Chapter 4

Analytical Techniques for Metrics Guided Risk Management

4.1 Introduction

Risk management of a software project is concerned with identifying the project's high-risk items as early as possible and resolving them. It can reduce long term costs and help prevent software disasters. Some of the important risk factors include excessive delays in schedule, unrealistically high development cost, inconsistencies in requirements, unreliability of a product, non-upgradability of a product and its non-reusability, product not conforming to standards, failure to meet real-time requirements, susceptibility of a product to security breach, using non-standard technologies and unproven environments, difficulty in maintenance, user unfriendliness, etc. Unless identified and controlled early, these risks can create serious problems. As mentioned in earlier chapter major steps in risk management consist of risk assessment and control [BOE89]. In the risk assessment step, loss probability and loss magnitude for each risk item are assessed and risk items are prioritized according to their expected loss. In the risk control step, the plans to resolve the risk items are generated and executed. These two steps are applied repeatedly throughout the life-cycle of the software development process in the form of the following queries:

1. What is the normal outcome of a software production process so that they can predict future outcomes?
2. When a process has resulted in an unusual outcome, what factors may impact the quality of the final product and when is their impact significant?
3. How can we diagnose the causes and predict consequences of such abnormal results?

Answer to these queries requires extensive knowledge that can be obtained by combining the information contained in software metrics data collected from the project with the manager's experience and knowledge of the organizational resources. Data from past projects can also enhance the effectiveness of the management process.

Considerable research has been conducted in software engineering in the past two decades. especially on the modeling aspects of the software development process, and accordingly building increasingly powerful analytical techniques for risk management. Major emphasis has been placed on developing suitable cost estimation and reliability models, and direct application of software metrics [FEN91,PAU93,BAS84c]. During this decade, there has been an increasing awareness about using an integrated process-based approach to manage software quality. We have seen the emergence of such metrics in the form of the *integrated process-based approach.* This evolution reflects the growing concern of the software engineering community of developing project metrics which can aid in cost estimation and product quality control. During the past few decades, there has been an increase in the awareness about collecting software metrics which can impact quality, reliability, complexity and maintenance.

Most of the existing tools for software metrics merely collect and display simple abstractions of the raw metrics data. However, software metrics data tend to be of high dimensionality, highly correlated, and often exhibit non-linear distributions. Simple examination of metrics data, as discussed in chapter 3, generally does not provide enough information to identify problem areas in the project. Accordingly, powerful modern analytical techniques need to be employed to process the metrics data in order to gain knowledge and detailed insights into the software development process. The objective is to assess the quality of the product and the

risk involved in the completion of the project. In this and the following chapters we discuss a number of techniques for analysis and to depict inter-dependency of metrics data. We then propose a multi-faceted approach for risk management based on using a variety of the approaches along with the set formal software metrics presented in Chapter 2.

Such approach can provide answer to the complex queries in Table 3.1. These queries can span a broad spectrum of risks and quality management aspects of the product. As discussed in Chapter 2, simple examination of metrics data cannot not provide answer to most of these queries. Techniques are needed to identify inter-relationships among the metrics data sets prior to providing answers to these queries. Such identification is an essential component of the decision-making processes. For this purpose various quantitative and knowledge-based approaches can be utilized. For example, an influence diagram based approach can capture the statistical dependence and independence among the metrics data in the temporal domain, over the whole life-cycle of the software development process, and can also provide a powerful mechanisms for probing and screening information embedded within the metrics data.

Additional approaches such as classification trees and neural networks may be needed to obtain a wide variety of analytical results, providing more complementary information. For example, probing may be targeted on finding the cause for certain problematic outcomes and isolating abnormal data points. For such purposes, multiresolution and singularity based analyses can provide powerful methodologies over other approaches due to its capability to carry out low level resolution analysis which is based on its dynamic threshold control features. This methodology allows powerful noise reduction capabilities in the metrics data and can provide smoothened data even at very low levels of resolution. On the other hand, if one needs to devise an appropriate corrective action, based on some findings, influence-diagram based expert system

methodology can readily provide decision alternatives. Similarly, for processing those queries which are focused on predictive assessment of corrective actions, techniques based on neural networks or classification trees can be quite suitable.

It is important to note that a single technique may not be able to provide answers to all the aspects related to risk and quality management. We need a comprehensive set of techniques, where each technique has its own characteristics and capabilities. In this and the following chapters we discuss seven such techniques and present an integrated approach that can provide answer to the kind of queries listed in Table 3.1. The approach leads to a framework for risk and quality management of large software projects. The techniques to be discussed include: Entity-Relationship (E-R) modeling; Influence Diagrams; Principal Component Analysis; Multiresolution Analysis; Singularity Theory Analysis; Classification Trees, and Neural Networks approach. Both E-R modeling and Influence Diagrams techniques emphasize the inter-dependency and correlation aspects of metrics data.

4.2 Data Dependency Techniques for Software Metrics Data

4.2.1 E-R Modeling Technique and an E-R Model for T&E Metrics

One of the primary requirements for a software metrics system is to manage metrics data in a manner that ensures referential integrity and consistency of the data. The important step to achieve this goal is to find suitable logical data models for software metrics. Although a number of approaches can be utilized for such purpose, the E-R model provides a natural representation of the relationships between the different metrics and is simple to understand and interpret.

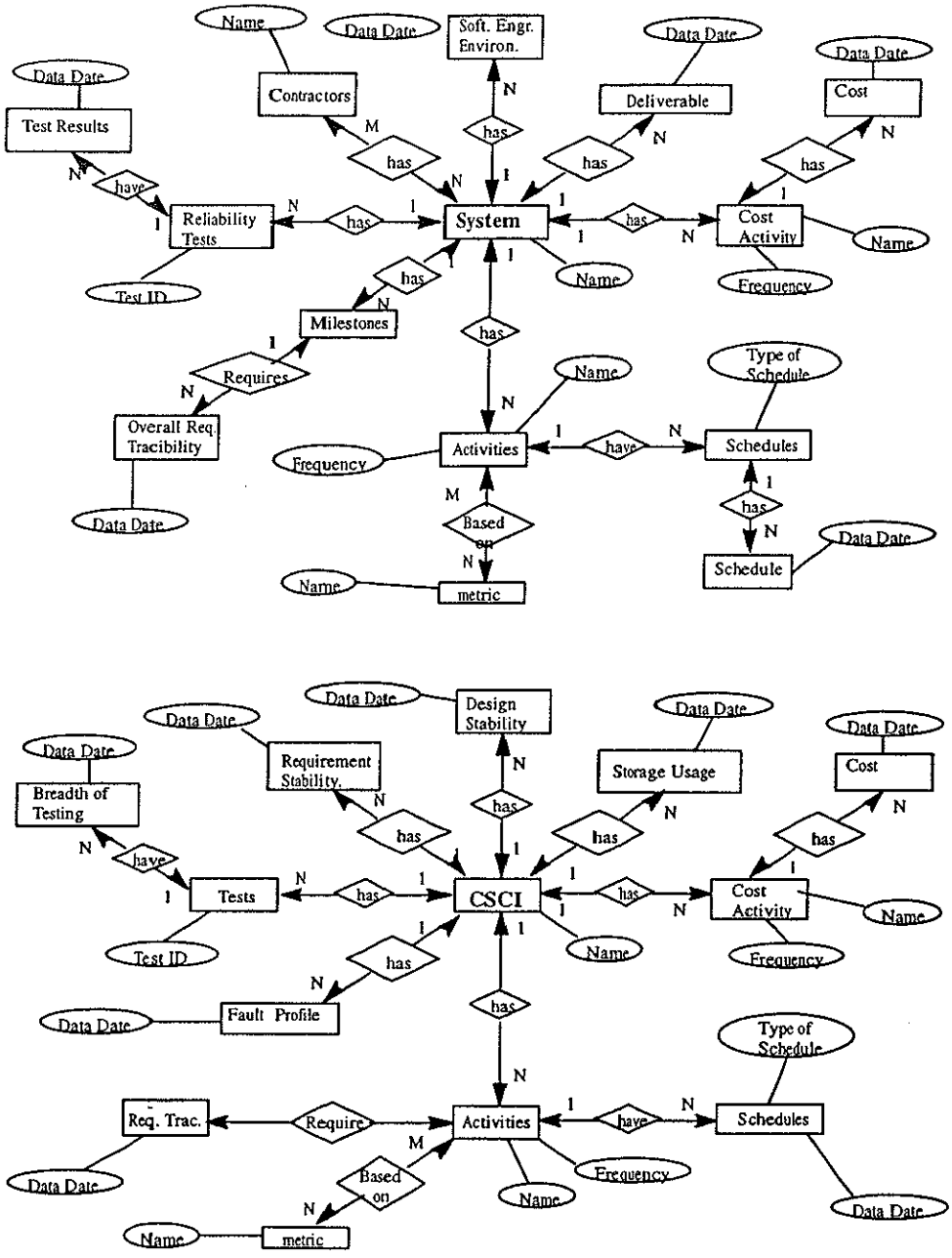

Figure 4.1: Entity-Relationship Diagram for the Proposed T&E Metrics Database

In Figure 4.1 we present an E-R diagram partially for the T&E metrics at the system and CSCI levels. As can be noticed from this figure, the system has one or more reliability tests, each of which has a distinct test identifier and is required to report test results. In addition, there are one or more Contractors assigned to one or more projects. The system also has a set of Deliverables as defined by management, which are accomplished by engaging in various Activities. Although not illustrated in figure , these activities may correspond to different phases of the software life cycle such as requirements specification, design, coding, testing, and maintenance. Each activity has a name and a set of metrics associated with it which has to be collected at regular intervals, as discussed in earlier. In addition, a set of schedules may be associated with each activity, and possibly an overall schedule encompassing the start and end date of all activities. Data for the schedule metric as discussed above is expected to be collected via schedule reporting at regular intervals. Furthermore, management may also associate one or more cost activities for the system, and prepare budgets for each cost item [CON85]. Examples of cost items would include the cost of manpower and computer resources allocated to each phrase of the software life cycle, as well as other overhead costs. Each cost activity is expected to have regular cost reporting in order to track actual versus budgeted costs. The system also has a number of milestones, each of which is associated with different activities; only regular overall requirements traceability reporting, which tracks the conformance of various milestones to system requirements is illustrated in the figure.

Figure 4.1 also describes the E-R model for the metric data associated with CSCI of a system. The E-R model has an obvious utility for some tasks, in particular performing logical database design. Many commercial database design tools allow the input of database constraints

via the E-R model. Such tools include capabilities for adding, deleting, and modifying the resulting E-R diagrams, and the generation of appropriate schema.

4.2.2 Influence Diagrams

Influence diagrams have been used to model a variety of situations, ranging from medical diagnosis [HEN91] to risk analysis of semiconductor manufacturing [AGO87,NAD91]. In such applications, influence diagrams have proven successful in combining the qualitative knowledge of human experts captured in the form of influence diagrams, and deriving models with less error compared to the use of first principles or statistical regression analysis. The combination of influence diagrams and neural networks has also resulted in new solutions to existing problems. Research on influence diagrams has also shown the advantages of reusing the same influence diagram in different inference phases [BEL94] due to the modularity requirements of keeping knowledge sources separated. Egar [SHA88] has also demonstrated the feasibility of generating influence diagrams automatically using a graph-grammar production system, easing the task of modifying them. Other work on influence diagrams includes the introduction of an arbitrary non-negative function (called a potential) instead of a conditional probability table to remove chance nodes directly without reversing arcs [NDI94], and the concept of stepwise decomposable influence diagrams to correspond to stepwise solvability.

An influence diagram is a network for probabilistic and decision analysis models. The nodes correspond to variables which can be constants, uncertain quantities, decisions, or objectives. The arcs reveal the probabilistic dependence of the uncertain quantities and the information available at the time of the decisions. Detailed data about the variables are stored within the nodes, so the program graph is compact and focuses attention on the relationships among the variables. Influence diagrams are effective communication tools and recent

developments allow them to be used for analysis. They have a graphical structure which makes them easy to understand and model the relationships in a complex system. They rest on the traditional Bayesian inference engine. Traditional techniques of analysis have been used to address questions of inference. We use the conditional independence implied by the diagram's structure to determine the information needed to solve a given problem. When there is enough information, we can solve it, exploiting that conditional independence. The same results are applied to problems of decision analysis. This methodology allows the construction of computer tools to maintain and evaluate complex models. Hence the major benefits of influence diagrams are that they are flexible, tractable, graphic and intuitive. They have a wide scope and enable reasoning at a purely quantitative level first. Using them, we can also do sensitivity analysis, model decision making, and flow of information in a complex system.

4.2.2.1 Characteristics of Influence Diagrams

An influence diagram is a network consisting of a directed graph with no directed cycles and detailed data stored within the nodes of the graph. Each node in the graph represents a variable in the model. This variable can be a constant, an uncertain quantity, a decision to be made, or an objective.

We call an influence diagram *probabilistic* if all of its nodes represent constants or uncertain quantities. Each variable in a probabilistic influence diagram has a data frame within its associated node, in which there is a finite set of *outcomes* (the values that the variables may take) and a *conditional probability distribution* over those outcomes. The *conditioning variables* for its distribution are indicated in the graph by arcs from (the nodes corresponding to) its conditioning variables into (the node corresponding to) it. If there are no arcs going into the node, then it contains a marginal (unconditional) probability distribution.

There are two types of variables in a probabilistic influence diagram. A *deterministic* variable has a degenerate conditional distribution and is drawn as a double oval(or circle); otherwise, it is called *probabilistic* and drawn as a single oval. We might be uncertain about the value of a probabilistic variable even after observing the values of its conditioning variables. In the case of a deterministic variable, we are certain of its value given the value of its conditioning variables, although we might be uncertain about its value if we could not observe their values.

4.2.2.2 Influence Diagram for the Software Metrics Data

In order to find an influence diagram for the software metric data we need to aggregate low level metrics into more abstract quantities which denote certain high level attributes. These attributes can further be abstracted into more high level aggregates which may denote "risk factors". There may be several cost nodes in the influence diagram representation of the complex system. There may be other deterministic nodes as well. There may be multiple decision nodes which represent risk management actions. In order to generate an influence diagram for the software metric data, we should take into account a number of factors. It is known that the use of ``software metrics" is a phase independent technique that can be applied at each phase of the software life cycle. Using the waterfall model of software development, metrics can be collected at the requirement, design, implementation, testing, and maintenance stages to provide guidelines for decision making at each stage and to determine whether each stage has been satisfactorily completed. As discussed in Chapter 2, at the requirement phase, we can use metrics to assess the time and effort needed at the later stages of the software life cycle. For example, requirement metrics can be used for project cost estimation and manpower allocation, and they can also be used to reduce the complexity of the requirement specification. At the design phase, metrics can be used to compare the quality of different design alternatives. Such metrics can serve as an

evaluation function for an automated system to choose between different design alternatives in its repository. Metrics collected at the coding stage determines the quality, maintainability, and understandability of the code. In addition, requirement, design, and code metrics should be used to guide the development at each stage for possible reusability.

The most useful metrics to be collected are during the requirement phase. This is because reducing the complexity and detecting errors at an early stage reduces the difficulties at later stages. Also, requirement metrics can aid project managers to schedule tasks, partition the work, and enforce certain performance standards for the software. Here, we suggest some of the characteristics that should be considered for metrics selection in the requirement stage. The dependency between software modules can lead to unexpected bugs if a module is modified, therefore dependency metrics need to be collected. At the requirement phase, we would also like to be able to predict the complexity of testing. Since local errors require less testing effort than errors involving many modules, requirement complexity metrics can be used to measure the amount of control flows between modules to predict testing complexity. The number of paths in a requirement control flow graph is an example of such a metric. Measurements of complexity due to concurrency, and structural and semantic understandability are other types of metrics that should be considered.

Based on this discussion, in Figure 4.2 we propose an influence diagram for the software metrics proposed in Chapter 2. The diagram can serve as a road map for diagnosis and probing of any issue related to quality and risk management.

As mentioned earlier, the use of influence diagram is tightly related to data analysis and correlation. For the purpose of completeness, we now discuss various analysis techniques and their potential role in risk and quality management.

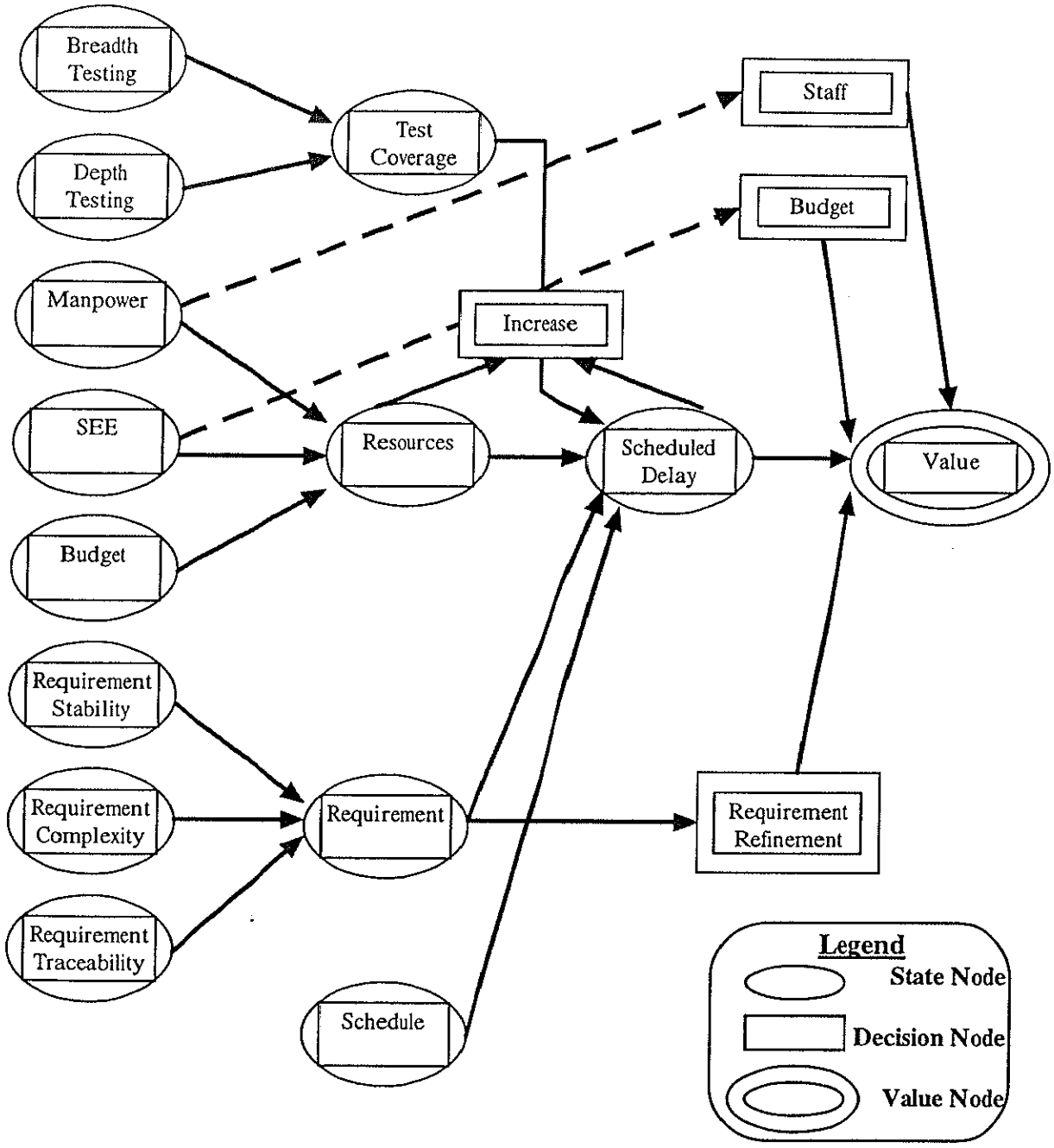

Figure 4.2: Influence Diagram for the Proposed Software Metrics

4.3 Analytical Techniques for Software Metrics Data

Various metrics data analysis techniques can be used for risk and quality management. There are a number of reasons for using these techniques. Some of them are listed below:

1. Effective graphical data presentation techniques can be used with the analytical techniques which can allow the manager to concentrate more on understanding what the data is telling them and less on simply endeavoring to grasp the data values and their inter-relationships.
2. Statistical techniques can help the eye to recognize and calculate trends, and exceptions to trends, in the data.
3. Expertise in managing software development consists in part of knowing what exceptions to look for, how to diagnose them, and what factors to consider in choosing an appropriate response. By classifying and cross-relating different kinds of exceptions, diagnoses and responses, it is possible to simplify the task of reaching a preliminary interpretation of the data, leaving the manager free to concentrate on modifying this interpretation in the light of his knowledge of the particular project concerned.

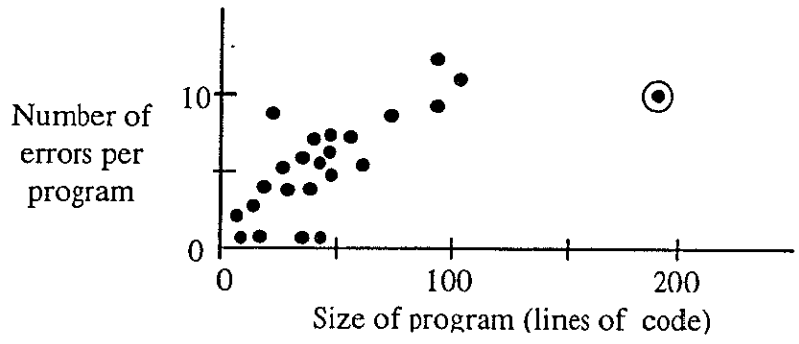

Figure 4.3: Scatterplot: Program Size versus Number of Errors

As an example, consider the scatterplot in Figure 4.3. This figure displays module size against the number of faults found during unit testing for a group of modules. When the size is small, we can observe that other factors outweigh it in determining how many faults a module has, producing the spread in the lower left quadrant of the graph. Larger modules show a strong relation between size and faults, and an exception to it is indicated by the circled point. Statistical techniques exist that can help the eye in calculating the exact relationship and measuring the degree to which exceptional points diverge from it. Formal statistical analysis is especially useful in more than two dimensions, and these techniques will be discussed in this chapter.

Possible causes of a module being large but with few detected errors include good coding, poor testing, simple control flow procedure, etc. By preparing tables cross-referencing each diagnosis to the results of other analyses that confirm or contradict it, a shortlist of plausible causes can be quickly obtained.

4.3.1 Criteria for Selecting Metrics Software Data Analysis Techniques

We first need to establish some criteria for selecting metrics software data analysis techniques. If an analysis technique is to be applied to software data, the result that the technique gives must be trustworthy. The technique must not be dangerously affected by a few erroneous or widely atypical values in the data. It must also not make assumptions about the data distribution. Classical statistical methods tend to assume that whatever data set one is working with was drawn from some underlying distribution of data commonly found in nature such as the Gaussian distribution; when this assumption is true, classical techniques are very efficient but

when it is untrue, the results can be dangerously misleading. Hence, it is very important to first fit a good distribution to the data before moving forward with the analysis.

4.3.2 Metrics Data Analysis Techniques

Some of the statistical and analytical techniques that can be used for analyzing software metrics data are the following:

- Univariate techniques
- Bivariate Techniques
- Multivariate Techniques
- Multiresolution Techniques

Each of these are discussed in greater detail in the rest of this chapter.

4.3.2.1 Univariate Techniques

Some univariate techniques include:

1. *Distribution plot* (or *frequency histogram*). This shows the values of a number of items such as the sizes of a number of modules plotted against an axis. An example of a distribution plot is illustrated in Figure 4.4(a).
2. *Boxplot.* A boxplot is a diagram that displays the lower, lower middle, upper middle and upper quarters of a one-dimensional distribution of data values. It summarizes the data without distorting it by oversimplification. The small amount of effort needed to become familiar with it is outweighed by its usefulness. An example of a boxplot is illustrated in Figure 4.4(b).

Effort to:

Design	12 hours
Document	8 hours
Code	25 hours
Test	9 hours

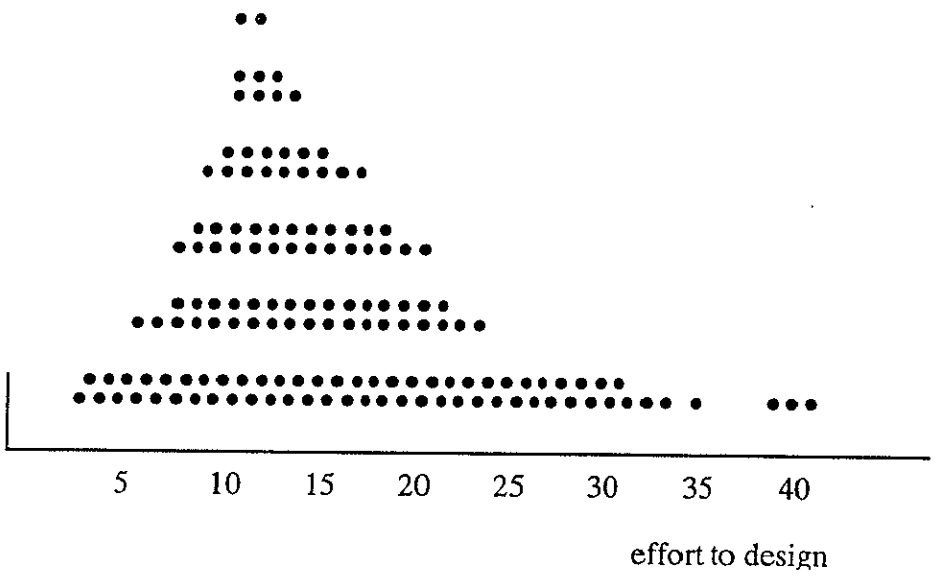

Figure 4.4(a): Distribution Plot of Effort to Design

Figure 4.4(b): Boxplot of Effort to Design

3. *Transformation*. Common transformation are: natural logarithm, square root, square of cube root, etc. There do not appear to be transformations which are optimal for all datasets, so the following strategy is (cautiously) suggested for selecting transformations:

a) for duration, effort and cost data use the natural logarithm transformation which has been used extensively for software cost models and conforms to general econometrics practice;

b) for counts based on incidents, failures and faults, two different transformations are particularly appropriate:

   i) the square root to stabilize the variance and so permit the accurate identification of outliers;

   ii) the square of the cube root to approximate normality and so permit cautious use of classical techniques.

   No single transformation can achieve both objectives.

c) for size metrics related to code or design, use the natural log transformation for metrics which do not take the value zero (or very rarely take the value), and the square root transformation otherwise.

4. *Classification* (often shown as a *histogram*). Often the data is naturally divided into value relating to a small number of items or categories. Where this is not so, it may be convenient to impose a classification scheme on a dataset (this is discussed in the next chapter) and count the number of items in each category. A bar histogram is an effective way of displaying such data. An example of a bar chart is illustrated in Chapter 3, (Figure 3.9 about the cyclomatic complexity of modules for a given project).

4.3.2.2 Bivariate Techniques

Bivariate techniques include the following:

1. *Scatterplot* or *Time-varying Plot.* A scatterplot or time-varying plot is the most common way of displaying software data for monitoring purposes. It can help comprehension if the axes of

a scatterplot also display the boxplot (or frequency histogram) showing the spread of data along that axis. An example of a scatterplot is illustrated in Figure 4.5 Visual examination of the plot will indicate whether techniques for investigating relationships, such as regression, curve fitting or smoothing, could be profitably applied. Alternatively, the data may be classified into groups with respect to one axis. The boxplot for each group, seen as a one-dimensional dataset along the other axis, can then be drawn above its interval on the first axis. The result can simplify a complex scatterplot.

2. *Regression.* Many relationships can be usefully approximated by a linear equation. However, the need to protect analyses from being distorted by the presence of exceptional points means that it would be desirable, in the calculation of such equations, to use special robust algorithms such as multiresolution analysis. Once an equation has been obtained, the differences between it and the data values (called residuals) should be examined. Points which deviate grossly from it should be removed from the dataset and the equation recalculated. Comparison of the first and second equations will reveal something of the effect of the exceptional points, and so indicate the trustworthiness of the equations. No regression should be relied on further than visual examination suggests is reasonable.
3. *Curve fitting.* Non-linear equations may appropriately be fitted to some time-varying graphs, especially those involved in measuring reliability. Outside a limited range of software applications, lack of robustness makes it essential to use these techniques cautiously.
4. *Smoothing.* Some relationships do not resemble any mathematical function and are best described by an approximation technique.
5. *Residual analysis.* If regression, curve fitting, smoothing or some other method of obtaining a predictive line has been applied to the data for a scatterplot or time-varying plot, then a

scatterplot of the differences between the data and the predicted line (called a *residual plot*) is often useful.

6. *Classification tables.* An ordinary two-dimensional histogram uses one dimension to show the classifications and one to show the histogram bars. Three-dimensional histograms use two dimensions to show the classification table categories and one dimension for the histogram bars.
7. *Tradeoff Surfaces.* These are three-dimensional plots to show either the relationship between three software metric components or two software metric components as a function of time.

Three-dimensional histograms and tradeoff surfaces are ideal for emphasizing gross trends in two-way classification tables, but, unlike ordinary histograms and charts, cannot have values easily read from them. Hence a table showing the numerical values should always be shown beside it. It helps to show, for each axis of the base of a three-dimensional histogram or tradeoff surface, an associated pie chart showing the total number in the table in each of the classes of that axis as a proportion of that total number in the entire table. This serves the same function as displaying a boxplot against an axis of a scatterplot.

4.3.2.3 Multivariate Techniques

These include the following:

1. *Scatterplot matrix.* A group of scatterplots of three or more variables against one another can be arranged in a matrix format to show multi-dimensional effects. The clarity with which such an effect is revealed can be enhanced by brushing (i.e. distinguishing by coloring or shading) a subset of the points on all the diagrams. The clarity of the resulting matrix, and whether it shows any multivariate effect, will determine whether to plot a group of variables as a scatterplot matrix or simply as a succession of individual scatterplots. Often it will be

best to combine the two styles of presentation by showing, first, a scatterplot matrix (reduced in size) and second, the scatterplots of which it is made up (each shown at full size). The scatterplot matrix acts as a contents list for the succeeding plots and emphasizes major effects.

2. *Profiles.* When the form of the relationship between a number of variables is unclear, they may be plotted against a common axis, or on radial axes of a circular graph. Studying the shapes produced for a number of cases may suggest patterns which can guide further analysis.

3. *Principal components and Multivariate regression.* The basic idea in this approach is to transform the metrics into an orthogonal system in such a way that the first principle component represents the largest variation. The second principal component represents the second largest variation, and so on. We now apply the same idea to regression analysis. In the outlier detection, we encounter all of the metrics in the analysis. Thus, the resulting principal components are based on the complete set of metrics. In the regression analysis, we consider some of the metrics as dependent variables and the remaining metrics as explanatory variables. The principal components are obtained only through utilizing the explanatory variables. For our situation, metrics x1 through x8 are considered to be dependent variables and the 14 metrics x9 through x22 are the explanatory variables. The principal component regression finds p orthogonal linear combinations (in our case: p=14) y1, y2, . . ., y p of the explanatory variables such that var (y1) >/= var (y2) >/=. . .>/= var (yp). Instead of using the explanatory variables, the transformed variables (so called principal components) are used in the regression model. The regression analysis is then performed. An attractive feature of the principal component regression is that the exclusion of any principal component from the model does not effect the contributions of the other components. In addition to making

predictions, the interpretation of these fitted models is easier than for the standard regression models.

4.3.2.4 Multiresolution Analysis

Multiresolution analysis (MRA) can be used to extract fluctuations of the metric data at different scales, that is, we can view the data with various resolutions. The Fourier transform:

$$f(x) = \frac{1}{4x^2}\int^{\infty} F(u)e^{I(ux)}\,du$$

has been the traditional tool for viewing the data with multiple resolutions, but it loses information on locations where big changes in the values occur. To remedy this drawback, we use the scaling functions and the wavelets based on Mallat's [MAL89] proposal for MRA. By computing the convolution with the scaling function, we can have a coarser view of the data, and the convolution with the wavelet detects the fluctuations in the data.

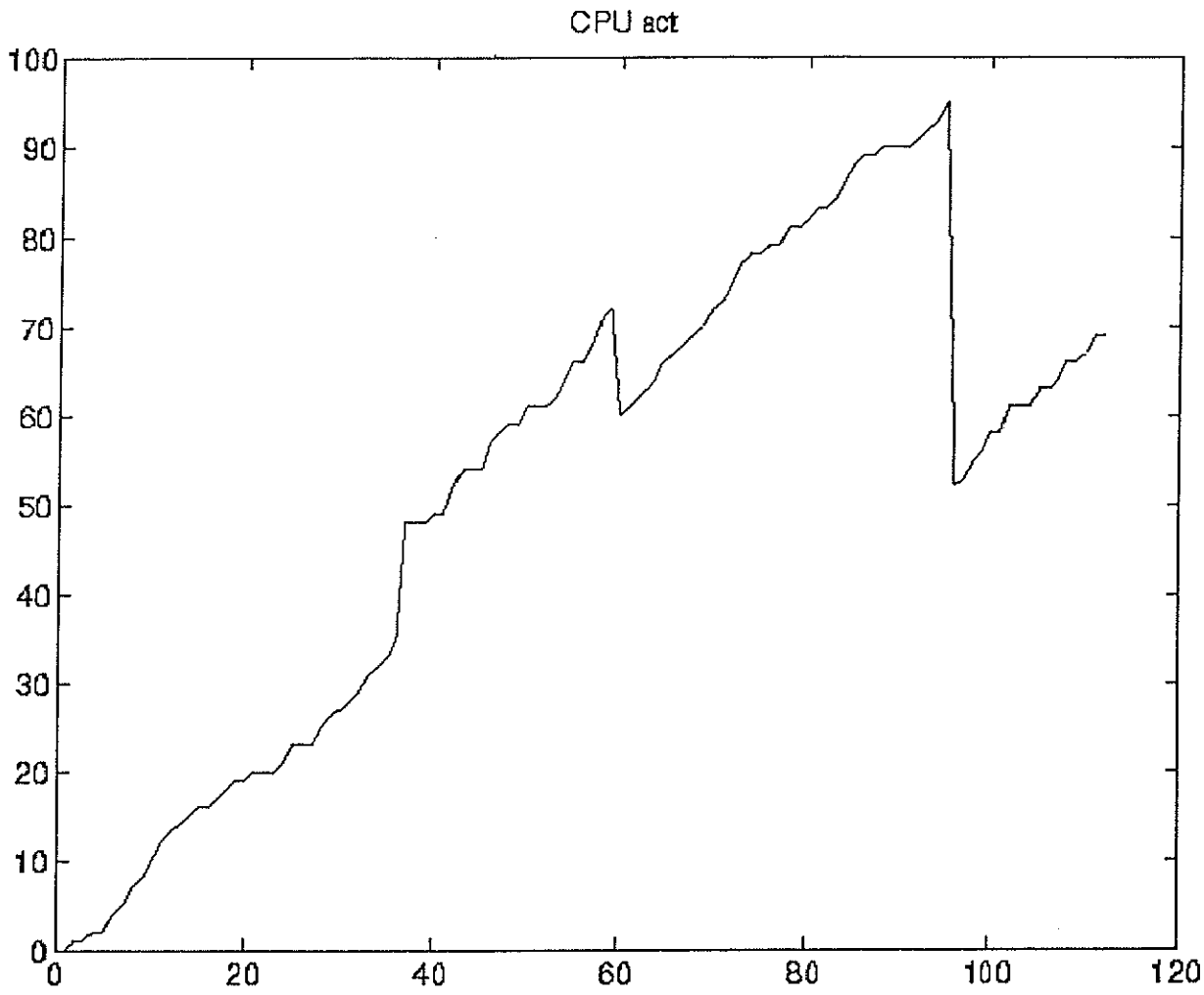

Figure 4.5: CPU Activity

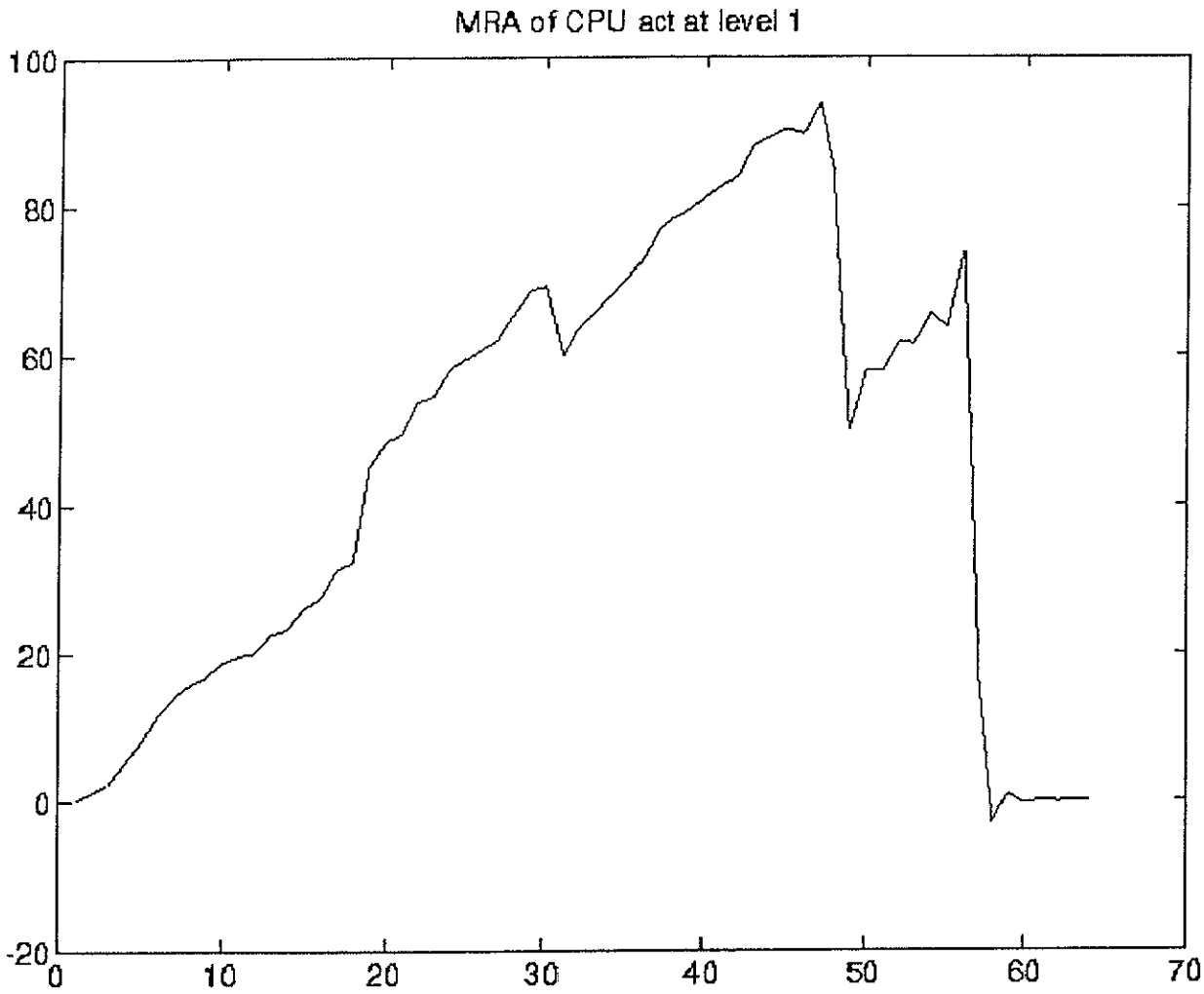

Figure 4.6: MRA of CPU Activity at Level 1

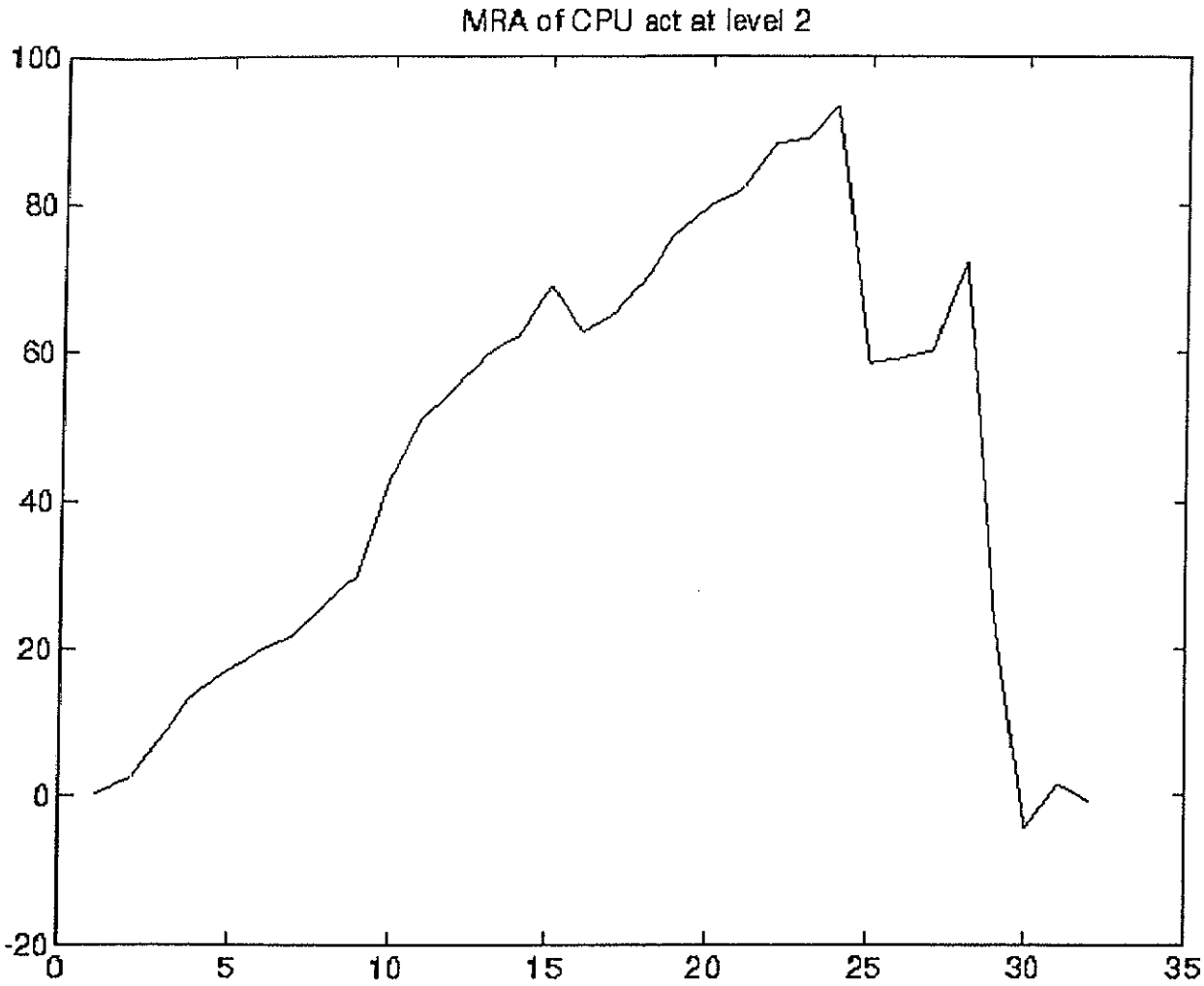

Figure 4.7: MRA of CPU activity at Level 2

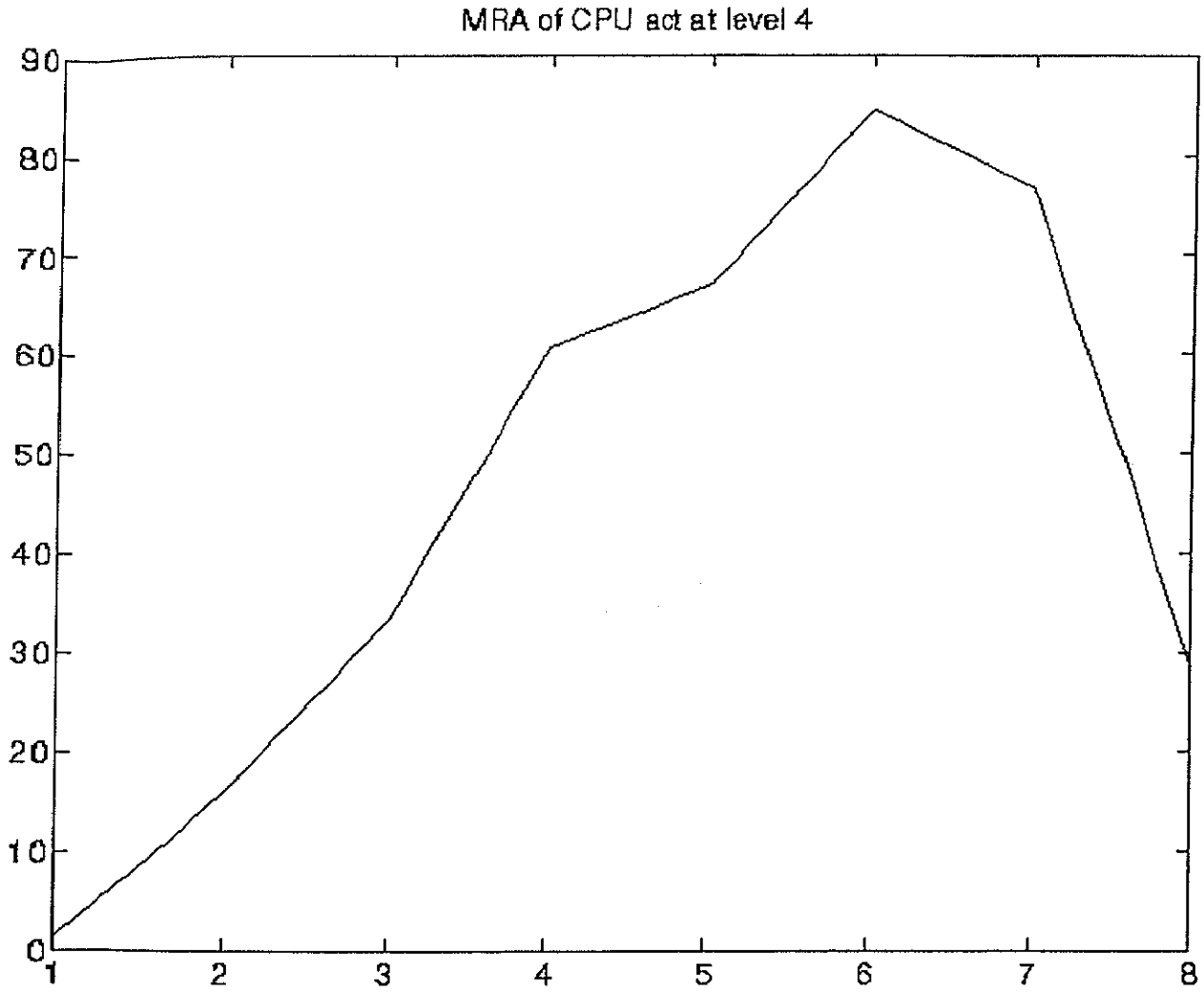

Figure 4.8: Multiresolution Analysis of CPU Activity Metrics

Consider, for example, metrics data of the CPU activity given in Figure 4.5. Suppose there is a sudden drop in the CPU activity in the middle of the software development life cycle. This is an indication of potential problems, such as drastic changes in requirements, system failures, sabotage, etc. We can then look at other metrics for possible causes:

- If there is a corresponding change in requirements during that period, then the drop in CPU activity corresponds to the change in requirements, and the cost is spent in changing the requirements.

- If the actual cost versus budgeted cost of various cost metrics is decreased according to CPU activity, then the cause is probably due to budget cuts, resulting in lower manpower and less resources available for the project.

In order to narrow down our our probing we must reduce the "noise" in the dark. This process can be achieved by selecting appropriate threshold parameter in MRA computation. Figures 4.6 - 4.8 depict the effect of removal of noise as we increase the threshold value and get smoother and smoother result.

4.3.3 Summarizing the Overall Process of Metrics Data Analysis

Figure 4.9 describes the overall process of using the above mentioned techniques to analyze the metrics data and extracting information useful for various decision making processes. Subsequently, the data extracted after this analysis can be stored in a database and can help in answering complex queries of the type depicted in Table 3.1. The detail about the handling of the queries and decision making process is given in the next chapter.

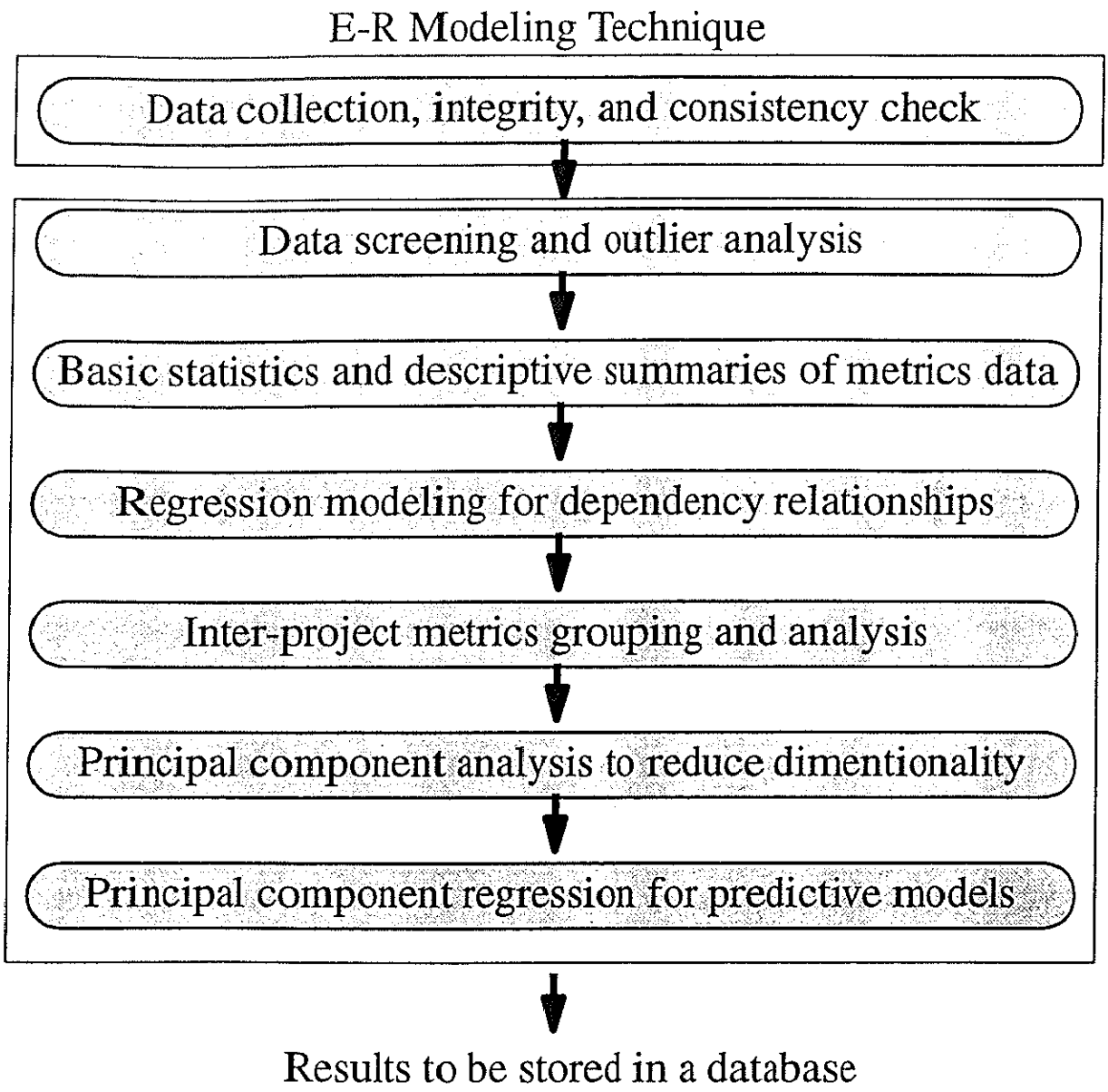

Figure 4.9: Overall Process for Metrics Data Analysis