Chapter 3

Queries On Metrics Databases

3.1 Introduction

In the previous chapter, we have proposed a set of raw software metrics that are considered to constitute a broad measurement base of the products and processes in the software development life-cycle. Raw metrics are useful only when they can be understood and analyzed to guide the decision making process in a large software project. This can be a very difficult task. Therefore, it is not only important to develop a framework that should also translate the metrics data into useful information but also facilitate the process of managing future software projects by learning and gaining knowledge from the historic data. In this chapter, we propose such a framework that can be used by the project manager or engineers.

The collection of software metrics data and their inclusion in a metrics database can be of great benefit to both current and future software projects. Management can make immediate use of software metrics data in the metrics database in determining the progress of their project and controlling their software development work. In order to enable software metrics data collection to take place to a sufficient standard to allow meaningful subsequent analysis, it is not enough to define software metrics data required, and guide companies on procedures for software metrics data collection. Rather we also need to define procedures to analyze the software metrics data retrieved from the database in order to produce meaningful information for software project control. In this chapter, we discuss three ways of analyzing data, corresponding to three stages in the gradual development and systematizing of software engineering knowledge:

1. *Simple examination.* At the beginning of metrics data collection and use, there will often be a lack of clear ideas on how to assign a meaning to the data. In such a situation a manager who feels that certain metrics are likely to indicate the state of the project may collect them and look at them. At this stage all interpretation proceeds from the manager, none from the analysis technique. The type of software metrics data that a manager may collect is discussed in detail in Chapter 2. Simple queries such as retrieving the start and end date for a given milestone from the metrics database, as well as determining project costs are examples of this stage. The theme of this chapter is to discuss such queries.
2. *Metric Guided Risk Management.* Risk management of a software project is concerned with identifying in a project high risk items as early as possible and resolving them. It can lessen long term cost and help prevent software disaster. The major steps in risk management consist of risk assessment and control [BOE91]. In the risk assessment step, loss probability and loss magnitude for each risk item is assessed, and risk items are prioritized according to the expected loss due to each of them. In the risk control step, the plans to resolve and assess the risk items are generated and executed. These two steps are applied repeatedly throughout the life-cycle of the software development process. The simple examination approach cannot provide the powerful techniques that are needed in order to identify interrelationships among the metrics data sets. For this purpose, in the next chapter we propose a framework that is based on the influence diagram [HOW84] which captures the statistical dependence and independence among the metrics data in the temporal domain, over the whole life-cycle of the software development process.

3. *Alternative Approach.* In addition we propose two alternative approaches, namely; classification trees and neural networks, as a form of analytical redundancy to complement the analytical approach in the above mentioned approach.

Starting from the attributes associated with the software development process, we first classify the queries into five categories. This classification provides an insight into the large dimensionalities associated with the metrics data. Since a key dimension is time, consequently the temporal attribute is essential for the analysis of data since a considerable number of queries require identification of events present in the metrics database.

3.2 Simple Examination and Classification of Queries for Software Metrics

Effective tracking and supervision can be carried out by consistently querying the metric database for the purpose of managing the project as well as the software development team. Such queries can provide the current and correct information that can lead to good management decisions, coordinated development (preventing members of the team from going in different directions), on timely schedules and costs within the budget. In general, queries can range from simple to complex, and can be arranged based on the semantics and the nature of processing required. The two aspects of classifying queries are shown in Figure 3.1. The processing based classification include range queries, summary queries and temporal queries. Both range and summary queries, in general, are concerned with aggregate information, suitable for high level management. However, the dominant dimension is the temporal one. Most of the information and knowledge in software engineering metric data is temporal, as the metrics are collected

over a long time span associated with the life-cycle of the software project. In Chapter 7, we elaborate on the temporal characteristics of the metric data and propose a formal framework for quality and risk management based on temporal processing of the metrics data. The second type of classification is based on the semantics associated with queries posed by the management team.

In Table 3.1, some semantic based sample queries are described, which can provide the gateway to the information and can be the essential ingredient for the proactive software development project management. Formally this is known as Goal-Question-Metric (GQM) paradigm, discussed in the next section.

3.3 Goal-Question-Metric (GQM)

In the Goal-Question-Metric (GQM) paradigm initiated by Basili [BAS94] and extended by Hewlett-Packard, each project has a set of goals. Each of these goals has a set of questions that a manager may ask to help him understand if each project is achieving its goals Many of these questions have answers that can be measured. The idea behind metrics-guided risk management is that the goals for each project are to reduce types of risk encountered during software development. Through the application of relevant metrics stored in the metrics database, it is possible to identify and isolate these risks so that corrective action can be taken.

Queries

Cost

Find the total cost of the system up to milestone II

Schedule and Risk Analysis

What is the most recent slippage in the planned end date for a given milestone ?
What is the total slippage in the planned start date for given activity?
For the entire system, find the percent of computer software configuration items (CSCI).

Productivity

For a given CSCI, find the percent of computer software units (CSU's) with hundred percent complete testing.

Quality Assessment

What is the overall test coverage (breadth and depth of testing) for the system requirement and functionality ?
For a given CSCI, for all the reporting dates, provide the following information to assess its quality

- Identify those CSCIs having the selected metrics value falling in a certain range. Also, identify the time period of program month.
- What percent of defects were found and corrected in 50 hours per defect.

Summary

Find the total number of relational requirements documents and functional description requirements for these two documents, find the percentage that is traceable to system specifications.

Management

Identify all CSCIs and System milestone slippage in the project current budget and Schedule
List all the CSU's with complexity rating higher than 5
List all requirements changed at system design review (SDR)
What CSU's were modified to reduce complexity?
What complexities were not traceable from high level specifications to low level test cases by quality assurance analyst and system engineer ?
Who determined the test success criteria when invalidated simulations were used to replace late equipment yet to be delivered ?
Who has been an operating system expert for six years ?
When were all test cases exercised successfully ?
Who got promoted from programmer to system analyst while at least one other programmer remained at that level ?
What critical and major software trouble reports were discovered at the end of system integration ?
When was the test completion date entered incorrectly as completed ?
The programmers CSCI failed to join the systems integration a month earlier ?
The test success ratio was low September 1995

Table 3.1 Sample Queries

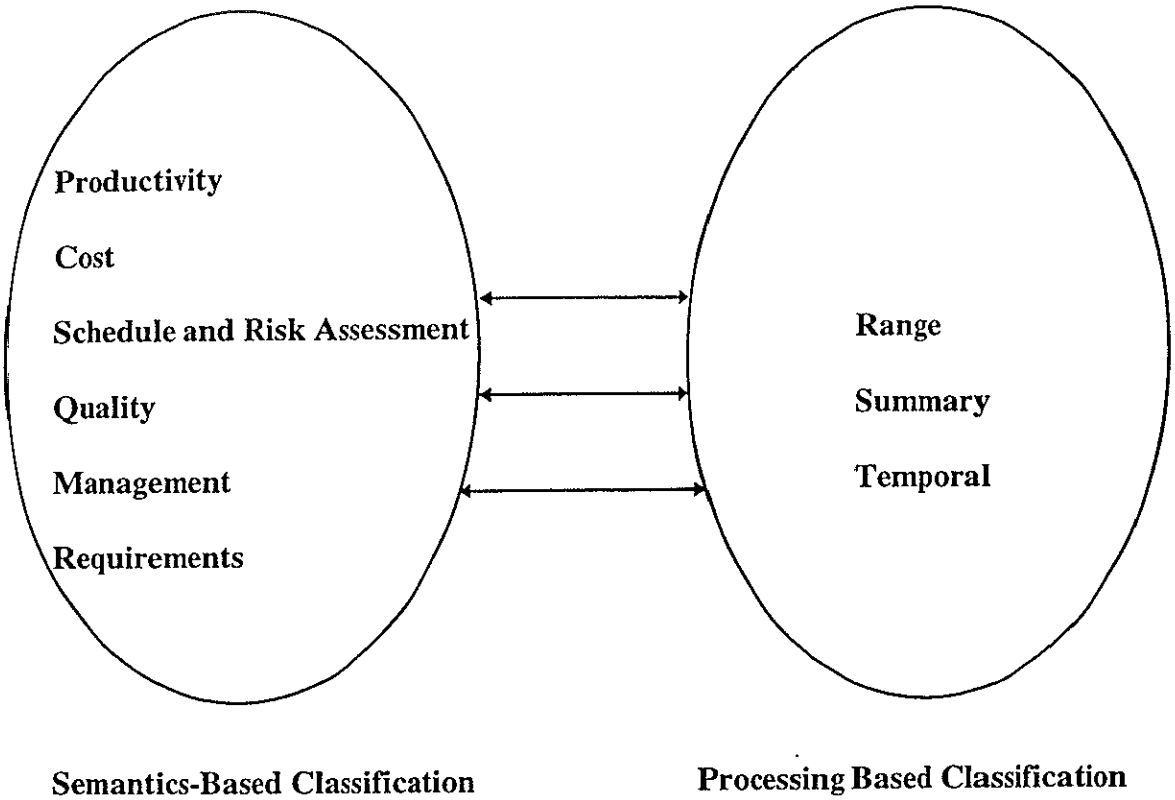

Figure 3.1: Two Aspects of Classifying Queries

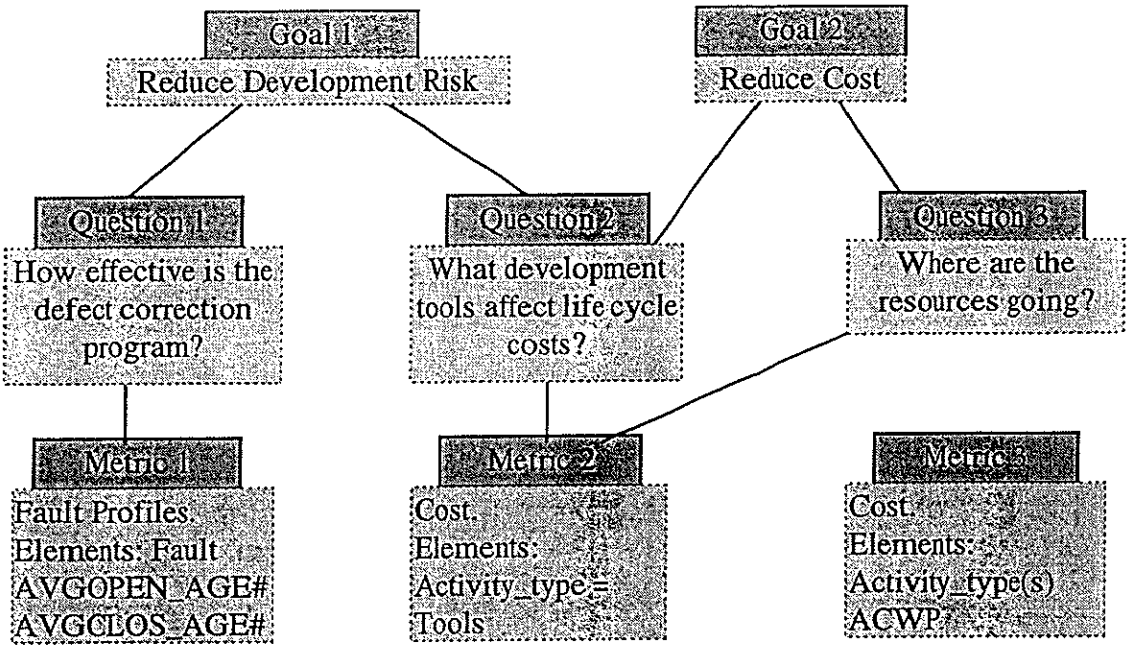

Figure 3.2: Goal-Question Metric - Measurement & Evaluation Paradigm

An example is illustrated in Figure 3.2. In this example, two project goals are established:

- reduce development risk, and
- reduce overall cost.

Although there are many questions one could ask that are related to these goals, three are represented here. From the test and evaluation metrics set, two metrics and related data elements that can help managers achieve their goals are also represented. Once goals and questions are established, it is possible to locate which measures from the test and evaluation metric are related.

Risks encountered during software development include the following:

- schedule
- technical
- cost
- acquisition.

Managers pose queries on the metrics database with the objective of obtaining more information related to the progress of the project in order to reduce project risks. In the rest of this chapter, we present examples of scenarios with related questions that managers may ask, and which can be either directly obtained by simple retrieval of raw metrics data or from more complex analysis of the metrics data in the database.

3.4 Schedule Scenarios

We now present some typical queries that a project manager may pose to the metrics database relating to project schedule. The questions that a manager may ask to reduce schedule risk in a software engineering project include the following:

1. What is the most recent slippage in the planned start date and the planned end date for a given milestone?
2. What is the total slippage in the planned start date and the planned end date for a given milestone?

In this example, slippage in the planned start date refers to the difference between the planned start date and actual start date for a given milestone. Therefore, if the planned start date for the milestone is Project Month 2 and the actual start date for the milestone is Project Month 3, then the slippage in the planned start date is one month. Likewise, slippage in the planned end date refers to the difference between the planned end date and the actual end date for that milestone.

The schedule metric, when plotted over time, can provide indications of problems in meeting key events or deliverables. The higher the slope of the trend line for each event, the more problems encountered. The schedule metric can be used in conjunction with several other metrics to help judge program risk.

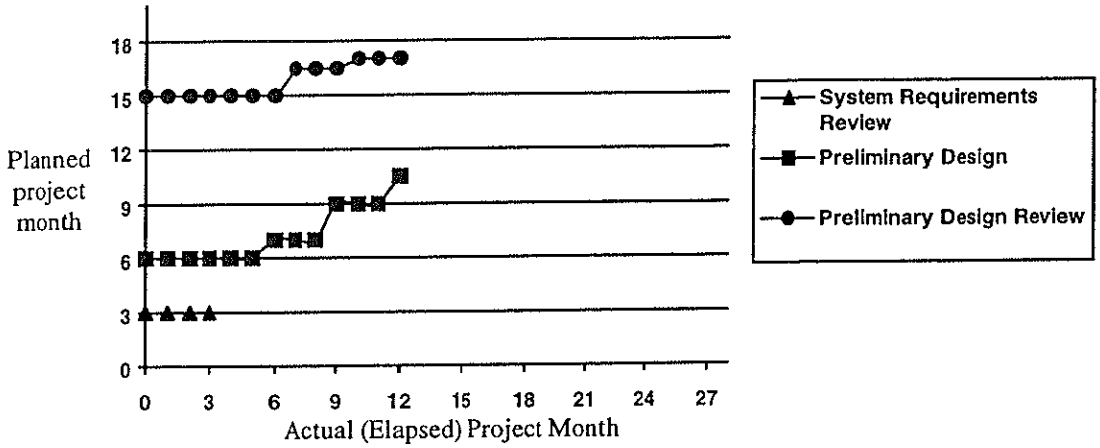

Figure 3.3: Planned Start Date for Schedule Example

For instance, in answer to the first question about the most recent slippage in the planned start date for a given milestone in the current project, we can plot a graph as illustrated in Figure 3.3. Assume that we are now in Project Month 12. The horizontal axis in Figure 3.3 depicts the elapsed time in project months, while the vertical (y) axis depicts the month in which the start date for the given milestone is planned. As we can see from the graph, the planned start date for the System Requirements Review was month 3, as indicated by the y-axis on the graph in the upper left hand quarter, and this plan remains constant throughout project months 0 through 3; there is no schedule slippage in the planned start date for the System Requirements Review. For the Preliminary Design milestone, the planned start date was originally scheduled for month

6. When program month 6 actually arrived, the start date was re-scheduled to month 7, so the schedule slippage is 1 month. As the project progressed to month 12, the planned start date for the Preliminary Design slipped to month 11, giving a cumulative schedule slippage of 5 months. For the Preliminary Design Review milestone, the planned start date remained at month 15 for the first 6 project months. In month 7, the planned start date for this milestone slipped to month 16, and in project month 10, the planned start date slipped one more month to month 17, and remained at that date till project month 12.

Figure 3.4 gives the planned end date for the schedule example. As we can see from the figure, the planned end date for the System Requirements Review milestone remained at month 6 for the first 5 months of the project. In month 6, it slipped 1 month to end in month 7, and in month 8 it slipped 2 more months to end in month 9. The schedule continued to slip by another 2 months in month 10 with a planned end date of month 11, and remained constant for the next 2 project months. From the beginning of the project to project month 12, therefore, the planned end date for the System Requirements Review has slipped by a cumulative total of 5 months, from the initial planned end date of Project Month 6 to the current planned end date of Project Month 11. Likewise, the Preliminary Design milestone has slipped by a cumulative total of 1 month, from an initial planned end date of Project Month 15 to the current planned end date of Project Month 16. The Preliminary Design Review milestone has also slipped by a cumulative total of 1 month due to the slippage in the schedule for the Preliminary Design milestone.

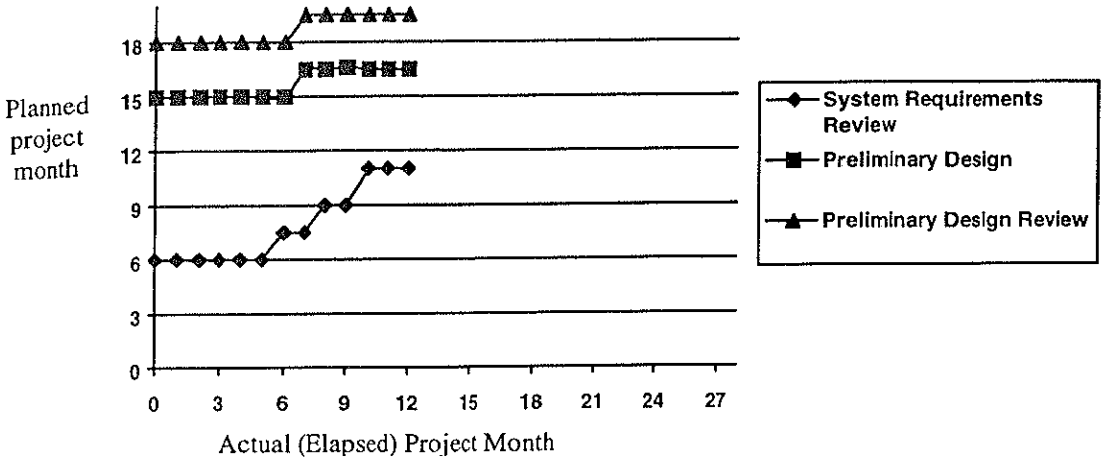

Figure 3.4: Planned End Date for Schedule Example

Therefore, in answer to the second question for the System Requirements Review milestone "What is the most recent slippage in the planned start date and the planned end date for the System Requirements Review milestone?", we look at the slippage in planned start dates and planned end dates for Figures 3.3 and 3.4, and the answer in both cases is 0, that is, there is no slippage in planned start dates and planned end dates for the System Requirements Review milestone between month 11 and month 12. In answer to the second question relating to the same milestone, "What is the total slippage in the planned start date and the planned end date for the System Requirements Review milestone?", the answer is 0 for the planned start date, and 5 months for the planned end date. This is because there is no change in the planned start date for the System Requirements Review from the start of the project to the present month. For the planned end date, however, the schedule slipped from an original planned end date of Project Month 6 at the start of the project to the current planned end date of Project Month 11.

3.5 Technical Scenarios

Project managers and engineers may wish to pose more technical queries to the metrics database. Examples of such queries would include:

1. Which modules in this project require extensive labor effort during development? What is the extent of labor effort required for each module in the project?
2. How completely has a given software project been tested?

We describe the solutions to the above queries in greater detail in the rest of this section.

3.5.1 Modules Requiring Extensive Labor Effort

An important objective of software project management is to identify high risk software components. Such components include modules that are likely to require the most effort in the software development life cycle. If such components are identified early, the project manager can use the information to reduce risks of schedule slippage by taking preventive measures such as allocating the necessary amount of manpower and sufficient time for completion. The extent of labor effort required for a software module varies proportionally with the size of the component (measured by the number of lines of source code), the number of decision paths, and the number of input/output variables. We may then want to follow some guidelines to determine whether a given module requires either low or high development effort. We use the following guidelines:

- A module requires high development effort if it has over 1500 lines of source code regardless of the number of decision paths and input/output variables;
- If a module has between 501 to 1500 lines of source code (inclusive), it requires high development effort regardless of the number of decision paths if the number of I/O variables exceed 120;

- If a module has 500 lines of source code or fewer, it still requires high development effort if it has over 90 decision paths, regardless of the number of I/O variables.

From the above guidelines, it is apparent that in order to determine whether a module requires high development effort, we must first divide the number of lines, number of decisions, and number of I/O variables into categories. One method of doing this is by using *classification trees*, as described in Chapter 5.

This query may, alternatively, be rephrased to determine the *level* of development effort required by a software module. Chapter 5 also describes an alternative classification technique using *neural networks* that can be used to answer such queries.

3.5.2 Completeness of Testing

How completely has a given software project been tested? This involves both the *Breadth of Testing* and *Depth of Testing* metrics. The *Breadth of Testing* metric addresses the degree to which required functionality has been successfully demonstrated, as well as the amount of testing that has been performed, while the *Depth of Testing* metric provides indications of the extent and success of testing from the point of view of coverage of possible paths and conditions within the software. Since a software project consists of a number of modules, which may themselves consist of further sub-modules, the response to this query requires determining both *Breadth of Testing* and *Depth of Testing* of not only the overall software project, but also of each module of the software project.

3.6 Cost Scenarios

Proper analysis of the cost metric is one of the key ways to monitor project progress. Questions that management may pose regarding project costs include?

1. Are costs under control?
2. Where are the resources going?

Each of these questions will be discussed in this section in the light of various scenarios.

3.6.1 Are Costs Under Control?

Are costs under control? Suppose we are only 6 months into a 30-month project and development costs are already over budget by 50%. This information can be obtained by comparing actual project costs against budgeted costs for various phases of the software life cycle. Although there is generally insufficient information early in the software life cycle to determine possible causes of the negative cost variance, the potential problem is significant enough to warrant monitoring. Continued negative cost variances in later phases of the software life cycle may indicate that the project is significantly under-budgeted and that more resources should be allocated for the project. Alternatively, this may be indicative of other problems, such as purchases of non-budgeted computer resources, high personnel turnover, etc.

3.6.2 Where Are The Resources Going?

Another question that management may ask related to cost is: Where are the resources going? Answering this query requires an examination of the costs of the activities for each module in the system. After reviewing each module, say we have determined that a certain module is taking up most of the resources prior to the Preliminary Design Review. This can be determined from the actual cost of work performed for that module. More questions must then be asked to determine why this is so. Suppose we determine that most of the costs incurred for this module can be attributed to the high cost of management and the acquisition of additional software that

was originally not part of the budget. Additional questions reveal that the developer is pursuing an aggressive integration of object-oriented development techniques into their present software development process. This accounts for the high level of software engineering management and the recent acquisition of automated tools that accommodate object-oriented analysis and design activities.

Further analysis and questions related to the potential problem module indicate that the preliminary design and requirements analysis activities have been conducted in parallel. This suggests a number of possibilities regarding the developer's software process, which can include one or more of the following:

- extensive prototyping within the selected domain
- early prototyping to explore alternative design solutions
- excessive analysis of the problem, where analysis has proceeded beyond the development of a reasonably complete model.

Further reference to the Software Engineering Environment metric or the Schedule metric should provide additional information.

It can therefore be seen from this scenario that although software metrics help to provide an indication of problems, analysis of software metrics alone may not be sufficient for all scenarios; management must follow up metrics analysis by probing more deeply into the problem. Furthermore, many management queries will require analysis of more than one software metric and analysis of the interactions between metrics.

3.7 Acquisition Scenarios

The manager may pose queries such as the following with regard to acquisition risks:

1. Is the technical maturity of the contractor sufficient?
2. Does the developer have sufficient knowledge of requirements?
3. Is the developer following the Software Development Plan?
4. Have the technical program complexities been addressed for life cycle management?

3.7.1 Technical Maturity

The manger may ask: "Is the technical maturity of the contractor sufficient?" A simple query to the metrics database can check, for example, that the technical maturity rating of the developer is 3. This should assure that the development process is stable enough to provide some predictability. Figure 3.5 shows the actual versus budgeted costs for the first 9 months of the project. Although it is early in the development life cycle, it appears from Figure 3.5 that the system cost for preliminary development activities is according to plan.

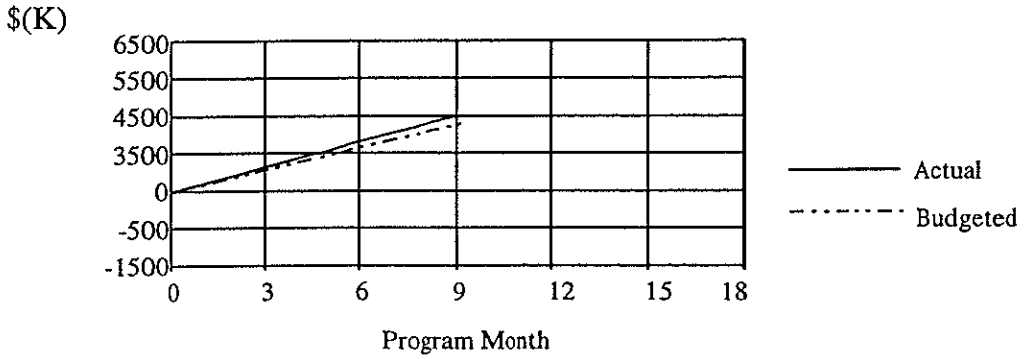

Figure 3.5: Actual Vs Budgeted Project Costs

3.7.2 Knowledge of Requirements

Another possible query to the metrics database is: "Does the developer have sufficient knowledge of requirements?" From Figure 3.6, it appears that requirements stability becomes a serious issue by month 14, where the developer can no longer keep pace with changes to the requirements. Note that "closed requirements" refer to requirements that have been satisfied, while "cumulative requirements" refer to all project requirements, both satisfied and unsatisfied.

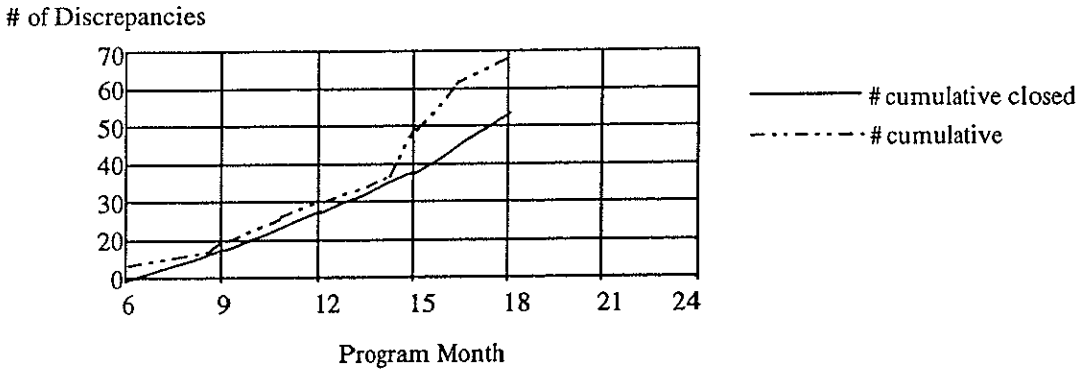

Figure 3.6: Discrepancies Between Cumulative and Closed Requirements

There are several possible causes for requirements instability, one of which is explained in Figures 3.7 and 3.8. Figure 3.7 suggests that prior to month 14, both the user and the developer seemed to have a common understanding of the system's requirements. An unplanned loss and replacement of personnel as illustrated in Figure 3.8 has led to many proposed changes by the developer.

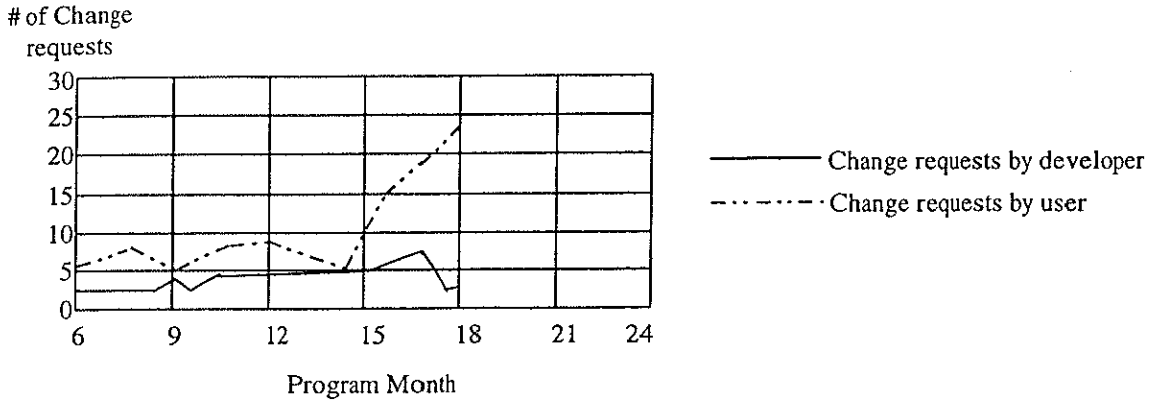

Figure 3.7: Discrepancies Between Change Requests by User and Developer

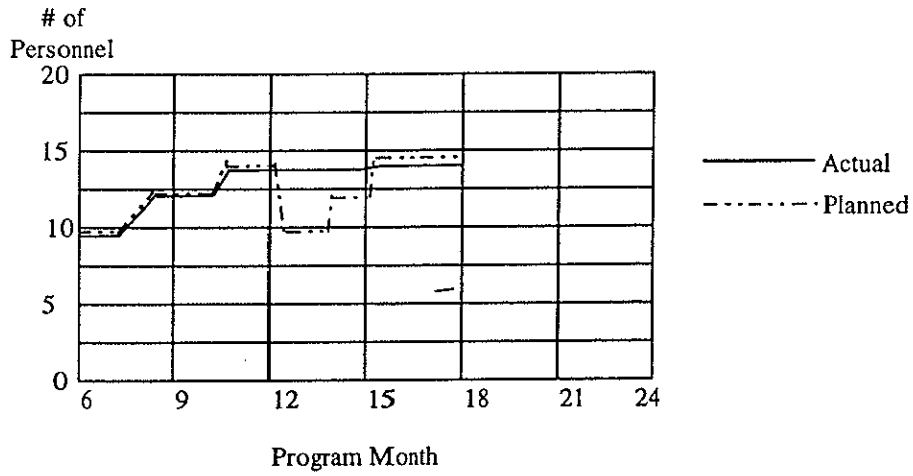

Figure 3.8: Actual and Planned Manpower Resources

3.7.3 Adherence to Software Development Plan

Another possible query to the metrics database is: "Is the developer following the Software Development Plan?" One way to address this question is to look at the variances between the planned, successful activity and the actual activity expressed in terms of various metrics, such as the schedule and breadth of testing metric. For example, we may plot the actual versus planned coverage for the Breadth of Testing metric and ensure that there are no major deviations between the actual and planned coverage's.

Another way is to determine the Requirements Traceability for the Software Requirements Specification (SRS) to various program modules. Volatility of the requirements specification process may lead to unplanned variances in the following:

- the budgeted cost of work performed versus the budgeted cost of work scheduled
- the traceability of the software requirements to the product itself
- the ratio of the number of successful functional tests to the number of tests conducted.

3.7.4 Addressing Technical Complexities

Another query that management may pose on the database is: "Have the technical program complexities been addressed for Life Cycle management?" Certain metrics specifically address the program's technical characteristics and can be an indicator of the future manageability of the program. For example, as illustrated in Figure 3.9, the developer may have difficulty addressing future faults given the high number of unresolved software bugs and the large share of modules that have cyclomatic complexity greater than 14. Also, suppose the planned and actual CPU usage's are 80% and 90% respectively at some point in the project, whereas the targeted CPU usage is only 50%. The high planned CPU usage level does not provide for bringing usage levels below the 50% target.

The earlier a program's "stress points" can be identified, the earlier the risks associated with life cycle management can be isolated. The dilemma for program management is that until some activities have been completed, for example, designs established, code generated, or tests conducted, there are few indicators of a program's status. With proper analysis of software metrics, a program's management can at least isolate problem areas and characterize trends for the future.

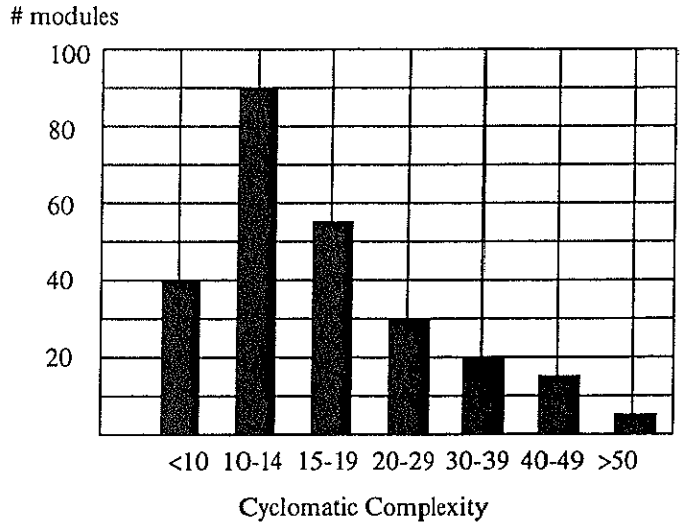

Figure 3.9: Cyclomatic Complexity of Modules for Given Project

3.8 Conclusions

In this chapter we have examined only a handful of the numerous possible queries that can be made on the metrics database. Our main purpose for providing this discussion is to build a framework for discussion of various classification and analysis techniques that may be required for more complex queries as described in Chapters 4, 5, and 6 as well as developing high level abstractions as described in Chapters 7 and 8.