Chapter 1

Introduction

1.1 Research Objectives

The main objective of this research is to propose a comprehensive framework for quality and risk management in software development process based on analysis and modeling of software metrics data. Existing software metrics work has focused mainly on the type of metrics to be collected and the analysis of metrics data, particularly with respect to complexity and reliability analysis of large software programs. The underlying data model with which the metrics data are stored has not been a consideration for software researchers to date, and researchers have adopted various data models with little regard to their relative efficiency. However, the underlying data model should not be taken for granted. The adoption of the "correct" data model greatly improves the efficiency of queries and the maintainability of the metrics database, if the type of queries that will most likely be posed can be predicted.

Figure 1.1 illustrates a high-level view of all the important components of a software metric database and data analysis system. The shaded areas of this figure highlight the research issues to be addressed in this work. As illustrated in this figure software metrics data are collected and input to the metrics database via the DBMS Application Interface. In the case of relational databases, for instance, this can be via SQL. The DBMS Application Interface sends the data down the data management layer to the Storage Management Layer, which then stores the metrics data on disk. To analyze the metrics data, it must be retrieved from disk. This is performed by the application via the DBMS Application Interface, which sends the request down to the data management layer and then to the storage management layer, which then sends

metrics data up the same route to the metrics analysis software. Our research is focusing on the following three issues, which provide the major contribution of this dissertation:

- determining a core set of metrics, hereafter referred to as the Test and Evaluation metrics, that can most effectively be used across various organizations for risk and quality management,
- identifying which metrics data analysis techniques are best suited to the type of diagnostic and prognostic analyses to be performed with the Test and Evaluation metrics thus identified. Accordingly we present a framework that integrates a suite of analytical technique for assisting in diagnostic and prognostic analysis.
- proposing a comprehensive data model for the metrics database, given the nature of temporal characterization of the data and the type of complex queries to be performed on metrics data. Subsequently we compare various data modeling approaches within the context of practical implementation of a system.

The contributions are discussed in more detail in the following chapters. The remaining sections of this chapter provide background in software metrics and notion of risk/quality management with a software project. We also provide a classification of the proposed metrics and provides a brief description of each metric. The detailed discussion of the metrics is given in Chapter 2.

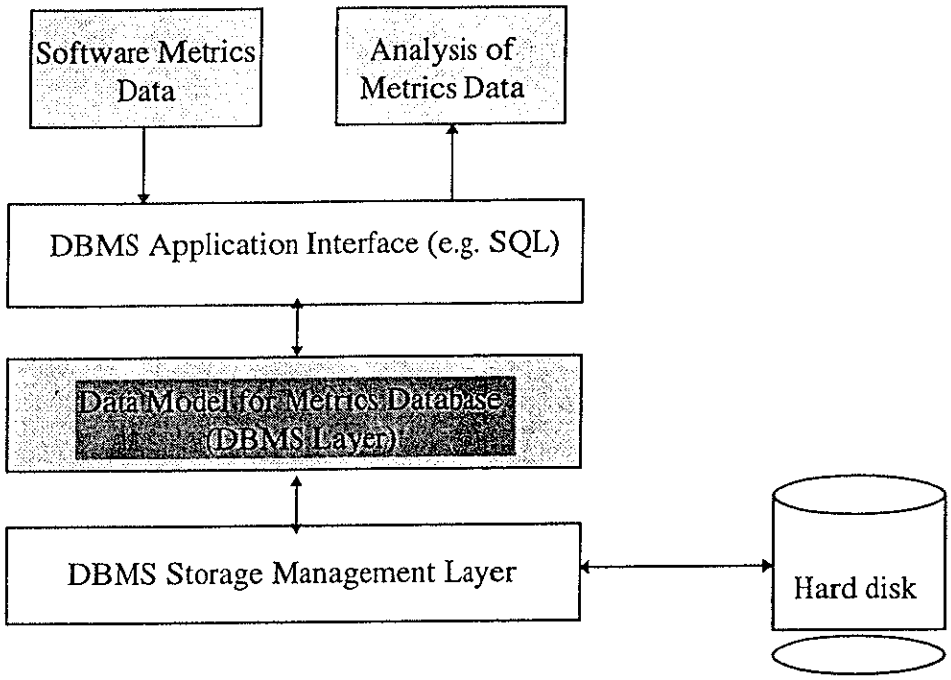

Figure 1.1: Areas of Research Focus

1.2 The Concept of Software Risk

From an engineering viewpoint, risk is "the combination of the probability of an abnormal event or failure and the consequence(s) of that event or failure to a system's operators, users, or its environment" [BOE91]. Risk is present in every phase of the software life cycle. In the requirements phase, for example, end-user requirements may change frequently, affecting the coding of the software. Likewise, the software design may be faulty, either with respect to end-user requirements or to system functionality. Hence the design may need to be modified. This can result in substantial schedule delay. For the end user, such delay may result in lost business opportunities. Therefore, scheduled delay is considered as a typical high risk element in software development.

In order to deal with risks properly and effectively, a framework is needed for identifying and analyzing potential risks and developing appropriate contingency plans [BOE89]. By dealing with risks quantitatively, risk analysis and management can be more proactive and objective. Prioritizing high risk items then becomes easier and more accurate. Objective analysis and contingencies help to promote effective management analysis techniques as standardized engineering methods.

Risk analysis can be classified into three major phases: (1) risk identification and assessment, which involves identifying the potential risks in the software life cycle and assessing their impacts; (2) risk minimization/prevention, which involves taking steps to minimize or prevent the risks identified during the early phase, and (3) risk compensation, which involves taking appropriate steps to correct the risky behavior or characteristic after it has occurred.

1.3 Motivation for Metrics

Software is pervasive though largely unnoticed by the developers. One would be hard put to name any industry in which software is not now a significant component. The software systems being developed today are much more sophisticated than the simple payroll and accounting systems of a few years ago. Today's systems operate in real time and may be used by many people simultaneously on networks. They launch space shuttles, command missile defense systems and control billions of dollars through money transfer systems.

The economic potential and growing significance of software as an agent of productivity is, however, not sufficiently appreciated. We might take as an example manufacturing. Fortune magazine in November 1994 featured an article on this topic leading off with the statement: "With speed and flexibility that leaves the Japanese agog, US. manufacturers have come roaring back after years in eclipse. What is the secret? It's in the software". The use of software in

manufacturing is not just a help, it has become a key factor. "by the year 2000, well over 50 per cent of the cost of producing manufactured goods will be the software necessary in that manufacture" as mentioned in [GIB94]. Despite its emerging role as the critical technology for various sectors of economy, software development is far from being a mature engineering discipline it ought to be. Studies have shown that for every six new large-scale software systems that are put into operation, two others are canceled. The average software development project overshoots its schedule by half; larger projects generally do worse. And some three quarters of all large systems are "operating failures" that either do not function as intended or are not used at all." Typically, the larger the system, the more difficult the problem.

The overall cost of producing and maintaining software are significant factors in a country's economy. The outlay for producing and maintaining software systems in the United States is about $100 billion a year [KEY93]. These figures approximate 1% of GDP and, although not mind-boggling, they are certainly significant. Costs range from $1,000 a line for debugged "mission-critical" software [KEL90] to $50 to $100 a line for marketable "shrink-wrapped" PC software. At the low end, that's still more than $25 million for a word-processing system [KEY93]. The cost of producing the initial software is not as major a part of the cost as software maintenance is. Maintenance over the lifecycle of the software will cost more than twice what it cost to produce the original software. The costs involved in debugging, delays and project cancellations are enormous. They constitute as much as 90% of the total cost of software. If those costs could be eliminated, the direct savings to the community would be a real windfall - perhaps $90 billion a year for the United States.

One industry rule of thumb suggests that initially there may be as many as 25 errors for each 1,000 lines of code. Review and testing soon remove most of the errors but "debugging", as

it is called, is expensive, time-consuming and ultimately unsatisfactory because there can be no certainty that all the bugs have been identified and removed. An IBM study [KEY93] revealed that in well debugged programs, about a third of the remaining bugs are so deeply hidden that they would probably cause only one failure in 5,000 years. There are likely to be errors in even the best computer programs which in our opinion is OK, as long as we have some estimate of the risk associated with having such errors in the software; also, one should not do a 'worst-case scenario' analysis since that is unnecessarily alarmist.

In so-called "shrink-wrapped" software [KEY93], most of the tasks are not critical, so the incidence of bugs can be expected to be higher in PC software than in so-called mission-critical applications such as the software used in "fly-by-wire" airliners like the Airbus A320 or the Boeing 777, or, indeed the software used in a broker's mainframe to process share transactions or a payroll program. Mission critical software is very reliable indeed, and the reliability of mainframe software is about four times better than PC software. As in other fields, the cost of software reliability increases exponentially with the degree of reliability demanded by the user [GOE79].

The traditional software development process bankrolled by billions of dollars, backed by a myriad of methodologies, exotic programming methods, CASE (Computer Aided Software Engineering) tools, and some of the brightest minds in the discipline, is still falling short of the quality needs of software. In 1968, 25 years after the first computer programs were written, the NATO Science Committee convened with some 50 top programmers, computer scientists and industry leaders to chart a way out of the difficulties being experienced with large software systems - the so-called "software crisis". Not much was achieved but a new name was coined: "software engineering", defined as "the application of a systematic, disciplined, quantifiable

approach to the development, operation and maintenance of software. According to [GIB94]: "After another 25 years software engineering remains a term of aspiration." The vast majority of computer code is still hand-crafted from raw programming languages by artisans using techniques they neither measure nor are able to repeat consistently." But, given the intractable nature of the problem, encouraging progress has been made. Indeed the search for the Holy Grail may be coming to a close.

The creation of software began as an art, not as a science with formal rules and physical laws to govern it. The introduction of more rigorous and formal methodologies to keep out the bugs, while preserving the essential creative component, is proving difficult. The acceptance of new ideas has always been a fairly slow process. The U.S. National Institute of Standards and Technology reports that it takes 55 years for 90% of US. manufacturers to adopt a technology. In Japan the comparable figure18 is years [KEY92]. Acceptance of technology seems to depend on four issues: Does it do something new or better than before? Are the results or effects demonstrable? Does it provide sufficient added value for the money? Does it seriously impact the status quo?

Early attempts to improve product quality and productivity depended to a considerable extent on the 10% rule - that the best software will be produced by hiring within the top 10 percentile. But not every software developer can hire from the top echelon and how do you decide who is in that group anyway? There are extraordinarily wide variations in programming ability between individuals - up to a factor of 25. Obviously, choosing the best people helps but it is not a sure-fire answer and does nothing to move software creation into a more disciplined engineering environment [BAS87b]. To this end the industry has seen the emergence of standards, new languages, new tools and new programming methods, all intended to encourage

developers along the smooth paths of best practice and away from the rocky roads that promulgate bugs and cost time and money.

As might be expected, computer software itself is being used to develop tools, i.e. the CASE tools, to help in the software development process. These tools cover all facets of the software development cycle, and are commercially available. CASE tools can be classified into two classes [HAM90]: upper CASE tools for design, and lower CASE tools to automate code production. But CASE tools, although useful, have been a disappointment. They are hard to fit together and they leave gaps which have to be bridged using manual methods. The software development process is one of the most complex processes a human can perform. CASE tools help with this process but they do not appear to be the final answer.

Reuse of proven components helps some, but a new class of problems--interface problems--can arise when the proven components are joined. A NASA study found that three quarters of all software errors occur at the interfaces of program modules [CAR90, KEY93, MCG93]. However, the potential benefits of reusability are so high that we ought to focus on reducing interfacing errors rather than give up on reusability.

Serious attempts have been made to introduce software metrics to measure productivity as a step toward better analysis of the software creation process. A multitude of tools have been devised to help bridge the very wide gap between the user requirements and the software produced [HAM90].

1.4 Motivation for Metrics Databases

Traditional project management has typically proceeded in an intuitive and ad-hoc manner, with inconsistent collection of metrics data across different projects for different periods of time [ABD89]. Several factors can cause inconsistent interpretation of these data. The leading

factors are: the lack of standard techniques for metrics data analysis; the large dimensionalities and non-linearities; mixture of continuous and discrete, quantitative and qualitative data; highly skewed distributions; and limited data points. Also, the metrics data are rarely validated, further aggravating the potential inconsistencies in interpretation.

The management and analysis of metrics data for software engineering projects can provide a consistent mechanism for risk identification and assessment. Integration of metrics data of past projects into a metrics database enables us to use past experiences effectively for current and future projects. Past experience is no guarantee of future performance and no two projects are identical [WOV87]. However, metrics data analysis can provide a mechanism through which an ongoing project can be compared, both quantitatively and qualitatively, with previous similar projects. Such an approach can be a decision aid, allowing management to revise project plans and adjust expectations. A detailed plan can be developed that will help management evaluate project progress. Such a plan should be applicable at every stage of the development cycle and should assist in monitoring and predicting potential risks associated with the project. The Test and Evaluation metrics discussed in the next chapter provide the basic framework for monitoring software engineering risks.

1.5 Research Issues

As indicated in Figure 1.1, we address three major research areas in this thesis. These include:

- Software Metrics
- Software Metrics Analysis Techniques
- Data Modeling of Software Metrics

We now briefly elaborate these research areas.

1.5.1 Software Metrics

Software metrics are used for the measurement and prediction of progress of software projects, in terms of resources, processes, and product attributes. The overall role of software metrics in the development process is as follows:

- improve the current product
- improve the future process and products

The science and engineering of metrics thus include measurements on processes, projects, products, their analysis, and evaluation. Metrics data capture and encapsulate the previous experiences, help develop models and theories of the observed phenomena, and thereby support prediction of project, product and resource requirements.

The metrics proposed in this thesis can be divided into three broad categories, namely management, requirements, and quality. This is the core set of metrics that should be used in tracking project progress and product maturity and quality. The set provides at least two means which are applicable in each of the following phases of the software life cycle: Requirements definition, design, coding, and maintenance. *Management metrics* are used to track project costs and schedule and determine personnel and computer resources to be allocated to a project. *Requirements metrics* track changes from the original requirements and requirements implementation. *Quality metrics* track product quality. Figure 1.2 illustrates the classification of software metrics that are addressed in our research. A detailed discussion of these metrics is provided in Chapter 2.

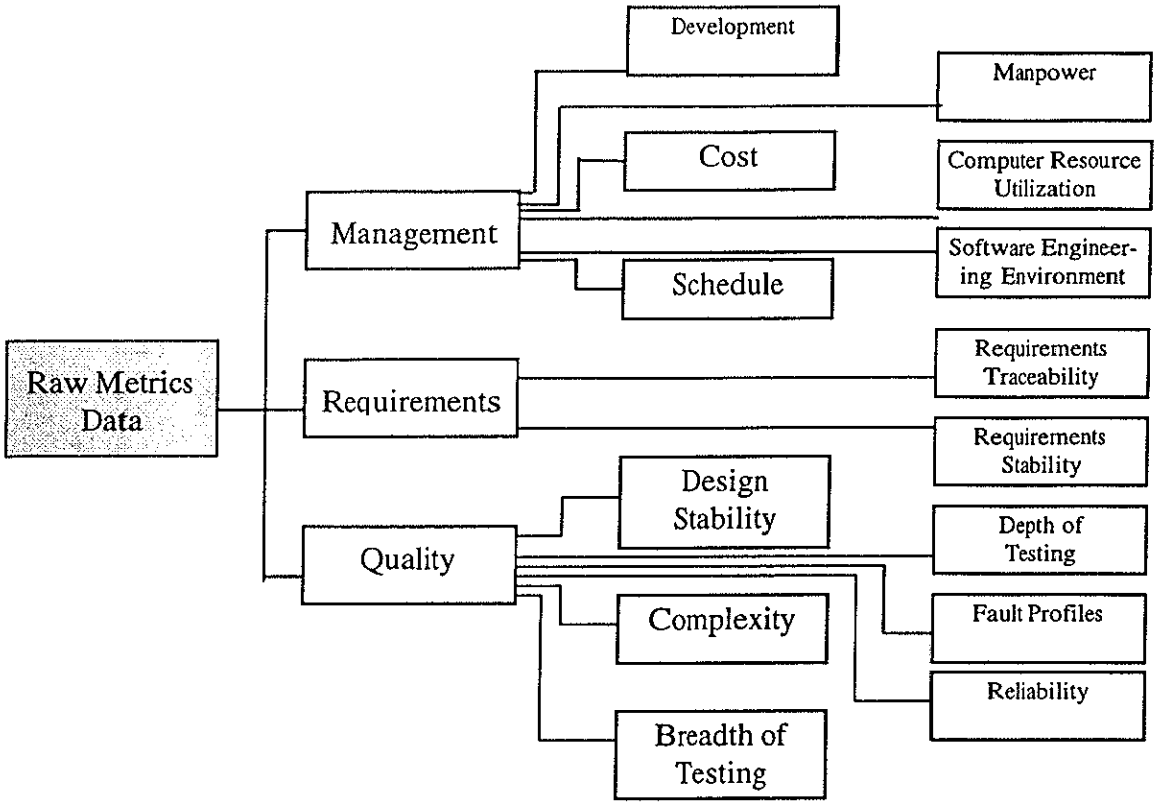

Figure 1.2: Types of Software Metrics

1.5.2 Software Metrics Analysis Techniques

Figure 1.3 illustrates the techniques used to classify and analyze metrics data. Analysis of metrics data implies the determination of two important pieces of information:

- recurring trends
- deviation from normal trends.

Metrics classification techniques are used to determine the level of risk associated with development plans by analyzing past project trends. These techniques help in answering simple queries such as:

- Which components in the software development process are error-prone?
- Which components in the software development process require high development effort?

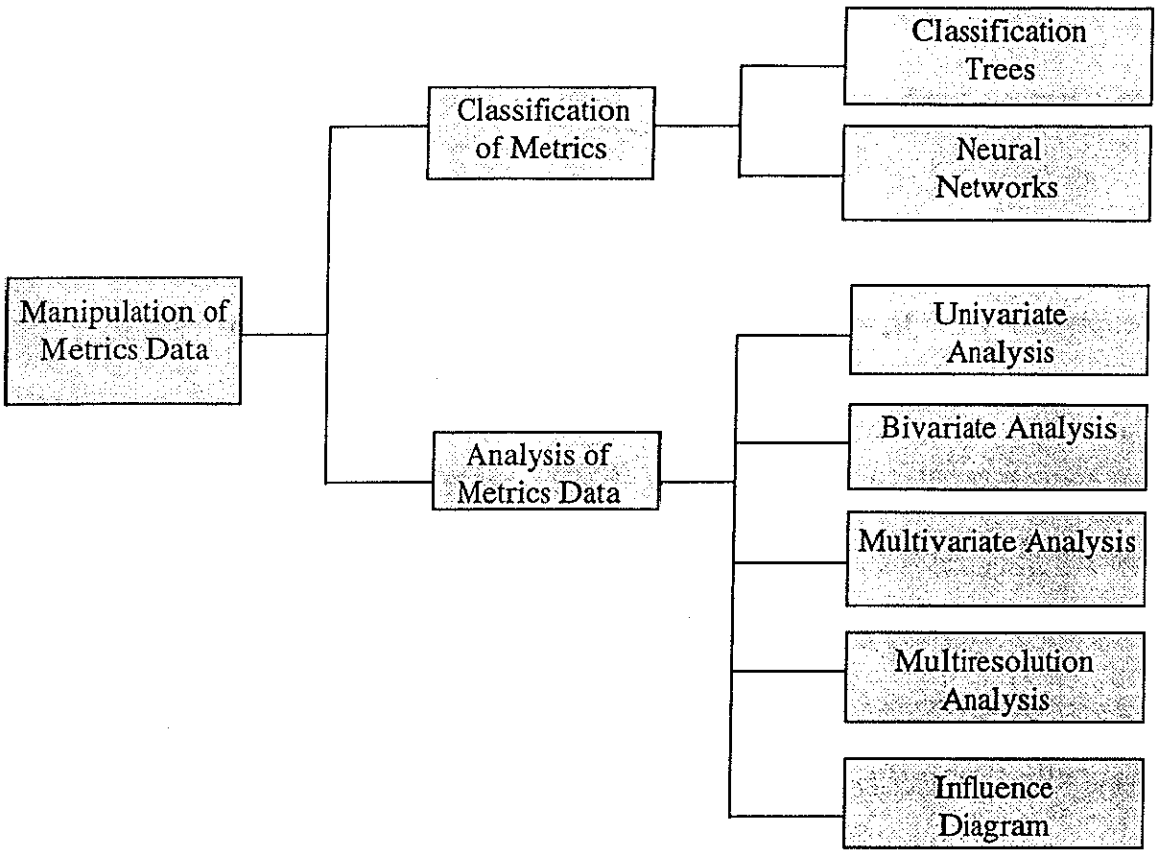

Figure 1.3: Metrics Data Classification and Analysis Techniques

Answering these queries can be regarded as a metric-based classification problem. For example, we may choose to categorize components into classes based on different degrees of risk, or different levels of reliability. Classification trees and neural networks are among the techniques available for classification of metrics data as explained in Chapter 5.

Simple database retrieval techniques may not be sufficient for complex queries; in which cases more sophisticated analysis techniques are needed. There are several statistical techniques available to analyze software metrics data, including multiresolution analysis and multivariate analysis. Software metrics analysis techniques are discussed in Chapter 4.

Effects of CASE tools and productivity/learning ability of development team are significant on the software project development process. Suitable mathematical models are

needed to study these effects. In Chapter6, we propose two such models based on realistic set of assumptions. A thorough experimentation is described to validate these models.

1.5.3 Data Models for Software Metrics

DBMS data models discussed in the literature have relative advantages and drawbacks with regard to storage and retrieval of metrics data. The effectiveness of the data model depends on the nature of the queries to the metrics databases which, in turn, vary according to the purpose for which software metrics are used. One key aspect of software metrics data is its temporal dimension which is fundamental to many important queries. Another key aspect is the semantics related to quality and risk of a software project. Figure 1.4 illustrates the major DBMS data models that will be analyzed to determine their effectiveness for specifying various types of queries, and their semantic power to express various views of quality and risk. This discussion is given in Chapters 7 and 8.

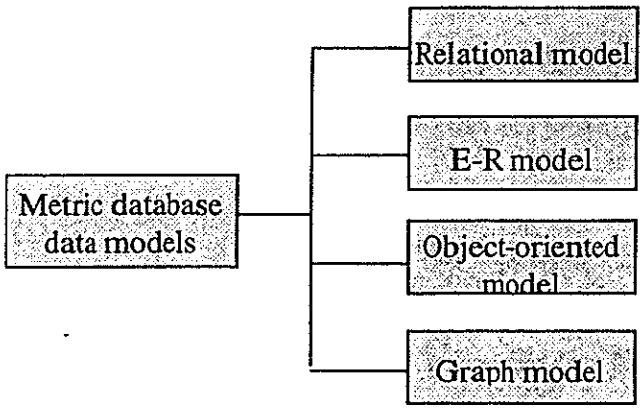

Figure 1.4: Data Models for Metric Databases

1.6 Thesis Organization

This thesis is organized into nine chapters. We have just discussed the importance of metric databases and presented an overview of the research issues involved in the collection, classification, and analysis of software metrics, as well as the data models to be discussed for

metric databases. In Chapter 2 we propose the types of software metrics to be collected for software engineering projects. Chapter 3 discusses typical queries that may be posed on the metrics. Chapters 4 and 5 discuss various analytical techniques. In Chapter 5, we then present a framework for integrating these analytical techniques with software metrics to develop a powerful environment for the management of software development process. In Chapter 6, we propose two mathematical models to analyze the effect of productivity of software team and CASE tools on the quality of software product. Furthermore, these models are used to estimate cost and resource requirements of software engineering environment. Extensive experimentation of these models validate them. Chapter 7 elaborates on various data modeling techniques. We then propose a unique temporal data model framework using Petri Net for metric databases. Based on this framework, in Chapter 8, we propose a unique methodology based on the theory of recursive graphs (R-graphs) that can be used to model different levels of abstractions for risk and quality management of software project in a systematic manner. Based on the research presented in this thesis, we then propose a pragmatic architecture for software metrics database system. Chapter 9 concludes by discussing the research contributions of this thesis and potential future research.