
Figure 32  Various symbols used in system flow charts ......... 141
Figure 33  An example of a systems flow chart ................ 141
Figure 34  The components of data flow diagrams ............ 142
Figure 35  An modified example of sub-levels 4 of a data flow
diagram ........................................................... 143
Figure 36  Symbols used in entity-relation diagrams ........... 144
Figure 37  Example of entity-relation diagram ................ 144
Figure 38  The various entries that are required in maintaining a data
dictionary ......................................................... 145
Figure 39  An example on the use of decision table .......... 147
Figure 40  An illustration of a decision tree ................. 148
Figure 41  An illustrative example of the Nassi-Schneidermann .. 148
Figure 42  An illustration of the selection and iteration criteria
            supported by Jackson charts .......................... 149
Figure 43  Stock Inventory Database in Warnier-Orr diagram
            format ................................................. 150
Figure 44  Warnier-Orr diagram representing program structure .. 150
Figure 45  HIPO — Hierarchical level 1 for updating inventory file . 151
Figure 46  Illustration of a Structured Chart ..................... 152
Figure 47  Concept of prototyping ............................... 154
Figure 48  Evolutionary Prototyping ............................. 155
Figure 49  The phases covered by the CASE tools ................ 160
Figure 50  Components of CASE workbench ....................... 163
Figure 51  Five Generations of Software Technology ............. 167
Figure 52  Fourth-generation technology vs CASE ................ 170
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Specifications in natural language</td>
<td>58</td>
</tr>
<tr>
<td>Table 2</td>
<td>Analysis of the prototypes</td>
<td>74</td>
</tr>
<tr>
<td>Table 3</td>
<td>Summary use of the prototypes.</td>
<td>75</td>
</tr>
<tr>
<td>Table 4</td>
<td>Viability of measuring the metrics.</td>
<td>79</td>
</tr>
<tr>
<td>Table 5</td>
<td>FORM I</td>
<td>116</td>
</tr>
<tr>
<td>Table 6</td>
<td>FORM II</td>
<td>118</td>
</tr>
<tr>
<td>Table 7</td>
<td>Data from SQL prototype.</td>
<td>119</td>
</tr>
<tr>
<td>Table 8</td>
<td>Data from C prototype.</td>
<td>121</td>
</tr>
</tbody>
</table>
Chapter 1 INTRODUCTION

§ 1.1 Background and Thesis statement

1.1.1 Software Development

Software development is still a new field and it is appropriate to seek analogies and borrow solutions from other well-established disciplines. Building and using prototypes to initiate the final solutions is a common activity in hardware engineering disciplines. This approach is currently being used in software development with relative success [28].

The term software engineering was first introduced in the late 1960s at a conference held to discuss the software crisis in the industry. This crisis was due to the introduction of third generation computer hardware that made development of large software applications feasible. A need for a well-defined method of software development was identified, as the existing methods were found inadequate to handle the complexity and size of the software required.

The notion that software development is an engineering discipline established the belief that software development was similar to other engineering processes. A model of the software development process was derived from other engineering activities [65]. Software engineering must use a quantitative approach to become a true engineering discipline and to satisfy the future demands of software development [58].

The current industry-endorsed method for requirements definition is rigorous definition. The logical needs of the application must be thoroughly and completely specified. The Software Development Life Cycle (SDLC) consists of various phases or steps involved in the recognition of a problem and the approaches needed to solve it for the
benefit of the organization, and the maintenance features required to accommodate the changes as and when required by the organization.

The phases of the SDLC are Requirements Analysis or Systems Analysis; Systems Design; Coding; Testing; Implementation; and Maintenance. Until the end of the 1970's the waterfall model of the SDLC was widely used [62]. In the waterfall model the phases cascade from one phase to another. However, it soon became apparent that the waterfall model was not sophisticated enough to keep up with the methods used in software development. The activities identified in the model did not correspond with the realities of the complex and variable process of software development.

A software engineer is defined as a person who "is involved with the entire software life cycle - from the conception of the system, through the development phase, and finally to the end of its operational life".

Software development is a creative task. It involves cognitive skill which is limited by the capacity of the human brain. "Most software projects are group activities, involving all the complexities of group dynamics, communication networks, and organizational policies. The study of group behavior in software development is in its infancy, but like the study of individuals, it promises to improve our understanding of the development process, particularly the front end [58]." Software developers often work with highly volatile requirements to provide the system as desired by the user. This volatility often produces inefficiency and quality degradation.

Basili and Musa [4] predict that the emphasis in the 1990's will be on the quality of

1 REFER TO APPENDIX F
software developed. Quality is a multidimensional concept. These dimensions include the entity of interest, the viewpoint of that entity, and the quality attributes of that entity. Entities consist of the final product, the intermediate products such as requirements documentation, and the process components such as the design method. The quality attributes relevant to any situation depend on both the entity and the viewpoint of that entity.

Requirements specification is a mutual process of discovery of the user’s real requirements or needs by the analyst and the user. The needs of the user often differ from his or her wants. The analyst tries to satisfy the user's requirements provided they are useful and are needed. The user tries to clarify and establish the necessity for those requirements.

Successful projects are often based on a joint process whereby the developer learns about the application domain or area, and the user about design realities and available choices.

What is an application area? It is quite appropriate at this stage to define what is meant by an application area in the field of software engineering. Pressman [58] feels that it is quite difficult to compartmentalize the various software applications as the complexity increases. However, according to him some of the software areas that present the applications possible are:

- **SYSTEM SOFTWARE**: This is a collection of programs that are written to service other programs. Examples of system software that process complex but determinate information structures are compilers, editors and file management utilities. Examples of systems software that process indeterminate data are operating systems compo-
nents, drivers, and telecommunications processors. This application area involves a large amount of interaction with computer hardware, heavy usage by multiple users, concurrent operation that requires scheduling, resource sharing, and sophisticated process management, complex data structures, and multiple external interfaces.

- **REAL-TIME SOFTWARE**: Any software that measures, analyzes or controls real world events are known as real-time software. Such a systems must react within a specific time period else would have disastrous results.

- **BUSINESS SOFTWARE**: The business informations processing is the largest software application area. Management Information System (MIS) software accesses large databases of business information.

- **ENGINEERING AND SCIENTIFIC SOFTWARE**: These basically are the "number crunching" programs. Applications include such diverse areas as astronomy, volcanology, automotive stress analysis, space shuttle orbital dynamics, molecular biology, and automated manufacturing.

- **EMBEDDED SOFTWARE**: This resides in read only memory and is used in control products and systems for consumer and industrial markets. These systems perform very limited functions, and provide significant function and control capabilities.

- **PERSONAL COMPUTER (PC) SOFTWARE**: Word processing, spreadsheets, computer graphics, entertainment, data base management, personal and business financial applications, and external network or data base access are some of the many applications of personal computer systems.

- **ARTIFICIAL INTELLIGENCE (AI) SOFTWARE**: Use of non-numerical algo-
rithms to solve complex problems that are amenable to computation. Expert systems or knowledge-based systems is the area that is being currently explored. Other application areas include pattern recognition (image and voice), theorem proving, and game playing.

1.1.2 Resources required in software development

Implementation of suitable hardware and software tools are essential to develop any software system. Pressman [58] reports that "every resource consists of: description of the resource; a statement of availability; chronological data that the resource will be required; duration of time that the resource will be applied."

![Diagram of software tools by Pressman](image)

**Figure 1** Categorization of software tools by Pressman

**Human resources** Qualified people are a primary software development resource. Software development requires persons such as managers, software engineers, technical staff, and so on. Senior staff are involved actively in project planning, requirements analysis, design, and the final testing stages, whereas the junior staff are involved in later stages of design, coding, and the early testing stages. The number of people required
for the development of any software system depends on the estimations of the planning phase. Scarcity of qualified personnel can halt a development process.

**Software resources** Any software tool that can be successfully utilized to build the new software comes under this category. The tools that are available in industry can be categorized as follows:

- **Code-oriented tools** include programming language compliers, editors, linkers and loaders, debugging aids, and an array of language-specific support.
- **Methodology tools** are used by software engineers for the analysis and design of the system. These tools support project planning, requirements analysis, design, testing, configuration management, maintenance and other similar activities.
- **Fourth generation tools** provides data base query languages and other nonprocedural techniques to assist the developer.

**Hardware resources** According to Pressman [58], selection of appropriate hardware for the development is quite difficult and it usually depends on the following factors:

- Components of the hardware are packaged as individual building blocks.
- Interfaces among components are standardized.
- Numerous off-the-shelf alternatives are available.
- Performance, cost and availability.

**1.1.3 Role of prototyping in software development**

If the system to be developed is large and complex, it is probably quite difficult to envision the system before it is built. Exploratory programming starts with a vague understanding of the system requirements. The systems developed by this approach may
not be specifiable. The system is built and modified as new needs are identified. This is where prototyping is applicable. The conventional prototyping approach is intended to discover the requirements specification, and create an output of the prototype development phase which meets that specification [65]. The differences in prototyping and exploratory programming are illustrated by Figure 2.

![Diagram of exploratory programming and prototyping](image)

Figure 2: Exploratory programming and prototyping

"A prototype is a trial version of an artifact (or its components) built for the purpose of exploration, experimentation and/or evaluation³". Prototyping — the process of building and using a prototype — often plays a crucial role in various types of engineering⁴. The role that prototyping can play in software engineering was recognized in the 1980s.

Prototypes focus on the functionality of systems rather than the efficiency of execution. They can be used as a means for obtaining user requirements. This is a major use of prototyping in software development and is the focus of this thesis. The prototyping process recognizes that requirements may initially be unclear or ambiguous. A prototype software system may be developed from an outline specification. Users experiment with

³ Pg 131, B. Racilfr "Early and Not-so-early Prototyping — ⁴ REFER TO APPENDIX F
Rationale and Tool Support" IEEE Software 1988
the prototype to refine and improve the specification. Therefore, the use of formal specification languages is not appropriate and a natural language description of requirements specification is often the starting point. While a function described in a specification may seem useful and well-defined, it may reveal the fact that the user's initial view was incorrect or incomplete when used with other functions.

Reliable user requirements are important for successful system development. It is difficult, if not impossible, to design complex software without a clear specification of user requirements. Prototyping can be defined as an approach for clarifying and establishing systems requirements through user interaction. Prototyping is especially important in software engineering as it reduces the usually considerable effort, cost, and time that is needed in the maintenance of the system. Prototyping involves the building of prototypes and is an iterative process [56]. The steps involved in this iterative process are: the interactions of the user with the prototype, the revision and refinement of the requirements, and the subsequent revisions to the prototype based on the user's request for change. Prototyping helps the user to view a working model of his or her requirements. These models are known as prototypes.
Figure 3 presents the taxonomy of the prototyping approach based on WHY and WHEN the prototypes are required in software development [59]. The principle reason for prototyping is to validate software requirements. The system requirements are validated because users discover requirements errors or omissions early in the software process.

There are four stages in prototype development:

- Establish prototype objectives.
- Select functions for prototype inclusion and make decisions on what non-functional requirements must be prototyped.
- Develop the prototype.
- Evaluate the prototype system.
Sommerville [65] presents a software process model that is based on the prototyping stage in Figure 4. This extends the requirements analysis stage to include the prototyping model in order to reduce the costs incurred in the life cycle. The diagram implies that the prototype is developed from an outline of the system specification. It is then used by the user and modified until the user is satisfied with its functionality. A specification is derived from the final prototype and the system is developed using a conventional software process model.

Software prototyping can be expensive if the prototype is developed using software which is not a suitable prototyping tool. Usually, non-functional requirements such as speed and space are not crucial. Error handling and management may be ignored or may be rudimentary unless the objective of the prototype is to establish a user interface. Standards of reliability and program quality may also be reduced.

1.1.4 The thesis

The thesis statement to be defended in this report is that:
a. Well-defined graphical notations can be used to describe various prototyping process for requirements\textsuperscript{5} analysis\textsuperscript{6}.

b. It is possible to identify well-defined\textsuperscript{7} metrics by which prototyping processes can be analyzed.

c. Prototyping processes used in requirements analysis are amenable to empirical study.

§ 1.2 Importance of the thesis

Software projects are usually big and complex, specifications are imprecise and volatile, the correctness of a solution differs with each participant in the application development, and the participants vary in skill level and may work on more than one project at any given time OR often work concurrently on two or more projects.

1.2.1 Need for better analysis in software development

Application of structured methodologies and techniques does not always produce the required system. Software development gained importance with the reduction in cost of hardware and software, and with the increase in the availability of qualified computer professionals. Some of the methodologies developed in 1970s are still being widely used. Though industry is receptive to new methodologies, it is quite reluctant to implement them owing to the risks involved and its viability. Hence, structured methodologies are a major obstacle in the acceptance of new technologies.

\textsuperscript{5} Requirements define "what" of the problem

\textsuperscript{6} Requirements analysis is the process of analyzing the various functionality of the existing system in great detail.

\textsuperscript{7} Well-defined or invariant to some extent.
The use of structured methodologies based on incorrect or incomplete assumptions increases work in the maintenance phase. "More than any other single factor, a specification that inadequately defines system requirements can jeopardize a project's success [59]." These erroneous assumptions are due to the lack of proper communication between the user and the systems analyst concerning the changing user needs. "In the software lifecycle the specification phase is the most crucial one [38]." The systems analyst often ignores certain user needs due to various constraints and reasons. Errors that go undetected or are not rectified at this stage could require extensive changes at the implementation stage.

Though the software crisis described in the previous chapter is yet to be resolved, there have been vast improvements in software engineering methods and techniques. Also, in tools used in software development and the skills of the computer personnel involved. But the increase in demand for software is faster than the improvements in the productivity of the software development process [65].

Glass [28] states that software crisis is caused by "Software that is always over budget, behind schedule, and unreliable". This is due to:

- Unstable requirements. Users are unable to clearly state what they need.
- Poor contracting. The contract does not clearly state the terms, conditions, processes, and products involved.
- Optimistic evaluation. Setting unrealistic time limits.

A software project needs planning and budgeting. "Software development is the most complex undertaking ever attempted by humanity. For any given hardware problem, especially those involving high reuse of component parts, there is only a limited number
of ways that the problem can be solved successfully. But for software, even considering optimized solutions, there is a near-infinite number of possible correct solutions [28]." Often the software products do not perform as expected and the development process may face many difficulties. Some of these difficulties according to Alan Wingrove⁹ are:

- a general lack of visibility⁹
- incomplete and ambiguous requirements
- imprecise and incomplete specifications
- a lack of agreed terminology
- uncertainties in software/hardware apportionment
- rapidly changing technology
- suitability of languages
- inability to model systems
- difficulties in resource and cost estimation
- inability to predict and measure reliability
- difficulties in progress monitoring
- complicated error and change control
- a lack of agreement on test metrics
- problems with interfacing
- problems with integration
- difficulty in controlling maintenance
Wingrove states that a number of these difficulties are faced in all software projects. Consequently, this causes a major setback to the concept of project management.

"There has been little success to date in the transfer of new software engineering methodologies and tools into active practise, despite the potential benefits of improving the development process in an industry of such size and importance [58]." This failure is attributed to the fact that software engineering is a process rather than a product. "It is an abstract intellectual ability with limited visibility, which makes it much more difficult to transfer. Most practicing software engineers are not aware of all the possibilities for improvement that exist [58]."

Another defect that is hardly realized in this industry is that most researchers are unaware of the entire range of problems that have to be successfully addressed before any of the new technologies gain practical significance. Tools and methodologies are very often difficult to learn or use or both. This is visible in the software industry where one can observe the coexistence of old\textsuperscript{10} and new\textsuperscript{11} languages and technologies.

Basili and Musa [4] recommend that these defects must be addressed. They feel that the means to overcome this hinderance should be sought through widespread education of new methods and tools. Methods and tools must be easy to learn in order to gain wide and rapid acceptance. Industry must also be receptive and willing to make change. New technologies are subjected to evolution and change as more knowledge is gathered on them by usage. Success and failures present the new technologies in an objective manner.

Software development should lay emphasis on the need for better analysis since the entire software product depends on the accuracy of the requirements. Inaccurate or

\textsuperscript{10} eg. COBOL  \hspace{1cm}  \textsuperscript{11} eg. C++
improper analysis produces a system that is inappropriate to the application area and does not produce user satisfaction. This often leads to extensive maintenance that is both cumbersome and expensive. Therefore it is profitable for the analyst to pay attention to evolving user needs and to incorporate the changes in the system into the requirements specification.

Many of the problems discussed above can only be tackled if the software development process itself is better understood. Therefore the software development process must be amenable to analysis. In particular, the prototyping process which is of paramount importance in obtaining user requirements must be amenable to formal analysis if it is to be better understood and better utilized.

1.2.2 Need for better analysis of prototyping processes

Emphasis is laid on effectiveness of using prototyping as a basis for providing the requirements specification in this thesis. Most books and papers on this topic acknowledge the importance of using prototypes to gather initial specifications. They also maintain that the value of using prototypes will be realized if it is possible to accumulate data on them. Kraushaar and Shirland [34] conclude from their experience with developing prototypes for various applications that “prototyping can provide ontime and within-budget systems for both large and small application projects that are typically developed by information systems specialists and/or end users. The prototyping methodology clearly has the potential to improve productivity over a wide range of applications”. A prototype may be developed from an outline specification. Users experiment with the prototype to refine and improve the specification. A working model of the application is initially built as a base and that is used to gather subsequent refinements until an acceptable
solution is reached. The benefits of developing a prototype early in the software process are summarized below:

- Early understanding of the system by software developers and users through demonstration of the system functions.
- Detection of missing user services.
- Identification and refinement of difficult-to-use or confusing user services.
- Identification of incomplete and/or inconsistent requirements as the prototype is developed.
- Availability of a working system to demonstrate the feasibility and usefulness of the application to management.
- The prototype serves as a basis for writing the specification for the system.

According to Vonk [74], “the costs of prototyping depends on the quality and the features of the tools being used to develop the prototype. The cost will further depend on the number of iterations needed to arrive at the acceptable prototype”. He concludes that the number of iterations depends on the uncertainty that continues to exist about the system.

Figure 5 The prototyping process as described by Luqi and Berzins
Prototyping is a means of communication between the user and the analyst. It is used to depict their respective perception of the system. The user, by evaluation and requesting changes, tries to get the system he needs. Figure 5 is a model of the prototyping process given in Luqi and Berzins [43].

Prototyping is a preliminary version of a future entity. It is an original model. There are essentially two types of prototypes. The first is built to represent the input and output of the system, and to give the user the ability to visualize the functionality of the final system. The second is a scaled-down version of the whole product which gives the user and the analyst a feel of the workability of the final product [28]. In both, the underlying feature is the prototype which helps visualize the requirements in order to identify the requirements which will form the basis for the final system. Thus, a prototype can be viewed as a skeletal version of the final software product. The prototype concept is an important one and in certain types of requirements problems there is no substitute for its use [Brooks].

1.2.3 Need for standardized software processes and notations for describing processes

The requirements that are used to build software systems are highly volatile owing to the fact that the user’s needs are never fixed. The needs constantly change until they have been identified in the requirements analysis phase. Other needs could also be identified in the other phases, especially the maintenance phase. Software developers work with these highly volatile requirements to provide the system desired by the user. This volatility is owing to lack of proper communication between the two participants.

12 "Build One to Throw Away," The Mythical Man-Month. Addison-Wesley, 1975
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

Successful projects are quite often based on the joint process whereby the developer learns about the application domain and the user about design realities and available choices. A process can be defined as a means that "supports feedback, learning and the refinement of the models of the environment [58]."

Models in the software environment are necessary. Models focus attention on the various issues of the product. "Experience models must be integrated into an experience base that can be accessed and modified to meet the needs of new projects [5]." These are the experience factories\textsuperscript{13}.

The thesis, however, does not focus on building a database of experience models. Such an endeavor is dependent on the various software tools, methodologies, and languages available in the industry. Understanding is achieved only when the components of the software system can be isolated, can be represented using well-defined notations and their properties can be measured. This thesis concentrates on the development of such notations and measurements.

Software developers feel that the fundamental principles of software development differ from that of manufacturing engineering but, are in some ways the same. There is no single model for a process and measurements are needed to verify the various models. The formalisms offer a pictorial view of a few prototyping processes. They are a means to continuously upgrade and clarify the requirements gathering process or models. A set of standardized processes and notations could be used to overcome these drawbacks that are present in the software development industry.

1.2.4 Need for better metrics

Estimating the resources that will be required to provide a system as envisaged by the user is an arduous task. The cost of software development depends on the quality of the developers, hardware, software and other resources available. Software development is a volatile area which has been subjected to study since the 1970's. Researchers claim that prototyping would help reduce maintenance costs significantly while increasing user satisfaction.

The ultimate goal is user satisfaction [28]. The specification of the product has to satisfy the explicit and implicit user needs, and be within the budget. The cost of software development depends on the strategy adopted and other factors. Human cognitive processes and social dynamics in software development affect product quality. Cognitive psychology is defined as the study of problem solving. Its disciplines can be used to study different intellectual activities in the software development process. Cognitive psychology in software development is largely unrealized and currently it focuses only on the human-computer interface. This focus is "implicit" in computer-aided software engineering tools.

Metrics are the means of quantifying and qualifying the various processes and models. Measurements are based on models that need to be monitored to evaluate their functionality.

The metrics identified in this thesis are meant to provide an idea of the resources involved in using prototyping. Some metrics that seem ideal to measure based on common-sense are often quite difficult to measure in reality.
1.2.5 Relationship of the thesis to problems in software development

It is clear that there is a need for more analysis of software development processes. There is also a need for more work on metrics. The use of prototypes will only be of value if the prototyping process is well understood. This thesis which is concerned with the formalization and analysis of the use of prototyping in requirements specification is clearly of great importance in the software industry. Ince and Hekmatpour [31] feel that the most important aspect of prototyping is the evaluation of prototypes. The evaluation process depends on meaningful measures\textsuperscript{14}.

§ 1.3 Work undertaken to support the thesis

In order to support the thesis, a number of tasks were carried out:

- A survey was conducted on the use of computer-aided tools in the analysis and design of database systems\textsuperscript{15}. It was found that prototyping is gaining use in software development. Prototyping as a means of gathering requirements specification was identified as an area suitable for the thesis work.

- A formalism was developed for describing prototyping processes used in requirements analysis\textsuperscript{16}. Since most of the conventional notations in software engineering are pictorial representations, the notations of the formalization are pictorial.

- The formalism was tested by using it to represent four prototyping processes\textsuperscript{17}.

- A survey was conducted to measure the clarity of the notations used to formalize the prototyping process. The respondents were also asked to come up with a prototyping

\textsuperscript{14} Riddle W. E. 'Advancing the state of the art in software system prototyping'. Approaches to Prototyping, Springer-Verlag, Berlin FRG (1984)
\textsuperscript{15} REFER TO APPENDIX F
\textsuperscript{16} REFER TO CHAPTER 2, SECTION 2.2
\textsuperscript{17} REFER TO CHAPTER 2, SECTION 2.3
process of their own\textsuperscript{18}.

- In order to determine whether or not natural language specifications might be `analyzable`, a small experiment was conducted\textsuperscript{19}.

- A student record system was built based on the natural language specifications.

- An experiment was conducted to validate part of the thesis which states that prototyping processes are amenable to empirical study. The empirical study of the use of a prototyping process was conducted on the student record system.

- A set of well-defined metrics associated with the prototyping process were identified. Forms were developed to aid the analysts record values for these metrics during the prototyping process\textsuperscript{20}.

- Appendix D contains a software guideline for entering information regarding applications that can be maintained in experience factories.

\textsuperscript{18} REFER TO APPENDIX B. \hspace{1cm} \textsuperscript{19} REFER TO CHAPTER 4, SECTION 4.1.1

\textsuperscript{20} REFER TO APPENDIX E
Chapter 2 FORMALIZING THE PROTOTYPING PROCESS FOR REQUIREMENTS ANALYSIS

§ 2.1 Why is it important that the prototyping process be represented unambiguously?

Various prototyping processes have been used, and are growing in use in the development of software. Because there is no formalism to describe these different processes, it is difficult to compare them. It is also difficult to transfer experience gained in one application area to a new one.

Prototyping has been stressed as a very important aspect of software development by many researchers. They maintain that it helps reduce maintenance effort and cost. It has been and is being used in many instances but, the standardized material available is insufficient to substantiate the claim. Many researchers active in this field have expressed that the entire software industry requires extensive research, and that prototyping also deserves a lot of work to conclusively prove its usefulness. Formalization helps represent the prototyping process in a standard notation whereby, it gets a structure. This, then forms a guideline to measure various aspects of the process that might prove useful to the entire development of the product.
§ 2.2 A New notation for describing prototyping processes

The system of symbols presented in Figure 6 was identified as being adequate and appropriate for the specification of various prototyping processes. Using this notation, the description of a prototyping process can be as abstract or as specific as necessary.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Name of Notation</th>
<th>Thing Denoted</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Actor</td>
<td>The actor X</td>
</tr>
<tr>
<td>X</td>
<td>Activity name</td>
<td>The activity X and the topics that are appropriate to perform X</td>
</tr>
<tr>
<td>T</td>
<td>Tool used to perform activity</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Data Communication</td>
<td>The abstract representation of data X</td>
</tr>
<tr>
<td>M</td>
<td>Manner of representation (optional)</td>
<td>The use of the option specifies that X is physically represented using the media M</td>
</tr>
<tr>
<td>X</td>
<td>Concurrence</td>
<td>X and Y are concurrent activities</td>
</tr>
<tr>
<td>Y</td>
<td>Output</td>
<td>Y is the output from activity X</td>
</tr>
<tr>
<td>X</td>
<td>Access of Document</td>
<td>X is accessed by actor Y</td>
</tr>
<tr>
<td>Y</td>
<td>Execution</td>
<td>X is executed by actor Y</td>
</tr>
<tr>
<td>Y</td>
<td>Flow of Control</td>
<td>Indicates that activity Y follows directly after activity X</td>
</tr>
<tr>
<td>X</td>
<td>Multiple Access</td>
<td>The data communication X is accessed by more than one activity</td>
</tr>
<tr>
<td>X</td>
<td>Update</td>
<td>The data communication X has been updated</td>
</tr>
<tr>
<td>X</td>
<td>Extended Update</td>
<td>The data communication X has been updated (updated the input with +)</td>
</tr>
</tbody>
</table>

Figure 6: Notations used in the Formalisms
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

**Actors** are the principle participants in the prototyping process. The participants are the user or the systems analyst or both. Each actor does some activity. Each **activity** is identified by a unique activity name and the tools that are used by the actors to perform the activity.

All activities that occur in sequence are connected by the **flow of control** symbol. The concurrent activities X and Y\(^1\), that occur in a loop over a short period of time and have a lot of interaction with actors, are placed in a **concurrent box**.

The **output flow** is used to indicate that Y is the output from the activity X. A **data communication** is an output or input to an activity. In the notation, a data communication is denoted as an abstract representation of data. But, it can be made specific by adding the option **manner of representation** to represent the documentation: manual, on-line, verbal or a program\(^2\). The **access of documentation** denotes that the data communication provides the actor Y with a document X, that supports the prototype. The **execution** denotes that the data communication X provides an executable version of the prototype to the actor Y.

The **multiple access** placed on the top right corner of a data communication informs the actors that that particular data communication is used as input by more than one activity. The **update** denotes the modified version of a data communication. The **extended update** denotes an updated version of an update. An example illustrating this is shown in the figure depicting prototyping process III.

---

\(^1\) The number of activities are not restricted to two.  
\(^2\) The mode of communication of ideas between the system analyst and the user is immaterial to this thesis.
§ 2.3 Examples of the use of the new notation

In this section, four different prototyping processes are described using the new formalism. These processes were chosen to be abstract representatives of some prototyping processes.

2.3.1 Prototyping process I

In this process, illustrated by Figure 7, the system analyst (SA) designs and builds the prototype that the user evaluates. Any changes that the user requests are addressed by the SA. This involves a period of intense clarification of alterations to the requested changes and identification of alternative changes. When a set of feasible changes have been accepted by the user, the prototype is modified to exhibit them. This is a cyclic process which stops when the user and the SA feel that all requirements have been identified.
This process begins with the data communication specification of requirements, being accessed by the systems analyst. The manner of representation of the specifications of requirements is not essential as the above diagram depicts an abstract view. The access of documentation denotes that the specification of requirements is a document that the actor (system analyst) uses as input for the design of prototype activity. This activity could make use of formal methods\textsuperscript{23} and/or CASE tools\textsuperscript{24}. The multiple access denotes

\textsuperscript{23} REFER TO APPENDIX F \textsuperscript{24} REFER TO APPENDIX F, section F.5
that this document will be used as input by other activities. The output from this activity is a data communication, the *prototype specification*.

The next activity is the *building of the prototype* using the computer and software language or tools. The actor performing this activity is the system analyst, who utilizes the prototype specification as input. The two outputs from this activity are the *prototype user manual* and the *executable program of the prototype*. The outputs are used as inputs by the user for the next activity.

The prototype user manual is a document as indicated by the access of documentation, and the executable prototype is a software program that is executable as indicated by the *execution*. The inputs are used by the user for a hands on experience of the prototype. The output contains the changes requested by the user. This could contain all the functions that he feels are useful and are not present in the prototype, or if there are any functions that are irrelevant to him as the user of the final system.

The *request for changes* is a document that is accessed by the system analyst for the activity of *evaluating requests*. This activity is concerned with analysing the feasibility of the user’s requests in accordance with the resources, budgets, and requirements of the system. The system analyst produces a set of *questions* that requires feedback from the user.

The concurrent activities involved in the clarification and identification of alternate solutions to the user’s requests for change occurs next. The two activities are *alteration and alternative* performed by the system analyst and *clarification of requests* performed by the user. The former activity is concerned with the alteration of current requests and identification of alternative feasible options. The two activities occur over a period of
time and involve a lot of interaction between the system analyst and the user.

The user uses the questions as input and tries to answer and express his needs during the *clarification of requests* activity. The system analyst refers to the answers during the *alteration and alternative* activity to identify more questions if possible. He or she makes use of verbal or written means of expressing his queries. The resulting set of questions forms the output. This activity results in additional requests or modified versions of the user’s requests for change. When the system analyst and the user have reached a consensus relating to the expected and feasible changes to the prototype, a formalized version of the changes forms the output.

The next activity performed by the system analyst is the *updating requirements specification*. He or she accesses the *changes* document and the *specifications of requirements* document. Appropriate and desired tools are used during the activity to complete the update and a modified version of the specification of requirements is obtained. The extended symbol indicates the modification. The new *specification of requirements*+ document is subsequently used as the initial input for the next cycle of the prototyping process. The flow of control in each cycle of the prototyping process is denoted by the *flow of control* symbol.

---

25 The mode of communication of ideas between the system analyst and the user is immaterial to this thesis.
2.3.2 Prototyping process II

The systems analyst (SA) designs and builds the prototype based on the specifications. The user and the SA concurrently evaluate the prototype, discuss various changes, and additional features that are required. This is a cyclic process as well and it terminates when it is felt that the requirements got from the prototype quite accurately exemplifies the required system.

![Prototyping process II diagram]

Figure 8 Prototyping process II

The first two activities performed by the SA in Figure 8 are identical to the corresponding activities in the formalism represented in Figure 7. The next activity evaluation and clarification is composed of two parts: evaluating the prototype and clarifying the SA's proposals. The prototype user manual and the executable prototype
are used for the evaluate the prototype. The user identifies certain needs that require to be incorporated into the prototype and communicates the same to the SA. The SA evaluates these needs by referring to the specification requirements documentation. He or she forms appropriate queries to gain a better understanding of the user needs. The evaluate and clarification and evaluating requests activities occur concurrently. The user then tries to satisfy the SA’s queries either by verbal interaction or using the prototype to emphasize his or her points of view.

The two activities occur over a period of time and a set of changes that satisfy the user and the SA are produced as outcome. These changes, together with the specification of requirements document, are used in the updation of specification requirements activity. An updated version of the document denoted by specification of requirements+ is produced. The flow of control between the various activities is clearly indicated. The specification of requirements+ forms the input document for the next cycle of the process.
2.3.3 Prototyping process III

This prototyping process can be viewed as being composed of two parts represented by Figure 9 and Figure 10, where the second part is cyclic in nature. The first part, i.e. Figure 9, involves the design and building of the prototype by the user. This prototype is evaluated by the system analyst (SA), who relies on the user to clarify his or her doubts. The second part of the process involves the SA modifying the prototype to suit the changes in the requirements. The subsequent stages beginning with the user interacting with the prototype is similar to the corresponding steps of prototyping process represented by Figure 7.
The first two activities are similar to that of prototyping process illustrated by Figure 7 except that the actor for both activities is the user and not the system analyst. The third activity use of prototype performed by the system analyst makes use of the documents prototype user manual and specification of requirements, and the executable prototype. The output of this activity is the request for clarification document. Since the user has built the prototype to be used as a means to communicate his needs to the SA, the latter is required to understand the prototype before he proceeds with other activities.
The next activity is clarification of user requests, quite similar to that of the prototyping process I illustrated by Figure 7. The output of the process is used by the system analyst to update the specification of requirements document. The prototyping process IIIA terminates with the start of prototyping process IIIB.
Figure 10 illustrates the second part, namely prototyping process IIIB which is cyclic in nature. The specification of requirements document and the prototype specification document are used by the system analyst for evaluation and alteration of designs.
This results in a modified version of the prototype specification, namely the *prototype specification*.

The *prototype specification* document is accessed along with the *executable prototype* and the *prototype user manual* by the system analyst for the altering *prototype activity*. The outputs of this activity are *prototype user manual* and *executable prototype*. They are employed by the user for the next activity *use of prototype*, producing the *request for changes*.

*Request for changes* and *specification of requirements* are used by the system analyst for the activity of *evaluating requests*. The subsequent activities are similar to those in prototyping process I. The output of the final activity *updating requirements specification* is the *specification of requirements* document. This is used as input for the next cycle of the prototyping process. All subsequent cycles commence evaluation and altering of the prototype design based on the new specifications. The *specification of requirements* document replaces the *specification of requirements* document and the *prototype specification* document replaces the *prototype specification* document.
2.3.4 Prototyping Process IV

Figure 11 illustrates that the prototype, designed and built by the system analyst (SA), is evaluated. Inconsistencies and inaccuracies are identified, and feasible changes are discussed by the user and the SA. The entire process is cyclic and terminates with the prototype exhibiting all of the characteristics that are desirable.

The first two activities of the prototyping formalism IV in Figure 11 is identical to the corresponding activities of the prototyping process in Figure 7. The subsequent activity of evaluation and clarification of requirements is performed by the SA and the user simultaneously using the appropriate documents and software and producing an output that presents the feasible changes. The next activity is the updating requirements.
specification which modifies the specification of requirements document to specification of requirements document which is used as the input for the next cycle of this process.

§ 2.4 How the new notation was developed?

Over a period of three months, the formalism and notations underwent numerous changes. The following sequence of diagrams describe these refinements. It was felt that it would be prudent to keep the formalisms as abstract as possible in regard to data communications and implements used in each activity.

2.4.1 The first-notation to be developed

![Diagram of the notation process](image)

Figure 12 An example use of the first notation that was developed

The first notation, presented in Figure 12 that was developed was used to depict the process of designing and building a prototype based on the initial specifications. The prototype is evaluated by the systems analyst and the user. The systems analyst incorporates the changes into the system. This process is cyclic and the final output is the requirements specification document.
The drawbacks in the notations were the lack of proper input and output indication, and lack of uniform symbols. The evaluation activities and the other activities are not depicted using a standard symbol. For the evaluation activity, there is no indication of who performs an activity.

2.4.2 The first refinement to the notation

![Diagram showing the refined notations]

Figure 13 An example use of the refined notations

Figure 13 illustrates the use of a refined version of the notations. The input and output are denoted. The different kinds of arrows show that there is inherent differences in the various flow of information. The problem of using different symbols for the evaluation and other activities had not been rectified. The evaluation of the prototype activity indicates that both the systems analyst (SA) and the user access the prototype either independently or jointly. However, the activity *Prototype built by SA* in Figure
12 has been split into *design of prototype* and *prototype built by SA* in Figure 13. The modification activity now has an entity who performs it.

The final specification which is the output of the entire process is used in the systems design phase of the SDLC.

### 2.4.3 The second refinement to the notation

![Diagram](image)

**Figure 14** An example use of the second refined version of the notation

Figure 14 illustrates the refinements to Figure 13 with regard to the absence of the requirement specification activity. Here, the evaluation of the prototype is depicted as being accessed by the systems analyst and the user simultaneously. They interact and clarify any queries that might arise.
2.4.4 The third refinement of the notation

At this stage of refinement there has been a drastic change in the pictorial representation of activities, inputs, outputs, access, entities and flow of control. In Figure 15 the inputs are shown to be accessed by the activities and not by the entities which is the way it should be. Also, the factor of time between each activity is brought forth. The access of inputs by the entities is clearly depicted here.
2.4.5 The fourth refinement to the notation

Figure 16 An example use of the fourth refinement of the notation

Figure 16 illustrates each activity clearly. The entire process is depicted as a sequence of stages linked by the time factor. There have been some modification to the appearances
of the notations in Figure 15, but in essence they emphasize the underlying process. The tools used by each activity are incorporated in the diagram.

2.4.6 The fifth refinement to the notation

Figure 17 An example use of the fifth refinement of the notations

Figure 17 illustrates that the concept of stages is removed. Inputs and outputs are no longer boxed with each activity.
2.4.7 The sixth refinement to the notation

![Diagram of the sixth refinement to the notation]

Figure 18 An example use of the sixth refinement of the notation

Figure 18, the time factor symbol is replaced by an arrow to clearly depict the flow of control.

§ 2.5 Feedback from potential users of the new notation

In order to identify an appropriate formalism for describing prototyping processes, various notations were developed and tested by describing various prototyping processes.

A survey\textsuperscript{26} was prepared which contained one of the prototyping processes and completed by a number of persons over a period of three months.

The survey consisted of two parts: the first part was to study the clarity of the notations and the second part to study the expressiveness of the notation.

\textsuperscript{26} REFER TO APPENDIX B

43
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

The subjects in the survey included undergraduate and graduate students in Computer Science and other disciplines, faculty and persons working in the software industry.

The feedback from the survey can be obtained from Dr. Richard Frost, School of Computer Science, University of Windsor. The results after analysis are presented in tabular form in Appendix C.

2.5.1 Result of analysing feedback

Analysis of the results of the survey shared that:

- A number of respondents had difficulty with the +, *, SA. concurrent activity, and execution symbols. Only a few respondents had some difficulty initially, with respect to the prototyping process described.

- The number of respondents with difficulty with the symbols decreased after reference to the key to the notations. The textual description provided removed all doubts with respect to the prototyping process and symbols being used to represent them.

- From the survey, it become apparent that the concurrent activity symbol still posed some problems for some of the respondents.

However, it was felt that these difficulties could be overcome through explanatory notes and did not warrant modification to the notation.
Chapter 3  WELL-DEFINED METRICS FOR ANALYSING PROTOTYPING PROCESSES

In any field, there is a need to understand the components and their relationships. Software engineering is in its adolescence and, therefore, merits an experimental method of analysis. Experimentation involves an iteration of hypothesis and test process. "Models of the software process or product are built, hypotheses about these models are tested, and the information learned is used to redefine the old hypotheses or develop new ones" [5].

If the cost and value of prototyping in requirements analysis is to be fully understood, it is necessary to experiment and therefore to identify some well-defined metrics by which experience in using prototypes can be analyzed.

§ 3.1 What is a metric?

A study of what we can measure and how we can measure it, is known as metrics [28]. According to Sommerville [65], a software metric is any measurement which relates to a software system, process or related documentation.

Pressman [58] states that software is measured for the following reasons:

- To indicate the quality of the product.
- To assess the productivity of the people who produce the product.
- To assess the benefits of using new software methodology and tools.
- To form a baseline for estimation.
- To help justify requests for new tools or additional training.
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

Measures can be divided into direct measures and indirect measures. The cost and effort needed to build software, the number of lines of code, and other direct measures are easy to determine if specific conventions for measurement are established in advance. Quality, functionality, efficiency, and maintainability are comparatively more difficult to measure. Software metrics are further categorized into the following [58]:

- **Productivity metrics** that focuses on the output of the software engineering process.
- **Quality metrics** that indicate the extent to which the product adheres explicitly and implicitly to the customer requirements.
- **Technical metrics** is useful in measure of the logical complexity, degree of modularity, and other aspects of the software rather than on the process of the development.
- **Size-oriented metrics** are used for measuring the output and quality.
- **Function-oriented metrics** provide indirect measures.
- **Human-oriented metrics** is used to measure the effectiveness of the methodologies and the tools used by the developers.

Pfleeger [56] has described the use of metric databases containing various information about completed projects at the company Contel. These databases have been used to help compare the initial estimates of software projects with final costs. Corporate historical databases are defined as more than a simple collection of individual project databases, by Pfleeger. The databases held provided sufficient data for detailed analysis, including the analysis of information to evaluate corporate trends. According to Pfleeger, the data from programming projects could have been compared with the data from non-programming projects to analyse whether the environment improved quality or productivity.

---

27 A precise definition of what the system is to do. What logical transformations that the system logic has to perform, when they are performed and on what data?
The paper by Pfleeger is the only one in which there is any evidence of a certain degree of success associated with use of metrics. However, she feels that the metrics should be tailored to suit each project, i.e., only certain metrics can be measured in certain applications. Pfleeger recommends that simpler metrics are more likely to be recorded by the software developers. "It is not only natural but desirable for different projects to collect different metrics. There will probably be a large core of common measurements to provide sufficient data for organizational evaluation. But the focus should be on solving project and process problems first, with institutional or organizational problems to be addressed later [56]."

§ 3.2 
Some "metrics" that are often referred to in texts on software analysis

The following lists some of the metrics that are frequently referred to in texts on software engineering. While some of these metrics are clearly ill-defined, and others are irrelevant to prototyping process, we include them all to indicate the wide range of potential measurements and factors that were considered before an appropriate set of metrics was identified.

Technology (resources) used in developing software:

— Hardware platform.
— Operating system.
— Identification of appropriate software needed for the development.
— License cost of the software.
— Ease of use of the chosen software.
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

Manpower needed for the development of the software:

— Total time spent by end-user in project.
— Total time spent by system analyst in project.
— Time spent learning to use the software.
— Time spent on programming.

User satisfaction.

Type of application.

Size of application and its complexity.

Reuse of existing software.

Appropriate Documentation.

Conforming to budget.

Efficient use of resources.

Reduced maintenance.

Product quality:

— Reliability.
— Efficiency.
— Speed.
— Functionalities of the system addressed.
— User Interface.

Cost incurred due to misunderstandings by the system analyst and the user about the software product specifications.
Differences identified between the initial specifications and the final specifications:

— Number of errors detected in initial specifications.
— Additional features identified.

Functionalities of the system that were requested vs. those that were feasible.
Discrepancies in understanding the required functionalities.
Number of lines of code.
Amount of information on data required by the final product.
Effective use of prototyping technique in software development:

— Problems identified with use of the particular prototyping tools.
— Number of iterations of the process.
— Screens frequently accessed.
— Frequency of accessing a particular screen is accessed.

§ 3.3 How an appropriate set of metrics for analyzing prototyping projects was identified

The views and experience described in section 3.2 provided some understanding of the metrics that might be appropriate for analysing prototyping projects:

• To be useful, a metric must be well defined, i.e., described unambiguously.
• Metrics must be measurable, i.e., measured with reasonable resources and with acceptable interference of the process of identifying requirements. Although obvious, this is very important. A well defined metric is of limited value if it is difficult to measure.
• Metrics can be categorized as direct or indirect. By the nature of prototyping, it is only feasible to measure direct metrics during a prototyping project.
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

- Simple metrics are more likely to be recorded by software developers.
- It may be desirable to record different metrics for different projects depending on the application area.

According to Schach, software quality and reliability are not important when building a prototype. Therefore it would not be appropriate to record metrics that are concerned with these issues. So, what kind of metrics should be recorded during prototyping? Before we can answer this question, it is important to identify the use to which the metrics will be put:

A. To determine the cost of using a prototype in requirements analysis.
B. To determine the value of using a prototype in requirements analysis.
C. To analyze the inner workings of the prototyping process itself.

In the thesis work, emphasis is placed on those metrics which could be used to support analysis C. This is necessary because many of the metrics for tasks A and B would require a longer time period than was available for this work.

In order to identify well-defined measurable metrics to support task C, the following approach was taken:

- All of the metrics listed in Section 3.2 were considered and a number were rejected immediately as being ill-defined or too difficult to measure.
- A list of proposed metrics was created.
- A case study was undertaken to determine the viability of measuring the proposed metrics.
- Three prototypes were built in three different programming environments and appropriate metrics related to the prototype itself were identified.
§ 3.4 Metrics that were identified

In this section, we describe the final set of metrics that were identified as being relevant to the prototyping process, well-defined, and measurable.

3.4.1 Cost of building tools.

The cost of hardware, firmware, i.e., operating systems, and software, i.e., programming language, used for building the prototype can be measured directly.

3.4.2 Length of informal specifications given in English.

The value of this metric is determined by the number of words in the specifications. The specifications are required to be as detailed as possible.

Length of the initial specifications. This metric is used for measuring the length or size of the specifications, irrespective of whether it is represented formally or informally.

Length of the final specifications. The length of the final specification represented either formally or informally, but in the same manner as the initial specifications. This metric can be used to measure the change in the system.

3.4.3 Total number of iterations of the prototyping process.

The number of iterations needed to gather requirements using the prototype is required to measure the complexity of the system. The experience of the analyst and the other resources used also plays a part in the eventual value of this metric.

3.4.4 Total number of reports required initially.

The outputs or reports that the user needs the final system to generate indicates the nature of the system. Since business applications involve the generation of reports for
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

management decisions. The initial reports could in many cases be the same as in the current system used in that organization.

3.4.5 Total number of reports required finally.

The number of final reports required gives an indication to the complexity and importance of the data being processed.

3.4.6 Number of lines of code.

The measurement of this metric can pose problems when one considers the various programming languages, concepts, and applications built using re-engineering techniques.

Number of lines of code of the initial prototype. The number of lines of code that make up the initial prototype is measured to get an indication for the size and complexity of the system being built.

Number of lines of code of the final prototype. The number of lines of code that make up the final prototype is measured to get an indication for the size and complexity of the system being built.

3.4.7 Number of iterations of prototyping process cycle.

The means of measuring this metric is straightforward.

3.4.8 Number of questions posed by the user.

These are any questions regarding the prototype and the system to which the user wishes clarification. These questions can be either answered by interacting with the prototype or verbally by the system analyst.

28 Reuse of code.
3.4.9 Number of questions answered by the analyst.

There are the questions that were answered by the analyst verbally while the user interacted with the prototype.

3.4.10 Number of questions answered by the prototype.

These are the questions that were answered by the prototype, i.e., the user interacted with the prototype and was able to have his queries answered without the help of the analyst.

3.4.11 Number of failures reported by user.

Any functionality that is lacking in the prototype and any graphic user interface of the prototype that fails to satisfy the user is identified as a failure. The value for this metric is user dependent.

3.4.12 Number of failures corrected by the system analyst.

This gives a numeric value to the number of failures identified by the user that the system analyst corrects. Some of these failures include error checking during data entry, screens not having required data entry options, screens not displaying required data, absence of required screens, and absence of required menu options.

3.4.13 Number of menu options: before the prototyping process.

The number of menus that are present in the system gives an indication of the expected degree of interaction between the system and the users. The various components of the menus that can be measured are the:

- Maximum depth of the menu options: the number of sub-menus available before a data entry or data display screen is reached.
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

- Average number of options on the menu: the options represent the choices that are available on each menu. This metric gives the average number of options offered to the user of the prototype.

- Number of screens: this metric gives the value of the total number of data entry and data display screens available to the user to manipulate the data stored.

- Total number of menus traversed: The number of menus that are linked in order to help the user go to a desired option without having to backtrack all the way to the menus higher on the hierarchy providing the required menu.

3.4.14 Number of menu options: after the prototyping process.

The number of menus that are required after the user has interacted with the prototype gives an indication of the changes that the user requires and the effectiveness of the prototype. The various components of the menus that can be measured are the same as the ones listed in the previous metric.

3.4.15 Number of times each menu options is used.

The term menu option is used to identify the number of user interface screens that provide menus to travel through the system. The various sub-components of the menu options are:

- Maximum depth of the menu option: the number of submenus present before reaching a screen.

- Average number of options on the menu: the various choices listed on each menu.

- Number of leaf nodes: the final menu screen.
• Total number of menus traversed: the link between the various menus. For example, the option to choose enter new input after generating reports without having to backtrack to the main menu to reach the required options.

This metric is used to measure the frequency of use and hence, the importance of the various menu options.

3.4.16 Number of screens.

These are the user interface screens other than the menu option screens. These include the error message screens, data entry screens, and screens to view the required data.

3.4.17 Number of times each screen is used.

This metric is used to measure the frequency of use and hence, the importance of the screens in the prototype.

3.4.18 Number of times each menu option is chosen accidentally.

The metric tries to obtain the value of the user choosing the wrong screen, due to either lack of familiarity with the system or due to erroneous menu options. Number of times each menu option is chosen accidentally, i.e., Number of times the previous screen option in each menu screen was chosen:

• Without accessing the screens for data entry or data display.
• After accessing the screens for data entry or display.

3.4.19 Ill-defined metrics

Many texts give convincing arguments concerning useful metrics. For example “user satisfaction” is an obviously important feature of any software system. But on a more careful consideration, this feature is clearly not a metric. How can one measure “user
satisfaction” with any consistency. According to Glass, “user satisfaction” is quantifiable when a system is considered by the user to adequately satisfy her or his needs. Other ill-defined metrics that are frequently referred to are:

- Functionality of the system.
- Product quality.

In this thesis work, an attempt was made to avoid such ill-defined metrics.

§ 3.5 Reasons why the metrics chosen may be useful

This thesis is concerned with the identification of well-defined metrics that can be viably measured not with the determination of the value of these metrics. But in order to identify sensible metrics some attempt was made to consider only those metrics that appeared to have some potential value. For example:

- The number of lines of code of the prototype, the programming language used, and the extent to which the prototype provides the functionality of the required system, could together give some indication of the complexity of the final product.
- The difference between the number of reports initially identified and the number of reports required after the prototyping process might be included in the estimate of the value of the prototyping process itself.

Forms were created to be used to enter values in a systematic manner for the metrics identified. It was possible to divide the useful metrics into:

- A set of metrics measured during interactive sessions between the user and the prototype.
- A set of metrics that can be derived from the directly measured metrics.
The experience gained in undertaking the case study suggests that in most cases the value of metrics may not be of use in deriving some numeric values by which prototyping projects can be quantitatively summarized or directly compared, but rather, in providing well-defined descriptions of the details of the specific use of a prototyping process. Such well-defined descriptions are one step in the direction of obtaining quantitative analyses of the cost and value of prototyping projects, but, without some standard languages, platforms and practices, the direct quantitative comparison between prototyping projects using any metrics is impossible.

However, use of metrics for quantitative analysis and comparison may be possible in large organizations which have standards to be adhered to in developing software.
Chapter 4 A CASE STUDY

§ 4.1 Introduction

A student records system was used as the case study. The prototyping process used in this case study is depicted in Figure IV in Chapter 2. Three different programming languages were used to develop three alternative prototypes.

4.1.1 Use of natural language for initial specifications

The specification of the system developed in this case study was recorded informally, using English, as a means of representation.

An experiment was conducted to obtain the specification of the student record system. Three Masters students in Computer Science and a PhD student in Electrical Engineering were involved in this experiment. The user verbally described the system that was required, and 4 separate specifications were written in natural language by the students. The table below represents the number of words used by each of the four students to represent the system. The specifications present only the function of the system without reference to implementation.

<table>
<thead>
<tr>
<th>Analyst</th>
<th>Number of Words</th>
<th>Normalised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arunita</td>
<td>275</td>
<td>100</td>
</tr>
<tr>
<td>Arindam</td>
<td>199</td>
<td>72.36</td>
</tr>
<tr>
<td>Farook</td>
<td>263</td>
<td>95.64</td>
</tr>
<tr>
<td>Arathi</td>
<td>254</td>
<td>92.36</td>
</tr>
</tbody>
</table>

Table 1 Specifications in natural language

29 REFER TO APPENDIX A
Surprisingly, the four different people produced informal specifications of the same requirement that are approximately of the same length. This suggests that it may be worthwhile to pursue the relationship between natural language specification and other aspects of required systems.

### 4.1.2 Formal description of the prototyping process used in this case study

The prototyping process that was used in this case study is given below.
4.1.3 Three languages used to build the prototype

Prototype developed using SQL/DS  SQL/DS — Structured Query Language/Data Systems was developed by IBM to manage databases. A database is a collection of interrelated or independent pieces of information stored together without unnecessary redundancy, to serve one or more applications. SQL is a 4GL or fourth generation language. Its primary use is to query information stored in a set of tables. Its user interface facilities are minimal but adequate.

SQL/DS automatically maintains information about the database in a set of tables called catalogs. They describe tables, columns, authorizations and other objects in your system. It provides a structured set of requests or queries with which one can access the data.

The system developed using SQL is not menu driven, i.e., user interfaces in the form of menus to guide the person using the system. SQL/DS is available on the CMS Mainframe at the University of Windsor.

The code for the prototype is presented in Appendix G, Section G.1.

Prototype developed using C  The programming language C was developed around 1972 by Dennis Ritchie as part of the Unix Operating System. Though C was originally developed for implementing the Unix operating system, C is now used for a wide range of applications. It has a number of control structures and operators which make it a flexible programming language.

The philosophy of C is to provide the programmer with as much power as possible to control the execution of the program and encourage compactness of expression.
C is a very small language with a large array of optional library functions. It permits very low-level access to hardware for systems software and provides very high-level control constructs for applications programming.

American National Standards Institute [ANSI] C is available on the SUN workstation at the University of Windsor.

The input/output library in C was used extensively to build a menu driven prototype of the required system.

The code for the prototype is presented in Appendix G, Section G.2.

Prototype developed using C++ C++ was invented by Bjarne Stroustrup in 1980 at Bell Laboratories, New Jersey. It was initially called "C with classes", and subsequently called "C++" in 1983. C++ underwent two major changes, once in 1985 and second in 1990. C++ is an high-level programming language that is based on the object-oriented concepts. It is a language that is under development, and many more changes can be expected to make it more of an object-oriented language than what it is currently.

The AT&T version of the C++ language is available on the SGI mainframe at the Computer Centre at the University of Windsor. It was possible to make use of the input/output facilities provided in this language to develop a menu-driven prototype.

The code for the prototype is presented in Appendix G, Section G.3.

§ 4.2 Experimental sessions with the prototypes

The interactions of the user with the prototype can be classified according to the user interface available, i.e., non menu-driven and menu-driven. The metrics relating to the prototyping process in form II presented in Appendix E were measured.
4.2.1 Experimental session with the SQL prototype

The prototype built using SQL was used to measure the metrics listed in the form II in Appendix E. Since the prototype was not menu-driven, the metrics relating to menus were dropped from the template. This prototype went through two iterations. The user interacted with the prototype along with the analyst. Initially, it was felt that the analyst would record the metrics, the questions raised by the user relevant to the functionalities of the system, and any additional requirements identified by the use of the prototype. However, as it was not a menu driven prototype, the user had to rely on the analyst and refer to the documentation provided to find answers to even the most simple queries that should have been answered by the prototype. The user felt more comfortable interacting with the analyst to obtain the answers. The examples provided in the documentation helped the user realize the capabilities of the prototype and also identify other related requirements.

The following is an example of the screen available on SQL:
The user was unhappy with the lack of customized error and help messages while using the prototype since, SQL which does not have any features to construct appropriate user interface. SQL generated error messages — syntax and semantic — were totally unsuitable. Some of the SQL constructs to generate the required data were useful since the user was able to clarify the requirements and stimulate the identification additional requirements. As query needed to interact with the prototype required the use of the table names and field names used, the user got interested in the structure of the database.

However, most of the outputs required the execution of complicated SQL query commands. The queries were long and required the user to have a through working knowledge of the database being accessed — the table and field names. The user felt that the less his or her use of complicated query commands the better, as it was tedious and time consuming to enter.

In addition to the initial specifications, two new requirements were identified by the user and some of the requirements in the specifications were clarified as discussed below.

The additional requirements identified by the user while interacting with this prototype:

- The need to have error checking for data input.
- Help and error message to be displayed appropriately.

Some of the requirements clarified by the user are:

- Flexibility to modify the scale for assigning grades.
- Flexibility for user to use his or her discretion in calculating the final grade for students who were absent for any particular test or assignment.
The metrics that could be measured from this prototype are found in Appendix E.2. The metrics identified were not easy to measure for this prototype. The lack of user interface eliminates many of the metrics. Due to slow response time, the user had time to explain his or her initial requirements.

No other iteration was done since the:

- User interaction with the prototype was not easy due to lack of appropriate interface screens.
- User rapidly lost interest in using the prototype due to slow response of the system.
- User found that he or she was relying on the documentation and the analyst than the prototype, to obtain answers about the system.

In conclusion, it was apparent that SQL was useful to build the database, but its drawback was in not supporting the development of user interface, since:

- The features of SQL/DS are geared to build a database.
- It has in-built query commands to manipulate and retrieve data.
- It can be considered as an Example Generator. If the probable queries and subsequent display of data were provided in the prototype's user manual, it would have been much faster and beneficial for the user to interact with the prototype on paper than on a console.

4.2.2 Experimental session with the C prototype

The prototypes built using C and C++ programming languages are menu-driven. The form was used to measure the metrics using the prototype built in C. Since the interface available on C++ prototype is possible in the one built using C, the C++ prototype was
not used. Only the C prototype was used in interaction with the user. The C++ was analyzed in terms of length of code and cost of software.

The user interacted with the C prototype in the presence of the analyst. The user found this prototype more appropriate to interact with than the SQL prototype. Error checking and the help facility provided for data entry helped the user clarify certain aspects about the data being entered. Due to the fast responses by the prototype the user’s interaction with it, his or her interest with the prototype increased. This prototype helped the user identify and clarify a number of requirements mainly due to the interface screens available. In fact, the user required the menu screens to be completely modified. This prototype maintained the user’s interest in the external interface features of the system, unlike the earlier prototype that got him or her interested in the internal features.

The following is an example of the type of screens available in this prototype:

![Example Screen]

The additional requirements identified by the user interacting with the two prototypes were:

- The need to have error checking for the data entered.
- Appropriate help and error message provided on the screen.
- Flexibility in changing the weights attached to each component of the grades, the value range for each such component, the grade scale, and to override the manner in which the grade was calculated for certain students.
The format for all the menus and screens were given by the user.

It was possible to measure nearly all of the metrics identified. Since the user was interacting directly with the prototype, the user interface for the system was changed considerably to suit the user's needs. The additional requirements identified have been listed above. Difficulty in measurement was due to the fast response time of the system which helped the user ask numerous questions about the system. Despite the fact that no data was available for this prototype, a number of requirements were identified by the user. The metrics that could be measured from this prototype are found in Appendix E.3.

§ 4.3 Analysis of the prototypes

Two sessions with the end-user were used to record metrics in the case study. In the first session, the **non-menu driven** prototype was used. In the second session, the **menu driven** prototype was used.

The 2 prototypes can be analyzed based on the metrics that can be measured. The cost of building tools can be measured directly for both the prototypes based on the operating systems and the license cost of the languages.

The number of iterations of the prototyping process cycle involves the session between the user and his prototype. This measurement is straightforward.

The number of lines of code of the prototypes depends much on various programming languages, concepts, and re-use of applicable software. The SQL prototype has a number of query constructs that are part of the language, whereas, the C prototype consists of a number of functions.
The number of words of the specifications increased after the experimentation. Though the final tally is nearly the same, the requirements added are not the same from the two prototypes. The additional requirements were appended to the specification IV in Appendix A, topic A.1.2 and are presented below.

The changes to the specifications after the experimentation with the SQL prototype are:

- Error checks during input of data.
- Display of error and help messages.
- Provision to change weights attached by arbitrary amounts.
- User friendly table names.
- Alias for Tables.
- Provision to change certain fields.
- Provision to view required fields.
- Provision to override computer generated grades and replace them with what the user enters.

The changes to the specifications after the experimentation with the C prototype are:

- Error checks during input of data.
- Display of error and help messages.
- Provision to change the weights attached as required.
- Provision to specify the value range for the weights.
- Provision to change the grade scale.
- Flexibility to change the manner of grade calculation for specific students.
The reports that were required initially:

The reports that were required finally after experimenting with the:

- SQL prototype:
  a. List of courses.
  b. List of students.
  c. List of student grades.
  d. Name of the relations — tables, field, etc. to help the user in queries.

- C prototype:
  a. View of all students in a particular course.
  b. View of all students in all courses.
  c. View of particular students in all courses.
  d. View of particular students in a particular course.

The questions posed by the user while interacting with the:

- SQL prototype:
  a. How is the prototype initiated?
  b. What are the names of the tables?
  c. What was the reason behind choosing these names?
  d. What are the queries that can be tried out?
  e. Are there any help messages?
  f. Are there any error messages?
  g. Is there any error checking facility available?
  h. Is there any error checking for data types?
i. How can the tables be viewed?

j. Is it possible to list out the tables?

k. Is it possible to list out the fields of each table?

l. Is it possible to rename the tables?

m. Is it possible to create aliases for these tables names?

n. Is it possible to change the structure of the tables?

o. Is it possible to change the gap between the grades by an arbitrary amount?

p. Can the data be stored in a sorted order despite the order in which the data is entered?

q. Can the ordering be done in ascending or descending?

r. Is it possible to customize the "views" to view the data?

s. Is it possible to customize the manner in which the grades are calculated for each course?

t. Is it possible to sort the student records by major?

u. Can different weights be used for the grades?

v. Can the grades be generated according to the format of courses with respect to the tests and assignments?

w. Is it possible for the user to override the computer generated grades?

• C prototype:

a. Does it have error checking facility?

b. Does it have help messages?

c. Does it have error messages?

d. Is it possible to insert records for adding new students to a course?
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

e. Is it possible to insert a particular record for adding new students to a course?
f. Can the required data be sorted?
g. Can the required data be ordered by any user specified key?

The questions denoted by a.b.c.d.e.f.i.j.k.l.m.n.p.r.s.t.u, and w as listed above were answered by the analyst while the user interacted with the SQL prototype. Similarly, the questions denoted by a.b.c.d.e. and g were answered by the analyst while the user interacted with the C prototype.

The questions denoted by g,h.o.q, and u were answered by the prototype while the user interacted with the SQL prototype. Similarly, the questions denoted by f was answered by the analyst while the user interacted with the C prototype.

The failures reported by the user while interacting with the:

- SQL prototype:
  a. Lack of error messages.
  b. Lack of help messages.
  c. No error checking facility.
  d. Inflexibility to view the required data.
  e. Queries are too long to enter.
  f. Prototype is too slow to interact.
  g. Certain requirements, namely a pictorial representation of histogram not available.
  h. Absence of flexibility to shift the grade gaps by arbitrary amount.
  i. Lack of flexibility of prototype to allow the user option to override certain data manipulations as required, namely grades calculation.
j. Inability to view all the grades in a certain course. This is because complicated manipulation was required to calculate the grades. The list of grades had to be viewed as two lists, one with the F grades and the other with non-F grades.

• C prototype:
  a. Lack of error messages.
  b. Lack of help messages in general.
  c. Lack of specific help messages.
  d. No error checking facility.
  e. Required histogram not available.
  f. Inability to correct data entry mistakes, by jumping to previous data field.
  g. Inability to enter an arbitrary number of student records.
  h. Menu options are too confusing.
  i. Screens do not contain satisfactory data entry fields.
  j. Insufficient data components displayed.
  k. Update screens are not user friendly.
  l. Unable to specify constraints for certain fields in a course.
  m. Unable to specify order by which data should be sorted before display.
  n. Unable to specify ranges for the fields that hold numeric values.
  o. Lack of user friendly options to enter data for particular student record.
  p. Lack of user friendly options to enter data for all student records.
  q. Lack of user friendly options to enter data for particular course record.
  r. Lack of user friendly options to enter data for all course records.
  s. Lack of user friendly options to view data for particular student record.
t. Lack of user friendly options to view data for all student records.

u. Lack of user friendly options to view data for particular course record.

v. Lack of user friendly options to view data for all course records.

The metrics relating to menus and screens were relevant to the C prototype and are discussed below.

The number of menu options before the experimentation were:

- Main menu.
- Input data menu.
- Update records menu.
- Reports menu.
- Student menu.
- Create new records menu.

The number of menu options after the experimentation were:

- Main menu.
- Input data menu.
- View data menu.
- View various student list menu.
- View a particular student menu.

The sub-menus and screens that were viewed by the user were:

- Main menu.
- Histogram for courses.
- Screen to view list of various students.
The following table summarizes the metrics and the measurements obtained.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>SQL Prototype</th>
<th>C Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of informal specifications given in English:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Length of the initial specifications.</td>
<td>254</td>
<td>254</td>
</tr>
<tr>
<td>* Length of the final specifications.</td>
<td>305</td>
<td>300</td>
</tr>
<tr>
<td>Total number of reports required initially.</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Total number of reports required finally.</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Number of iterations of prototyping process cycle.</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Number of questions posed by the user.</td>
<td>23</td>
<td>7</td>
</tr>
<tr>
<td>Number of questions answered by the analyst.</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td>Number of questions answered by the prototype.</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Number of failures reported by user.</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>Number of menu options : before prototyping process.</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>Number of menu options : after prototyping process.</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>Number of times the menu options were used.</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Number of times screens were used.</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Number of times each menu option is chosen accidentally.</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2 Analysis of the prototypes

From Table 2 it is possible to see that it is possible to viably measure the metrics.

§ 4.4 Findings during the case study

4.4.1 Use of natural language for initial specifications

The representation of the initial specifications in natural language in the case study by 4 persons in approximately the same length suggests that the use of natural language for specifications should be investigated further.
However, the length of a natural language specification may only be relevant if and only if each requirement of the system is stated explicitly and completely in the specification. For example, reference to other similar systems would clearly produce shorter specifications than if such references were not made. Also, the experience and background of the analyst will affect the length of the specifications. The graduate students who were involved in this experiment had no previous experience writing specifications. So, there may be correlation between the length of the specifications and the experience of the analyst.

4.4.2 Use of different languages for prototyping

Three different languages were used to build the prototype for the case study in this thesis. The 2 prototypes that the user interacted with highlight the different aspects of the system that were focused on by the user. The ability of the prototype to capture and retain user interest, elicitation of requirements, and clarification of existing ones depend on the features of the language. The following table presents a summary of use of SQL and C languages in developing the prototypes to support this thesis.
<table>
<thead>
<tr>
<th>SQL prototype</th>
<th>C prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample database was created.</td>
<td>No database was created.</td>
</tr>
<tr>
<td>No user interface was available.</td>
<td>User interface was part of the prototype.</td>
</tr>
<tr>
<td>No data entry screens were present.</td>
<td>Data entry screens were available.</td>
</tr>
<tr>
<td>Data entry was possible using SQL input commands.</td>
<td>Data entry was possible using the screens.</td>
</tr>
<tr>
<td>No error messages provided.</td>
<td>No error messages provided.</td>
</tr>
<tr>
<td>No help messages provided.</td>
<td>Some help messages provided.</td>
</tr>
<tr>
<td>Tedious to enter long queries and that requires a working knowledge of the commands and the database.</td>
<td>Menu options were available.</td>
</tr>
<tr>
<td>Slow response time.</td>
<td>Fast response time.</td>
</tr>
<tr>
<td>Did not retain user interest, but elicited some new requirements.</td>
<td>Maintain user interest and elicitation of new requirements.</td>
</tr>
<tr>
<td>Useful as a Example Generator on paper.</td>
<td>Useful as an on-line Screen Generator.</td>
</tr>
</tbody>
</table>

Table 3 Summary use of the prototypes.

The cost of software needed to maintain the system in a required platform and the programming language depends on the resources of the organization. Refer to metric 3.4.1 in Chapter 3.

The number of lines of code appears to be easily measurable in theory but in fact, the length of code depends on the various programming languages, concepts, and re-use of applicable software.

The usefulness of eliciting further user requirements and clarifying existing ones, would appear to depend to a great extent on the user interfaces available in the prototypes.
as seen in this case study. The availability of such user interfaces depends on the programming language used.

4.4.3 Viability of measuring metrics identified

The metrics that could be measured depended largely on the user interfacing capabilities of each prototype. It was virtually impossible to obtain any meaningful measure for some of the metrics in the experimentation with the SQL prototype. The metrics pertaining to menus, screens, and others had to be ignored for this prototype.

In the experimentation with the C prototype, it was possible to obtain measures for the metrics. The problem encountered was in keeping track of all the user questions and at the same time updating the measures.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Viability of measuring the metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost of building tools.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Length of informal specifications given in English.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of iterations of prototyping process.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Total number of reports required initially.</td>
<td>Easy to measure once the required reports are identified.</td>
</tr>
<tr>
<td>Total number of reports required finally.</td>
<td>Easy to measure once the required reports are identified.</td>
</tr>
<tr>
<td>Number of lines of code.</td>
<td>SQL Prototype</td>
</tr>
</tbody>
</table>

Table 4 Viability of measuring the metrics. (Continued) . . .
<table>
<thead>
<tr>
<th>Metric</th>
<th>Format aids the measure.</th>
<th>Coding format aids measure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of iterations of prototyping process cycle.</td>
<td>Easy to measure.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of questions posed by the user.</td>
<td>Easy to measure.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of questions answered by the analyst.</td>
<td>Easy to measure.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of questions answered by the prototype.</td>
<td>Easy to measure.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of failures reported by user.</td>
<td>Identification of aspects that do not satisfy the user.</td>
<td>Identification of aspects that do not satisfy the user.</td>
</tr>
<tr>
<td>Number of menu options: before prototyping process.</td>
<td>Menus not available.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of menu options: after prototyping process.</td>
<td>Menus not available.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of times the menu options were used.</td>
<td>Menus not available.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of times screens were used.</td>
<td>Menus not available.</td>
<td>Easy to measure.</td>
</tr>
<tr>
<td>Number of times each menu option is chosen accidently.</td>
<td>Menus not available.</td>
<td>Easy to measure.</td>
</tr>
</tbody>
</table>

Table 4 Viability of measuring the metrics.
4.4.4 Conclusions

From the above findings it is possible to conclude:

- Metrics can be measured depending on the programming language.
- The 2 prototypes appear to be useful in identifying new requirements, but in different ways.
Chapter 5  CONCLUSION

§ 5.1 Analysis of findings

The thesis to be defended in this report consists of three statements:

1. Well-defined graphical notations can be used to describe various prototyping processes.

2. It is possible to identify well-defined metrics by which prototyping processes can be analyzed.

3. Prototyping processes used in requirements analysis are amenable to empirical study.

We now consider each of these statements in turn discussing the extent to which the work undertaken supports them.

Well-defined graphical notations can be used to describe various prototyping processes.

The work undertaken to provide evidence in support of this aspect of the thesis included the following:

- A literature survey on *Computer Assisted Analysis and Design of Database Systems*\(^{30}\) was referred to in order to determine the different kinds of notation that have already been used to describe prototyping processes.

- A set of well-defined notations was developed to graphically represent various prototyping processes\(^{31}\).

- Various prototyping processes were identified.

\(^{30}\) REFER TO APPENDIX F

\(^{31}\) REFER TO CHAPTER 2, SECTION 2.2
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

- The notations were used to graphically represent the various prototyping processes\textsuperscript{12} that had been identified.

- A survey was conducted to study the ease of use of the notation developed\textsuperscript{13}.

It was found that:

- No single method for representing all prototyping processes was identified in the literature survey. That is, no one prior to this work had attempted to undertake the task of representing the prototyping process in a well-defined manner.

- The new notation that was developed could be used to describe all prototyping processes that were identified.

- The survey of users of the new notation enabled it to be modified until it clearly met the requirements of unambiguity and well-definedness.

- The respondents of the survey were unable to come up with a process that could not be represented using the notation.

The conclusions that can be drawn are:

- Well-defined graphical notations can be created to describe prototyping processes.

- Also, there is some indication that the prototyping processes presented in this report are representative of all possibilities, and therefore that the notation developed can be used to describe all prototyping processes.

It is possible to identify well-defined metrics by which prototyping processes can be analyzed.

In order to support this aspect of the thesis, the following tasks were completed:

\textsuperscript{12} REFER TO CHAPTER 2, SECTION 2.3

\textsuperscript{13} REFER TO APPENDIX C and Forms Completed in this

80
- A study of the literature on quality assurance and metrics relating to software development was carried out.
- A number of metrics presented in the texts relating to software development were identified.
- A preliminary use of the prototype developed in SQL by the analyst helped identify additional metrics.

It was found that:
- Some metrics proposed by others and which appear reasonable at first are difficult if not impossible to measure because they cannot be defined with sufficient clarity\(^{34}\).
- It was possible to identify 19 metrics that are well-defined.

The conclusions that can be drawn are:
- That it is possible to identify well-defined metrics by which prototyping processes can be analyzed has been clearly supported.
- A great deal of careful consideration is necessary to analyze the suitability any metric identified.

**Prototyping processes used in requirements analysis are amenable to empirical study.**

The work undertaken to provide evidence in support of this aspect of the thesis included the following:

- A data collection sheet\(^{35}\) for recording values of the metrics was developed.
- An experiment for representing specifications in English\(^{36}\) was conducted.

\(^{34}\) REFER TO CHAPTER 3, SECTION 3.3, TOPIC 3.4.19  \(^{35}\) REFER TO APPENDIX E  
\(^{36}\) REFER TO APPENDIX A
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

- The resulting specifications from the experiment were analyzed\textsuperscript{37}.
- A prototype was developed in three programming environments: SQL, C, and C++\textsuperscript{38}.
- Metrics were recorded while the user interacted with the prototypes\textsuperscript{39}.
- New user requirements were identified and recorded.

It was found that:

- The set of metrics identified in Section 3.4 could be measured. However, it was also found that the measurement did interfere significantly with the prototyping process. Since every feedback from the user had to analyzed to identify what metric value it was.
- The data collection sheet mentioned above was not very useful for the collection of data during the interactive sessions. In particular, it was difficult for the analyst to keep record of what was happening during the session and at the same time increment the value of the appropriate metric.
- There was an unexpected consistency in the length of the English specifications.
- The different programming languages used to develop the prototypes highlighted the various aspects involved in the prototyping process\textsuperscript{40}.

The conclusion that can be drawn are:

- Prototyping processes used in requirements analysis are amenable to empirical study.
- The metrics and the identification of certain additional requirements are dependent on the nature of the programming environment.

\textsuperscript{37} Refer to Chapter 4, Section 4.1.1
\textsuperscript{38} Refer to Appendix G
\textsuperscript{39} Refer to Appendix E, Section E.2, Section E.3
\textsuperscript{40} Refer to Chapter 4, Section 4.1.3
• If the interactive session could be recorded using an electronic device namely, audio recorder, it might be much easier for the analyst to complete the process without any loss of data, and inappropriate interference.

§ 5.2 Additional findings

In addition to the conclusions above the case study suggested that some common perceptions about prototyping deserve further investigation.

**Common Perception 1: Prototypes are easy to build.** The following findings suggest that prototypes are not as easy to build as indicated in literature, since:

• The analyst is building a model of the system or aspects of it which may not be concrete.

• The analyst is required to understand the user’s needs, this is especially difficult if the user is uncertain about the requirements.

**Common Perception 2: The prototype must completely meet user requirements with the exception of efficiency.** The following findings suggest that this may not always be the case, since:

• It was found that the interpretations of the analyst of certain aspects of the system helped the user become more interested in the prototype build in C. This was despite that fact the C prototype had no database.

• It did not matter whether the analyst’s interpretation extended the requirements along the recognized requirements of the user presented in the initial specifications of requirements or not.
PROTOTYPING PROCESSES FOR REQUIREMENTS ANALYSIS

- It was possible for the user to identify other requirements even with prototypes that did not display any data.

Therefore, slow but completely functional prototypes might be of very limited use in eliciting new requirements. However, additional work is required to fully substantiate this.

Common Perception 3: Efficiency and robustness are not integral to prototypes. Though it is often said that efficiency and robustness are not required in prototypes, the following findings suggest that they may be necessary to maintain the user's interest in experimenting with a prototype, since:

- When the user was interacting with the prototype developed in SQL/DS as part of the work done to support this thesis, the efficiency factor played a major role.
- The slow response of this prototype caused the user to lose interest while interacting with it.
- The fast response time of the C prototype helped maintain user interest and elicit new requirements.
- Robustness is necessary in the prototypes in order that the user get a feel for the final system.

Since prototypes are models of the final system a certain degree of efficiency and robustness may be necessary to maintain user's interest and involvement with the development of the final system.

Common Perception 4: Data entry is necessary to keep user interested in the prototype or "A lot of user interaction is necessary". Many descriptions of the
prototyping technique stress the need for a great deal of interaction by the user with the prototype. However, the following findings seem to suggest that this might not always be the case:

- In the prototypes built using C and SQL, it was found that it is not essential for the user to enter data while interacting with the prototype though, it seems to be sensible if the user is to maintain interest in the prototype.
- The C prototype was found to satisfy its primary purpose for eliciting requirements even though it did not have data entry options.
- In fact, the new requirements identified from using the C prototype were directed towards the functionalities of the final system.
- Though data entry is possible in the SQL prototype, the user ceased interacting with it as it was too slow.

**Common Perception 5 : Informal specifications are not an appropriate means to represent the initial user requirements.** It seems that informal specifications in English might be an appropriate means to represent initial requirements, since:

- Surprisingly, it was found that the English specifications written by three of the four persons involved in the experiment were nearly of the same length, and this deserves further investigation.

**Common Perception 6 : A single prototype is sufficient.** Experimentation with the two prototypes in the case study facilitated the identification of new and/or the clarification of existing user requirements. Some of these requirements were identified in interaction with both prototypes. However, some requirements that covered entirely different aspects
of the system were identified by the user and these requirements were dependent on the features of the programming languages used, and therefore on the prototypes.

§ 5.3 Future work

It would appear that analysing prototyping processes for comparison and cost/value estimates is only possible if there is sufficient commonality in prototyping language and other available resources. It may only be possible to build a useful ‘experience’ library where such commonalities exist. For example in a single large organization where there may be some degree of standardization. Therefore, an appropriate development of this work would be to determine the value of the notation developed and metrics identified in the construction of an ‘experience’ library for prototyping projects in a large organization that has some degree of standardization and which is seemingly interested in improving its software development process.
Appendix A  Using natural language for systems specification

§ A.1 The process to be studied:

1) Create a specification in English
2) Build a prototype: the interface will be the screens developed to aid the user in identifying the important aspects required for the system
3) End-user interacts with the prototype
4) Modify according to user requests

If a correlation is found between the length of the natural language and the complexity of the first prototype that is built, the result suggests that the length of the natural language is independent of its author.

For example: 10 lines of natural language = 1 line of actual code

A.1.1 Initial specifications for the student record system

Specification I  From: Arunita Jaekel <arunita> 275

Design a system capable of handling student records for 1 course. Each student record has the following fields: student name, student ID, class test 1, class test 2, final, assignment1, assignment2, ... assignment10). All the marks are numeric. Each of the 10 assignments is out of 10, each of the two class tests is out of 100 and the final is out of 100. We want to be able to do the following operations on a record: delete a record, insert a record, sort records by student name, sort records by student ID, get the average of any entry and generate the total marks for each student.

The total mark for a student is calculated by multiplying each component mark by a suitable weight and adding all the weighted components together. We want to be able
to generate a list of all the students and their records. We also want to include an extra field in each record which contains the total mark for the student. We want to generate the (letter) grade for each student. This is calculated from the total mark using a given scale. We should be able to easily shift the scale by a fixed amount and generate a new set of grades. We need to generate a histogram from the grades.

We need to develop a procedure which selects a particular student record and one of the component marks (e.g. assignment3 or class test 1 etc) and calculates the final grade by readjusting the weights for each component. The weights are adjusted so that selected component has no effect on the final grade (0 weight) and its previous weight is distributed properly among the other components.

**Specification II**  From: Arindam Das <arindam> 199 [not counted hints and comments]

Design Specification for STUDENT RECORD SYSTEM

A Student Record System is required to be implemented. The Specifications are given below. The Record should be as follows:

Record

name_of_the_student, student_id, Marks_in_First_Class_test (maximum 100), Marks_in_Second_Class_test (maximum 100), Marks_in_Final_Exam (maximum 100), Marks_in_Assignment1 (maximum 10), Marks_in_Assignment2 (maximum 10), Marks_in_Assignment10 (maximum 10).

Following Operations are expected to be executed by the system:

1. Delete Records
2. Insert Records
3. Sort Records by student_id
4. Find average of Marks_in_First_Class_test obtained by all the students
5. Find average of Marks_in_Second_Class_test obtained by all the students and similarly for other entries
6. Generate the Total (Total = No. * Wt)
7. Generate a list of all students with all of their records with any additional field.
8. Grades are to be computed using a scale. This scale should be user defined and there should be scope for changing it interactively by the user
9. The user must be easily able to change the scale by a fixed offset interactively
10. Generate a histogram of the grades
11. for any student, if his work is incomplete, e.g. he has not submitted the assignment, then the system must be able to ignore that particular entry

**Specification III**  From Farook Fri Apr 30 16:33:06 1993 263

**SPECIFICATION FOR STUDENT GRADING SOFTWARE SYSTEM**

**TASK**: To develop a software system that facilitates in grading students for a particular course. The system should be able to perform tasks, such as, entering student's marks, id, name, generate grade, find average marks obtained by students, generate histograms etc.

**INPUT**: The following will be available to the system:

For each student,

- Name
- Id
- Course name
• Marks obtained in first test
• Marks obtained in second test
• Marks obtained in final exam
• Marks obtained in class assignments

For each course,

• Grading scale
• Weightage for test, exam, and assignments

OUTPUT: The system should be able to allow the following:

• Delete a student’s record for a particular course
• Insert a student’s record for a particular course
• Calculate the average marks obtained by student’s for a particular course
• Calculate the marks, grade and total for a student in a particular course (the system should be flexible enough to allow the inclusion of additional fields in the grading report for a particular student)
• Generate the grading report for a class for a particular course (the system should be flexible enough to allow the inclusion of additional fields in the final report)
• The system should allow the user to interactively change the grading scale and weightage for test, exams, and assignments for a particular course
• The system should be able to generate appropriate graphs
• The system should be able to suppress the inclusion of any stated test and/or exam and/or assignment(s) while calculating the marks/grade and at the same time adjust the other weightages (weightages which are used in the calculation of grades/marks) appropriately
STUDENT RECORD SYSTEM

To develop and implement a system that maintains the student records for a particular course in a given semester/term. The record system is to contain data pertaining to each student in the course:

- name of student
- student identification #

Marks of:

- Class test #1 out of 100
- Class test #2 out of 100
- Final exam out of 100
- assignment #1 out of 10,
- ..........
- assignment #10 out of 10

The systems must be able to handle:

- deletion of records
- insertion of records
- sorting of records by student identification #
- average marks for any record
- total marks for any record
The total marks is generated by multiplying each individual mark by a weight fixed by the instructor before the commencement of the course, and then, adding the products. The system should generate the following output:

- List of all students
- Individual student records
- List of grades for individual and all students
- A field that displays total marks of students
- Appropriate grade is deduced from the scale that contains the list of grades and its corresponding numeric value (range)
- Provision to change the value by a fixed amount and to regenerate the grades
- Options to view grades using a histogram
- Provision to generate list of students absent for any test or exam
- Generation of grades by ignoring the absent mark
- To develop a procedure that can be applied to any selected student record to give a particular component of work.

A.1.2 Final specifications for the student record system

The new requirements are appended to the initial specification IV.

Specification IV after experimentation with the SQL prototype From: Arathi Ranganathan 305

STUDENT RECORD SYSTEM
To develop and implement a system that maintains the student records for a particular course in a given semester/term. The record system is to contain data pertaining to each student in the course:

- name of student
- student identification #

Marks of :

- Class test #1 out of 100
- Class test #2 out of 100
- Final exam out of 100
- assignment #1 out of 10,
- ........
- assignment #10 out of 10

The systems must be able to handle :

- deletion of records
- insertion of records
- sorting of records by student identification #
- average marks for any record
- total marks for any record

The total marks is generated by multiplying each individual mark by a weight fixed by the instructor before the commencement of the course, and then, adding the products.

The system should generate the following output :

- List of all students
- individual student records
- list of grades for individual and all students
- A field that displays total marks of students
- Appropriate grade is deduced from the scale that contains the list of grades and its corresponding numeric value(range)
- provision to change the value by a fixed amount and to regenerate the grades
- Options to view grades using a histogram
- Provision to generate list of students absent for any test or exam
- Generation of grades by ignoring the absent mark
- To develop a procedure that can be applied to any selected student record to give a particular component of work.
- Error checks during input of data.
- Display of error and help messages.
- Provision to change weights attached by arbitrary amounts.
- User friendly table names.
- Alias for Tables.
- Provision to change certain fields.
- Provision to view required fields.
- Provision to override computer generated grades and replace them with what the user enters.

**Specification IV after experimentation with the C prototype** From: Arathi Ranganathan 300

**STUDENT RECORD SYSTEM**
To develop and implement a system that maintains the student records for a particular course in a given semester/term. The record system is to contain data pertaining to each student in the course:

- name of student
- student identification #

Marks of:

- Class test #1 out of 100
- Class test #2 out of 100
- Final exam out of 100
- assignment #1 out of 10,
- ..........
- assignment #10 out of 10

The systems must be able to handle:

- deletion of records
- insertion of records
- sorting of records by student identification #
- average marks for any record
- total marks for any record

The total marks is generated by multiplying each individual mark by a weight fixed by the instructor before the commencement of the course, and then, adding the products.

The system should generate the following output:

- List of all students
• individual student records
• list of grades for individual and all students
• A field that displays total marks of students
• Appropriate grade is deduced from the scale that contains the list of grades and its corresponding numeric value(range)
• provision to change the value by a fixed amount and to regenerate the grades
• Options to view grades using a histogram
• Provision to generate list of students absent for any test or exam
• Generation of grades by ignoring the absent mark
• To develop a procedure that can be applied to any selected student record to give a particular component of work.
• Error checks during input of data.
• Display of error and help messages.
• Provision to change the weights attached as required.
• Provision to specify the value range for the weights.
• Provision to change the grade scale.
• Flexibility to change the manner of grade calculation for specific students.
Appendix B  Questionnaire for study on notation

§ B.1 A study of a notation for representing prototyping processes

A. Ranganathan

School of Computer Science.

University of Windsor

B.1.1 STUDY I: A study of the clarity of the representation

The following analysis is related to a project in which an attempt is being made to formalize prototyping processes.

We are asking you to help us to determine the clarity of a particular notation that we have developed to describe prototyping processes. Please, read the instructions carefully before completing each step. Thank you for your help.

PLEASE COMPLETE THE FORM IN BLACK INK.....LEGIBLY !!!
Data on respondent

<table>
<thead>
<tr>
<th>Name :</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Your occupation :</td>
<td></td>
</tr>
<tr>
<td>Qualification, please list degree(s) or year of undergraduate studies:</td>
<td></td>
</tr>
<tr>
<td>Have you designed any software before ?</td>
<td></td>
</tr>
<tr>
<td>If yes, how many years ?</td>
<td></td>
</tr>
<tr>
<td>Have you used any pictorial representation, namely flow charts, etc. :</td>
<td></td>
</tr>
</tbody>
</table>

Step 1  Examine Figure 20 on the next page, do not look at any other pages at this point.

Attempt to identify the process that is being depicted in the Figure 20. Stop when you feel that you have a good understanding of the process or when you feel that you cannot obtain any further information from Figure 20. Use form1 below to record any difficulties that you encounter.
<table>
<thead>
<tr>
<th>Start Time :</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name :</td>
<td></td>
</tr>
<tr>
<td>List difficulties in understanding the figure</td>
<td></td>
</tr>
</tbody>
</table>

Do you think that you fully understand the process being represented? If not, list aspects of the figure that you do not understand:

| Finish Time :        |          |

Figure 19 Form 1
Prototyping Process — Figure 20

Specification of Requirements

"Design of the Prototype"
Using formal methods and/or CASE tools

Prototype Specification

"Building the Prototype"
Using the computer and Software (language or tool)

Executable prototype

User

"Use of Prototype"
Using the computer and Software (language or tool)

Request for changes

Specification of Requirements

"Evaluating Requests"
Use of documents

Concurrency

Questions

User

"Clarification of Requests"
Use of Answers

SA

"Alterations & Alternatives"
Use of Question

Answers

SA

"Update Requirements Specification"
Use of appropriate and desired tools

Specification of Requirements

Feasible changes

Specification of Requirements

20 Formalism
Step 2  Refer to the key given on the following page and see if your understanding of the notation used in Figure 20 is consistent with that given in the key.

Use form 2 below to record your findings.

Key to notation  The table below provides a key to the notation that is used in the formalization of prototyping process given in the following page.
Start time:

Symbols

Was your understanding of the symbol consistent with the description in the key? [N / Y]

\[ \times \]

\[ \times \]

\[ \times \]

\[ \times \]

\[ \times \times \]

\[ \times \times \times \]

\[ \times \times \times \]

\[ \times \times \times \]

Which of the symbols in the figure if any, do you still not fully understand?

Finish time:

Figure 21 Form 2
<table>
<thead>
<tr>
<th>Notation</th>
<th>Name of Notation</th>
<th>Thing Denoted</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Actor</td>
<td>The actor X</td>
</tr>
<tr>
<td>X</td>
<td>Activity name</td>
<td>The activity X and the tools T that are appropriate to perform X</td>
</tr>
<tr>
<td>T</td>
<td>Tool used to perform activity</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>M</td>
<td>The abstract representation of data X</td>
</tr>
<tr>
<td>M</td>
<td>Manner of representation (optional)</td>
<td>The use of the option specifies that X is physically represented using the media M</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>X and Y are concurrent activities</td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td>Y is the output from activity X</td>
</tr>
<tr>
<td>X</td>
<td>F</td>
<td>X is accessed by actor Y</td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td>X is executed by actor Y</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>Indicates that activity Y follows directly after activity X</td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>The data communication X is accessed by more than one activity</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>The data communication X has been updated</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
<td>The data communication X has been updated ( updated the input with + )</td>
</tr>
</tbody>
</table>

Figure 22 Notations used in the Formalisms
Step 3  Now, refer back to Figure 20 using the key on the previous page to better
determine what is being represented. Use form 3 below to record any difficulties that
you still have in understanding figure 20.

<table>
<thead>
<tr>
<th>Start Time :</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name :</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>List difficulties in understanding the figure.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Do you think that you fully understand the process being represented? If not, list
aspects of the figure that you still do not understand : |  |
|              |  |
| Finish Time : |  |
Step 4  Finally, read the textual description of the prototyping process given in page 105 and see if your understanding of figure 20 is consistent with the textual description. Use form 4 to record your findings.

<table>
<thead>
<tr>
<th>Start Time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name :</td>
<td></td>
</tr>
<tr>
<td>List the discrepancies between the textual description of the process and your understanding of the process from figure 2</td>
<td></td>
</tr>
<tr>
<td>Finish Time</td>
<td></td>
</tr>
</tbody>
</table>

Figure 24 Form 4

Prototyping process — Description  In the process illustrated by Figure 20 the systems analyst (SA) designs and builds the prototype that the user evaluates. The systems analyst assesses any changes requested by the user. This involves a period of intense clarification and alterations to the requested changes and identification of alternative changes. When a set of feasible changes have been accepted by the user, the prototype is modified to
exhibit them. This is a cyclic process which stops when the user and the SA feel that all requirements are met.

This process begins with the data communication specification of requirements, being accessed by the systems analyst. The Figure 20 depicts an abstract view. The manner of representation of the specifications of requirements is not essential. The access of documentation denotes the specification of requirements is a document the actor (systems analyst) uses as input for the design of prototype activity. This activity could make use of formal methods and/or Computer Aided Software Engineering (CASE) tools. The multiple access denotes that this document will be used as input by other activities. The output from this activity is a data communication, the prototype specification.

The next activity is the building of the prototype using the computer and software language or tools. The systems analyst utilizes the prototype specification as input. The two outputs from this activity are the prototype user manual and the executable program of the prototype. The outputs are used as inputs by the user for the next activity.

The prototype user manual is a document as indicated by the access of documentation, and the executable prototype is a software program that is executable as indicated by the execution symbol. The inputs are used by the user for a hands on experience of the prototype. The output contains the changes requested by the user. This could contain all the functions that he or she feels are useful and are not present in the prototype, or if there are any functions that are irrelevant to him or her as the user of the final system.

The request for changes is a document accessed by the systems analyst for the activity of evaluating requests. This activity is concerned with analysing the feasibility of the user’s requests in accordance with the resources, budgets, and requirements of the system.
The systems analyst produces a set of questions that requires feedback from the user.

The concurrent activities involved in the clarification and identification of alternate solutions to the user's requests for change occurs next. The two activities are alteration and alternative performed by the systems analyst and clarification of requests performed by the user. The former activity is concerned with the alteration of current requests and identification of alternative feasible options. The two activities occur over a period of time and involve a lot of interaction between the systems analyst and the user.

The user uses the questions as input and tries to answer and express his or her needs during the clarification of requests activity. The systems analyst refers to these answers during the alteration and alternative activity to identify more questions if possible. He or she makes use of verbal or written means of expressing his queries. The resulting set of questions forms the output. This activity results in additional requests or modified versions of the user's requests for change. When the systems analyst and the user have reached a consensus relating to the expected and feasible changes to the prototype, a formalized version of the changes forms the output.

The next activity performed by the systems analyst is the updating requirements specification. He or she accesses the changes document and the specifications of requirements document. Appropriate and desired tools are used during the activity to complete the update and a modified version of the specification of requirements is obtained. The extended symbol indicates the modification. The new specification of requirements + document is subsequently used as the initial input for the next cycle of the prototyping process. The flow of control in each cycle of the prototyping process is denoted by the flow of control symbol.
B.1.2 STUDY II: Study of the expressibility of the prototyping process

Please, attempt to identify and describe in English a prototyping process which you think cannot be represented using the notations given in the key on page 101.
Appendix C  Summary of the responses to the survey

§ C.1 Frequency table : Study I

<table>
<thead>
<tr>
<th>GIVEN THE FIGURE AVERAGE TIME: approximately 10 mins</th>
<th>NUMBER OF RESPONDENTS WHO HAD DIFFICULTY UNDERSTANDING THE NOTATION [out of 21]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION</td>
<td></td>
</tr>
<tr>
<td>Flow of Control</td>
<td>19</td>
</tr>
<tr>
<td>Significance of the concurrency</td>
<td>3</td>
</tr>
<tr>
<td>symbol</td>
<td></td>
</tr>
<tr>
<td>Implications of symbols : *, &amp;, +</td>
<td>10</td>
</tr>
<tr>
<td>The term &quot;SA&quot;</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROTOTYPING PROCESS</th>
<th>NUMBER OF RESPONDANTS HAVING DIFFICULTY UNDERSTANDING THE PROCESS DEPICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clarity of concurrency not clear</td>
<td>2</td>
</tr>
<tr>
<td>Clarity of Process, no</td>
<td>2</td>
</tr>
<tr>
<td>Symbols : information flow, execution: *, &amp;, +</td>
<td>10</td>
</tr>
</tbody>
</table>
### Given Key to the Notation

Average Time: approximately 6 mins

<table>
<thead>
<tr>
<th>Symbols in the Notation</th>
<th>Number of Respondents Having * No * Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>19</td>
</tr>
<tr>
<td>Activity</td>
<td>20</td>
</tr>
<tr>
<td>Data Communication</td>
<td>12</td>
</tr>
<tr>
<td>Concurrence</td>
<td>13</td>
</tr>
<tr>
<td>Output</td>
<td>19</td>
</tr>
<tr>
<td>Access of Document</td>
<td>15</td>
</tr>
<tr>
<td>Execution</td>
<td>9</td>
</tr>
<tr>
<td>Flow of Control</td>
<td>20</td>
</tr>
<tr>
<td>Multiple Access</td>
<td>7</td>
</tr>
<tr>
<td>Update</td>
<td>11</td>
</tr>
<tr>
<td>Extended Update</td>
<td>6</td>
</tr>
</tbody>
</table>

### Given the Figure Along with the Key to the Notation

Average Time: approximately 5 mins

<table>
<thead>
<tr>
<th>Notation</th>
<th>Number of Respondents Having Difficulty with the Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency</td>
<td>5</td>
</tr>
<tr>
<td>Activity</td>
<td>1</td>
</tr>
<tr>
<td>++ Symbol</td>
<td>1</td>
</tr>
<tr>
<td>SA</td>
<td>3</td>
</tr>
<tr>
<td>Flow of data</td>
<td>1</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td></td>
</tr>
<tr>
<td>Easily understood</td>
<td>16</td>
</tr>
<tr>
<td>Unclear</td>
<td>3</td>
</tr>
</tbody>
</table>
GIVEN THE FIGURE WITH A TEXTUAL GUIDE TO THE PROCESS

AVERAGE TIME: approximately 11 mins

<table>
<thead>
<tr>
<th>PROTOTYPING PROCESS:</th>
<th>NUMBER OF RESPONDENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENTS</td>
<td></td>
</tr>
<tr>
<td>No discrepancies</td>
<td>19</td>
</tr>
<tr>
<td>Unclear Cyclic nature</td>
<td>1</td>
</tr>
<tr>
<td>Some Practical Difficulties</td>
<td>1</td>
</tr>
<tr>
<td>Concurrency</td>
<td>1</td>
</tr>
</tbody>
</table>

§ C.2 Frequency table: Study II

None of the 21 respondents could identify a prototyping process that could not be accommodated by the notations.
Appendix D  Guidelines for building experience libraries

The following can be viewed as a guide to building an experience library in order to study the impact of the prototyping process in gathering requirements of a system and hence, on software project planning.

- Type of Application being built.
- Description of the Application.
- Resources that was available.
- Resources that were used in building the prototype and the final product.
- Budget that was made.
- Actual utilization of the resources.
- Components reused.
- Aspects of the system that was developed using the prototyping technique.
- The prototyping process used described using the notation described in this report.
- An analysis of the data that was gathered for using the prototyping process to indicate the time and effort involved.

The potential users of this library could be systems analysts, software development planners, managers, and designers.
Appendix E  Form for data collection during a prototyping project

§ E.1 Template

These are some of the information required for the entire requirements analysis phase so that it can be used to report on the actual resources utilized, number of system analysts involved in this phase, and the place of work.

Name of Analyst :

Location of Analyst :

On-site / Off-site

Date :

Start Time :

Finish Time :
This is a form to be filled by the systems analyst when the user is interacting with the prototype.

Start Date : Start Time :

Finish Date : Finish Time :

<table>
<thead>
<tr>
<th>Length of Initial Specifications :</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of Final Specifications   :</td>
<td></td>
</tr>
<tr>
<td>Total Number of Iterations in the Prototyping Process :</td>
<td></td>
</tr>
<tr>
<td>Number of Lines of code in Initial Prototype :</td>
<td></td>
</tr>
<tr>
<td>Number of Lines of code in Final Prototype :</td>
<td></td>
</tr>
<tr>
<td>Total Number of Reports Required Initially :</td>
<td></td>
</tr>
<tr>
<td>Total Number of Reports Required after the Prototyping Process :</td>
<td></td>
</tr>
</tbody>
</table>

**Number of Menu Options :** Before Prototyping Process

* maximum depth of the menu option
* average number of options on menu
* number of screens
* total number of menus traversed

**Number of Menu Options :** After Prototyping Process

* maximum depth of the menu option
* average number of options on menu
* number of screens
* total number of menus traversed

Table 5 FORM I
<table>
<thead>
<tr>
<th>Prototyping Process Iteration Number</th>
<th>Frequency</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of times each of the following screen is used:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Screen 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Screen 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Screen 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of times each of the following menu option is chosen:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Menu 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Menu 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Menu 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of times each menu option is chosen accidentally i.e. Number of times the Previous Screen Option was chosen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Without Accessing Screens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* After Accessing Screens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of questions the user has about the final system:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- answered by interaction with prototype</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- answered by the analyst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of failures reported by user:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- error checking while data entered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- screens not having required entires</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- screens not displaying required information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- absence of required screens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- menu options unavailable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- menu options in wrong menu</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6 FORM II
§ E.2 Data collection form — SQL prototype

This is a form to be filled by the systems analyst when the user is interacting with the prototype.

Name of Analyst: Arathi Ranganathan

Date: Nov 13th 1993

Location of Analyst: University of Windsor

Start Time: 2:40pm

Finish Time: 3:30pm

On-site

<table>
<thead>
<tr>
<th>Iteration Number: 2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Prototyping Process Iteration Number</th>
<th>Frequency</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of questions the user has about the final system:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- answered by interaction with prototype</td>
<td>I</td>
<td>5</td>
</tr>
<tr>
<td>- answered by the analyst/documentation</td>
<td>IIIII IIIII IIIII IIIII II</td>
<td>22</td>
</tr>
<tr>
<td>Number of failures reported by user:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- error checking while data entered</td>
<td>III</td>
<td>3</td>
</tr>
<tr>
<td>- screens having required entires</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>- screens not displaying required information</td>
<td>IIIII III</td>
<td>8</td>
</tr>
<tr>
<td>- absence of required screens</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td>- menu options unavailable</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>- menu options in wrong menu</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7 Data from SQL prototype.

§ E.3 Data collection form — C prototype

This is a form to be filled by the systems analyst when the user is interacting with the prototype.
**Name of Analyst:** Arathi Ranganathan  
**Date:** Nov 20th 1993

**Location of Analyst:** University of Windsor

**On-site**

**Start Time:** 2:15pm  
**Finish Time:** 3:50pm

**Iteration Number:** 1

<table>
<thead>
<tr>
<th>Prototyping Process Iteration Number</th>
<th>Frequency</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of times each of the following screen is used:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A] Course Information - Entry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B] Student Information - Entry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C] Student Marks - Update</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>D] Student Marks - Delete</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>E] Assignment Marks - Entry</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>F] Test Marks - Entry</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of times each of the following menu option is chosen:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A] Main Menu</td>
<td>III</td>
<td>3</td>
</tr>
<tr>
<td>B] Input Data Menu</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>C] Report Menu</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>D] Update Records Menu</td>
<td>II</td>
<td>2</td>
</tr>
<tr>
<td>E] Create New Records Menu</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>F] Student Menu</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of times each menu option is chosen accidently i.e. Number of times the Previous Screen Option was chosen</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>* Without Accessing Screens</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>* After Accessing Screens</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of questions the user has about the final system:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>answered by interaction with prototype</td>
<td>III</td>
<td>4</td>
</tr>
<tr>
<td>answered by the analyst</td>
<td>III</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8  Data from C prototype.  (Continued) . . .
<table>
<thead>
<tr>
<th>Number of failures or insufficient features reported by user:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- error checking while data entered</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>- screens not displaying required information</td>
<td>I I I I I</td>
<td>6</td>
</tr>
<tr>
<td>- absence of required screens</td>
<td>I I I</td>
<td>4</td>
</tr>
<tr>
<td>- menu options unavailable</td>
<td>I I I I</td>
<td>5</td>
</tr>
<tr>
<td>- menu options in wrong menu</td>
<td>I I</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8 Data from C prototype.
Appendix F  Survey report — Computer Assisted  
Analysis and Design of Database Systems

§ 1 Introduction to the survey

The decisions taken by managers of organizations depend on data. The computerization  
of manual data processing in an organization is the result of an application of Systems  
Development Life Cycle (SDLC). The systems development life cycle consists of vari-  
ous phases or steps involved in the recognition of a problem and the approaches needed  
to solve it for the benefit of the organization, and the maintenance features required to  
accommodate the changes as and when required by the organization [67]. The systems  
development life cycle and the various modifications to it are discussed in section F.2.  

The birth of Computer Aided Software Engineering (CASE) tools and prototyping  
was due to a crisis in the software industry. User dissatisfaction and plummeting hardware  
costs have led software manufacturers to make a concentrated effort to improve their  
products. CASE tools facilitate the system development process by providing automated  
support to the various development methodologies. Prototyping is an automated and  
interactive approach to systems development that is gaining wide recognition in industry.  
The interaction between the user and the system developer makes use of the automated  
tools to demonstrate the user interface ability and the features provided by the system.

1.1 The problem area

An involved user interaction is often necessary in order to develop a satisfying product.  
Other than the fact that the user wants the system to be computerized, she/he is often  
unclear about the objectives of the system. The systems analyst has to interact with the
user and identify the requirements. The software development process consists of well-defined phases which lead to well-defined and maintainable software. Many software development organizations fail to follow one or more phases due to time and/or budget constraints. Invariably, the urge to begin coding and implementing the system takes precedence over the important requirements analysis and design specification phases. But, the analysis of the system is crucial and the design phase communicates the system to the programmer. Inaccurate or incomplete analysis leads to the development of incorrect software, and an improper design leads to low quality implementation. The problem is the need to allocate successfully resources available to obtain a complete and accurate identification of the system and user requirements, and to translate them into a successful computerized application system. The practicality of utilizing the tools and methodologies for transferring the logical design into a physical design of the system depends largely on the talents of the system analysts.

**F.1.2 Organization of the report**

In this document, the results of a survey into computerised aids for the analysis and design of database systems, is presented. The structure of the report is as follows:

In Section F.2, we discuss the system development life-cycle that forms the basis for both prototyping and CASE technology.

In Section F.3, the tools and methods that are used in the systems development life-cycle are presented with examples.

Section F presents a discussion on prototyping and describes the technology that has enabled prototyping to become more widely used. The report contains papers that present data gathered from experiments conducted in the field of prototyping.
In Section F.5, Computer Aided Software Engineering (CASE) tools and methodologies are discussed.

F.1.3 Computerized aids to the development process

Software development has not changed much but for the automation of some labour-intensive tasks [48]. Software automation is exemplified by highly interactive and responsive environments for systems development [14, 49]. Luqi et al. [42] report that Yourdon lists the following as the important features of automated tools for systems analysis and design of database systems. They are:

- Graphics support for multiple types of models.
- Error-checking features to ensure model accuracy.
- Cross-checking of different models.
- Additional software engineering support.

<table>
<thead>
<tr>
<th>Traditional Development</th>
<th>Software Automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>* emphasis on coding and testing</td>
<td>* emphasis on analysis and design</td>
</tr>
<tr>
<td>* paper-based specifications</td>
<td>* rapid iterative prototyping</td>
</tr>
<tr>
<td>* manual coding</td>
<td>* automated code generation</td>
</tr>
<tr>
<td>* manual documentation</td>
<td>* automatic documentation generation</td>
</tr>
<tr>
<td>* software testing</td>
<td>* automated design checking</td>
</tr>
<tr>
<td>* maintain code</td>
<td>* maintain design specifications</td>
</tr>
</tbody>
</table>

Table 25 Illustration of the underlying differences between the traditional and automated methods

McClure [49] uses the above table to list the differences between the traditional approach and the automated approach to software development.
Prototyping Reliable user requirements are important for successful system development. It is impossible to design a system without a clear specification of user requirements [55]. Prototyping is an iterative process. The steps involved in the iterative process are the interactions of the user with the prototype, revision and refinement of the requirements, and revision of the prototype based on requirement changes. Prototyping helps the user visualize the proposed system. The product of prototyping is called a prototype. Rapid prototyping facilities replace manual labor intensive paper-based specifications, verification and validation checks, generation of code and documentation during system design. Prototyping facilities include libraries that store data for reuse, intelligent tools for knowledge base and natural language interfaces, and expert system functions for navigation during development of the system.

CASE tools Computer Aided Software Engineering (CASE) provides automated assistance to the analysts, designers, and builders of computer systems through computerised tools, techniques, and methodologies which facilitate system development [17]. CASE supports documentation assistance, analysis enhancements, project coordination, and application generators [74]. McClure [48] reports that the organizations Touche-Ross, Deere & Co., and DuPont, had expressed their belief in software automation 7 years ago and had vouched their trust in CASE having positive effect on software production and quality. “CASE is a combination of software tools and structured software development methodologies [48].”
§ F.2 Processes used in the development of database systems

The need for a computerized database system is recognized and the development process is initiated by the management of an organization. Systems development commence when the computer professionals interact with the management. On-site work is any work related to the system development conducted in the client's premises. Off-site is the work done at the computer consultancy firm. A software process consists of several well-defined steps or phases that lead to well-defined and maintainable software. These phases are Requirements Definition, Systems Analysis, Systems Design, Coding, Testing, Implementation, and Maintenance. These phases are utilized in a cyclic process and are called the Systems Development Life Cycle [69, 37, 55, 20, 32, 64, 57, 36, 65, 10, 15].

F.2.1 Conventional systems development life-cycle model

The systems development life cycle works best when specialists are responsible for the different phases. If all the steps of the life cycle are performed by the same project team, they often lose perspective of the system requirements and tend to take short cuts with regard to design and coding. This leads to poor quality and less reliable software that basically does not address the user requirements. Since a compromise was made between the user and the analyst, the system usually falls short of user expectations. But, the user accepts it grudgingly since a large amount of resources has been invested. The system satisfies the analyst more than the user. From a certain point of view the system is perfect, but when viewed from the user's point of view it falls short of expectations. In many ways, systems analysis is the most difficult part of the development cycle. The systems analyst must be proficient in the latest data processing technologies. The analyst must
be able to declare user requirements clearly and match what is possible with the current
technology that which is feasible for the user. The analyst must be able to succinctly
establish what the user wishes by means of the various methodologies currently present
in the industry. The system analyst must have a complete understanding of the system
before composing the software blueprint that shows what to build and how to build it. Many software development projects are characterized by problems leading to late
delivery, cost overruns, and occasionally, dissatisfied customers [7, 37, 1].

Figure 26: Traditional Systems Development Life-Cycle

Figure 2 represents the manner by which the various phases of the development life-
cycle are linked. The inputs and the outputs that aid the linkage is also shown. The
above figure is modified from the following books: Sommerville [65], Fisher [22], and
Chantler [15].
**Initial study or requirements definition**  Requirements definition is the first phase of the systems development life-cycle. In this phase the user expresses an interest in computerization of some tasks. Informal initial discussions between the computer consultants and the user ratify the needs of the latter. The computer consultants initially conduct a *feasibility study* [15]. The feasibility study is a formal, high level, and brief study of the entire system. This study produces an analysis of the problem area, and the system objectives are clarified. The requirements specification is the process involved with producing documentation about the system or organization. The requirements analysis or feasibility study is the process of analyzing the functionality of the existing system. This system may be a manual system or a computerized one. If it is a computerized system then, the problem is to identify successfully the functionalities that need to be modified. If it is a manual system, then, the functionalities of the various manual process must be analyzed. The specification must define user requirements in terms of functionality, design constraints, interface, and performance of the system [7]. Based on the outcome of this study, the need for computerizing the organization is strengthened. This is the most crucial decision of the entire project.

**Systems analysis**  The requirements analysis or systems analysis, is the process of analyzing the various functionality of the existing system in great detail. It answers the question, *what* are the various functionalities of the current system? It is an intensive study of the system, and involves user interaction. The analysis results in a detailed representation of the system in the form of a logical model. Requirements specification is the product of the requirements analysis phase. It is usually a document describing the system or organization. "Analysis is the art of considering everything relevant at a given
point and nothing more[61].” In the traditional approaches to systems development, the validity check of the analysis was done during the implementation phase of the system. Inaccurate analysis causes user dissatisfaction, and the costly maintenance work needed is not always successful [20, 32, 64, 57, 36, 65, 10, 15]. Since user satisfaction measures the success of any system approaches used in analysis have been modified to include user interaction as much as possible. Prototyping and CASE tools are of invaluable assistance to the analysts for user interaction. It involves an intensive effort and this was something that was never quite achieved by the traditional tools and methodologies in the same duration.

**Systems design** Systems design answers the question of *how* the objectives of the system are to be solved. Usually, the analyst and the designer are the same person. Otherwise, the designer has to understand the system based on the documentation produced during the analysis phase, this increases the duration of development. Consequently, having different teams for different tasks as discussed in Section 2.1 of this survey has its share of advantages and disadvantages. The logical model of the systems analysis phase is transformed into a physical model. “The specifications developed during detailed design is analogous to the engineer’s blueprints[20].” The design phase is represented according to the hardware components, software components, supporting documentation and the data components available.

**Coding** It is during this phase that the database system is actually developed. The programmers take over and the purpose is to express the system design as programs using a suitable programming language that can be executed by the machine. The database is established and the system design is translated into the code of the language selected to
represent the system. The entire system is represented as modules and programs that should be well supported by documentation. The system is evaluated systematically during the coding phase at regular intervals. The coding phase and the testing phase are interactive.

**Testing**  Each piece of code must be tested extensively and debugged. Testing is carried out to check whether it satisfies the specification of the system. It does not improve the program but it demonstrates the processing ability of the programs. The programmers must devise a good testing strategy that would help them. The strategy includes the choice of appropriate test data, identification of errors, correction of errors, and the aids that are used for the debugging. The modules are integrated and the resulting system is tested and this is called integrated testing.

**Implementation and parallel running**  The system represented by programs is implemented at the user site. The user tests the system. This system testing is part of the integration testing and can be viewed as on-site testing. Code walk-through helps ensure quality and reliability in the implementation. Any minor modifications with regard to the screen formats for input or output can be handled by the programmers. Any major modifications that the user requires are handled by the systems analysts in the maintenance stage. If the existing system continues to function, while the new computerized system is implemented it is known as *parallel running*. It is useful when the end-users are being trained to use the new system. Parallel running is extremely practical as it is a gradual introduction of the system to the users, when compared to the abrupt installation of the entire system.
**Maintenance**  The systems analyst has to interact with the user and refer to the systems analysis and design documents to provide solutions for the bugs or problems found by the users. The new user requirements could either be solved with slight modifications to the present system or require extensive rework. These modifications cost a great deal and often are the outcome of improper analysis of user specifications or a changed user outlook or both. Enhancement requests are collected for continued development.
The above figure provides a detailed view of the various phases of systems development [15]. The main loops provide the forward links between the phases and the feedback loops connect each phase with one or more predecessors. These feedback loops indicate reexamination of the existing specifications and the necessary modifications are carried out by the system developers.

**F.2.2 Current system development models**

With the advent of automated aids that are being incorporated into the traditional systems development life-cycle, it is useful to view the life-cycle in two ways: The Waterfall
model and the Spiral model [12]. The life-cycle has evolved into an interactive model and is no longer a sequence of steps followed mechanically.

**Waterfall model** The *waterfall model* of the systems development life-cycle is a variation of the traditional model. Each phase in this model flows smoothly to the next phase. Any discrepancy in a phase requires a modification of the previous phase. This model is the result of modifying the traditional systems life-cycle model. The two important modifications are the concepts of feedback and prototyping. The feedback concept makes use of looping and iterations and this is seen when compared with the traditional life-cycle model. It is extremely useful in the areas of incremental development, parallel development, software development, validation, and risk analysis. It has succeeded in replacing the traditional model. According to Boehm [12], this model has had a great impact whenever used. In the traditional model, the various phases flow from one to another and the only feedback that is present is from the maintenance phase to the initial requirements specifications stage. In the waterfall model, the main flow of the logic is from one phase to another but there is also a feedback loop that connects the successor phase to its predecessor, and the major feedback loop that connects the maintenance phase with that of the requirements specification phase. One difficulty with the waterfall model has been the emphasis on the extensive documentation required as a completion criterion for requirements analysis and design. This has often produced voluminous documentation that complicates the user interface, the design constraints and specifications. Also, if the phases of the waterfall are pursued in the wrong sequence the projects may fail.
A diagram from [12] that illustrates the concept of the waterfall model is presented on the following page.

![Waterfall Model Diagram](image)

Figure 28  Systems Development Life-Cycle : Waterfall Model

**Spiral model** The *spiral model* is another approach to the systems development life-cycle, and is the result of extensive modifications to the waterfall model and the traditional model. It is called the spiral model due to its shape. A spiral begins when it is felt that the software process could be improved by further software development. It uses risk-analysis and prototyping approaches that are present in industry. A cycle of the spiral starts with the identification of the objectives to be satisfied in that cycle and explores the alternatives available and the constraints that are applicable to them. It is based on the concept that each cycle of the spiral involves the same steps for each phase.
of the development life-cycle. Each phase of the systems development undergoes an intensive iterative process of specification verification by which modifications are made [12]. It includes the underlying concepts of the traditional and waterfall models but with a increased use of interactive prototypes for each phase. The risk-analysis and the prototypes utilized in each phase of the systems life-cycle produce a more accurate and realistic view of the entire systems development process. They also help the discovery of alternate ways for effectively utilizing the resources and satisfying the user at the same time.

Boehm's [12] illustration of the spiral model helps clarify the underlying concept.

Figure 29 Systems Development Life-Cycle: Spiral Model
Investment of capital such as hardware and software, human resources and time is required for developing a system. The systems analyst provides the human interface between the user and the product. When the system fails the systems analyst is invariably the one held responsible, since it was the analyst who dealt with the user and identified the processes to be computerized.

Systems analysis and systems design phases are often confused during systems development. Though the principles of the two phases differ, the techniques and functions are not as clearly distinguishable. This often causes confusion. Guidelines for systems development using the traditional approach are stated in the IEEE Guide to Software Requirements Specifications and the British Standard Guide to Specifying User Requirements for a Computer System [2]. The specifications must depict the needs of each individual user and their combined needs in a conducive manner. Requirements development has been viewed from traditional approaches, prototyping, and incremental development. Incremental development is when the system is developed in stages to the complete satisfaction of the user. It is now possible to construct prototypes of database systems using advanced software products such as 4GL tools, which include relational database management systems, fourth generation languages, report writers, query languages, rapid screen design tools, application generators, code generators, and re-usable software components.

§ F.3 Conventional tools and methods

There are two distinctive approaches to solving the problems presented to computer professionals. They are the decomposition approach and the composition approach. In the decomposition approach, the problem is divided into interdependent subsystems that
are easy to solve. But, these subsystems are incomplete models and are complete only when the various details are filled in. This is also known as the top-down approach. The composition approach or the bottom-up approach starts with the modeling of parts of the system. As parts become complete they are integrated until the entire system is built. Most systems analysts and designers make use of both approaches, this aids a better understanding of the problem and forms a guideline to ensure that the system is the required one. The various tools and methodologies that are employed in the various phases of the development cycle are discussed in this survey. References are made to these tools and methodologies throughout the survey.

State transition diagrams, decision trees, and flow charts are examples of tools that are used in systems development. They directly interact with the changing environment to provide specifications regarding the control features of systems.

Orr [2] maintains that “A methodology in software engineering terms, is a collection of methods based on a common philosophy that fit together in a framework called the systems development life cycle”. According to Constantine [2], “Methodology actually means a study of methods”.

According to Orr [2], “Method is a procedure or technique for performing some significant portion of software life cycle”.

**F.3.1 Structured analysis methodology**

The structured analysis methodology specifies what the product is to do? It was developed by Edward Yourdon and Tom DeMarco. It presents the workings of the system, by means of diagrammatic representations to the designers and the programmers. It has replaced the tedious and involved methods that have been conventionally used to represent systems.
The objective of structured analysis is to produce a structured specification that presents the functional aspects of the system. The diagrammatic representations that are possible in this methodology convey the workings of the system to the designers and programmers.

**Control view** State Transition Diagram, Decision Trees, Flow Charts are diagrammatic methods that present a view of the activities of the system. These activities are extremely dependent on the system environment. Constant reactions to the changes in the system environment are extremely critical.

**3.1.1.1 State transition diagrams** State transition diagrams are used to specify the flow of control. It is a graphical representation of the states and the events that cause the transitions between the states in the system.

**3.1.1.2 Decision trees** The description of decision trees along with an example are presented in sub—section F.3.4.

**3.1.1.3 Flow charts** Flow charts are diagrammatic representations of the flow of logic. Two kinds of flow charts are available: program flow charts and system flow charts.

**3.1.1.3.1 Program flow charts**

Flow charts focus only on the flow of control as its name implies. This diagrammatic representation is extremely easy to use. A flow chart represents a very low level view of the processes within the system, in fact each symbol of the diagram can be considered as a mapping to a line of code on a one to one basis. This low level representation often hinders the analysis of control flow. This tool is often avoided by software professionals as it involves a large amount of paperwork. CASE tools that are currently present in
the market and which help in the process of representing the system by flow charts successfully reduce the amount of paperwork and labor involved [22].

The following illustration is an example of a flow chart. Consider, for example, a list of items and each item must be verified against corresponding item entries in the inventory file. If any of these items is insufficient or depleted then they are ordered.

![Flow Chart Diagram]

**Figure 30** Set of symbols used in the representation of the flow chart
System flow charts

These are diagrammatic representations of the physical system. A physical system is the form into which the logical model of system analysis is converted during system design. The flow chart symbols indicate the hardware, programs, files, procedures, and so on. This represents the flow of logic through the various components of the entire system.
Figure 32 Various symbols used in system flow charts

Figure 33 An example of a systems flow chart

Functional view: data flow diagrams Data flow diagrams depict systems from the viewpoint of the data elements flowing through the processes. It is a graphical depiction of the various data items in a system and their movement between the various processes. Data flow diagrams reveal the data flow and not the control flow. A data dictionary
is a catalog of all the data found in the data flow diagram. Data flow diagrams aid communication of the system functionalities between the analyst and the user.

*Data Flows* are the individual data items that are transmitted and received by processes. They are represented in the data flow diagrams by labelled vectors with arrows to indicate the direction of flow.

*Processes* transform the input data into the required output data. In the diagrams the processes are represented by bubbles. The process nodes can be expanded to reveal secondary data flow diagrams.

*Data sources and sinks* are the originators and receivers of data flows and are represented by square boxes.

*Files and Databases* contain the data that is required for the processes. These are represented by horizontal bars.

Figure 34 illustrates the components of the data flow diagram and figure 35 presents a modified representation of the level 4 of the data flow diagram from Gane and Sarson [25].

![Figure 34 The components of data flow diagrams](image)
Data view: entity-relationship diagrams  Entity-relationship diagrams were developed by Peter P. Chen. They are a graphical depiction of the data in a database and how individual programs make use of this data. Data dictionaries are used to organize the various data elements in the application program. Entity-Relationship diagrams are used to represent the relationships between the data in the conceptual model of the system or process. These diagrams consist of representations of entities and relations. Davis [20] describes the entities as data objects or structures that represent persons or abstract concepts, and the relations as links that depict how the entities are connected. The entities are data representing entities. The links are unidirectional. The data in the entity-relationship diagrams are stationary unlike the data in the data flow diagrams. Many commercial CASE tools support entity-relationship diagrams. The dictionary has the definition of entity as “an organized array of parts or elements forming or functioning as a unit".
F.3.2 Data dictionary

Every data item present in the database system should be documented in a data dictionary. The data dictionary is a store of all definitions of data — raw or processed, of the database system. “All of the attributes of any particular piece of data can be found in the data dictionary [22].” In current CASE tools, the data dictionary is an inherent feature and it is essential even if the structured approaches are not applied. Data elements, also known as data items or fields, are the smallest piece of data that can be stored and of value to the database system. The two relations that exist between the items are based on the data elements and on the data flows. Each data element of the system must be documented. Data flows and data stores are the combinations of related data items called records.
Documentation of the relation of the data flow’s connection to data stores, entities, and processes helps keep the data dictionary consistent with the data flow diagram [20, 22].

<table>
<thead>
<tr>
<th>Item name</th>
<th>Description of Item</th>
<th>Type of item</th>
<th>Size of item</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>name given to the item</td>
<td>a description of</td>
<td>indicating</td>
<td>varies</td>
<td>any special</td>
</tr>
<tr>
<td>as referred in programs</td>
<td>the item for</td>
<td>numeric, alphabetic or</td>
<td>according to</td>
<td>conditions</td>
</tr>
<tr>
<td></td>
<td>readability</td>
<td>alphanumeric</td>
<td>the type of data</td>
<td>attached to the item</td>
</tr>
</tbody>
</table>

Table 38: The various entries that are required in maintaining a data dictionary

F.3.3 Pseudo-code

Pseudo code, an alternative to structured english is an informal representation of the input data flow and the output data flow associated with the functioning of the system. The structure of pseudocode is based on the structure of programming languages like COBOL, FORTRAN, and Pascal. Pseudo code is a combination of English statements and more formal representation of algorithms. It is used to replace flow charts in specifying the functions of the system [20, 22].

Start;

counter <- 0;

open transaction file;

open stock file;
read transaction record:

while not end of file

do read stock record;

if stock.item = transaction.item and stock.item_qty > transaction.item_qty

then process transaction record;

  counter <- counter + 1;

  stock.item <- stock.item + transaction.item;

  write stock record;

  print stock.item

read transaction record;

endif;

print counter;

close files;

stop.

The above pseudo-code is an example of reading an order file, and when there are insufficient items in the inventory file or stock file, the instructions in the order file is carried out. A local variable ‘counter’ is incremented each time the iteration is activated.

F.3.4 Decision tables and decision trees

Decision trees and decision tables are two alternative tools to represent algorithms. They are used to overcome the difficulties of using structured English, pseudocode or flowcharts.
**Decision tables**  A decision table is a diagrammatic representation of an algorithm. The table is divided into four sections. A condition stub at the upper left, a condition query at the upper right, an action stub at the lower left, and an action entry at the lower right. Questions are listed in the condition stub; and the associated actions are listed in the action stub. The responses are recorded in the condition entry and the appropriate actions in the action entry. The responses are marked by the symbols ‘Y’ and ‘N’ or ‘X’ and ‘_’, to indicate yes and no respectively [15].

<table>
<thead>
<tr>
<th>Condition Stub :</th>
<th>Condition Entries :</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) stock.item = transaction.item</td>
<td>Y  N  Y</td>
</tr>
<tr>
<td>2) stock.qty &gt; transaction.qty</td>
<td>Y  _  N</td>
</tr>
<tr>
<td>3) order process</td>
<td>_  _  Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action Stub :</th>
<th>Action Entry :</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Issue item to customer</td>
<td>X  _  _</td>
</tr>
<tr>
<td>2) Report to Supply department</td>
<td>X  _  _</td>
</tr>
<tr>
<td>3) Report to Purchase department</td>
<td>_  _  X</td>
</tr>
</tbody>
</table>

*Figure 39  An example on the use of decision table*

**Decision trees**  Decision trees are graphical representation of decisions, events, and consequences. Decision trees are frequently used by analysts. These are also known as 'decision science tools' or 'management science tools'. Each tree begins with an 'act fork' represented by a box. This has two branches which represent the decisions. The dot represents the 'event fork', which denotes what happens when a certain decision is taken. Each decision branch terminates in an outcome based on the decision taken. Probabilities are associated with each branch. The expected values for each outcome is calculated and provides guidance in the decision making process [20].
F.3.5 Nassi-Schneidermann diagrams

The Nassi-Schneidermann diagrams are especially useful in depicting the "control flow and code organization [22]." The selection criteria that are present in the programs are presented in a diagrammatic form and, based on this diagram the programmer can complete the code. It is used for checking the logic flow of the program and the diagrams are much easier to debug than the source code [22, 15]. The following example depicts the "if-then" conditional inside a "while-do" loop.

![Decision Tree Diagram](image)

Figure 40 An illustration of a decision tree

F.3.6 Jackson charts or Jackson structured design

Michael Jackson was the developer of the Jackson charts, also known as Jackson Structured Design (JSD) in the early 1970s. This design method has been used successfully for
specifying program operations. It combines structure charts with notations that specify execution of algorithms. Presented below are illustrations of an example of a Jackson chart and the selection and iteration criteria as presented in the charts [22].

![Diagram](image)

**Figure 42** An illustration of the selection and iteration criteria supported by Jackson charts

### F.3.7 Warnier-Orr diagrams

A Warnier-Orr diagram is a flexible tool and is used to represent data structures, program structure and the program logic. This was developed in 1970s by Jean-Domonique Warnier as an alternative to HIPO technique. Its capability was extended to include system design by Ken Orr. Though it was originally developed to express program structure, Fisher [22] states that it is more widely used in describing data structure composition. These diagrams can be regarded as an organizational chart turned on its side. As the principle use of Warnier-Orr diagrams according to Fisher, is designing data structures and file formats. The following figure taken from [22] exhibits this nature of the diagrams:

146
Warnier-Orr diagrams can be successfully used to represent program structure as well. The following figure taken from Fisher, illustrates that instead of data structures, the outer levels are replaced by modules and files, and the inner level with subroutines, Do loops, and IF-THEN statements. The $\theta$ denotes a selection criterion.

Program Representation in Warnier-Orr Diagrams

Generate Shipping Invoice \{ Process Item (1, N) \} Check Inventory Control \{ Process Backorder \{ Post as Backordered Update Backorder File \} Update Invoice \{ Post an Invoice Update Stock Database \}

Figure 44 Warnier-Orr diagram representing program structure [22]

F.3.8 HIPO technique

HIPO — Hierarchy plus Input Process Output.

This technique displays the set of specifications associated with each program that is specified in the systems flowchart. The results of applying this technique is presented in the form of a hierarchy chart which represents the top down structure of the program.
and an IPO (Input Process Output) chart that describes the input and output forms, and the processes. Data dictionaries help define the input and output data. The processes are defined by 'algorithms' [20, 22].

![HIPO Chart](image)

Figure 45 HIPO — Hierarchical level 1 for updating inventory file

The update inventory module is the main control module and it controls the order in which the lower level modules are invoked from left to right.

**F.3.9 Structured charts**

Structured charts are of great assistance in keeping track of the data involved in the design of the database system through the IPO (Input Process Output) chart structures. This is feasible as only the data required for a module are presented. They graphically present the hierarchical structure of the system. Fisher [22] states, “The structure chart, coupled with data structure definitions, comprises a software system’s basic architectural design”. Each module should ideally be separable from other modules. The following figure from [22] illustrates a structured chart that presents the hierarchy and executional information.
Structured charts are one of the most difficult methods present. This difficulty to a great extent has been minimized with CASE tools. According to Kenneth Orr, as reported by Fisher [22], “Something is structured if and only if (1) it is hierarchically organized and (2) the pieces of each function are related to one another either by sequence, alternation, or repetition — the basic forms of logic.”
§ F.4 Prototyping

F.4.1 What is prototyping?

Prototyping can be defined as an approach for clarifying and establishing systems requirements through user interaction with incomplete or inefficient systems that are known as prototypes. "Prototypes can be considered as 'mock-ups' of the user interface by means of software" [27].

The application of structured methods and techniques for requirements specification does not always lead to efficient systems. Often, the organizations are faced with considerable maintenance problems in spite of following structured methods. This is due to the fact that these structured methods are often based on incorrect or incomplete assumptions. These inaccurate assumptions are due to the communication gap between the user and the systems analyst, and changing user needs as the development process is going on. The latter fact is often ignored by the systems analysts. The analyst must pay attention to the user’s evolving needs and incorporate it into the system. The systems analyst often uses technical jargon that the user fails to comprehend, and vice versa. To overcome this deficiency, graphical methods are employed. Although the problem is solved, users at times fail to recognize the database system based on this description. Interactive systems require user interaction. Many of the structured methodologies, such as Yourdon’ Structured Design, and SofTech’s SADT, give no importance to the user interface [74].

F.4.2 Software aids to prototyping

Prototyping systems Prototype are systems which model aspects of a system. Users
have difficulty visualizing the features and functionalities of the system without the aid of prototyping. Prototypes that are used to model the external appearance of a database system using screen, reports, and dialogues with no data processing are called user-interface prototypes or mock-up prototypes. A three-level approach to database systems is made up of the information level, data level, and the internal level. Prototypes are best suited to model the information level or the user interface. During the early iterations of the prototype as with the analysis phase, the conceptual aspect of the system is most important. “Rapid prototyping is the prototyping that occurs early in the software development life cycle [71].” The availability of powerful CASE tools such as relational DBMS (DataBase Management System), report generators, screen painters, a data dictionary, and a very high level language are essential for successful prototyping.

![Figure 47 Concept of prototyping](image)

Gomaa [29] in his article claims that most users are not sure about their requirements until they use the system. At that stage it is too late and too expensive to consider the modifications that the user subsequently requests. The prototype is a functioning model of the user requirements and it provides a compact view of the system. Prototypes that allow the storage and simple processing of data are called Functional Prototypes. Prototypes
are classified according to the time frame in which they are developed. *Exploratory Prototypes* are used for discovering the user requirements. *Experimental Prototypes* are used to discover the adequacy of the system during the design phase. *Evolutionary Prototype* prototypes successive prototypes are based on their predecessors. If this prototype is retained then it is a *keep-it prototype* else if it is discarded, it is called the *throw-away prototype* [74].

![Diagram](image.png)

*Figure 48 Evolutionary Prototyping [29]*

Rapid and radical development in the field of information technology has flooded the market with various systems development strategies, structured development methods, and software tools. It is essential that the correct development approach be chosen. The prototyping approach is at times used without considering the advantages and disadvantages. There are many situations where prototyping is quite disadvantageous. User interface prototyping is usually feasible. But, the use of functional prototypes depends upon the cost effectiveness of the prototype. The cost of software development depends on the approach used, the language selected, and the capability of the developers. The benefits of choosing the prototyping approach are the increase in validity and consolidation of system features. Maintenance is greatly reduced as the requirements
are as accurate as possible, and the resulting database system has a good chance of satisfying user requirements.

**Languages that support prototyping**  The choice of a language suitable for prototyping depends on the features of error detection and data manipulation. The tools used widely for rapid prototyping are application generators and fourth-generation languages. “Fourth-generation languages and application generators have been effectively used to prototype interactive database oriented information systems [29].”

Very high level languages such as APL [29], C and Ada [44], LISP, SETL, PROLOG and SNOBOL [31] have been used quite successfully in prototyping. The selection of language suitable for prototyping depends on the features of the language. Flexibility of the language, coding and testing speed, and self-documentation are the important features needed in the language. COBOL and PL/I do not provide the features required for prototyping. APL [50] is fast gaining acceptance as an adaptable prototyping language.

Application oriented high level languages are used for modelling applications by customizing the parameters of the languages successfully. Functional languages are based on mathematics and support prototyping. Functional languages include SASL, MIRANDA and ML [31]. HOPE, OBJ and CLEAR [31] are functional languages that use algebraic approaches and are well suited to prototyping. GIST is an operational specification language that produces specification of operational closed systems, these specifications are ideal for prototyping [3]. According to Ratcliff [59], JSD (Jackson System Development) methods are used for systems that are described as a collection of processes and the following languages generate executable specifications:

- OBJ produces specifications in the form of algebraic axioms.
- Me-too constructs specifications in the form of functions.
- Prolog makes use of Horn clauses for the representation of the system.
- Gist is a 'system behavior' or operational specification language.
- PAISley is well suited for prototyping real time, embedded systems.

**Object-Oriented paradigm**  A current trend in industry is the use of object-oriented languages for the development of a product. Object-oriented languages such as C++, Smalltalk, and other languages have been introduced and are rapidly gaining acceptance. The concept of object-oriented design is quite different from the traditional concepts used in programming languages. Object-oriented languages aid the process of simulation which is an essential part of prototyping.

The focus of object-oriented design is on data structures rather than on algorithms. Each data structure is called an object. Messages instruct the objects to perform an action. Each action is associated with a particular method for the object. Objects can be placed in classes. A class is an object template. All the objects present in a particular class share the same properties, messages and methods of that class.

The approach used in object-oriented program design is similar to the approach of prototyping. The object-oriented software development life-cycle makes use of coding, testing and implementation. This concept of developing the system by using code, tests and implementation processes is akin to that of prototyping. In fact, the development process of object-oriented design consists of a series of prototypes which evolve into the final design. Evolutionary prototypes encourage the development of alternative designs for the system which is essential to provide a suitable design. The use of prototypes in object-oriented design methodologies helps in the crucial decisions involving the
requirement for certain features of the system. The implications of prototyping certain user-interfaces, is that it aids the process of better designs [49].

Automated tools for prototyping Prototyping tools are really important in the early stages of the systems development life cycle. The CASE tools that are used for prototyping include:

Screen painters.

Report generators.

Menu builders.

Fourth-generation languages, and

Executable specification languages.

The screen painters, report generators and menu builders are used in prototyping of user interfaces. The fourth-generation languages provide the facility to include system functions but are not concerned with the validity of the entries. The system is converted to a set of iterative processes by means of executable specification languages. These iterative processes are then used to check the validity of the system.
§ F.5 Computer Aided Software Engineering (CASE) systems

The basic objective of CASE is to support each phase of the systems development life-cycle (SDLC) with a set of automated labor-saving tools. Computer-Aided Software Engineering (CASE) is a name for a number of automated tools supporting the analysis and design of software systems. CASE tools are an extension of existing software tools. The tools help the systems analysts and designers with requirements analysis and design phases besides aiding in the implementation, testing, and release phases. Most CASE tools are aimed at supporting analysis and design of the systems [49, 70, 45, 33, 23, 22]. Many of the CASE technologies are based on the structured methodologies that were developed in the 1960s and 1970s.

Current CASE tools in industry aid the creation and updating of a number of systems analysis and systems design diagrams, with upgraded supportive documentation. It arrests the spiralling costs and time delays in software development [66]. CASE tools can be categorized as supporting requirement analysis and design, data modelling and design of user-interface [24, 47, 45, 63, 9, 33]. Current CASE methodologies include object-oriented principles, re-engineering and user-interfaces. CASE tools include all software tools for programming control, program design, testing, and requirements analysis, as well as tools such as program editors, compilers, and debuggers. Fourth-generation languages and some prototyping systems are considered part of the CASE technology. The 4th Int’l workshop on CASE, Dec ’90, produced a paper [53] which presents the general consensus of technical researchers, practitioners and tool vendors who attended the conference. It lists over 450 features concerning CASE. The components of CASE systems can be summarized as follows:
F.5.1 Categories of CASE tools

Nilsson [51] maintains that "The need for more precise concepts have led to different prefixes and suffixes to CASE, upper, middle, lower, toolkit, workbench and integration being the most commonly used". Gibson [26] and Nilsson [51] divide CASE into — upper, middle, and lower tools. Upper CASE is called computer-aided planning as it is used during the phases prior to systems analysis. Middle CASE are the tools supporting the analysis and design phases of the SDLC, and lower CASE tools support systems development or construction.

In other instances, upper CASE or front-end tools are diagrammatic tools that deal with the planning and design phases, and lower CASE are non-diagrammatic tools that are used for the implementation and support phases [16, 73, 51]. Different CASE tools focus on different phases of the software development life-cycle.

![Diagram of CASE tools](image)

**Figure 49** The phases covered by the CASE tools [49]

**CASE methodology companions** A CASE methodology companion is either a CASE toolkit or a CASE workbench, that provides automated aid for a particular structured software development method. Information regarding the method in the form of help
menus and screens is presented by the companion. Each stage of the methodology must be completed according to the criteria present in the companion in order to start on the next stage of the methodology. Data Resources Leverage With Janus Section aids the prototyping development methodology [49].

5.1.1.1 CASE toolkits CASE toolkits are the simplest type of CASE tool. CASE toolkits are a set of integrated tools that support a particular phase in the software development life-cycle, such as systems design or a particular type of job, such as system analyst. These toolkits can be called phase-level tools as they focus on particular phases. They are also classified according to the hardware and operating system environment which is needed. SOFTWARE THROUGH PICTURES, STATEMATE, TAGS, TEAMWORK/RT, EXCELERATOR, and DESIGNAID [49, 22] are analysis toolkits.

Analysis toolkits aid the automated implementation of good systems analysis and design techniques. These toolkits consist of structured diagramming tools (data flow diagrams and entity relationship diagrams), prototyping tools (user interface painters and executable specification languages), repositories, and specification checkers (automatic checking for correctness and consistency of specifications).

Design Toolkits support the automatic creation of the logical and physical design of databases and files of the system. Examples are AUTO-MATE PLUS, SQL*DESIGN DICTIONARY and CHEN toolkit [49, 22].

There are three kinds of toolkits

* programming toolkits for code generation,
* maintenance toolkits for system evaluation and assessment, and
* reengineering toolkits for the reuse of components of the existing system.
Certain toolkits aid automated aid for programming which involves code generation, for maintenance which involves tools for system evaluation and assessment, and for reengineering.

Framework is a CASE toolkit that aids in the integration, customizing, and management of individual software tools. This toolkit aids the integration of both old and new CASE tools and increases portability of database systems that are created.

5.1.1.2 CASE workbenches  A CASE workbench is a set of integrated tools that supports the entire software development life-cycle [49, 51, 65, 22]. It involves all of the phases and output from a phase is directly input to the next phase. The final product is a documented executable software application system. Examples include Information Engineering Workbench (IEW), TekCASE, and Autocode [22].

CASE workbenches have been developed to support exclusively analysis and design phases of the software process. They present graphical support based on the tools and methodologies for system development. CASE workbenches usually support a specific structured methodology, for example the Yourdon methodology, or provide graphical support of diagrams that is common to all the methodologies. According to Sommerville [65], the components of a CASE workbench are:
Structured diagramming tools provide assistance to create data flow diagrams, structure charts, entity-relationship diagrams, and other diagrams. These tools are not the automated replications of the paper and pencil diagrams. These tools capture information about the entities and is able to store it for further reference by the designer.

Design analysis and checking tools which analyse the design for errors and report them. When integrated with an editing system it provides on-line reports.

Query language functionalities aid browsing of stored data and completed designs of the system functionalities.

Data dictionaries maintain the data of the system. This supports the design of the system by providing the data that is present and that can be used in developing the system.

Report generation functionalities automatically generate the appropriate system documentation based on the stored information.

Form generation tools provide support for screen and document format specification.

Import/Export facilities allow the exchange of information between the stored information in the system and other tools.

Code generators aid the generation of skeleton code based on the system design.
In spite of the extremely versatile options present in workbenches, there are certain drawbacks to them, namely:

* The lack of integration of workbenches.
* Lack of adequate standards.
* Provides rigid generic methods that are not suitable for use in customization.
* Generation of substandard documentation.
* Diagram drawing consume a lot of time.

**F.5.2 CASE tools for database work**

CASE tools and environments are widely used to manage database projects and support the implementation of such projects. The tools are used for automating the process of drawing the many diagrammatic tools of data flow diagrams, structure charts, data dictionaries, entity-relationship diagrams, state-transition diagrams and Jackson diagrams [see section F.3]. Many of the analysis and design CASE tools are based on the structured analysis methodology [25]. Structured techniques emphasize on graphical approaches which enhance the definition of the resulting software application. The use of CASE tools reduces errors, improves documentation and the user interface [22], and reuse [35]. Data flow diagrams produced by the structured analysis process can be translated into module hierarchies through a structured design process. Structured analysis methodology was developed by Edward Yourdon and Tom DeMarco as a method to focus on the application’s data flow rather than the control flow [22]. Different CASE tools using the approach of structured analysis lay emphasis on one of the techniques more than on the others. The structured analysis methodology [25] has three complementary techniques namely data flow diagrams, data models and control specifications. The CASE tools
support a structured analysis approach and are applicable to systems development life-cycle. Most of the CASE tools present in the market today are suited for prototyping. The analyst workbenches and fourth-generation development tools are referred to as CASE tools by Vonk [74]. Fourth-generation development tools support the development of example systems. Similarly analyst workbenches are extremely important for use in the traditional methodologies and are part of a larger set of tools that support the entire database systems engineering. The tools that are useful to the systems analyst and the systems designer are diagram editors, text editors, prototyping facilities, data dictionaries, and report generators.

**F.5.3 Integration of CASE tools or “I-CASE [52]”**

Structured analysis is a requirements analysis methodology. The focus is on modeling data and information flow. Data flow diagrams are especially useful for establishing communication links between the analyst and the end-user. The structured analysis based CASE tools hasten the composition and editing of the data flow diagrams. Requirements analysis is an iterative process involving a great deal of interaction between the user and the analyst. Many of the CASE tools available in the market are primarily requirements and design specification editors. The labor intensive paper work of drawing the data flow diagrams, entity-relationship diagrams, module hierarchies, and Warnier-Orr diagrams, is eliminated.

Traditional top-down specification methods lay emphasis on the flow of control of the systems and the documents produced by this method are text oriented. The primary goal of structured analysis is to produce a functional requirements document, that is largely graphic based, called the *systems requirements document*. 
CASE tools eliminate the extensive manual effort needed to represent the structure and processes of the systems. The conceptual schema is captured and the features and capabilities of the system are enumerated. The most crucial components of specification such as user interface and form designs are considered to be most difficult by McClure [49]. Schema specification is highly iterative, and involves great participation by the user with the analyst.

Individual CASE tools are useful and cost-effective. But, integration of tools [30, 55, 51, 52] is necessary for the cost-effectiveness of the overall system development. Standardization of tools is necessary to make such integrations feasible. The CASE environments can be either open or closed. In an open environment there are facilities to support the introduction of new tools to either replace the existing ones or to provide additional support, whereas in the closed environment the tools are integrated in the infrastructure of the environment. Introduction of new tools from outside this environment is impossible.

Basically, there are three kinds of integration options available in CASE technology [49]:

- data integration where a data model is shared by the CASE tools,
- user interface integration is based on the presence of a common user interface, and
- activity integration, where the environment incorporates a model of the software process. This model is used for the coordination of the tools.

**F.5.4 Software technologies**

There are five generations of software technology at present. The first and second generation technology deals with the low-level languages, namely machine languages
and assembly languages respectively. Third-generation tools are compilers, program editors, and libraries, test data generators and comparators, and performance monitors and optimizers. Fourth-generation tools are DBMS (DataBase Management System), fourth-generation languages, application generators and code generators. Fifth-generation tools are natural-language interpreters, speech recognizers, voice synthesizers, and parallel processors. CASE technology is a combination of the third-, fourth-, and fifth-generation technologies and it integrates the various techniques and tool components [49].

![Diagram of Five Generations of Software Technology](image)

**Figure 51** Five Generations of Software Technology [49]

**Third-generation technology** McClure [49] writes that high-level programming languages, like PL/1, PASCAL, C and Ada are associated with third-generation technology. Since these languages have been widely used in the development of application systems, they are likely to remain in use for a while in the future. The maintenance of such systems
is easy when CASE technology is used. High-level programming languages are extensively used in engineering and scientific fields. They facilitate embedded products like SQL (Structured Query Language). They are also used extensively in commercial and system application areas. Third-generation languages are called structured or modern programming languages. They are highly procedural and have data structuring capabilities. These languages are classified into general-purpose high-order languages and specialized languages. The technical characteristics of high-order languages include formal language theory and specification, and functional comparisons of specific languages. Specialized languages are designed to satisfy special requirements and often possess an unique syntax and form. The specialized languages that are used in software engineering are PROLOG, Lisp, Smalltalk, APL, and FORTH. These specialized languages are extremely useful in translating requirements into a design and for the implementation of such designs. But, the portability and maintainability of specialized languages is more difficult than that of general-purpose languages.

Fourth-generation technology Fourth-generation technology began in the late 1970's as simple report-generation tools and evolved in the 1980's to include capabilities of simple query operations for databases and very high level programming languages. Fourth-generation tools are classified as end-user tools and professional programmer tools. Examples of end-user tools are FOCUS, NOMAD, RAMIS-II, QMF, and QBE (Query By Example) [49]. Examples of professional programmer fourth-generation tools are SQL (Structured Query Language), IDEAL, ADS / ONLINE, and NATURAL [49]. Most of the tools are useful for the generation of required data and are useful in report generation and querying through non-procedural code. Fourth-generation languages are
classified according to the complexity of the applications that can be developed, and the user for whom it is suitable. Fourth-generation languages require less labor intensive programming work and relieve the developer of the procedural aspects of the programs. These languages are mostly non-procedural, as the programmer specifies what is to be done and not how it is to be done. The language and language supporting software are integrated for easy use and maintenance. Fourth-generation tools are successfully used in small-to-medium sized enterprises, and productivity is increased due to the use of non-procedural coding facilities and programming functions. If the application is concerned with accessing data that is to undergo very little or no processing, then the non-procedural query languages or report generators can be used where the user is willing to accept generic formats for screens, reports, and error messages. These languages provide the facilities of both third generation languages and automated tools.

The benefits of using fourth-generation technologies stated by McClure [49], include the fact that:

* application development time is faster,
* testing is easy due to the reliability of the code that was generated,
* the programs are easy to understand,
* the fourth-generation languages are user friendly,
* linked to data base management systems,
* self-documenting, and
* end-users can easily write and maintain fourth-generation application systems.

Since many of the fourth-generation tools are DBMS and operating-system specific their portability is limited, the tools lack concurrent data update features, recovery and
restart facilities, and are incompatible with systems written in second- and third-generation languages. The languages are not standardized, and therefore produce less efficient code and utilize a large amount of computer resources. The following table by McClure [49] lists the various underlying differences between fourth-generation technology and CASE technology.

<table>
<thead>
<tr>
<th>4GL</th>
<th>CASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>* End user</td>
<td>* Professional software developer</td>
</tr>
<tr>
<td>* Mainframe-based</td>
<td>* PC-based</td>
</tr>
<tr>
<td>* Tied to DBMS</td>
<td>* Tied to structured methodologies</td>
</tr>
<tr>
<td>* Special purpose</td>
<td>* General-purpose</td>
</tr>
<tr>
<td>* Menu-assisted</td>
<td>* Graphic-assisted</td>
</tr>
<tr>
<td>* Streamlining development process increases productivity</td>
<td>* Automated development process increases productivity</td>
</tr>
<tr>
<td>* Generic systems</td>
<td>* Customized systems</td>
</tr>
<tr>
<td>* Target implementation</td>
<td>* Target whole life cycle</td>
</tr>
</tbody>
</table>

Table 52 Fourth-generation technology vs CASE

**Fifth-generation technology** Fifth-Generation Technology has been used to deal with software systems that perform tasks usually associated with human intelligence. Examples of the fifth-generation technology are robots, expert systems, natural language processors, and automatic theorem provers [49].
§ F.6 Future trends

Each year there are many CASE tools being introduced in the market and, concurrently, many of the existing tools become obsolete. The current market for CASE tools is capricious, and it is imperative that the systems analyst be aware of the current methods and tools that can be successfully implemented.

CASE technology is a rapidly growing area, and the market for these tools has been growing steadily. CASE technology is very powerful right now and is expected to become more powerful as it evolves.

Individuals involved in software engineering and software development, have recognized the importance of having requirements specification that is as accurate as possible in the allocated time frame. Rapid prototyping has proved in many instances to be a valuable means of producing requirements specifications that satisfy the user and convey the system to the programmers. Prototyping and, especially, rapid prototyping is being recognized as a very important and integral part of SDLC.

§ F.7 Concluding comments

The entire database application system should be contemplated from the user’s point of view. The computer professionals have to bear in mind that the user takes precedence over other essentials. Earlier, the computer professionals were merely concerned with systems that work. Whether these systems addressed the user requirements or merely demonstrated the expertise of the computer professionals, remained unclear. The applicability of the systems were discovered only at the testing phase with user interaction in the actual working environment. Computer professionals involved in systems developed started using the methodologies and various tools and methods available in the industry,
with the realization that systems development is lucrative. Various organizations have taken and continue to play an important role in this rapidly evolving field of software engineering, and many standardization procedures were introduced. Yourdon, DeMarco, Gane, Sarson, Orr, Constantine, Chen and many others were instrumental in the development of structured methodologies since the 1970’s. The 1980’s brought prominence to the role of the user in the development of such systems. Since then, rapid strides have been made to increase user interaction in various stages of the software development life cycle by the use of automated tools and techniques.

The systems analysts and designers had ample support from tools and methods during the development process. Initially these tools and methodologies required intensive 'paper and pencil' labor. The concept of user interaction made the application of the methodologies and the use of tools time consuming. The increase in labor and the time consumed during the initial phases of requirements definition, analysis, and design caused the projects to often exceed the allocated resource levels. Owing to the fact that the projected allocations were exceeded, most systems analysts and designers continue with the remaining phases with inaccurate and/or incomplete specifications. These allocated resource levels were usually projected by the systems consultants based on the requirements and resources availability. The final system is often the direct outcome of such inaccurate specifications. The responsibility of creating the required system to user specifications depends solely on the systems analysts.

The technology of prototyping, to a great extent, was born out of the necessity to interact with the user. Prototyping has now evolved into a major area in software engineering. Active research is being conducted in the field of prototyping as can be
seen in the reviews of papers pertaining to this field. The need to use graphically
designed [75, 36, 42] and easy to modify prototypes to clarify user requirements and
user interface was the first step taken towards the introduction of computerized aid in
software engineering. Computer aided software engineering (CASE) is a relatively new
concept which gained recognition in the mid 1980s. The support and research in this
area by the industry has been tremendous. The scope of CASE tools and prototyping
is mind boggling, and except for the well-established tools the market is continuously
flooded with many short-lived CASE tools. The selection and introduction of CASE tools
for use during the database system development requires a lot of financial and hardware
resources. The system analyst must be well acquainted with the current technologies in
the market and must bear in mind the feasibility of upgrading these tools in the future.

CASE tools make a dramatic difference to the entire concept of systems analysis and
design. They improve the user interfaces, expedite development, and automate the work
of analyzing and designing systems. A great deal of research is directed to this vast and
rapidly evolving field of software engineering.

Prototyping and CASE tools have added an entirely new dimension to software
development.
Appendix G  Program Listing

§ G.1 SQL / DS

Student Grading Application System in SQL/DS
******************************************************************************

This is to create the various databases required for the application system.

Table "p_crse" contains information on various courses offered in a department. By default, it is assumed to be the School of Computer Science.

Master File.

create table p_crse -
  (crse_id smallint not null, -
   course_id char(9) not null, -
   semester char(3), -
   crse_desc varchar(254), -
   instructor varchar(254))

Table "p_std_rec" contains information of the students identification number and name.

Master File.

create table p_std_rec -
  (student_id integer not null, -
   student_name varchar(254) not null)

Table "p_exam_marks" holds the information of the students mid-terms and final exam marks.
Transaction File.

A mark of -1 is assigned to the who are absent for that test or exam.

create table p_exam_marks -
(crse_id smallint not null, -
student_id integer not null, -
class_test1 decimal(6,2), -
class_test2 decimal(6,2), -
final_exam decimal(6,2))

Table "p_assig_marks" holds information of the marks of the 10 assignments for each course.

Transaction File.

create table p_assig_marks -
(crse_id smallint not null, -
student_id integer not null, -
assig1 decimal(5,2), -
assig2 decimal(5,2), -
assig3 decimal(5,2), -
assig4 decimal(5,2), -
assig5 decimal(5,2), -
assig6 decimal(5,2), -
assig7 decimal(5,2), -
assig8 decimal(5,2), -
assig9 decimal(5,2), -
assig10 decimal(5,2))

Table "p_lw_lt" holds the lower limit for each grade except the 'F' grade for each course.

create table p_lw_lt -
(crse_id smallint not null, -
grade_l char(2) not null, -
lower_l smallint not null)
Table "p_gap" holds the old scale and new scale of the grade ruler. The new scale gap is initialised to the old gap by default.

create table p_gap -
  (crse_id smallint not null, -
   gap_old smallint not null, -
   gap_new smallint not null) [initialise to gap_old]

Table "p_wt" holds the various weightage assigned to each of the mid term test, final exam and for assignments.

create table p_wt -
  (crse_id smallint not null, -
   test1_wt decimal(4,2) not null, -
   test2_wt decimal(4,2) not null, -
   exam_wt decimal(4,2) not null, -
   assig_wt decimal(4,2) not null)

*********

Following views created automatically by SQL based on the data held in the tables. By default, it lists all the courses and students in each course, requires group by and order by clauses in the select statements. Can be used to view information of a particular course or student.

This view holds the information of the raw scores of all students in the tests, exam and assignments.

create view p_sorted_lt (course, student, name, -
test1,test2,exam,a1,a2,-
a3,a4,a5,a6,a7,a8,a9,a10) -
\text{as select e.crs\_id,e.student\_id,student\_name, -}
\text{class\_test1,class\_test2, -}
\text{final\_exam,assig1,assig2,assig3,assig4,assig5, -}
\text{assig6,assig7, -}
\text{assig8,assig9,assig10 -}
\text{from p\_std\_rec s,p\_exam\_marks e,p\_assig\_marks a -}
\text{where e.crs\_id=a.crs\_id -}
\text{and e.student\_id=a.student\_id -}
\text{and s.student\_id=e.student\_id}

This view holds the information of the weighted scores of all students in the tests, exam and assignments.

create view p\_tot (course,student,name,test1, -
test2,exam,assig) -
\text{as select e.crs\_id,e.student\_id,student\_name, -}
\text{test1\_wt*class\_test1, -}
\text{test2\_wt*class\_test2,exam\_wt*final\_exam, -}
\text{assig\_wt*(assig1+assig2+assig3+assig4, -}
\text{assig5+assig6+assig7+assig8+assig9+assig10) -}
\text{from p\_std\_rec s,p\_assig\_marks a,p\_exam\_marks e, -}
\text{p\_wt w -}
\text{where e.crs\_id=a.crs\_id -}
\text{and e.crs\_id=w.crs\_id -}
\text{and e.student\_id=a.student\_id -}
\text{and e.student\_id=s.student\_id}

This view holds the information of the total marks of all students

create view p\_tott (courseid,studentid, -
studentname,total\_marks) -
\text{as select course,student,name, -}
(test1+test2+exam+assig) -
from p_tot

This view tabulates any changes of the grade scale.

create view p_gaps (course,diff) -
as select crse_id,gap_old-gap_new -
from p_gap

This view calculates the lower limit to each grade 'A+' to 'D-' for every course.

create view p_grades (course,grade,lower_limit) -
as select l.crse_id,l.grade_l,(l.lower_l+g.diff) -
from p_lw_lt l,p_gaps g -
where l.crse_id=g.course

The grades to students are assigned [A+ to D-]

create view p_f_grades (course,student,marks,grade) -
as select courseid,studentid,total_marks,g.grade -
from p_tott t,p_gap ga,p_grades g -
where t.courseid=ga.crse_id -
and t.courseid=g.course -
and total_marks between lower_limit and -
(lower_limit+gap_new)

The range for the F grade is calculated

create view p_fail (crse,fail_up_lt) -
as select crse_id,lower_l -
from p_lw_lt -
where grade_l='D-'

List of students who have failed a course
create view p_f1_grades (course, student, marks, grade) -
    as select courseid, studentid, total_marks, 'F' -
      from p_tott t, p_fail f -
      where courseid = f.crse -
      and total_marks between 0 and fail_up_lt -

List the students who were absent for any test or exam.

create view p_ab (course, student, name, test1, -
    test2, exam) -
    as select e.crse_id, e.student_id, s.student_name -
      , class_test1, class_test2, final_exam -
      from p_std_rec s, p_exam_marks e -
      where class_test1 = -1 or class_test2 = -1 or -
      final_exam = -1 -

Sorting records by student identity

To view the exam mark database.

select * from p_exam_marks -
      where crse_id = '??' -
      order by student_id -

select student, name, test1, test2, exam -
      from p_sorted_lt -
      where course = ?? -
      order by student -

To view the exam mark database.

select student, name, a1, a2, a3, a4, a5, a6, a7, a8, -
        a9, a10 -
      from p_sorted_lt -
      where course = ?? -
      order by student -

176
To view the student info

select * -
from p_std_rec -
order by student_id

select student.name -
from p_sorted_lt -
where course = ?? -
order by student

DELETION of records

delete from p_std_rec -
where student_id='??'

delete from p_exam_marks, p_assig_marks -
where p_exam_marks.crse_id='??' -
and p_assig_marks.crse_id=p_exam_marks.crse_id -
and p_exam_marks.student_id='??' and -
p_assig_marks.student_id=p_exam_marks.student_id

INSERTION of records

input table_name [default allfields]

insert into table_name -
values ( , , , )

TOTAL marks

select * -
from p_tot -
where course ='??' -
union -
select total_marks -
from p_tott -
where courseid=p_tot.course -
and studentid=p_tot.student

GRADES

select * -
from p_f_grades -
where course=100 -

select * -
from p_f1_grades -
where course=100

HISTOGRAM

cREATE VIEW p_g (crse,grades,number) -
AS SELECT course,grade,COUNT(DISTINCT grade) -
FROM p_f_grades -
group by course,grade

select * from p_g

cREATE VIEW p_f (crse,grades,number) -
AS SELECT course,grade,COUNT(DISTINCT course) -
FROM p_f1_grades -
group by course

select * from p_f

GAP

update p_gap -
set gap_new = '??' -
where crse_id
DATA : AVAILABLE ON THE SYSTEM

input p_crse

100,'03-60-100','F93','Introduction to Computers','Dr.Frost'
104,'03-60-104','F93','Basics in computer science','Doug Thistle'
110,'03-60-110','W92','Turing Course - I','Dr.Kent'
111,'03-60-111','W92','Turing Course - II','Dr. Channen'
212,'03-60-212','F94','C Programming Concepts','Steve Karamatos'

input p_std_rec

970236578,'Gregory Peck'
890000384,'Scarlett OHara'
903247834,'Tom Cruise'
904378432,'Robert Redford'
890343848,'Julian Sands'
904374837,'Winston Churchill'
869832747,'Scarlet Pimpernel'

input p_exam_marks

100,903247834,null,null,null
100,890000384,90,100,100
100,904378432,85,67,90
100,869832747,89,98,87
100,890343848,98,87,99
100,970236578,-1,-1,90
100,904374837

input p_assig_marks

100,903247834,9,8,4,5,0,0,9,6,10,null,null
100,890000384,9,10,10,10,10,10,9,8,8,10
100,904378432,8,5,6,7,9,0,0,null,null,10
100,869832747,8,9,9,8,8,7,8,10,10,10
100,890343848,9,8,8,7,9,9,8,8,9,5
100,970236578,10,9,8,7,5,10,10,10,10,9

179
input p_lw_lt

100,'A+',96
100,'A',92
100,'A-',88
100,'B+',84
100,'B',80
100,'B-',76
100,'C+',72
100,'C',68
100,'C-',64
100,'D+',60
100,'D',56
100,'D-',52
100,'F',48

input p_gap

100,4,4
212,4,4
110,4,4
111,4,4

input p_wt

100,.2,.2,.4,.2
110,.15,.2,.5,.15

§ G.2 C

#include <string.h>
#include <stdio.h>
#define SCALE 4
#define CRSE '03-60-'
int id=0;
long sid=0;
int amarks[10]={0,0,0,0,0,0,0,0,0,0};
int emarks[3]={0,0,0};
int scale=0;
int no=0;
int wtsscale[3]={0,0,0};
int wts=0;
int entry=0;
char new_ln;
int cresse[3]={0,0,0};
long student[3]={0,0,0};
int ans=0;

struct students_rec {
  int student_id;
  char student_name[15];
} info_s;

struct students_marks {
  int cresse_id;
  int stud_id;
  float class_test_1;
  float class_test_2;
  float exam_marks;
} info_se;

struct assig_marks {
  int cresse_id;
  int student_id;
  float assig_marks[10];
} info_sa;

struct weights {
  int cresse_id;
  int wt_class_1;
  int wt_class_2;
  int wt_exam;
  int wt_assig;
} info_w;

/* the lower limit is set for grades from
A+ to D- : with the gap a */
/* constant 4 points */

struct grade_scale {
    char letter[2];
    int lower;
};

struct grades {
    int crse_id;
    struct grade_scale grade[13];
} info_sc;

struct course_rec{
    int crse_id;
    char course_id[9];
    char semester[3];
    char crse_desc[15];
    char prof[10];
}info_c;

struct gap_rec {
    int crse_id;
    int gap;
} info_gap;

struct grade_info {
    int crse_id;
    long student_id;
    float total;
    char grade[2];
} info_g;

main()
{
    int scr1op, scr2op;
    int scr0(), scr1(), scr2(), scr3(), scr4();
    int scr5(), enter();
    void new(int), rpt(int);
}
char qu='n';
char x;
while(qu=='n')
{
    switch(scr0()) {
        case 1:
            scr1op=scr2();
            if (scr1op<3)
                new(scr1op);
            break;
        case 2:
            scr2op=scr1();
            if (scr2op<5)
                rpt(scr2op);
            break;
        case 3:
            printf("\n Do you really want to quit [n/y] ? : ");
            scanf("%c",&x);
            scanf("%c",&qu);
            break;
        default:
            qu='y';
            printf("\n Aborted (error) \n");
            break;
    }
}
printf("\n You have completed session ! \n\n");
}

int scr0()
{
    entry=0;
    do {
        printf("\n \nA : MAIN MENU \n");
        printf("\n \n1 : Input Data, \n");
        printf("\n \n2 : Reports \n");
        printf("\n \n3 : Quit \n");
        entry=enter();
    }while(entry>3);
    return(entry);
```c
}

int enter()
{
    int temp=0;
    printf("\n \n Please enter valid choice : \t");
    scanf("%d", &temp);
    return(temp);
}

int scr1()
{
    entry=0;
    do {
        printf("\n \nC : REPORT MENU \n");
        printf("\n l : Student List \n");
        printf("2 : Histogram \n");
        printf("3 : Student Grades \n");
        printf("4 : Absentee Lists \n");
        printf("5 : Previous Screen \n");
        entry=enter();
    } while(entry>5);
    return(entry);
}

int scr2()
{
    entry=0;
    do {
        printf("\n \nB : INPUT DATA MENU \n");
        printf("\n l : Create New Records \n");
        printf("2 : Update Records \n");
        printf("3 : Previous Screen \n");
        entry=enter();
    } while(entry>3);
    return(entry);
}

int scr3()
{
    entry=0;
    do {
        
        
```
```
printf("\n\nD : UPDATE RECORDS MENU \n");
printf("\n\n1 : Insert Student Record \n");
printf("\n\n2 : Delete Student Record \n");
printf("\n\n3 : Update Course Record \n");
printf("\n\n4 : Update Student Record \n");
printf("\n\n5 : Update Grade Scale \n");
printf("\n\n6 : Update Weights \n");
printf("\n\n7 : Previous Screen \n");
entry=enter();
    } while(entry>7);
return(entry);
}

int scr4()
{ entry=0;
do {
    printf("\n\nF : STUDENT MENU\n");
    printf("\n\n1 : All Students \n");
    printf("\n\n2 : Individual Student Record \n");
    printf("\n\n3 : Previous Screen \n");
    entry=enter();
    } while(entry>4);
return(entry);
}

int scr5()
{ entry=0;
do {
    printf("\n\nE : CREATE NEW RECORDS MENU \n");
    printf("\n\n1 : New Course \n");
    printf("\n\n2 : New Student \n");
    printf("\n\n3 : Previous Screen \n");
    entry=enter();
    } while(entry>3);
return(entry);
}

void new(rec)
int rec;
{
    int scr5op,scr3op,l_rec;

185
void info(int), update(int);
char cont='y';
l_rec=rec;
while (cont=='y') {
    switch(l_rec) {
    case 1 :
        scr5op=scr5();
        if (scr5op<3)
            info(scr5op);
        break;
    case 2 :
        scr3op=scr3();
        if (scr3op<7)
            update(scr3op);
        break;
    case 3 :
        printf("\n Previous screen coming right up ! \n");
        cont='n';
        break;
    default :
        printf("\n Encountered error");
        cont='n';
        break;
    }
    l_rec=scr2();
}

void info(rec)
int rec;
{
    extern int crse[3];
    char temp[9], cont='y';
    int l_rec, i;
    int ccheck_id(int);
    int check_stdid(int);
    int check_str(char temp[], int);
    extern int id;
    l_rec=rec;
    id=0;
printf("\n \n \n Please enter : \n");
if (l_rec==1)
{
printf("\n Course id [eg. 212] : \t");
scanf("%d",&id);
if(check_id(id)==0)
{
    for(i=0;i<3;i++)
    {
        if(crse[i]==0)
        {
            crse[i]=id;
            info_c.crse_id=id;
            printf("\n DATA ACCEPTED !!!");
i=4;
        printf("\n Course code [eg. 03-60-212] : \t");
        scanf("%s",temp);
        if(check_str(temp,id)==0)
        {
            strcpy(info_c.course_id,temp);
            printf("\n Semester : \t \t");
            scanf("%s",info_c.semester);
            printf("\n Description of course : \t");
            scanf("%s",info_c.crse_desc);
            printf("\n Instructors name : \t \t");
            scanf("%s",info_c.prof);
            printf("\n Successfully received data \n");
        }
        else printf("Wrong not 03-60-XXX");
        }
    }
else if(l_rec==2)
{
printf("\n Student Id [eg. 890788364] : \t");
scanf("%d",&id);
info_s.student_id=id;
if(check_stdid(id)==0)
{
    for(i=0;i<3;i++)
{ 
if(student[i]==0) 
{ 
student[i]=id; 
i+=4; 
printf("\n Name of Student : \t \t"); 
scanf("%s",info_s.student_name); 
printf("\n Successfully received data \n"); 
} 
} 
} 
/* 1_rec=scr5(); */ 
}

int check_str(identity,id) 
char identity[]; 
int id; 
[ 
extern int crse[3]; 
int result=0,i,temp1=0,ccrse=6; 
char temp[4]; 
char tmp_crse[7]=['0','3','-',',','6','0','-',',',']}; 
result=strncmp(identity,tmp_crse,6); 
for(i=0;i<4;i++) 
{ 
temp[i]=identity[ccrse++]; 
temp1=atoi(temp); 
if (temp1!=id) 
{ 
result=1; 
return(result); 
}

int check_stdid(identity) 
int identity; 
[ 
extern long student[3]; 
int i,result=0; 
for(i=0;i<3;i++) 
{ 
if(student[i]==identity) 
{ 

188
result=1;
printf("\n Student record exists !!");
i=4;
}
else printf("\n Student record does not exist !!");
}
return(result);

int check_id(identity)
int identity;
[
extern int crse[3];
int i,result=0;
for(i=0;i<3;i++)
[ if(crse[i]==identity)
[
result=1;
printf("\n Course exists !!");
printf("%d,%d,%d",i,crse[i],result);
i=4;
]
else printf("\n Course does not exist !!");
]
return(result);
]

void update(rec)
int rec;
[
int l_rec;
int check_id(int);
int check_stdid(int);
void enters(char,int,int);
char cont;
char check;
char indication;
cont='y';
check = 'n';
}
indication='a';
1_rec=rec;
while(cont=='y') {
    switch(1_rec) {
        case 1: printf("\n Insertion of student marks");
            printf("\n Enter course id ");
            scanf("%d",&id);
            if(check_id(id)==1)
            {
                printf("\n Enter student id");
                scanf("%f",&sid);
                if(check_stdid(sid)==1)
                {
                    printf("\n Indicate assignment or exam marks [a/e]");
                    scanf("%c",&indication);
                    enters(indication,id,sid);
                    printf("\n Entry completed");
                }
            }
            break;
        case 2: printf("\n Deletion of records");
            printf("\n Enter course id ");
            scanf("%d",&id);
            printf("\n Enter student id");
            scanf("%f",&sid);
            printf("\n Are your sure ? [y/n]");
            scanf("%c",check);
            if (check=='y')
                printf("\n Record for course %d , Student %d deleted",id,sid);
            else printf("\n not deleted");
            break;
        case 3:
            printf("\n Enter course id ");
            scanf("%d",&id);
            printf("\n updating course records");
            break;
        case 4:
            printf("\n update student record");
            printf("\n Enter student id : ");
            scanf("%d",&sid);
            }
printf("\n Assignments or exam [a/e] : ");
    indication=getchar();
    enters(indication,id,sid);
    break;
    case 5:
    printf("\n update grade scale data");
    printf("\n Enter Course : ");
    scanf("%d",&id);
    printf("\n Enter grade scale : ");
    scanf("%d",&scale);
    break;
    case 6:
    printf("\n update weights data");
    printf("\n Enter Course : ");
    scanf("%d",&id);
    scanf("%d",&no);
    printf("\n Enter weights change : ");
    scanf("%d",&wts);
    wtsscale[no]=wts;
    break;
  default:
    printf("\n default option Invalid");
    break;
  }
printf("\n Do you wish to continue ? [n/y]");
scanf("%c",&new_ln);
scanf("%c",&cont);
}
printf("\n Over");
}

void enters(indication,crseid,stdid)
char indication;
int crseid;
long stdid;
{
    char cont;
    int mark;
    int no;
cont='y';
while(cont=='y'){
    switch(indication){
    case 'a':
        printf("\n Enter assignment number [1..10] : ");
        scanf("%d",&no);
        printf("\n Enter mark");
        scanf("%d",&mark);
        amarks[no]=mark;
        break;
    case 'e':
        printf("\n Enter [1] test 1, [2] test 2,
               [3] exam marks : ");
        scanf("%d",&no);
        printf("\n Enter marks");
        scanf("%d",&mark);
        emarks[no]=mark;
        break;
    default: break;
    }
    printf("\n Continue [n/y] ");
    scanf("%c",&new_ln);  
    scanf("%c",&cont);
}
printf("\n \n Entries over");
}

void rpt(rec)
int rec;
{
    int l_rec,scr4op,code;
    void generate(int,int);
    char cont='y';
    l_rec=rec;
    while(l_rec<5 && cont=='y') {
        printf("Enter course code \t : ");
        scanf("%d",&code);
        switch(l_rec) {
        case 2:
            printf("\n VIEW of the Required Histogram \n");
            break;
        }
case 5:
    printf("\n \n Previous screen \n\n");
    cont='n';
    break;
default:
    scr4op=scr4();
generate(l_rec,scr4op);
    printf("\n Done");
    break;
}
l_rec=scr1();
}
printf("\n\nRpts over & out \n\n");
}

void generate(rpt_value,all_ind)
int rpt_value,all_ind;
{
    int std;
    switch(rpt_value) {
    case 1:
        if (all_ind==1)
            printf("\n Display all student records");
        else {
            printf("\n Display Individual record ");
            printf("\n Enter student identity : ");
            scanf("%d",&std);}
        break;
    case 2:
        if (all_ind==1)
            printf("\n Display all student grades");
        else {
            printf("\n Display Individual student grade ");
            printf("\n Enter student identity : ");
            scanf("%d",&std);}
        break;
    case 3:
        printf("\n Displaying all absentees");
        break;
default:
break;
}
printf("\n Done display");
}

§ G.3 C++

/* Prototype 3rd cycle */
/* proto3.c++ */

#include <iostream.h>
#include "protobas.h"
#include "protocrs.h"
#include "protostd.h"
#include "protobas.c++"
#include "protocrs.c++"

main()
{
    base ob;
    course crse_ob;
    student std_ob;
    int crseid;
    char stdid[11];
    char option='n';
    char op='n';
    cout << "\n\nWelcome to the Course/Student Grade system \n";
    cout << "\n\tDo you wish to start the process [y/n] ? : ";
    cin >> option;
    while(option=='y' && op=='n')
    {
        switch(ob.main_menu())
        {
        case 1 : switch(ob.input_menu())
        {
        case 1 : cout << "\nCreate new course offering\n";
            crse_ob.get();
            break;
case 2: cout << "\nModify Course offering\n";
crse_ob.modify();
break;
case 3: cout << "\nAdd a student to course\n";
cout << "\nPlease, enter the course id [eg. 60-100];
cout << " should be entered as 100] \n";
cin >> crseid;
crse_ob.add_std_to_crse(crseid);
break;
case 4:
cout << "\nUpdate all student records in a course\n";
cout << "\nPlease, enter the course id [eg. 60-100 ];
cout << " should be entered as 100] \n";
cin >> crseid;
std_ob.update(crseid);
break;
case 5:
cout << "\nUpdate a student record in a particular ";
cout << "course\n";
cout << "\nPlease, enter the course id [eg. 60-100 ];
cout << " should be entered as 100] \n";
cin >> crseid;
cout << "\nPlease, enter the student id ";
cout << "[eg. 890-123456] \n";
cin.getline(stdid,11);
std_ob.update(crseid,stdid);
break;
case 6: cout << "\nPrevious screen !!!\n";
break;
default: cout << "\nInvalid option\n";
break;
}
break;
case 2: switch(ob.view_menu()) {
case 1:
cout << "\nView various student list\n";
switch(ob.view_all_std_menu()) {
case 1: cout << "\nPlease, enter course id.\t";
cin >> crse;
break;
case 2: cout << "\nDisplay of all students in";
cout << " all courses\n";
break;
case 3 : cout << "\nPrevious screen";
    break;
default : break ;
}
break;
    case 2 :
cout << "\nView student list\n";
cout << "\nPLease, enter student identity";
cout << " number : ";
cin >> std;
switch(ob.view_a_std_menu()) {
case 1 : cout << "\nAll courses\n";
    break;
case 2 : cout << "\nPLease enter course ";
    cout << " identity : ";
cin >> crse;
cout << "\nStudent " << std;
cout << " information in course " << crse;
    break;
case 3 : cout
    break;
case 3 : cout << "\n View of course records\n"
    break;
case 4 : cout << "\nPrevious Menu"
    break;
default : << "\nPrevious screen"
    break;
default : break ;
}
break;
    case 3 : cout << "\nListing of all course"
    cout << " descriptions\n"
    break;
}
break;
case 3 : cout << "\nDo you really wish to quit"
    cout << " [y/n] ? : ";
cin >> op;
break;
cout << "\n\nProcess terminated as requested !\n";
return(0);
}

/* protobas.h */

class base {
private:
  int prim;
  int sec;
  int tert;
  int quart;
public:
  void printing()
  { cout << "\n\tOutput Being printed ! "; }
  int main_menu();
  void ordered_by();
  int view_menu();
  int input_menu();
  int view_all_std_menu();
  int view_a_std_menu();
  base() { prim=0; sec=0; tert=0; quart=0; }
  ~base() { }
};

/* protobas.c++ */

int base::main_menu()
{
  cout << "\n |---------------------------------";
cout << "----------------------------------|\n";
cout << "\n\nt A Main Menu";
cout << "\n\n\nt 1] Input (Data) Menu";
cout << "\n\n\nt 2] View Menu" << "\n\n\nt 3] Quit";
    cout << "\n\n\nt Please, enter choice : ";
int i=0;
cin >> i;
return i;
}

int base::input_menu()
{
cout << "\n |----------------------------------";
cout << "\n|\n";
cout << "\n\nt B Input (Data) Menu";
cout << "\n\n\nt 1] Create new course offering";
cout << "\n\n\nt 2] Modify course offering";
cout << "\n\n\nt 3] Add student(s) to course";
cout << "\n\n\nt 4] Update all student records in a ";
cout << " course";
cout << "\n\n\nt 5] Update particular student";
cout << " in a particular course";
cout << "\n\n\nt 6] Previous Screen";
cout << "\n\n\nt Please, enter choice : ";
int i=0;
cin >> i;
return i;
}

int base::view_menu()
{
cout << "\n |----------------------------------";
cout << "\n|\n";
cout << "\n\nt C View Menu";
cout << "\n\n\nt 1] View various student list";
cout << "\n\n\nt 2] View a particular student record";
cout << "\n\n\nt 3] View of course records";
cout << "\n\n\nt 4] Previous Screen";
cout << "\n\n\nt Please, enter choice : ";
int i=0;
cin >> i;

return i;
}

int base:: view_all_std_menu()
{
    cout << "\n|----------------------------------------|
    cout << "\n----------------------------------|\n";
    cout << "\n\n\n\t D View Student Record Menu";
    cout << "\n\n\n\t 1] View all students in a course";
    cout << "\n\t 2] View all students in all courses";
    cout << "\n\t 3] Previous Screen";
    cout << "\n\n\t Please, enter choice : ";
    int i=0;
    cin >> i;
    return i;
}

int base:: view_a_std_menu()
{
    cout << "\n|----------------------------------------|
    cout << "\n----------------------------------|\n";
    cout << "\n\n\n\t E View A Student Record Menu";
    cout << "\n\n\n\t 1] In all course offerings";
    cout << "\n\t 2] In a particular course offering";
    cout << "\n\t 3] Previous Menu";
    cout << "\n\n\t Please, enter choice : ";
    int i=0;
    cin >> i;
    return i;
}

void base:: ordered_by()
{
    cout << "\n|----------------------------------------|
    cout << "\n----------------------------------|\n";
    cout << "\n\n\n\t F Ordered by";
    cout << "\n\n\n\t 1] Course\n\n\n\n\t 2] Date \n\n\n\n\t 3] ;
    cout << "\n\n\n\n\t Please, choose any three options for ";

cout << "order of sorting answers.\n";
cout << "\n\n  Primary key [eg. 1] : ";
cin >> prim;
cout << "\n\n  Secondary key [eg. 2] : ";
cin >> sec;
cout << "\n\n  Tertiary key [eg. 3] : ";
cin >> tert;
cout << "\n\n  Final key [eg. 4] : ";
cin >> quart;
}
Bibliography


An Assessment of the Prototyping Approach to Information Systems Development.


The STARTS Guide Vol I and II: A Guide to Methods and Software Tools for the
Construction of Large Real-time systems.


Operational Specifications as the Basis for Rapid Prototyping.


A Survey of Software Engineering Practice: Tools, Methods, and Results.


An Extendable Approach to Computer-Aided Software Requirements Engineering.


DBE: An Expert Tool for Database Design.


Formal Methods in Software Development Requirements for a CASE.


*Software Engineering: A Holistic View.*


*Application Prototyping: a requirements definition strategy for the '80s.*


*A Spiral Model of Software Development and Enhancement.*

       Prototyping versus Specifying: A Multiproject Experiment.

       Some Experience with Automated Aids to the Design of Large-Scale Reliable Software.

       Programming Techniques and Practice.

       Software Technology People Can Really Use.

       Computer-Aided Software Engineering (CASE).

[18] Mary Ann Cummings.
       The Development of User Interface Tools for the Computer Aided Prototyping System.

       A Hypertext Based Software Engineering Environment.


CASE Seen From Both Sides of the Fence.


Tools Fair: Out of the Lab, onto the Shelf.


The CASE Philosophy.

Stop the Life-Cycle, I want to get off.


[28] Robert L. Glass.
Building Quality Software.

[29] Hassan Gamaa.
The Impact of Prototyping on Software System Engineering.

Integrated CASE for Cleanroom Development.

Software prototyping — progress and prospects.


[33] Jeff Kramer.
In Third European Software

[34]James M. Kraushaar and Larry E. Shirland.
A Prototyping Method for Applications Development by End Users and Information Systems Specialists.


Foundations of Software Design.

[37]Chi Y. Lin and Reuven R. Levery.

[38]Luc Duponcheel Luc Jadoul and Willy Van Puymbroeck.
An Algebraic Data Type Specification Language and its Rapid Prototyping Environment.

[39] Luqi.

Knowledge-Based Support for Rapid Software Prototyping.


Winter.

[40] Luqi.

Software Evolution Through Rapid Prototyping.


[41] Luqi.

Computer-Aided Prototyping For A Command-And-Control System Using CAPS.


[42] Luqi, P. D. Barnes, and M. Zyda.

Graphical Tool for Computer-Aided Prototyping.


[43] Luqi and Valdis Berzins.

Software Engineering with Abstractions.


[44] Luqi and Mohammad Ketabchi.

A Computer-Aided Prototyping System.


[45] Charles F Martin.


Introducing Relational Databases.

NCC — The National Centre for Information Technology, Manchester, UK, 1983.


The CASE for Structured Development.


The CASE Experience.


CASE is Software Automation.


CASE Tools and Software Factories.


Working Together to Integrate CASE.


CASE at the Start of the 1990’s.


[54] Ken Orr, Chris Gane, Edward Yourdon, Peter P. Chen, and Larry L. Constantine.

Methodology : The Experts Speak.


An Integrated Environment for Requirements Engineering.


[56] Shari L. Pfleeger.

Lessons Learned in Building a Corporate Metrics Program.


boyd & fraser publishing company, 1990.

[58] Roger S. Pressman.

Software Engineering : A Practitioner’s Approach.


[59] B. Ratcliff.

Early and Not-so-early Prototyping — Rationale and Tool Support.

[60] Donald T. Ross.

Applications and Extensions of SADT.


Structured Analysis for Requirements Definition.


Software Engineering.

Aksen Associates Incorporated Publisher, 1993.

[63] Veikko Seppänen, Marko Heikkinen, and Raino Lintulampi.

SPADE : Towards CASE Tools that can Guide Design.


Systems Analysis and Design.


[65] Ian Sommerville.

Software Engineering.

Addison-Wesley Publisher’s Ltd., 1992.

CASE makes strides toward Automated Software Development.


[67]Daniel Teichroew and Ernest A. HersheyIII.


Rapid Prototyping in Software Development.


Guest-Editor’s Introduction.


*System and Software Requirements Engineering*.


[70]Iris Vessey, Srikka L. Jarvenpaa, and Noam Tractinsky.

Evaluation of Vendor Products: CASE Tools as Methodology Companions.


[71]Scott V. Gordon and James M. Bieman.

Rapid Prototyping and Software Quality : Lessons From Industry.


Experimentation in Software Engineering.

[73] John Voelcker.

Automated Software: proceed with caution.


[74] Roland Vonk.

Prototyping: The effective use of CASE Technology.

Prentice Hall. 1990.


A Graphical, Extensible Integrated Environment for Software Development.

Appendix H  VITA AUCTORES

Arathi Ranganathan was born in 1969 in India. She graduated from Sacred Heart Matriculation Higher Secondary School in 1986. From there she went on to the Stella Maris College, University of Madras, India where she obtained a Bachelor of Science in Mathematics in 1989. She then completed an International Diploma in Computer Studies, National Computing Centre (UK), India in 1990. She is currently a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in the Spring of 1994.