Measurement for the management of software maintenance

Cooper, Simon D.

University of Durham

School of Engineering and Computer Science
(Computer Science)

Measurement for the Management of Software Maintenance

Simon D. Cooper

Ph.D.

1993

Abstract

Measurement for the Management of Software Maintenance

Simon D. Cooper

This thesis addresses the problem of bringing maintenance, in a commercial environment, under management control, and also increasing the profile of maintenance in a corporate picture, bringing it onto a par with other components of the business. This management control will help reduce costs and also the time scales inherent in maintenance activity.

This objective is achieved by showing how the measurement of the products and processes involved in maintenance activity, at a team level, increases the visibility of the tasks being tackled. This increase in visibility provides the ability to impose control on the products and processes and provides the basis for prediction and estimation of future states of a projects and the future requirements of the team. This is the foundation of good management. Measurement also provides an increase in visibility for higher management of the company, forming a basis for communication within the corporate strategy, allowing maintenance to be seen as it is, and furnished with the resources it requires.

A method for the introduction of a measurement strategy, and collection system, is presented, supported by the examination of a database of maintenance information collected by a British Telecom research team, during a

commercial software maintenance exercise. A prototype system for the collection of software change information is also presented, demonstrating the application of the method, along with the results of its development and the implications for both software maintenance management and the technical tasks of implementing change.

Acknowledgements

Firstly I would like to thank my supervisor, Mr. M. Munro, for his help and guidance throughout this project, and without whom none of this would have been possible.

I would also like to express my thanks and appreciation for the many valuable contributions from Mr. J. Foster at British Telecom Research Laboratories, and all the members of his research group, RT3121.

Many people have contributed to this work through their discussion, support and encouragement. My thanks to all those who have helped, especially Miss R. Kenning and Mr. R. Freeman. My thanks also to all my friends, colleagues and acquaintances who made my time at Durham such an enjoyable experience.

Finally I would like to thank the Science and Engineering Research Council, and British Telecom Research Laboratories who funded this project, and British Telecom for providing the data on which part of this project is based.

Copyright

This work is dedicated to my wife, Karen

Contents

Chapter 1

Introduction

1.1. Software Maintenance

Software maintenance, as defined by the IEEE [IEEE84], is:

> The modification of a software product, after delivery, to improve performance or other attributes, or to adapt the product to a new environment.

In other words, software maintenance includes all work done on a software system after its delivery into its working environment.

In keeping with this definition, maintenance activity can be divided into four categories [SWANSON76, PRESSMAN87]:

- *Perfective maintenance:* the alteration of code so that it conforms to a new specification. This normally involves the addition of functionality.

- *Adaptive maintenance:* the alteration of code so that it runs in a new or changed environment.

- *Corrective maintenance:* the alteration of code to remove errors. That is, to make the software conform to its specification.

- *Preventive maintenance:* alteration of the code in an internal sense only, i.e., no change in functionality. This is normally performed in order to make future maintenance work easier and less costly [WADE88].

The definitions above show that during most of the life-time of a software system, which in many cases is 25 years or more, it is in the maintenance phase of its life-cycle. Software maintenance is accepted as being the most costly phase in the this life-cycle. It is quoted as accounting for between 50% and 80% of all software expenditure and effort [LEINTZ79, MORISSEY79] and this is likely to be the case for the foreseeable future [SCHNEIDEWIND87].

With maintenance being such an important part of the life-cycle, it is important to find methods of reducing the cost of maintenance. This is perhaps more important than finding new methods of developing software as existing software is going to be with us for the near, if not the long-term, future [SCHNEIDEWIND87].

1.1.1. The Problems with Maintenance

The high cost of software maintenance can be attributed to a number of factors, an important one being the lack of close and effective management of the maintenance process at the line management level. This is due in part to the

special difficulties of controlling maintenance activity [KAFURA87]. If the cost of software maintenance is to be decreased and the quality improved, we must impose stronger and more rigorous control over the whole process.

The general principles of management are well defined and understood, allowing projects to be completed on time and within budget [WINGROVE86], but there seems to be resistance to applying these principles in the maintenance field.

Maintenance poses special problems to a manager [KAFURA87]. A more diverse group of people, over a longer period of time, work on the software, with fewer defined work standards or methods, than in any of the other phase in the software life-cycle. A large proportion of this work consists of trying to respond rapidly to change requests due to the direct impact on a customer, or the business function of a customer, so the maintenance activity takes on a responsive or 'fire fighting' role. This role causes the backlog of less urgent requests to increase, and rules out any more controlled preventive maintenance work with a view towards reducing problem areas.

The lack of control and the rapid response nature of the work allows the natural degradation of the system due to the maintenance activity, described by Belady [BELADY76], to go unhindered. The most noticeable symptom of this degradation is an increase in system complexity. As the systems complexity grows rapidly, so the ripple effect - the introduction of new errors, or adverse changes, while making a required change - becomes a major problem, increasing the workload and the backlog. In a study conducted by Collofello and Buck [COLLOFELLO87] it was concluded that more than 50% of errors were introduced by previous changes. The difficulty of *'fighting the fire while*

feeding the flames' is apparent. The result, as the backlog builds and the error rate increases, is that the system is 'maintained to death' [BROWN80].

Maintenance activity, because it is driven by the people actually using the system, is invariably put under heavy time constraints. The result of this, combined with the large amount of maintenance done in an uncontrolled way generating more work, has lead, in many places to a maintenance backlog. This is a queue of work waiting to be done, sometimes years old. This adds to the pressure on the maintenance teams and escalates the problem.

Real management of the process is needed to bring maintenance under control and allow for future planning and scheduling. This would lead to more efficient use of time and other resources, a reduction in the backlog of work and allow for preventive maintenance, and, as an end result, reduce the cost of this most expensive phase of the software life-cycle. It has been shown in other fields that management control can achieve these objectives, it therefore must be a requirement for the software maintenance field.

1.1.2. Maintenance in the Corporate Strategy

One of the major problems faced by the maintenance community as a whole, and particularly by managers of maintenance departments in large commercial organizations, is the lack of recognition by senior management of software maintenance as part of the corporate strategy. Maintenance is often regarded as an unimportant sideline to software development [LIENTZ80], an annoying waste of money, fixing problems caused by bad development.

This lack of recognition of the problems of software maintenance, and lack of recognition of the benefits afforded by its effective application, introduces major stumbling blocks in the path of management of the process and its overall control.

The reasons behind these problems is a lack of effective communication between the managers of the maintenance teams and the senior or corporate level management responsible for running the business. This corporate level management tend not to regard the organization's software portfolio as a company asset, and fail to realise the cost of keeping this software asset in working order, and keeping it in line with current business and practical requirements. They do not realise because they cannot be told in a practical way. A requirement therefore exists for maintenance teams and, more particularly, their managers to talk the '*language of business*' in order to present their case effectively [COLTER88].

Software, in any large commercial environment, represents a significant investment, and maintenance work is further investment that is required in order to maintain the value of the software as a corporate asset. In these terms, software maintenance has a cost, and it also has a benefit in the corporate strategy. These costs, their projected values and their comparison is how a company should view its maintenance component. This is the '*language of business*', a language that high level management can work with and expects.

Communication about maintenance in these terms is, however, not possible at present. This is because the values required cannot be quantified. They cannot be quantified because of a lack of understanding and knowledge about the content of the values, and how to go about producing them. The best attempt at producing these values is estimation (finger in the air?) by managers with

experience of the maintenance role within the specific environment. The target, however, must be to produce these figures on a routine and accurate basis. The only answer is for measurement of the processes involved in order that the underlying components and so the values themselves, can be generated [COOPER89].

Measurement of a process (and the development of a measurement system), increases the visibility and understanding of that process. This greater understanding, and the increased visibility, combined with the experience of managers, will allow the estimates of cost to become more accurate, their basis to become more demonstrable and will allow the measurement to improve. The end result is the ability to talk with confidence about the maintenance environment and its role within a corporate strategy.

The greater visibility and understanding, along with the evidence to support it, will provide the basis for demonstrating the part played by maintenance in the corporate strategy. In this way the profile of maintenance as an important phase in software development, and in the business, can be increased, and put on an equal footing with other elements of the overall business strategy.

1.1.3. A Model of Software Maintenance

The processes involved in software maintenance and the organization of the tasks are of crucial importance to the success of the activity. There is, however, no accepted framework in which the processes of software maintenance and the organization of the constituent tasks can be placed. This is a particular problem when comparisons are to be made between different teams or organizations. A uniform model is required onto which any maintenance team

can be mapped, allowing a generally applicable discussion and comparison to take place. Such a model is presented here in preparation for discussions later in this thesis.

A number of models of the maintenance process have been proposed. A general discussion and comparison by Collofello can be found in [COLLOFELLO86]. These models are generally directed at the technical aspects of performing a maintenance task [BOEHM76, MARTIN83, PARIKH82, PATKAU83].

John Foster et al. [FOSTER89] presents a model based on observations of actual maintenance teams rather than a theoretical starting point. Within this model the technical and managerial issues can be presented and discussed from a common stand point, applicable to any maintenance team. The model addresses all aspects of maintenance. The model consists of seven levels, each representing a different view point on the maintenance process, from the corporate view, down to the technical level. The seven levels are:

- *Asset Level:* the software as a company asset. It considers the entire set of software owned by the company and the overall costs and paybacks associated with it.

- *Portfolio Level:* the set of software items owned and used by the company. It concerns the set of products that support particular business functions of the organization.

- *Network Level:* concerns the interactions between the levels of resource applied to software products, in teams responsible for more than one product.

- *Product Level:* one single software product. Concerns the overall activity involved with a particular product.

- *Team Level:* a maintenance team. The processes related to a maintenance team.

- *Function Level:* a function performed within a team.

- *Topic Level:* components of functions. Concerns individual actions performed by members of the maintenance team.

The model represents a useful anchor point on which to base further discussion and to allow discussion to be based on a common ground with common reference points.

In the context of this thesis, we will particularly address the Team Level of this model. This represents a level of abstraction away from the actual tasks involved in making a change to the software source code and deals more with the overall area of concern of a line maintenance manager. We are also interested in the communication paths to the higher levels of the model, representative of corporate level management.

1.1.3.1. The Maintenance Team

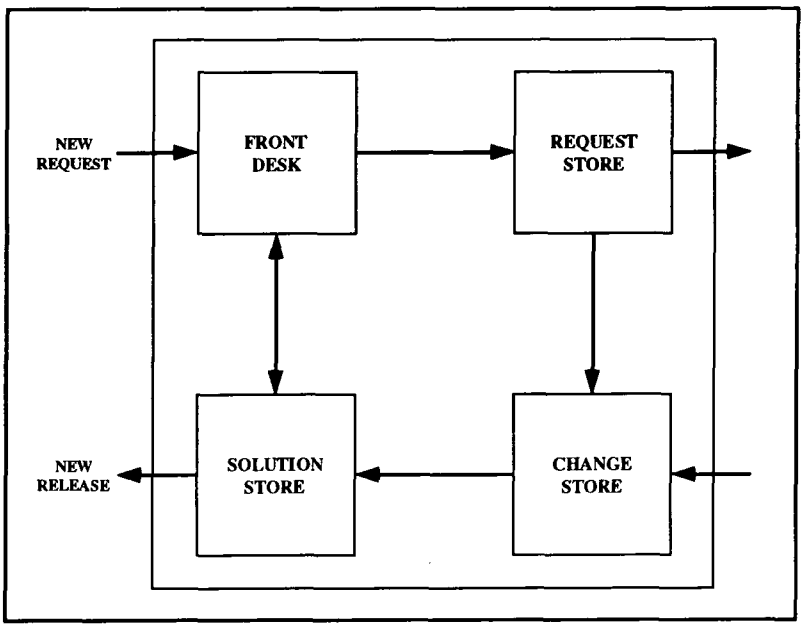

Fig. 1 The maintenance team

Figure 1 is the conceptual model of a single team in the maintenance organization. It is idealized in the sense that a real team may not exhibit all of the features shown in the diagram, although in general, they are present in some form.

The diagram represents objects, or duties in the maintenance team, with the arrows representing the flow of information, which is related to the actual work being done. This representation allows the model to be applied to maintenance teams of any size, from a large many-man operation to a single maintainer responsible for all stages in the model.

The larger rectangle in the diagram is the organizational boundary of the team, outside which the team has no control. To the left are the customers, generating requests to the team, and taking receipt of new releases of the software or other products of the team, such as updated user instructions etc.

Requests are received by the front desk task. The front desk may be able to offer an instant solution from previous work, in which case the solution is released. These known solutions are stored in the solution store, which may be documented knowledge or knowledge in the heads of members of the team. This solution store is fed by all the activity of the team.

If the front desk cannot offer a solution, the request is passed to the request store where it is queued for further work. This is generally a prioritized list of requests awaiting action. In the optimum case, this store will always be empty, but in reality it exists in some form.

When a request gets to the head of the queue, work is done on the analysis of the request, and the design of the change required. This designed change is then stored in the change store. Designed changes remain in this store until a decision is made to implement a subset of the available changes. At this time, the changes are implemented and tested, and thus move from the change store to the solution store, ready to be shipped to customers.

Two more important features are also represented. Firstly there is a feedback loop from the solution store to the front desk if a problem is found during implementation of a change. This starts a new iteration of the loop.

The second feature is the communication to the right of the diagram. If a change request is outside the scope of this team it may be passed on to another team. The current team become the customer to another team. When the change has been completed, the design returns from the client team into the change store and continues round the loop.

In any particular maintenance team, as stated before, some of the features described above will not be obvious. The features do, however, generally exist in some form, whether it be one person doing all the tasks with the solution store in his head and the request store in his *in tray*, or a large team with carefully apportioned jobs. Even in an organization where changes are made directly to the code, immediately on request, it can be represented as a fast transition around the loop.

1.1.3.2. Maintenance Network

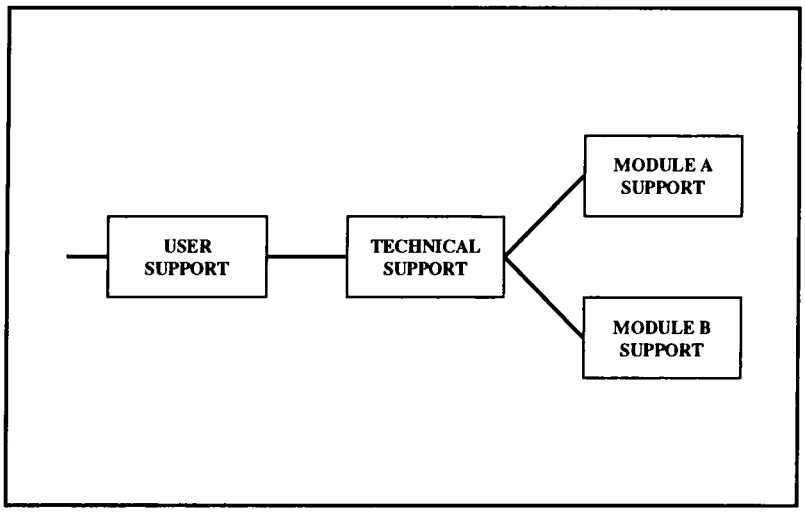

Fig. 2 The maintenance network

Figure 2 shows a maintenance network (not related to the network level), in which each box represents the outer box from the maintenance team diagram - the maintenance team organizational boundary, and the lines represent two way customer/client communication. This represents a more complex, and generally more normal, form of the *Team Level* of the model. It is rare for a single team to be responsible for all maintenance activity in any but the smallest environments. The diagram represents an example of the organization of a

maintenance department with two levels of user support before the change requests are passed to the actual change process. In this case two bottom level maintenance teams are present, each responsible for a different part of the system.

In this way, any organization of maintenance effort can be represented in terms of the teams and their communications.

1.2. The Thesis Position

Software maintenance, at present, suffers from a lack of effective line management and from a poor image at the corporate level.

This thesis attempts to address the problem of bringing maintenance under management control, and also increasing the profile of maintenance in a corporate picture, bringing it onto a par with other components of the business. This control will help reduce costs and also the time scales inherent in maintenance activity.

Software maintenance is a very costly part of the software life-cycle. In fact, it is now widely accepted as the most costly, and certainly lasts for the longest time. This position will only get worse as hardware costs get lower, and software development methods improve. In commercial environments, the reduction of maintenance costs is assuming more and more importance, and to this end, research in this area is desperately needed.

One of the reasons identified, for the high cost of maintenance, and the lack of acceptance in the commercial world of the strategic importance of software

maintenance, is the lack of effective management of the maintenance process and the teams involved in this work. This poor management leads to higher costs and poor communication with higher management resulting in the situation seen in many places.

In order to introduce more effective management into the maintenance arena, measurement of the products and processes is required to increase the visibility of these products and processes, to increase understanding and to provide a basis of knowledge about software maintenance and allow communication of that knowledge.

1.2.1. Management Through Measurement

The prime objective of this thesis is to show how the measurement of the products and processes involved in maintenance activity, at a team level, increases the visibility of the tasks being tackled. The increase in visibility leads to greater understanding and provides the basis for imposing control on the products and processes.

The increase in visibility, and the availability of data about the products and processes both in their current state, and historically, also provides the basis for prediction and estimation of future states of the projects being undertaken and the future requirements of the team. This prediction requires models, which require observations to develop and validate.

Prediction and estimation are necessary ingredients for forward planning. The ability to forward plan, the ability to see what is currently happening and the ability to control produce an environment for good management.

Measurement also provides the basis for communication with higher level management of the company. The visibility that measurement produces is in a form that can be understood by these people - figures, forecasts, targets and progress - the language of business. In this way, measurement provides the basis for communication within the corporate strategy, allowing maintenance to be seen as it is and furnished with the resources it requires.

1.2.2. Maintenance Research

An important product of the measurement of the maintenance products and processes is the opportunity provided for research.

In order to provide software maintenance research with a firm basis in the real world, large quantities of real world observations of the maintenance process are required. This data will provide the basis for software maintenance model creation and validation, as well as calibration of models to particular environments and working practices.

There is only a certain distance one can go in pure research without validating ideas and showing them to be correct in real situations. Research results can never be truly accepted unless they are shown, in a practical sense, to be true. This is a lesson learned from research into most fields, but particularly the sciences. Research into software maintenance is necessary if a true understanding is to be formed, and it is truly to become a science. This science evolution requires measurement of the products and processes involved.

1.2.3. The Goal/Question/Metric Paradigm

Rombach and Basili [ROMBACH87] present a method of developing a measurement system based on a top-down schema.

The method involves the careful definition of goals to be fulfilled by the measurement system. The goals are definitions of pieces of information required at a management level. These goals are therefore, normally, fairly abstracted from the products and processes of maintenance. An example is *'Examine the effectiveness of the maintenance effort.'*.

The goals defined are broken down into sets of questions that provide the information required to satisfy the goals. The questions are nearer to the products and processes but still represent a level of abstraction. An example is *'Is the user satisfied with the function, performance, etc.?'*.

The questions themselves are then broken down into sets of measurable metrics that provide the answers to the questions. In this way a system of measurement is produced, that measures the products and processes involved and generates, by means of answering questions, information that fulfills the goals defined. These goals, being management defined, provide information to aid in the management of maintenance.

This method has been shown to be applicable, and to work, providing management information from measurement. It has also been shown to be useful [ROMBACH87].

The method does, however, have a number of shortfalls.

Firstly, the paradigm depends on the definition of a set of specific goals before the method can be applied. This identification and definition of goals can be time consuming and difficult, at the best of times, but is required to be accurate and complete, as the rest of the method and the final measurement system depends wholly upon it.

The process of breaking down goals into questions can also be a very difficult, and often inaccurate job. This is particularly true when the goal is an abstract, perhaps corporate goal, without any direct relation to the product or processes in the maintenance environment. The next stage, that of production of metrics from the questions, can again be complex and involved, and may not be possible. The metrics produced, or required, may also not be measurable.

The application of this method, therefore, requires a substantial initial investment in time and effort to define the metrics, sometimes with limited results, and also guaranteeing a long time lag between inception and the first results from the system. It is also often true that the implementation of the metric collection system developed is unworkable in the environment to which it must be applied. This being primarily true because the system does not take into account the working environment or current work practices.

Once a system of metrics has been defined, the measurement system must be implemented. This now highlights another problem with the paradigm. Unless careful attention has been paid in the early stages, a system of metrics must now be collected that perhaps bear no relation to the procedures currently in use and the available data set. In these cases, a large investment is again needed to implement the system, including effort on the part of the maintenance teams, who are already under time constraints.

The effectiveness of the measurement system, and its real value, only now can be established. The most likely outcome, as with any new project, is that changes are required. This, however, becomes another costly exercise. Any change in goal may result in a large amount of rework effort, and a completely new measurement requirement.

With these shortfalls in mind, it must be remembered that the method has been shown to produce valuable and useful results once the measurement system is implemented. It therefore provides a good basis for further work.

1.3. Criteria for Success

The basic premise of this thesis is that measurement of software maintenance products and processes produces visibility and understanding, leading to better management of the software maintenance environment at both line and corporate level. This has been shown by a number of studies, including the Goal/Question/Metric paradigm.

The hierarchy presented in the Goal/Question/Metric paradigm of collection of data providing answers to higher level questions is valid and useful. This thesis, therefore, addresses the shortfalls of the Goal/Question/Paradigm by applying a bottom-up design approach. This approach will help target the hierarchy, overcome the overhead problems which are a major consideration in an industrial environment, and provide for an evolution of the system to take account of goal changes and gathered experience.

1.3.1. The Goal

The goal of this thesis is to present a method of introducing a measurement system into an industrial software maintenance environment.

The purpose of the measurement system is to increase the visibility of both the products and processes involved, leading to improved management control at the line level, and greater ease of communication with corporate level management.

The method must produce a system for both the collection and the use of information about the products and processes, and provide for the evolution of the system to reflect changes in the measurement requirements and the tailoring of the system to better satisfy existing requirements, as knowledge and understanding within the environment increase.

The method must also allow for certain basic necessities within an industrial environment. The first is the need for as little overhead, in both effort and cost, as possible for the implementation of the system. Specifically, the amount of effort required to develop and implement the initial system, and the amount of impact the system has on the resources within the environment must be kept to a minimum. The second is the need for immediate feedback of results from the system in order that any impact that the system has can be justified immediately, and the benefits and drawbacks can be assessed. These criteria are of utmost importance in an industrial setting, but have not been addressed in other work.

The thesis draws on the experience of the Goal/Question/Metric paradigm to show the usefulness of collection of data, but provides a solution to the shortfalls of that method.

1.4. Thesis Overview

This thesis begins with a description and analysis of measurement of software. This includes the rational behind the need for software measurement and metrics research and the advances that have been, and must be made. A large amount of work has been done on the application of metrics to program code, however, very little published work addresses the higher level problems of the measurement for management of the software, and particularly software maintenance. For this reason, the concept of *configuration metrics* is introduced, being metrics abstracted away from the actual code of the system, and more applicable to the management level.

In Chapter 3 there is a discussion of the problems associated with data collection and analysis and the presentation of a method for developing a data collection and analysis system in an industrial maintenance environment. The evolution of the measurement system is also addressed.

Chapter 4 introduces a British Telecom Research Laboratories project to collect software maintenance data, and describes the application of the method presented in Chapter 3 to this maintenance project.

Chapter 5 discusses a prototype data collection and analysis system whose design is based on the preceding discussions. This prototype system is specific

to the British Telecom environment, but has components that are generally applicable.

Chapter 6 brings together the conclusions from the preceding chapters, and evaluates these conclusions with respect to the Criteria for Success. Chapter 7 discusses the potential for further work based on what is contained here.

Chapter 2

Metrics and Measurement

2.1. Metrics

2.1.1. Definition of Metrics

A *Software Metric* is a quantitative measure of a certain feature, or collection of features of the software in question or the processes that went into producing the software [DEMARCO82].

The *Software* consists of the programs and documentation, in all their representations, which result from a software development and maintenance process [INCE90].

These definitions mean that Software Metrics are quantitative measures derived from, for example, the source code of a system, its design documents, system documentation, quality control documents or error report documents. The features of these documents that the metric measures can be anything that can be quantified. It can be readily seen that the possible metric set from any software is both varied and infinite.

The size and structure of this set of possible metrics leads naturally to the notion of a *Useful Software Metric* being a member of the set of possible metrics that communicates some information about the software which is of use for our current requirements. The definition and measurement of these metrics is, however, far from straight forward.

2.1.2. Types of Metrics

In general, software metrics are divided into two classes, product metrics and process metrics:

- *Product Metrics*: These are measures that apply to the products of the software development and maintenance processes. The products include such tangible items as source code, requirements documents, specifications documents, etc. Product metrics are based on the finished product, that is, when the process concerned has been completed, or can be estimated values based on the incomplete product.

- *Process Metrics*: These are metrics that measure the processes involved in the software development and maintenance. Examples are the rate of document production or speed of error correction. These metrics can only be produced while the process is on-going. It may be noted that these are often derived from changes in product metrics. For example, rate of document production is produced from the relative size of the documentation at two distinct times. Size of documentation is a product metric, as it is based on a measure of the documentation, a product of the process. This shows

in general how closely related the two types of metric are. It is often only feasible to measure processes by applying product metrics.

Metrics are values associated with certain features of the software. For a metrics to be useful it must be well defined, and clear as to exactly what feature the metric characterises or to what process the metric relates and exactly what that relation is. It must also be clear what factors effect the value of the metric. These, in general, are very difficult conditions to satisfy, but are required if any metrics theory is to have a firm foundation. Building on uncertainty can only lead to increased uncertainty.

As stated above, a metric is a quantitative measure of the software. Quantitative measures are those that define a position on a scale. Baird and Noma [BAIRD78] divided possible measurement scales into four categories :

- *Nominal scales*: Measured items are classified into groups. Each group has a unique and constant value for the measure, so there is no ordering of the members of a single group with relation to each other. Different groups may have an implicit ordering associated with them, although this is not necessary and is usually intuitive depending on the desired value for the measure. An example is dividing cars into groups depending on their basic colour i.e. red, blue, green, etc. Here there is no ordering of two blue cars, they are just blue. The only ordering of groups may come from a desire to have certain type of colour. For example, if a bright colour is required, the red group may be better, or higher on the scale, than the blue and green groups.

- *Ordinal scales*: Measured items are individually ranked. Each item has a value associated with it. This value, and the requirements of the observer impose and ordering on the items dependent on the value. The gap between items, however, has no meaning. An example is taking the first letter of the surnames of a group of people. The letter is the value associated with each person, and the ordering of these letter in the alphabet impose a ranking on the values, and so the people, called alphabetic order. The fact that there is a gap of, say, four letters between two people is , however, irrelevant and meaningless.

- *Interval scales*: Measured items are individually ranked, and also their relative separations are given on the scale. This type of scale requires a unit of measurement to be defined. Examination results presented as a grade from A to E are presented on an interval scale. The unit is a grade. Individual results can, not only, be ordered on the scale, B is better than D, but also we can say B is two better than D where as A is only one better than B. Looking back at the previous example of ordering alphabetically, although the same relations can be quoted, the gap between people in the ordered list is irrelevant and meaningless.

- *Ratio scales*: This type of scale is similar to the Interval scale, but a zero is defined on the scale. This is an important addition. On this type of scale, ratios of values have meaning. An example is the length of pieces of string. The pieces of string can be ordered by length, statements about the difference in lengths can be made successfully and statements such as *'piece A is twice the length of piece B'* also have meaning. Notice that this is not the case with the

grades example above. It is also worth noticing that on a ratio scale mathematics can be used in a meaningful way e.g. adding the lengths of two pieces of string. The ratio scale is the minimum requirement for this type of manipulation.

From these definitions it can be seen that a measure on a ratio scale is the most flexible and useful. It not only is it the best defined scale, but it also allows mathematical manipulation. This is the highest form of scale, the nominal scale is the lowest. Most useful measures in common use are on a ratio scale, and certainly most of the useful ones in the sciences.

A measure on any of the scales can be converted into an earlier, less well defined one by a simple function. For example, if we group pieces of string whose lengths fall within certain bounds, into separate groups, they are now on an ordinal scale. Measures cannot, however, be converted to a higher scale without the collection of more information. Measures on different scales can not be combined in any meaningful way, the only course is to convert the measure on the higher scale to one on the lower scale. The result can only be on the lower scale or a lower one.

A nominal scale is the least useful as it provides least information, and no manipulation can be performed on the measures.

From the above discussion it can be seen that software metrics that are defined on a ratio scale are going to be the most useful. However, a scale of this nature is very difficult to define, and requires a very deep understanding of what is being measured. A nominal scale metric, on the other hand, is relatively easy to define, although usually its usefulness will be small.

2.1.3. Useful Software Metrics

A *Useful Software Metric* will be defined as a software metric that is defined exactly, as outlined earlier, and is also defined on at least an interval scale. The interval scale is chosen to allow some flexibility.

The definition above provides an important requirement for a useful software metric, that we must have a defined unit of measure for the metric. The remainder of this thesis will tend to concentrate on useful software metrics, although it is recognized that metrics on lower scales are valid. It should be borne in mind, that the aim of metrics research must be to find measures on ratio scales. Most, if not all, of the other sciences are based on theories and laws relating metrics on ratio scales. If computer science, and metrics research in particular, are ever going to exist on a par with other sciences this type must become the basis.

2.2. Measurement of Metrics

The definition of a useful metric, as described above, requires a definition of the feature to be measured along with the definition of a scale on which to measure it. The next stage is to find a way of generating the measure from the software.

The process of measuring a metric may be very straight forward if the metric has a simple relationship to data that can be collected from the software. An example may be a metric of the number of lines in the source code files. This is directly measurable in an obvious way. Metrics of this type we will call *direct*

metrics. Direct metrics are, however, only a small proportion of required metrics. The larger proportion consist of metrics that cannot be simply and directly measured from the software. These metrics, called *indirect* metrics, require the measurement of various facets or characteristics of the software that can be directly measured. These measures are then combined, according to certain rules, to produce a value for the indirect metric. It should be noted that the constituent measurements of the indirect metrics are themselves direct metrics.

This description leads to the notion of a hierarchy of metrics. The lowest level, being the direct metrics, measured directly from the software. The higher levels then represent combinations of the lower level metrics according to various combination rules.

No new information is imported into the structure as we rise up the hierarchy, only the representation of the information collected by the lowest level direct metrics. This does not take into account possible '*intelligent*' input into the structure to derive new information. This is because we want to make the derivation of information as objective as possible, as will be described later, which excludes '*intuition*' from the process. Specific rule-based inferencing can still be regarded as base level information being combined into higher level results.

Why does the base level information need to be combined into higher levels if the base level contains all the information required? The answer is in the representation of the information. The higher level metrics contain the information in a more abstracted and usable form than the larger number of lower level metrics. This leads to a new concept, that of a *Useful Metric Set.* A useful metric set is the set of metrics that communicate the information

required about the software in a usable way. Depending on the requirements of the measuring system, this may consist of a set taken from the lowest level only, the direct metrics, or a set containing metrics from various levels. An interesting aside is the comparison of levels of metrics with the users of those metrics. A comparison can be drawn with a business structure where the lowest level is more use to the technician, whereas higher levels apply more to managers requiring more of an overview and less specific detail.

A further requirement for useful metric measurement is that it is both repeatable and objective. Only in this way can it be ensured that the metric is a true measure of the relevant feature. This will already be a feature of the metric if the initial direct metrics and the rules for combination of metrics are well defined. If the measured metric exhibits all these features, it can be compared to other values in the knowledge that the comparison has validity. In general, this rigid definition of metrics will allow them to be automatically measured, thus ensuring their objectivity and repeatability, and also reducing the overhead in the measuring. In a usability sense, this can be put as an all encompassing requirement for a usable metric - one that is both useful and automatically derivable.

2.3. Uses of Metrics Systems

There are many ways in which metrics can help in all phases of software development and maintenance. A lot of work has been published on the subject, some of which is referenced below. A detailed overview of metrics research can be found in [HARRISON84, COTE88, WAGUESPACK87].

The following discussion is based upon a comprehensive metric set. This set, if it were available, would provide a useful metric for any item of information required. This is obviously a target for metrics research and a future possibility.

2.3.1. Product Metrics

2.3.1.1. System Description

The comprehensive metrics set would describe the software completely. This description would be both detailed and complete. It would contain measures of software features normally only determinable by expert judgment, such as 'quality' and 'reliability', as well as the more accessible external features. The description would allow meaningful communication about the system and the passing of knowledge and information in an concise and objective way.

This ability to describe a unit concisely and objectively is the foundation of any science. In physics, for example, there is a set of defined features for describing an object, such as weight, velocity, size etc. The description provides not only the basis for meaningful communication about the unit, but also for understanding the unit. Understanding of an object or system can never come if it cannot be described and documented quantitatively, in a concisely and objective way [EPICTETUS00].

The comprehensive metric set has not been defined. This set would be enormous, to say the least, and therefore impossible to use. The important point is that a subset of this comprehensive set, which contained enough information about the software or part of the software for our needs at the time, is possible, and this is the set that would be used.

2.3.1.2. System Comparison

Once the software system can be described objectively and in detail, using a standard set of metrics, comparisons are possible of different software systems by comparing the respective metric sets. This comparison is both objective and repeatable among a number of software systems. By concentrating on those metrics that are important in the current situation, software best suited to the current environment can be identified, for example, if maintainability was more important than size for two pieces of software performing the same job. This decision is based on figures as opposed to a detailed analysis of the software by an expert [INCE88].

2.3.1.3. System Specification

Using the comprehensive metric set, exact definitions of the properties required from a new software system can be laid down, at the requirements stage of development, as well as a definition of its functionality. The finished product can be compared to the specification in an objective and impartial way, to determine the suitability of a system. The comparison could even be part of the procurement contract.

This comparison can also be performed on the software during its maintenance lifetime to ensure the system continues to comply with the specification, and does not degrade.

There are many opportunities to improve Quality Assurance [COLLOFELLO87], by extracting objective and analysable measures of a software product, and comparing these with optimal, or required values. In this

way we go some way towards recognizing *'good software'*, by allowing tangible criteria to be laid down for deciding what is good. These criteria will vary from environment to environment, and from product to product, but the criteria can be laid down based on knowledge and past experience.

Lastly, by describing, exactly, a system's features, by assigning values to the features, decisions can be made about the relative importance of those features, and the final product can be checked to ensure it reflects these priorities. The metric set can give numeric measures of the relative importance of features to provide targets for development and product assessment. Do we want the software to be small or cheap or easy to understand and maintain?

2.3.2. Process Metrics

2.3.2.1. Process Modeling

If models of software and software development could be generated, that is, rules that govern the metric set of a piece of software and allow the evaluation of new metrics and the prediction of future values of metrics, it would go a long way to improving our understanding of software. As in all engineering disciplines it is traditional to be able to predict, or estimate attributes of a finished product from some initial data. This is a property a model has, but a model can only be based on metrics and generate metrics as results.

If models of software development were available, features of a software product could be predicted while it was still being developed. Such things as the cost, or how long it will take, how big will it be. At this stage the initial values used for the development method can be altered or tuned, cheaply, to

attain the desired result at the end, without expensive backtracking as the development reaches it's conclusion.

If a system's important features can be described by a finite set of numerical values, that systems description can easily be stored at various stages of the development, giving an historic record of the development. This database can then be called on in the future to compare with the state of a new system under development, to help predict the behavior of the new system or method of development. The database can also be analysed to find trends in the data that may show shortcomings in (or advantages of) various development methods so these features can be avoided (or enhanced) in the future. This forms the first stage of model development.

The end result is the ability to mathematically analyse data produced, to generate, for instance, optimal development configurations, and more importantly, to generate targets for systems, and development methods, to attain.

2.3.2.2. Progress Monitoring

The assessment of the current state of a software development or maintenance project is a very difficult, and hap-hazard task at present, using only subjective assessments. Generating a metric set for the current state of the project, provides an objective way of increasing the visibility of the project, and allows the monitoring of progress being made. In this way early indication can be provided of a project going off target or using poor development methods.

The advantages of an automatically generated metric set provide the ability to raise the visibility of the project state without crippling overheads and in a form managers can understand, particularly managers of the business, not necessarily knowledgeable about the project area. The visibility comes from reports quoting meaningful figures and displaying graphical trends and forecasts - the language of business.

2.3.3. Management

Increased visibility of the process, plus prediction of the future and comparison to the current state accurately provides a basis for good management. The process and so the product can be controlled [CARD87].

The ability to model a software project, at whatever stage of development and produce targets for the project, and the ability to measure, exactly, the actual project's state, is the necessary basis for managing the project much more closely than is at present possible [DRUCKER79]. Deviations of the development from the required goals would be spotted much earlier in the development, making corrective action easier to accomplish and also cheaper [ROOK86].

Project visibility is improved, and therefore there is greater accountability of members of a team or of methods being employed. Metrics could help Managers spot trouble areas, such as critical code, or areas in need of redesign, and they would be given the ability to better judge the effects of corrective action [HUFF86].

Managers would also be able to make decisions about priorities of features in a system and to estimate the effects of concentrating on certain features at the expense of others. Managers would be able to assess new production methods or tools and help a manager answer critical questions such as:

- Is my DP department any good; is it doing its job properly?

2.4. Metrics Applied to Maintenance

With maintenance being such an important part of the software life-cycle, it is important to find methods of reducing the cost of maintenance, perhaps more important than finding new methods of developing software, as existing software is going to be with us for the foreseeable future [COOPER89].

All the uses of metrics described in the previous section apply as much to the maintenance phase of software development as any other phase [HARRISON82], however, the use of metrics for managing maintenance is particularly important. The following section expands on, and details, some uses of a metrics system in the maintenance phase, and explains why such a system is necessary for true maintenance management. More information can be found in [ARNOLD86, BERNS84].

2.4.1. Monitoring

As described above, a major use of software product and process metrics is that of monitoring the state of the product or process, and the change of state with time. In other words, the monitoring of exactly what is happening. In the maintenance field, this visibility is particularly important in three areas.

2.4.1.1. Quality

A large amount of software maintenance activity involves changes being made to small areas of the source code, with very little, or no design work being done before hand. This, although it is undesirable, is often seen as the only way to meet time constraints and complete all the required work in the time allowed. This sort of maintenance activity is very difficult to monitor or control.

Automated software metric measurement provides a method whereby such work can be monitored [ARNOLD86]. A software metric system, applied to the code being altered can provide a first line indication as to whether the change being made is of adequate quality or not. This indication can be used to show up changes that are not adequate, and thus lead on to further review or rework.

This monitoring implies two requirements for the metric system used. Firstly the system should be automated. This is necessary to comply with the time pressures that are forcing the work to be done in this way. A system that represents a significant overhead on the change process is counter productive, as it would be far better to allow more time for the change to be made, and expect adequate analysis and detailed designing to be done.

The second requirement is a definition of quality [KAPOSI87]. A definition must be available so the metric values collected can be compared to what is acceptable in order that a conclusion can be reached. This is a subject in itself, and reference should be made to the earlier discussions of software metrics.

Simple definitions can be constructed, however, but are based on the environment and the work being done.

In this way, a metric system increases the visibility of the maintenance activity being performed on a system, providing an indication of whether the work is of a required standard, even for rush jobs, without implying a large overhead in analysis of the change and comprehensive reviews.

2.4.1.2. System Degradation

A normal feature of maintenance work is the degradation of the system being maintained, usually due to an increase in complexity and a reduction in performance [YAU80]. This degradation is manifested by an increase in the difficulty of working with the system, that is, of doing maintenance work. Eventually the system must be replaced by a new system as maintenance becomes too costly.

The reasons for the degradation are many, but include the facts that a larger number of people work on the system [SCHNEIDEWIND87], over a longer time than in any other phase of development and the time available is much shorter. Thus a maintainer is only interested in a small part of the code - the part to be fixed, or enhanced etc. This leads to a very narrow view of the software as a whole and a poor design of a change in the global system picture leading to unforeseen knock on effects and unplanned changes, accelerating the decline of the system. A large number of changes consist of small patches added to the code to implement a particular correction or enhancement. This work could often better be achieved by redesigning a whole section of code, but the time required is too great.

A metric system, providing a description of the software as a whole and in parts, would allow the plotting of the degradation of the system as a whole and its separate components and provide the necessary information for planning preventive maintenance.

The metric set would also provide the information required for discovering those methods that reduce system degradation, or keep it to a minimum. By identifying various areas that cause increased system decay, such as ripple effect, these factors can be combated and methods developed that reduce those factors. The feedback from the metric system will show if these methods are successful.

Not only does a metrics system providing a system whereby a manager has much closer control over the maintenance work, he also has the ability to assess the effects of the maintenance and therefore is given some criteria for judging '*satisfactory*' maintenance.

For very large systems, metrics could also provide a maintainer with a much wider view of the system as a whole, so the effects of a proposed change could be assessed and a change modified without the need for the maintainer to spend time and effort understanding the whole system or having to rely on the knowledge of others.

The planning of preventive maintenance would be assisted by the ability to identify '*bad*' areas of code, those parts where understanding and alteration will be difficult. The metric system could also help identify unreliable parts, those parts that are most likely to contain most errors and therefore require large amounts of maintenance.

Once identified, these parts can be the target for redesign and rewriting to a better standard, thus reducing the effort required in future maintenance. The metrics could also provide information about what priority should be assigned to the work -- which redesigned parts would have the greatest beneficial effect, and so should be tackled first.

2.4.1.3. Progress

The metrics system allows us to monitor the state of the software system at any particular time, and by examining its change in state over time we have a picture of the time dependent features of the system.

Monitoring metrics over time also allows the monitoring of work being done, such as the amount of work, and the time the work is taking. It also allows the monitoring of how the work is done, and the effects this has on other features of the process and product.

These are important factors. They provide an indication of the progress being made, and the factors that influence that progress. They also provide another major facet of the visibility of the product and the processes involved in maintenance.

All the above features increase the visibility of the maintenance project, allowing its detail and the global picture to be examined and represented in a way comprehensible to managers and people not experts on the system being maintained.

2.4.2. Prediction

The second major use of metrics derived from the current system, is as inputs to models that will predict the future state of the system. This future state can then be assessed in the same way as the current state in terms of its acceptability and resource requirements.

A metrics system is an important prerequisite for this future prediction as it provides a level of abstraction from the real state. This abstraction reduces the amount of information that has to be worked with, and thus reduces the necessary complexity of the model used for the prediction.

A level of subjectivity is also removed that would be present if a metrics system was not used. This is particularly important if a number of possible scenarios are to be investigated, and their outcomes compared. This can be a very inaccurate procedure at the best of times, obviously dependent on the models used, but at least we make some advance by removing a level of subjectivity.

This discussion assumes the presence of models that can describe the maintenance process and the advancing state of the software system. These models, themselves, can only come about if they are based on a metrics system as described above.

2.4.3. Management of Maintenance

The control of maintenance through proper management of the processes involved is a necessity if costs are to be controlled and efficiency maximized [CHAPIN88].

As described above, given the ability to monitor the state of the current project, and its change over time, and the ability to make some predictions about its future state, we have the basis for management of system maintenance [ROOK86].

A metrics system could help with the management of software maintenance in a number of ways, particularly by making it easier to answer some of these important questions:

- Maintain or Redesign? Is it worth trying to maintain this piece of software, or is it better to scrap it and rewrite. Does the whole system need rewriting, or is it necessary only to rewrite parts?

- Priorities? Which bits are most important, or will have the greatest beneficial effect, and so should be done first? Which changes can we postpone because they are not important, will take too long or are being dealt with by another change or rewrite? How long are the changes going to take, and what effects will the work have on the system as a whole?

- How long is it gong to take to complete the current maintenance work, and how much will it cost?

- Once this work is done, how much will it cost to continue to support the product, and how long will it be before this cost becomes excessive?

These are all examples of decisions a manager can make because he can clearly see, and assess, the state of the system being maintained, and can make educated predictions of what the future state, and resource requirements, will be. From this information the manager can make decisions to affect the future state and can monitor the progress of those decisions. Areas of the project that are causing problems can be identified and corrected, and those areas that are satisfactory can be expanded upon and learnt from [LIENTZ80].

This is management, and is the way forward for maintenance practice. The requirement is, as has been shown, a metrics and measurement system that provides the information required.

2.5. Metrics for Maintenance Management

Here, we introduce a new classification of software metrics. This classification is based on the level of abstraction of the software to which the metric relates. Three classification classes are defined:

- *Code metrics.*
- *Configuration metrics.*
- *Others.*

These are described below.

2.5.1. Code Metrics

Code metrics are related directly to the source code, or text, of the software in question. The source code is the input to the collection algorithm and the output is a measure of a feature of the source code of the system.

These metrics represent the base level of the metric hierarchy, and are in general the direct metrics on which other metrics can be based. As they are, they are useful only to those dealing directly with the code, i.e., programmers etc. They represent no abstraction from the actual tasks of doing maintenance.

These are, however, the base from which other metrics can be built. There are infinite numbers of possible metrics that could be measured. This thesis does not attempt to specify those that should be measured, but attempts to provide a method whereby the requirements for a metric set can be specified. The satisfying of this requirement, is dependent on the specific environment and work practices.

2.5.2. Configuration Metrics

These metrics refer to the software configuration, and represent a level of abstraction from the source code of the system. The configuration of the software is the collection of units that make up the system, along with their relationships. The units are any parts that make up the system, for example, modules, or code files.

This level of abstraction is important for a number of reasons. Firstly, it removed the language dependent features that are common among code metrics. Most languages divide a system into units of some form, thus

configuration metrics are applicable and general, whereas specific code metrics would be required for each language on which they are used. An example of such a configuration metric is number of errors per unit. This measure is independent of language or system, as its basis is a unit, whatever that may be.

Configuration metrics also represent a move up the metric set hierarchy, away form direct metrics and into the indirect, derived metrics. These two abstractions reduce the amount of data involved in a system description. This is especially important for line management. The manager is provided with information that has far less granularity that with code metrics, and therefore is easier to understand and use.

Research in this area of metrics is far less common. One of the major reasons for this is that the metrics that will be useful depend on the requirements for the metrics and the environment in which they are to be used. This is a major problem, as it means that research into this area cannot easily be driven by research interest alone, but must be driven by specific requirements of a user.

2.5.3. Other Metrics

The last classification, that of '*other*', includes the rest of proposed metrics that apply to higher levels of abstraction. These require the combination of other metrics and the abstraction of information. Again, those that will be useful depend on the requirements of the specific environment, and the feasibility of data collection and analysis. These metrics will be specifically useful to management, particularly higher management.

2.6. Summary

Metrics research is an important area if we are to generate a true understanding of software, its development processes and those of maintenance [SCHAEFER85]. Major advances have been made in the measurement of the fundamental features of software, but there is still along way to go.

A large amount of work has been addressed at the measurement of features of the code or other elements of the software at a low level. Most of this, however, lacks a real statement of the reasons why the measurement of the feature is important, and the detailed meaning of the metric value once it is derived. This lack of context and specification of measures makes it very difficult to fit them into an overall picture of measurement of the software and the processes involved in its production.

The above discussion fits the research into this global picture of what is required from metrics research. A particular area identified in which measurement is a necessity is that of management. To bring a system, and the processes acting on it, under proper control there must be the ability to monitor what is going on, and predict what will happen in the future. For this, measurement and metrication are a requirement. This is an area, however, that has been lacking in the research to date.

Maintenance activity is, generally, an ad hoc process completed under heavy time constraints and lacking control and planning. This therefore, is an area that requires the application of measurement to allow it to be brought under management control [CHAPIN88, ROMBACH89]. There is, however, a lack of a practical approach that will allow a maintenance organization to introduce a measurement system that will help manage and, therefore, control the

maintenance activity. This thesis presents such a method, with an example of how it can be achieved.

Measurement is the only way to introduce proper control over all phases of the software life-cycle, and the only way to gain true understanding of the products and processes involved [GRADY87]. As such it is, therefore, and important area for research and investment by all areas of the software industry and academia.

Chapter 3

A Method for Developing a Measurement System

This chapter presents a method for developing a data collection and analysis system. This system is primarily aimed at helping the management of a maintenance environment at the line level, but will be shown to have further reaching implications. The resulting system is specific to the environment in which it is developed, and therefore satisfies the requirements of that particular environment. The method allows for minimum impact of the system development and the system itself on the tasks being performed, while maximizing its usefulness. The method also allows the measurement system to evolve as the understanding of the environment increases and the requirements for the system alter. This also allows benefits and costs, in terms of money and effort, to be assessed and related decisions to be taken prior to major commitments of these resources.

3.1. Data Collection

A measurement system has two basic stages. The first is the collection of raw data from observations within the environment in question. The second is the analysis and use of the data.

3.1.1. Why Collect Data?

Chapter 2 presents a detailed review of why data collection is necessary in any environment if that environment is to be understood and controlled. There follows a brief summary of those reasons which have relevance to the maintenance management field.

- System and state description.

  A set of relevant data allows the description of the system or current state in an easy and objective form. The data set chosen reflects the information required about the system or state, and can exclude information that is not required. In this way the description is both objective and concise, representing a level of abstraction away from detailed or subjective system knowledge.

  The description is easily recorded, communicated and understood, provided the data set is understood. The change in the description over time, represents the progress being made in the domain of the data making up the set. In this way, it forms the basis of a useful reporting function.

- Management

  In order for maintenance to be brought under strict management control the visibility of the products and processes involved has to be increased to a level where managers can see what is going on [CARD87]. This, in any but the most limited of cases, requires measurement [KITCHENHAM84].

Measurement supplies the following tools to the maintenance manager:

- Progress Reporting

  The first stage of being able to manage a process, such as software maintenance, is to be able to accurately assess what the current state is, and how it is changing [GRADY87a].

  In general, there are two ways to achieve this visibility. The first is by experience, that is, detailed knowledge of what is going on from experience of doing the 'hands-on' job. This is a wide-spread method, but is time consuming, generally inaccurate and very subjective.

  The second is the measurement of the system and environment as described above. This provides quicker, objective readings of the situation in an abstracted way. This is the route of real management.

- Target Assessment

  This is the next important stage of management for which information is required. The current state is known, but knowledge about the probable future states is also required, so that planning and resource management can take place.

This, at present, generally relies on experience. But this experience is just a knowledge of past states and progress, to be extrapolated to the future. If this knowledge is 'woolly' and based on subjective and perhaps inaccurate assessment of the situation, the forecast is, at best, going to be 'woolly' and inaccurate. Objective, accurate past experience, in a documented form that can be reviewed and even graphed provides a much better basis for the application of experienced forecasting.

An enhancement of this process is the development of models that encapsulate that experience in order to predict future values. These models, however, can only work on explicit, quantified data and produce the same as results.

- Communication

  Another important weapon in the managers arsenal is the ability to communicate about his or her responsibility area.

  Once plans and resource requirements are formed, and knowledge about the environment is collected, this must be communicated and justified to those higher up the corporate tree. Objective, abstracted data is the only way toward this end, and is therefore a requirement.

- Store of Experience

  Data collected about a system or environment can be stored for future reference.

  A data set forms a reasonable experience store because it is an abstracted representation of the system or environment, hopefully containing details of the important characteristics of the system or environment, without the needless ones. The data set is also objective, therefore its meaning and terms of reference are available in the future. Data that is subjective could well lose relevance, as the terms of reference and conditions of collection cannot be specified.

  This store of experience is particularly important, as stated before, for the prediction of the future.

- Post-Maintenance Analysis

  This follows from the store of experience. Experience and knowledge is enhanced by the retrospective review of events and conditions. This gives insight into the related environment and the factors that effect it. This will improve the ability to predict the future, as well as isolate those factors that have negative influence and remove them, and those that have positive influence in order to enhance them.

- Research

  An important area if progress is to be made into the understanding of the maintenance process. The progress that can be made is limited, however, on a purely theoretical front,

without links and input from the real world. This is important on two main fronts:

- Validate models

  Models and theories need validation if they are to be accepted and used. This requires the collection of data in the real world if the model or theory is going to engender any confidence.

  As Basili notes, actual data is required for validation of models and also the generation of new models, and without the collection of data not only are models unprovable, they are also worthless [BASILI84].

- Generation of models

  Taking the experience of other sciences, it can generally be seen that, excluding the occasional notable exception, most progress of understanding happens by the careful and pains-taking study of observations to generate the theories and models that constitute the understanding of the science. The progress by pure hypothesis is, historically, limited.

  This, therefore, suggests that if advances are to be made in software science, it must be based on real observations. This required measurement.

- Visibility

  In summary, if software maintenance and its management are to be improved, we must increase the visibility of the products and processes involved. This requires measurement.

3.1.2. Collecting Data

The need for collection of data has been established. The introduction of a data collection strategy, however, poses some problems. The first is the method by which data will be collected. A number of possible strategies exist. There follows a discussion of these and a discussion of there general applicability.

3.1.2.1. By experiment

A collection method often supported is that of the carefully controlled experiment. A test project is devised and a set of data to be collected is decided upon. The project then goes ahead with the defined data being collected. Once the project is finished, the data is analyzed and conclusions drawn about the usefulness of the data that has been collected. The data produced by such an experiment is useful for designing collection systems for the future, and provides part of a store of experience that can be referred back to. The shortfalls of this type of experiment are, however, numerous.

Firstly, experiments of this type must normally be small so that the time between inception and the results and conclusions is reasonable. In a normal software engineering environment, such small experiments are not representative of the normal large scale work being carried out. Secondly, the cost in time and effort of performing these experiments would normally be

prohibitive in a commercial environment where schedules are tight, and resources are short. Thirdly, there is no feedback into the management cycle in this sort of experiment. Shortfalls in, and the benefits of, the data collected can be ascertained when the analysis is done, but, as this analysis is performed at the end of the experiment, it is not useful for the management of the project which must be an on going task. It is very difficult to decide what data would have been useful had it been available at the time.

In general, this is a very useful strategy for initial validation of ideas, as it can be conducted in a controlled way. It is, however, not of use to the industrial environment, for the management of projects or for the real world validation of ideas.

3.1.2.2. After the event

Data can be collected after the actual work has finished. This is done by such techniques as interviewing participants and filling in forms and the analysis of the finished product and the by-products of the exercise of interest. This is a method often used to get a feel for what happened during the project.

A lot of information is lost or unobtainable using this method, and the reliability of the data collected may well be suspect. Validation of collected data is virtually impossible. In general it is only possible to get the snap-shot view of the end of the project. This again, although it may be useful as a store of experience and for research, does not provide any facilities to help management.

What this method does provide is useful information about what data it might be useful to collect in the future. It provides pointers to those facets that is unnecessary to collect and those that may be useful. The time spent on this exercise after the work is completed is ,however, difficult to justify in a commercial environment, as the results may not be applicable in future work.

3.1.2.3. During the project

Data collection procedures can be incorporated into an actual project. A method of data collection is devised, and a definitions of the data to be collected. This is often in the form of collection forms, either paper or machine based, that must be filled in by the participants in the project.

This method has a number of advantages. The data being collected is relevant, because it is being collected about a real project. The data is available as the project progresses. This means it can be validated easily. The data can be analysed before the end of the project giving the necessary feedback into the data collection system, allowing the collection system to evolve into a form that is most useful. The availability of the data allows feedback into the management cycle, allowing problems to be seen and corrective action taken etc., in fact the increase in visibility required for a project to be managed correctly.

This, therefore, can generate a useful system of data collection for both research and project management. The major problem with the system is the overhead in the collection of the data, such as the filling in of forms. This is the kind of operation that will be ignored if time pressures become too great or resources get too scarce. If the data set becomes incomplete, the information it provides ceases to be useful so the system falls into disrepair.

3.1.2.4. Collection as a background task

This is in all senses the best method. Data is collected automatically as a background task to the ordinary tasks in the project. This is becoming more feasible with the advent of Integrated Support Environments and software tools to support software engineering projects. The important feature is that a machine based tool or system collects information about what is going on as it is being used. The tool or system must therefore be seen to be useful, or even indispensable [KITCHENHAM86], so that it is used.

This method incorporates the advantages of the previously defined methods. The overhead of the data collection is negligible, as the system is being used to help the tasks being performed, the data collection is automatic. The data collected is valid as it is collected automatically, and the data set remains complete. This all relies on the system being used at all times and having access to the information required. This can limit the information available to the collection system, but in general, if the system is designed carefully, it can have access to all the required data.

It is worth noting that users of the system should in general be aware of what data the system is collecting about them, not only does this lead to improved work, but it can overcome problems produced if people think they are being spied upon.

The second form of background collection, strongly linked to the above, is to make the collection system part of the necessary working practices and procedures in use in the environment in question. This means that instead of

being another task that has to be completed, such as, 'fill out a report after the change has been finished and incorporated into the system', the data collection task should be part of the one task, i.e., 'once the change has been completed it is signed off by completing the report and returning it to change control'.

In this way, data collection cannot be avoided if time becomes too short, or become incomplete because of laxity, because this would be indicated by a complete breakdown of the working practices.

The conclusion from this discussion is that data collection should be a background task wherever possible as this form of data collection proves the most useful to the management of the environment, and also the least prone to the problems symptomatic of other collection strategies. This collection process should also be automatic wherever possible to enhance the monitoring potential and minimize the scope for errors.

3.1.3. The Cost of Collecting Data

Data collection has a cost in time and effort and perhaps in systems to support the collection. This cost has to be weighed against the benefits described above, but are very difficult to quantify at the outset. A general requirement, therefore, particularly in the industrial maintenance environment is to minimize this cost overall, but more importantly, to minimize the initial investment that has to be made before any results can be assessed, and also to spread the investment as much as possible so cost/benefit decisions can be made from a position of knowledge.

For these reasons it is important to understand where these costs come from, so we can minimize, or avoid those costs.

Data collection requires planning an effort to achieve. The planning stage must define what information is to be collected and devise a strategy for its collection. The collection itself involves overheads on the processes that are being measured, either in machine time and system costs for data that is to be collected automatically, or people effort for the data that requires their input.

The optimum situation would allow the collection of all relevant data, automatically, but this cannot be achieved without a cost. The initial investment in planning, i.e., identifying the data to collect and its collection strategy, will reduce the impact of the collection on the environment. However, too much planning effort contradicts our main requirement of low initial investment.

As described earlier, different types of data are apparent, with different costs associated with them. This forms the basis of the method to be presented here, and the identification of the different types forms the core of the planning process. The data types are as follows:

- *Available data*: Certain data can be identified as essential to the processes themselves. This data is collected as a matter of procedure, in order for the procedures to function. Examples are, requirement sheets that are dated as they are received, test result documents, code walk-through results, etc. This kind of data may be filed, or often destroyed, but forms a major source of information about the processes involved. If this data can be captured and stored it can form the basis of a data collection system. This type of

data has almost no effort overhead, and very little cost associated with it.

- *Attainable data*: The second type of data is data that can be identified as useful, and which could be collected as routine with a small change in procedure. The overhead here is the change in procedure, and that has to be weighed against the value of the information. Certain data, however, can be made available with very little effort. An example is, requiring that all documents are signed and dated to provide information about who and when.

- *Collateral data*: Some information is available in an indirect form, and requires a certain amount of effort or investment to retrieve. Examples are, compile time for a changed module, modules changed, or size of a new release. For this type of data, most of the effort involved in its production has been completed already, as a background task, it is just a matter of the final collection. Again, however, a decision must be made about the relative value to cost, although this can provide some of the most valuable information.

- *Other*: Certain data requires more overhead to collect, such as, thought time spent to complete a change. A large amount of useful data can be identified that falls into this category, however, it is difficult to assess the cost/benefit relation, so care should be taken.

- *Inaccurate data*: This kind of data is data that is effected by the collection process. It is important to recognize this kind of data as it could lose its value by its collection. It is particularly person/performance data that falls into this category. From a

management point of view, some collection could improve the performance of the people involved, but it may also have detrimental effects. People management issues are, however, outside the scope of this thesis.

3.2. Data Analysis

Data analysis, as used here, is the process of taking the raw, collected data and transforming it in various ways, combining it with other data and presenting it in order to produce useful results.

3.2.1. Why Analyze Data?

Data analysis is required for a number of reasons including:

- Presenting data in a more easily understandable or demonstrative form, e.g., a graph of number of changes over time to show work loads.

- Reducing the amount of data to be presented by combining multiple data elements into a single result, e.g., taking a measure of the number of changes to each module in a system and combining them to produce a measure of the total number of changes to the entire system - this is a simple, intuitive, combination exercise.

- Taking data elements that have meaning in the collection domain and mapping them onto data elements that have meaning in the domain in which they are to be presented, e.g. the mapping of the

total number of changes outstanding on a system onto the expected man-effort required over the next month.

- Applying models to the collected data in order to produce new data such as expected values of measures, e.g., taking the total number of changes implemented over the last year, and using the data to predict the total number of changed expected next year, and so on to the expected cost for next year.

3.2.2. Analyzing Data

Data analysis can be divided into a number of types, with varying complexities and levels of knowledge and understanding required.

3.2.2.1. Presentation

Once the final values, or results have been derived it is necessary to present these values. There are many ways of presenting a set of quantified values from graphs to tables, etc. Data presentation is generally well understood, particularly in the management area. The important requirement is that the data is presented in a way in which it is useful to its audience.

This presentation is very important in the area of management. Not only must the results be of use to the project manager in order to feed back into his project, but they must also be of a form that can be communicated up the hierarchy of management. These different audiences require different presentation methods. In higher management the recipients of the results may well not be technically knowledgeable in the area of software maintenance, so the results produced, and their presentation method must be in the language of

business. A prime implication of this is not that the analysis must become highly complicated, but that, as the language of business is cost, cost projection and summaries of trends, the results produced must be able to be presented in these ways - summaries of trends and predictions of the future presented in simple to understand forms.

So, to produce a system where measurement is useful for the management of software maintenance we must tailor our data analysis and presentation to the goals and requirements of the audience.

3.2.2.2. Abstraction

The actual collection of data forms the first stage of abstraction from the problem domain. This happens in two ways. Firstly, the representation of features or characteristics of the software as measurements reduces the quantity of information that must be assimilated - now a value instead of a part or all of the code. Secondly, the representation as a measure reduces the number of assumptions about the underlying knowledge of the problem area. This is true because a value, with its associated scale, that is the units of measure and the meaning of the measure, forms an intrinsically more encapsulated representation of the feature or characteristic than the original software. In other words, an understanding of the metric does not require a detailed understanding of the software itself. In most cases, and unless a truly complex metric has been chosen, this understanding of the metric is easier than the understanding required of the software itself.

This abstraction of both quantity of data, and the knowledge base required for its understanding are the base reason for the collection of metrics in the first

place. This is, however, only the first stage of abstraction. This first stage may be enough for our purposes. If it is not, further abstraction takes place by data analysis.

Data analysis can provide data abstraction in a number of ways.

- Data Combination

  The main method of data abstraction is the combination of measures into single results. If a single metric is a measure of a certain feature of the software, the combination is a measure of a group of features of the software. In this way the number of measures that are required to describe a set of features of the software is reduced.

  This is a very useful method of producing a small set of data that provides the information we require, however, great care must be taken when adopting this procedure. The first problem to be wary of is the increase in the domain knowledge required as the metrics are combined. A metric combination of two features represents those two features, so knowledge of the two is required to understand the metric produced.

  The second factor to bear in mind is that metrics, as scalar representations of values, follow the laws of mathematics as applies to numeric values. This is something that is often forgotten by people developing metric based measurement systems. The rule that is most often forgotten is that of the combination of units of a measure. Two values with differing units of measures that are combined by a mathematic operation

will produce a result having units that are a similar combination of the units of the initial values. This is particularly important when there exists a situation where the combination of units is meaningless. In this case the result of the combination is also meaningless. An example of this is the addition of the number of lines of code in a system to the length of time it has taken to write. The result has units of (number of lines) plus (time) which is meaningless.

It is, of course, not necessarily wrong to do this operation, as long as its implications are understood. The basis of measurement, here, is to measure defined things. That is, to measure things that have defined units of measure, and whose method of measurement is well understood. If a combination occurs, such as that above, that produces a meaningless unit of measure, it now becomes a result with undefined units and a theoretically undefined meaning. It is just a figure. If this happens, the meanings of the values and their combinations is lost and therefore can produce a situation, very quickly, in which all the results have no defined meaning. This is not a recommended situation. In general, it is better to maintain the meanings of variables and their definitions. For this reason the observation of the rule of combination of units of measure is to be recommended..

- Data Transformation

  Data transformation, the changing of data using transformation rules, provides data abstraction by changing the knowledge base required to understand the metric. An example is the

transformation of number of lines of code into cost of producing the software, by whatever rules are accepted in the particular environment. The knowledge base for the initial metric is that of a programmer, in terms of what is meant by a line of code, how is it measured etc. The knowledge base of the second is that of business - just the cost - no technical knowledge about programming languages, etc. is required.

This method of abstraction of data usually happens in conjunction with the combination of measures described above. In fact it is very difficult to do one without the other. It is also worth noting that the application of one method may adversely effect the other. The transformation of data into a more easily understood form may necessitate the increase in the quantity of data. The advantage and disadvantage of this process is however something that varies with individual requirements and environments and so cannot be discussed here.

3.2.2.3. Translation

This result of data analysis goes hand in hand with the previous section. Data translation is the application of transformations or combinations in order to change the meaning of the data. This does not necessarily involve abstraction of data, just its representation in a new form.

This type of data analysis is important when the results of analysis must be used by a variety of end users. Programmers need results specific to their domain, that of programming and language specific information. Managers need more

technically independent information. This requires translation to the management domain as well as being abstraction as described earlier. Higher management need an even greater level of abstraction and an even more technically independent representation.

Although the necessity and definition of these different result areas are not too complicated, the specific translation requirements can be difficult to define and implement. It is in this area that research is needed to provide the methods for this translation. Some steps along the way can however be made without too much effort.

3.2.2.4. Prediction

Prediction based on measured values involves the application of models to the measured data to produce predictions of future values. These future values can be fed into the various translation and abstraction methods to produce information about the expected state of the project or software at a point in the future.

This production of expected values is at the heart of good management. If these expected values are unacceptable we can make changes now to avoid that future, unacceptable state. We can also measure the current state against what was expected, to discover if we are on course, or if things are going wrong. We can plan for the future, forecast budgets and resource requirements and ensure the smooth running of the project. These basic principles are the cornerstone of management and business.

It is this prediction that is particularly hard. The models required to provide enough detail are normally complex and have a large number of variables. As has been described before, these models can only be found if we start collecting the information required and empirically looking for the models. In this way we can expect to find the underlying models in our environment, and be able to do the prediction we require.

There are, however, things we can do. The ability to predict values is heavily dependent on experience and past values. The first stage is therefore to collect this information and produce the results we require from it. From these past values, extrapolation can produce future values to a certain level of accuracy. This level of accuracy is often enough for management to do its job. The conclusion from this is that the collection and analysis described earlier provide the ammunition we need to provide estimates of future conditions. This demonstrates the necessity for evolution of the collection and analysis process, as we learn from experience. It also enforces the view that, at least initially, these rules are going to be very environment specific, and require the knowledge of those involved in the environment.

3.2.2.5. Visibility

The above sections lead on to this important conclusion. The analysis discussed provides the visibility of product and process that a manager requires, in a form the manager requires. In this way it can lead to better management.

3.3. Development of a Measurement System

This section present a method for introducing a measurement system into a maintenance environment, based on the preceding discussions.

3.3.1. Requirements of System

The measurement system produced by the method must satisfy a number of requirements. These are:

- The measurement system is entirely dependent upon the environment, and is therefore sensitive to specific environment characteristics and motivations. The system, therefore, also satisfies all the requirements for the system within the environment in which it works.

- The measurement system is entirely applicable to the environment in which it is developed. That is, the results are those that are required in the environment, and the data collection is completely in harmony with the processes of the environment.

- The system must be usable. That is, the data collection must not be in conflict with processes in the environment, the data collected must be complete and valid, and therefore collected during the project it is measuring. The system should produce results that are useful to the management role within the environment.

- The initial investment in planning and implementation must be small, and the impact on the tasks in the environment must be minimal.

- The system should be capable of evolving as requirements for the system change and knowledge grows. This evolution potential will also allow the initial implementation to be limited in impact, but the potential system to be whatever is required as the benefits of measurement are seen.

3.3.2. Development Method

The method is a four phase approach to system inception and implementation.

3.3.2.1. Phase 1 - Initial Data Set

Phase one is the definition of the initial set of data that will be collected.

The initial set should be seen only as a first guess at that data that will be useful. The larger the set of data that is collected, the better. The more data there is to work with, the greater the chances of finding an optimum set of useful data quickly. This has to be weighed against the overheads in its collection and analysis.

The types of data, as listed above, available, attainable, collateral and other, are the starting point for the definition of this initial set. The data in each of these classes that is present within the environment, or would be liked, should be identified. From this set, the relative costs of the data items can be assessed.

In order to keep overheads at a minimum, we are looking for an automated method of data collection, and one that fits in to the working practices of our

environment. This implies special interest in available, attainable and collateral data within the environment. A good initial set can be produced by looking at the various tools and methods that are in common use during projects. The information that these require, collect and produce can form a good starting data set, with little overhead.

The set thus produced can be enhanced by looking at possible changes to procedures, or the implementation of procedures and the requirements for new tools to help in the tasks involved in a project. These procedures and tools may be worth investing in if they have the joint advantages of assisting in the work of the project, or reducing that work or other such advantage, along with producing useful data about the project.

The set can further be enhanced with data that it is intuitively felt will provide useful information. With data of this type it is important that the collection method be considered in order that the overheads of producing the information do not out weigh its benefits. This is an important point, at this stage we do not know the benefits of the collected data so we cannot make intelligent cost/benefit decision. For this reason we must ensure that costs remain at a minimum.

Special care must be taken to exclude data that cannot be validated, cannot be collected accurately, or cannot represent the true situation.

In summary, the initial set of data is best defined from that data that is already present, i.e. already collected although not used, in addition to any data that can be collected by automated tools that either will not impose undue overhead on the project, or the overhead can be laid off against other advantages. In

other words, the available data, with supplements from the attainable and collateral data for which the cost of collection is not too high.

This set will change with time, so not too much effort should be spend in the initial definition.

3.3.2.2. Phase 2 - Collection Strategy

Phase 2 is the definition of the method of data collection. Two important facets have to be considered. Firstly, the source of the data, and how it is to be collected into a single, usable repository. This will normally need to be machine based to allow easy manipulation of the data.

As the initial set of data has probably been defined based on procedure elements that already create data and on tools that support the project, the initial stage of the collection method is also defined - the tools themselves and the procedural elements. The collation of the data from, probably, a number of sources into a single repository is more complicated.

The second, and connected consideration is the instigation of procedures to ensure a complete set of data, or as complete as is practicable, is collected. It is, in general, no use collecting a data set from a tool that is only occasionally used by only a few members of a team and thus will not produce a representative data sample. We must install procedures so that the collection tools are used consistently and regularly. The same applies to procedural data. We must also ensure that this data is faithfully entered into the repository of information.

This validation requirement will generally require only small alterations to procedures that should be in place already. It is good practice, and certainly the first stage to decent control of a project to make sure working procedures are in force. These procedures can be enhanced to make the use of the data collection systems obligatory, and to install some form of validation at the time of collection.

This phase has feedback to phase one, as the problem of overheads is considered again. Any data collection that cannot easily be incorporated into the working procedures, or cannot be validated without causing unacceptable increases in the work overhead have to be reconsidered.

In summary, it has to be decided how the initial data set will be collected, and install procedures to ensure the data collected is representative of the general case, not just specific instances, and the data collected is true and valid. This in an important phase as all the results of the measurement system rely on these assumptions.

3.3.2.3. Phase 3 - Collection and Analysis

The first stage of this phase is the collection of data, and beginning to populate the repository. This begins as a test run for the data set and the data collection strategy causing feedback to the previous two phases. At this stage, great care should be taken to validate the data collected in order to validate the strategy. This is the foundation for the system, and problems here will be difficult to rectify later.

The second stage is that of use of the data, i.e. analysis. We now need to define a system of analysis that will help increase the visibility of the processes involved, help to bring them under control and proper management, and provide a method of presenting the results.

At this stage, very few analysis avenues are open, so the analysis should remain simple, and therefore require minimal effort. More complicated analysis will proceed from greater knowledge as the project progresses and greater insight becomes available.

The first step in the application of data collected is its presentation. There are many different ways of presenting data, and the choice of method and its use has a great influence on the utility of the data, and the course the collection and analysis will follow. The areas of application of the data must be identified and what use it is meant to be put to in order to decide on the method of presentation.

The area of application of particular interest is that of management. One of the objectives of the system is to be able to communicate the data to higher levels of management. The only choice in this environment is to present the data in the language of business. The language of business in this respect is summary reports, and graphical presentation methods such as graphs and diagrams.

Another objective of our analysis is to drive the development of the collection and analysis system. In this respect, the best method of trend analysis and the identification of relationships is by human eye, and particularly with respect to graphical representation of information. With these two arguments in mind, it is an obvious first step to use graphical representation of data, and summary reports.

This is the starting point of our analysis. The presentation of our collected data, or summaries thereof, in a graphical way particularly with a time baseline. This gives us information about flows in the system and so the processes going on, and lets us identify problems with the flows. It also gives us a picture of how things vary over a longer time scale, and using intelligence and intuition, the ability to predict future values. This is the first stage of seeing what is going on, managing what is going on, and predicting the future.

A use of the collected data that should not be overlooked, is the possible benefit to the actual tasks involved in the environment. Data is the basis of change control and configuration management and also represents the store of experience described earlier. This store may have potential uses for the technical people in the environment, so reducing the effort required on their behalf, and offsetting the overheads imposed by the measurement system.

3.3.2.4. Phase 4 - Evolution

This phase is probably the most important, and, like maintenance in software development, is never complete and encompasses the other three phases. In this phase we alter the data set collected, the collection method and the analysis applied as knowledge and experience increase and our requirements and expectations change.

One of the obvious drivers of this evolution is the results of the analysis of the data and the requirements for these results. There are other drivers also. There must be a constant re-assessment of the position with regard to the data collected and the method of its collection. This assessment must encompass several areas. The first is the availability of data.

As the environment changes and the procedures used and the tools employed to do the jobs change, the opportunity to collect data also changes. There must be an ongoing assessment of what new data is available that can be added to the collection set. This should also become a major consideration in the adoption of new procedures or tools - what data will they provide?

The data currently collect must also be assessed. How useful is the data? What use is made of the data? How much overhead is involved in its collection? Using questions like these, data must be identified that should be removed from the collection set, because it is to costly to collect, or is not of much use. This is where the collection of a large data set to start with provides a pay back. At this stage the data can be judged in terms of its benefit - that is, what use is made of it, and how useful is it - and its cost - what effort is involved in its collection, analysis (and storage). It is now possible to make cost/benefit decision, which were not possible before. As an aside, the potential usefulness of the data in the future must be taken into account, as both a store of experience and a project history for research and analysis.

The collection method can also be the subject of scrutiny. Are the procedures working? Is the data collected representative of the project in general? Is the data collected valid? These questions and others must be asked, and corrective action taken if necessary. We can also look at ways of applying the methods to a wider area, for example, other projects, or other tasks.

All the time during this evolution we are returning to the previous phases to ensure that the measurement system remains consistent.

In this way, the data collected and the method of collection are changed. The data set starts targeting the useful areas of data collection, with surplus data being removed from the set. The data set also increases to encompass new areas so that a wider picture can be built up and a greater understanding and better control are produced. The collection method is adapted to produce a system whose benefits outweigh the cost, and whose results are useful and correct. The data collection system migrates until we have a usable and useful system allowing the instigation of proper control and accountability.

The data analysis and presentation must also evolve as the knowledge about the system and the requirements for the measurement system grow, and the set of data collected evolves. This process of evolution of the data analysis is very closely linked with the evolution of the data collection system. In fact it is the evolution of the analysis that will drive the change in what data is collected.

The starting point is just a method of presenting the data we collect. This will quickly, however, suggest new presentation methods and methods of both combination and transformation of data to produce new results and methods of presenting the data. This evolution of the analysis and presentation is primarily driven by the intelligence and knowledge gained of the managers that use the data.

As the data is used, ideas will be generated as to new methods of presentation and analysis that could be applied. The methods of analysis and presentation undergo a constant appraisal of their worth, simply by being used. Those that are not useful, or do not provide useful information can be dropped from the repertoire as new methods are produced to take their place.

At this stage we can focus on the model of the software maintenance process that we chose to adopt. The Foster model, as described in chapter 1 defines the tasks and process flows within a maintenance environment. The tasks and process flows can bow be seen as targets for quantifying and reporting on the features the maintenance process. At this stage, reports on various of the features should be possible from the data collected. This may require combination and/or transformation of collected data, but some will be available. There will, however, also be those parts that cannot be quantified. This brings us back to the evolution of the data set collected.

The requirement to quantify features of the model that are not already possible, will define, or suggest new information that should be collected. The information can be defined based on the knowledge so far accrued about the system being measured, and the data already collected. Again, this data collection has to be weighed against the overheads of the collection, but knowledge will also be being gained to help make decisions of this nature.

3.3.3. Summary

The data collection system and the analysis system will evolve form an initial 'guess' toward a system that is useful to managers and provides information to higher levels. It is only by starting this data collection and analysis that its true benefits can be perceived, and the path toward a truly useful system can be mapped. A method has been presented here that will form a starting point for a useful collection system, and one who's benefits will out way the costs.

No turn-key measurement system can be presented that will satisfy all environments and conditions, but by the application of this method, a useful

and applicable system can be developed specifically attuned to the environment concerned. But it is only by applying the method that its benefit can be demonstrated.

Chapter 4

A Practical Application of the Method

4.1. British Telecom Project Data

In order to provide a practical platform for the research contained here, British Telecom Research Laboratories provided a database of information, about maintenance tasks, that had been collected by a research team while maintaining an on-line system for British Telecom. Although the data itself cannot be reproduced here, for commercial reasons, it is used as the basis for applying the measurement system development method to asses its practicality and worth.

4.1.1. The British Telecom Project

British Telecom run a comprehensive research facility at Martlesham Heath, Ipswich, England. At this facility there are research teams looking into every aspect of British Telecom's business including many areas of software development. One such team is actively researching the area of software maintenance. The team have examined various aspects of software maintenance over a period of more than 10 years.

The team has consisted of about 8 people on average, but the size has varied over its lifetime, with a reasonable turnover of personnel. To help in the study of maintenance activities, the team took on the support of an on-line telephone exchange control system, called the UXD5. Over a 9 year period, and two major releases of the software, UXD5A and UXD5B, the team have been responsible for all aspects of software maintenance from initial acceptance of a change request to the release of new software versions. A number of other parts of British Telecom acted as the 'customer', generating the requests for change and accepting and distributing new releases.

For the duration of the project, which has now finished, the team undertook research projects as well as the actual maintenance work. These projects were based on the maintenance tasks being completed, and many involved the collection of information about the jobs being done. As a result of this project, a database of information was created about various aspects of the tasks involved and the software system itself.

4.1.1.1. The UXD5B Project

The UXD5 is a self contained telephone exchange designed to serve remote areas. It caters for up to six hundred lines, and due to its location, is completely automatic. The onboard code consists of a total of about 300,000 lines of source code in languages including Assembler, PLM, CORAL 66 and KINDRA. KINDRA is a British Telecom in-house language based on a graphical representation of control flow.

The UXD5 has been through two major versions while under the control of the maintenance team. The first, the UXD5A, was maintained over a period of

about three and a half years. It was during this project that a bug reporting system was developed and adopted.

The second version, being both much larger and with greater functionality than the previous version was the UXD5B. This was maintained over a period of about five years. The bug reporting system was in place when this project began, so a complete set of data about all changes made to the system has been collected. This is the data that was made available to this research project. The discussion that follows, therefore, refers specifically to the UXD5B data.

4.1.1.2. The Change Procedure

From its instigation, the UXD5B maintenance project had a rigorous change procedure that was followed. This procedure changed slightly over the duration of the project as knowledge was gained, but its major attributes remained the same. This procedure, itself, was a research project. It was tried and tested with a view to producing a standard procedure for all maintenance activity.

The change procedure conforms very closely to the Foster model of software maintenance, and proceeded as follows.

Requests for both fixes to problems (bugs) and enhancements to the code were generated from a 'customer' - another area of British Telecom. These requests were based on both field testing of the supplied software, and actual field use of the telephone exchange. These requests would be received by one of the engineers in the team, who would record the details of the request and acknowledge the receipt of the request to the 'customer'. The request would be assigned a default priority, unless otherwise specified by the 'customer'.

The software system ran an independent, automatic exchange, designed to service remote areas of the country. This meant that when a new version of the software was released, it required to be installed at numerous remote sites around the country. This was an expensive exercise. As a result, the changes designed for the system were collected as they were produced, but were not incorporated into the final system until it was considered cost effective to produce a new release. At this stage many changes were implemented and a new release produced and distributed.

In terms of priorities, this in general produced two categories. Those that could wait for a new release, although the actual timing of a new release could be influenced by the need for the changes that were waiting to be implemented, and those that were critical and had to be implemented immediately. These would be completed and a new release generated. The priority of the change would therefore be primarily defined by the customer.

Once the request was received, and reached the front of any queue of waiting requests, it would be assigned to an engineer for analysis. The assigned engineer would then become responsible for that change, through to completion. The first task of the engineer was to assess the request. If the request was a problem report, he would first attempt to reproduce the problem on a test rig, with test software. The objective was to ascertain whether the problem was a problem in the software, or was a hardware or other system problem. The later cases would generate a reply to the 'customer' describing the reasons why the request could not be acted on by this team. If the change was a software problem, the engineer would estimate what the scope of the corrective action was likely to be. In the case of an enhancement, the scope of

the required change would be assessed. Against this the benefits of the change would decide whether the change went ahead or not.

Assuming the change was thought necessary, the engineer would design and code the required change, using a copy of the software specially designated for change testing. An important facet of this stage was that the design and the code changes were fully documented - a fact referred to in more detail later. The engineer would test the change on the test rig.

At this stage the changes would be reviewed by a Quality Review Panel to ensure everything that should be taken into account had been, and to attempt to remove any errors at this stage. The engineer would present his change to a panel of his peers, describing the rational, as well as the actual code changes. Any problems with the Quality Review would be dealt with by the engineer, then the change would be resubmitted to the review.

Once the change had passed the review, the change would proceed to a library of changes waiting to be implemented in a field release of the software.

When a number of changes had arrived in the library of changes, a decision would be made to build a new release of the software. At this time, a selection of changes was made from the library, not necessarily including all the available changes, and these would be implemented. The system would then go through rigorous unit and system testing before being released. This stage could well discover problems with the changes it implements. If these problems were easy to correct, the change would be made. If the problem was more complex, it could well lead to the generation of a new request for change, and the initiation of the whole cycle again.

This then was the procedure adopted by the maintenance team during this project.

4.1.2. The Maintenance Data

During the project described above, paper based information was collected about the task being requested and completed. An important feature of the software change procedure described above, is the intrinsic need to capture data about the maintenance tasks. This need is generated by the procedures themselves, in order that the various stages can be completed.

When a request is generated, the details of that request must be recorded so that a receipt can be generated, and also so the request can wait in a queue of pending requests, if one exists. This record of the request is the driver for the change design and implementation, and also defines the objective of the change for use in the review. The reasons that the change is required, and its 'customer' generated priority, are also important in the final choice about which changes are to be included in a new release.

This record of the request and its details, therefore, forms the first part of the information set for this particular change.

Once an engineer has assessed the change request, and designed and produced the required code changes, he has to present these to a review panel. This presentation requires that the change design, and the reasoning behind the decision to go ahead with the change, are recorded and available for the panel to review. They must also be available for changes to be made if it is thought necessary.

The record of the assessment and the design of the change is the next part of the information set.

The code changes necessary to implement the change are not incorporated into the system immediately, but are added to a library for later implementation. This requires that the code changes are fully recorded and documented so work does not have to be repeated when its becomes time to implement the change.

This library of required code changes forms the third part of the information about the change, and the change procedure. If changes to the design and implementation are required by the review, new details must be produced, of the new design or implementation. These new details are also added to the information set.

When it becomes time to produce a new release, and the changes are implemented, the unit and system testing procedures produce either a new system or new change requests. This forms the last part of the data set necessary to drive the software change procedure.

The data described above is that data that must be produced in order for the procedure to work. What happens to this data once a change has been implemented is not important, it is just a by-product of the change procedure adopted.

This data, in fact, was collected on a paper based, form system to drive the procedure. It also became very detailed historic record of all the changes made to the system. This data set was, in fact, filed for later use, and it is this set that has been used to investigate the measurement system development method.

4.2. Application of the Method

The data provided by British Telecom is a historic record of the work performed over the duration of the project. The re-enactment of a major maintenance project is outside the scope and resource of this thesis, therefore the data provided by British Telecom will act as a substitute.

The data can be used to theoretically re-create the project, and it is to this re-creation that the measurement system development method will be applied.

4.2.1. Development of a Measurement System

4.2.1.1. Phase 1 - Initial Data Set

The first phase is to define the initial data set for collection. The areas of interest are the five data types defined in Chapter 3:

- Available data: That data that is already available as a by-product of the processes already employed.

- Attainable data: That data that would be a by-product of the processes use given a small change in procedures more strict application of procedures.

- Collateral data: That data that is produced as a by-product of the processes but is not actually collected.

- Other: Data perceived as being useful but which required effort and expense to collect.

- Inaccurate data: Data that is inaccurate or impossible to collect accurately.

As described above, the maintenance procedures adopted during the British Telecom project produced a large amount of data as a by product (*available* data). This data, collected on paper forms, includes such information as:

- A description of the change required.

- The system and version of the software that the change was required for and the incoming priority.

- A reference to the initiating 'Customer'.

- The symptoms of the fault if it was a fault report, and a detailed description of the required performance after the change.

- A record of the engineer assigned to a change.

- The date of receipt of the request.

- Diagnosis of problem or the change request.

- A description of what changes should be made, if this was thought necessary.

- The date of the completion of analysis, change design and implementation.

- A reference to the files that required changes to be made.

- The actual changes to be made to the files. This section was a text entry that usually contained a copy of the statements to be changed, and the new version of those statements.

This *available* data must form the basis of the initial collection set. This forms a quite large set of data to collect, but we must also consider the other types of data.

Collateral data is a useful source of information, but is not available here. It will be considered later. The same is true of *other* data. *Attainable* data is also an important consideration, but its consideration can be left until the evolution phase.

This then forms the initial data set for the collection system. A full list can be found in Appendix A. It consists entirely of *available* data and so the overhead of its collection is, so far, negligible.

4.2.1.2. Phase 2 - Collection Strategy

The collection strategy must be carefully considered based on the data to be collected. The information is currently collected on paper forms by the engineer concerned, and filed. This poses several problems.

The first is that the use that can be made of the information on paper is fairly limited, without a large amount of expended effort. The second is that validation of the data is very difficult. That is, ensuring documents are filled in completely, and at the correct stage of the process and that the information contained on them reflects reality.

Both of these problems can be solved by the implementation of a machine based system for the data collection.

The system should be based on the set of forms currently completed, and require the same data to be entered, i.e. the defined initial set of data. This data is now electronically stored and so is easily retrievable. It can also be validated at the time of entry by two means. The first is by requiring certain fields be filled with appropriate data, and by filling others, such as dates, automatically. This stops incomplete data being entered.

The second validation step is ensuring the correct procedure steps are completed. That is, we can insist that a review takes place after the diagnosis of the problem. The machine can enforce this procedure by requiring review information before any further action can be taken.

This represents a sensible strategy for data collection. The overheads must also be considered. The major source of overhead, during the maintenance process, is very small. There are small procedural changes, but in general the process is the same from the engineers point of view, the forms are just machine based instead of paper based.

From an implementation point of view, the data entry system has to be procured. This cost, however, is relatively small and can be justified for the immediate benefit of more rigorous procedural control, as well as the potential visibility gain.

4.2.1.3. Phase 3 - Collection and Analysis

The first stage of this phase consists of validating the data collection set and strategy. The data set is self-validating in this case due to its source. It is already produced and available.

The collection strategy is more difficult to validate from the current position of historical data. A prototype system was designed to collect the information required, and a validation exercise was carried out on the paper based data during which time it was converted to electronic form. The prototype system (SCIMM) is described in the next chapter.

The data captured on the forms displayed several shortcomings, for which there was no remedy at this late stage. The major examples where data that was not completed correctly on the forms and data that was obviously incorrect. Both these factors have feedback into the collection system, specifying extra checks that must be performed.

A specific example of incorrect data, was that of completion dates of the various stages of the process. These dates, in some cases, were the same for all stages, even on some large scale changes. This indicated that the forms had been completed only after all the work had been done, thus, perhaps,

invalidating more of the data collected. This state of affairs, however, would be rectified by the machine based collection system.

This procedure mirrors very closely the real implementation of this phase. Problems and shortcomings are identified, and feed back to phase 1 and phase 2 to produce a workable data collection system.

The next stage of this phase is to collect and analyse data. This was accomplished using the now machine readable form of the data collected by British Telecom. The database now represents a long term use of the collection system, although some validation is missing.

From the data collected we can begin the analysis stage. This analysis takes the form of presentation of the data collected. Simple presentation techniques can be used on subsets of the data to demonstrate the usefulness as a management tool. Examples are given in the next chapter, and can been seen in Appendix B.

Here, the Foster model of maintenance is again considered as the guide to useful information about the maintenance processes. A number of the features of the Team Level can now be quantified:

- Frequency of incoming requests.
- Number of requests solved immediately.
- Number of requests rejected.
- Outstanding change requests by time.

- Outstanding change requests by assigned engineer.

- Numbers of changes awaiting a new release by time.

- Number of changes requiring rework.

- Time for each stage of a change.

These reports quantify facets of the model, but data can be presented to answer more general activity questions:

- Total changes per module by time.

- Total new releases by time.

These reports demonstrate information that can be retrieved about the ongoing project that would have been, at best, very difficult to ascertain previously. All these reports, however, contain information to make the management and control of the project easier and more complete. These reports also provide information to describe and support the work of the team to higher management level. All these reports, in addition, represent a simple display of the data captured.

Of course, the possible reports to generate are endless. More examples can be constructed easily. This is the stage at which evolution take over.

The last stage of this phase, however, should not be overlooked. That is the use of the data for other tasks, apart from management.

The data represented here forms an experience store of previous changes to the software, with their rational and associated problems. This data should, if made available to the maintenance engineers in a useful way, provide a valuable tool to help with the tasks of maintenance.

By providing a method of retrieval of the data, an engineer would have access to the content and reasons behind similar changes to the one currently being made. This should short-circuit a proportion of the necessary work, so reducing the required effort. It would also allow previous problems encountered to be avoided.

This technique has further ramifications. A proportion of changes made to a software system introduce errors themselves and so lead to further work. If the original change that caused the problem can be identified and altered, instead of introducing a new patch on a patch, this should reduce the overall complexity increase and improve the life-time of the software. This also introduces traceablility of changes into the measurement system - another useful item of data.

The ability to trace similar changes has a further result - the identification of duplicate change requests, or the collation of very similar outstanding request, so they can be dealt with together, as opposed to with separate effort. This, from a management point of view, improves the work scheduling and should reduce work required.

4.2.2. Evolution of the Measurement System

Evolution has already begun. The previous section described a number of reports that can be produced from the data available. Some may not be particularly useful in the particular environment concerned, these can be removed. Others can be envisaged that could give further useful information about the environment, such as, numbers of change requests that are impossible to implement. Thus, the measurement system changes to incorporate these new requirements.

Functional change requirements to the collection method have also been identified. Those of allowing access to previous change data by maintenance engineers. This requires the development of a data retrieval method that allows specific changes and types of changes to be accessed. Such a change is incorporated in the prototype described in the next chapter.

These are evolutionary changes defined by the practical collection of data done so far. Evolution of the data set must also be addressed. The data collected must be assessed in order to identify that data that is not useful, and to identify data that would be useful. The cost of these changes must also be addressed.

An important flaw in the British Telecom data, that has been referred to before, is the lack of accurate time stamp data. From a management point of view, the time tasks take to complete is of utmost importance as future resourcing and scheduling rely on this type of information. This can, therefore, be identified as data that is lacking from our collection set, and should be included. The cost of this data is relatively small, if we require that the collection system itself time-stamps the data as it is entered.

Another piece of information that is important to the work scheduling process is some information about the relative sizes of the solutions to change requests. The actual changes required are already entered as part of the collection system, but further information would be desirable. This requires an examination of the work environment to find a source of this information that is both accurate and easy to collect. Sources worth considering are compiler time or output when compiling a change, or changes to the testing scheme that a change necessitates.

These then are definition of steps for the evolution of the measurement system, and will themselves generate further steps. At each stage, however, the current system can be assessed with a view to its cost/benefit relation. As knowledge about the environment grows and potential data sources and uses are identified, the system will evolve.

Chapter 5

The SCIMM System

5.1. Introduction

The SCIMM (Software Change Information for Maintenance Management) system [COOPER89] is the detailed prototype implementation of the results of the initial application of the method, as described in Chapter 4.

SCIMM is a computer based system that stores information about requests for changes and changes made to software systems, with a view to easy access and retrieval of data, and the provision of analysis to allow managers to analyse this information and use it as an aid in prediction, planning and scheduling of maintenance activity, as well as for report generation and to raise the overall visibility of the project they are managing.

The SCIMM system also provides facilities to help the maintenance programmer with maintenance tasks on the software system in question.

The SCIMM prototype development was based on the principles, outlined earlier, for the collection and analysis of data, and forms a practical example of the application of those principles.

5.2. The SCIMM System

5.2.1. System Overview

SCIMM is a computer based system that stores information about requests for changes and changes made to software systems. This data collection occurs at every stage of the change process, from initial request to incorporation of the change into a new release of the software. The data set to be collected was based on the application of phase 1 of the measurement system development method, described earlier, to the British Telecom maintenance environment.

This data collection requires that the SCIMM system be central to all the activities of the maintenance team, and is used at all stages. To ensure this, SCIMM provides the basis for a comprehensive change control procedure, following closely the procedures already in place in the environment in question. The system provides functions that, ensure change procedures are followed and changes are complete before being signed off, allow change tracking from request to completion, allow quality assurance procedures to be carried out on the changes and provide a master store of the actual code changes for later incorporation into documentation or releases.

In order to offset the overhead of time and effort required to use the system, as well as providing the main objective of management visibility, the system also provides information in a form useful to the maintenance programmer doing a maintenance task.

The SCIMM system is designed, primarily, to demonstrate the application of the measurement system development method described in Chapter 3, and to show how the method can satisfy the goals defined for the method.

5.2.2. Data Collection

The data capture method of the SCIMM system is based on a series of preset forms that must be completed at various stages of the change process. Some of the fields are filled automatically, and as the data collection system evolves, and incorporates other tools, a greater proportion will be filled in this way.

The forms themselves define the method by which maintenance tasks will be completed, and provides a first level of visibility of the tasks being undertaken. This first level of visibility coming from the presence or absence of the data. The forms that have been completed define the current state of the change, and once all the necessary forms for a change have been completed, the change can be signed off as complete. The system stores information based on the maintenance task, which is the effort required to respond to a single request for change. The system also maintains a complete history of the change, including feedback, and the reasons for the feedback.

In this way the SCIMM system captures the maximum amount of the available information about the tasks being performed, and it will be shown how the overheads involved can be offset.

Each maintenance task has five basic forms, each corresponding to the end of a stage in the change process. These five basic forms are referenced by a task header that is created when the change request is received. The task header maintains information such as the state of the task, the engineers working on the change, the expected time for the change, the actual time taken for the various stages of the change, the current priority of the task and the result of

the task. A full list of the data elements captured by the SCIMM system can be found in Appendix A.

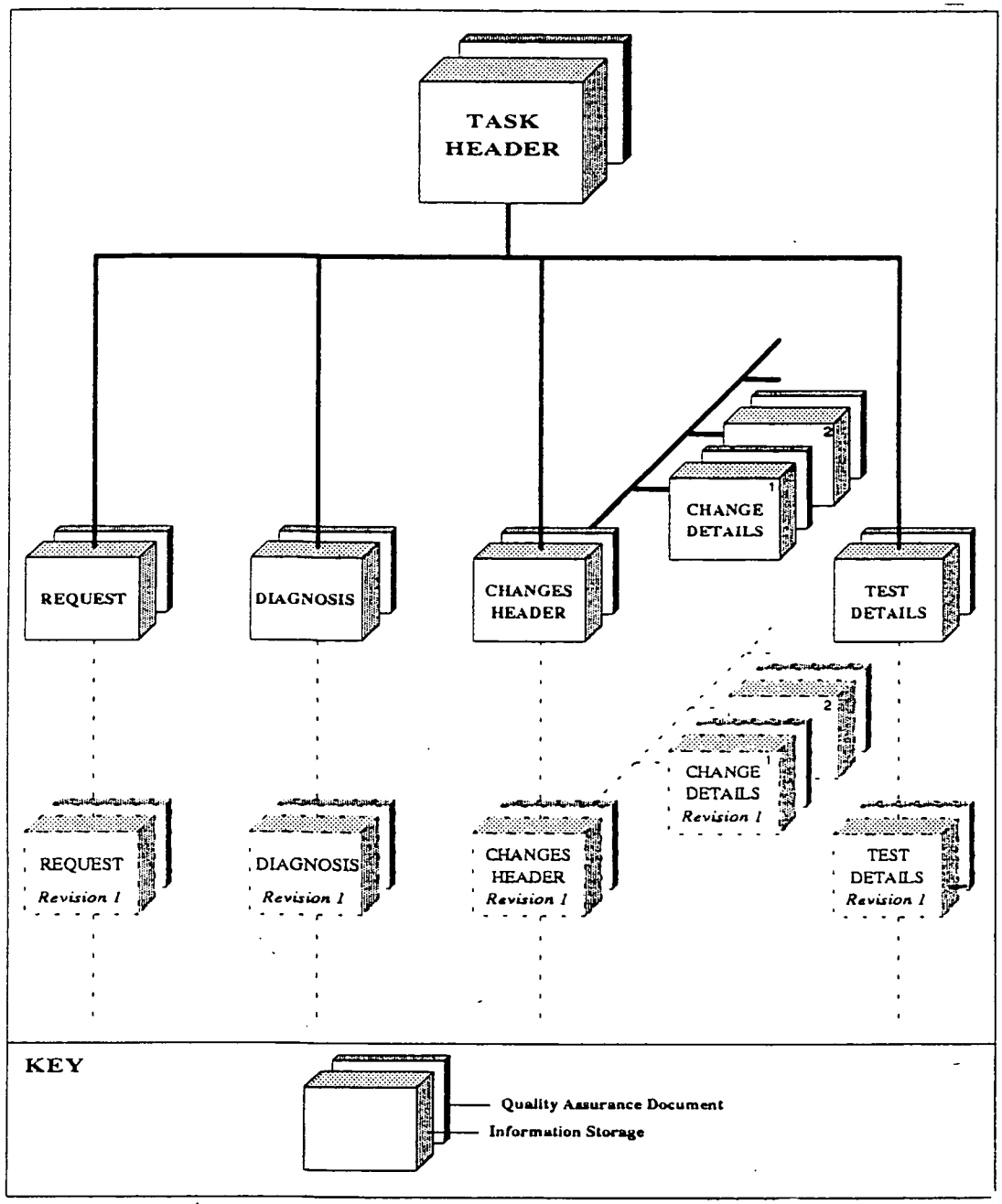

Fig. 3 The SCIMM system database structure

5.2.2.1. Change Request

The first of the five forms is the change request form. This form corresponds to the receipt of a request for a change to the software system. The request will normally originate from outside the maintenance team, but may be generated by a previous change, or from a planned maintenance schedule. This form is completed by the front desk personnel, if such exists, or by an engineer if not. The form records information such as a text description of the required and current performance, contact names for originator and the priority of the change from the originators point of view.

At this stage, a keyword description of the change request is also entered. This keyword description consists of a list from a set of pre-defined keywords and describes the change from the users perspective. The list contains information about the requirements, effects and areas of influence of the change from a users point of view, that is, the view of someone using the system. This keyword description is designed to allow analysis of the request, based on the users perspective of the problem. The keywords contain information about which screens are affected by the change, which controls accessed by the user are affected, and which parts of the display or other output of the system are effected. An example may be the keyword 'DIALLING' which means the change, or required change, effects the operation of the system while a dialling operation is underway. This perspective is very important for analysis, and for searching the database, as it is often only a users perspective that is available, without effort being expended by a trained programmer or someone knowledgeable about the system. This users perspective, therefore, provides a first line description of the change, and the keyword description allows analysis and searches based on this perspective.

The receiver of the request, may, at this stage be able to provide a solution immediately. This may come from the store of changes already completed, or the change may not be possible, or actually required. In this case the change can be written off as complete, with an appropriate result code stored in the task header. The change would then require no more work and the remainder of the forms would not be completed.

5.2.2.2. Change Diagnosis

The second of the five forms is the change diagnosis form. This form corresponds to the completion of the analysis of the request, and its implications, by a maintenance engineer. The method by which the change request is allocated to an engineer, and how changes to be worked on are chosen from any queue of requests awaiting work, are not dealt with here. A system for priority ordering, and request selection could, however, be incorporated into the system, corresponding to the system adopted by the maintenance team.

The form records information including a text description of the cause of the problem, or an analysis of the change required and a specification and design of the required changes to the code. This form also includes a keyword description of the change.

The keyword description, as on the request form, consists of keywords taken from a predefined set, that describe the change requirements from a system perspective. This perspective provides the second stage of a description of the change, allowing analysis and searches to be performed. The first stage being the description from the user perspective. This forms a very effective way of

describing the change in a way that allow automatic analysis and searches on change types. An example may be the keywords 'LONG DISTANCE' and 'CONNECT', which specify the change includes an effect upon the part of code responsible for making a connection on a telephone exchange, and only has an effect when the call being made is long distance. In the keyword description definitions, the keyword 'LONG DISTANCE' might contain a description of the factors that hold for a call to be long distance. From these keywords we can now select all the changes that effect the connection of calls. When this is combined with the user perspective, we can find, for example, changes that effect the connection of calls, but only when generated by a user dialling function.

As in the previous stage, the result of this analysis may be that no further work is required, for reasons such as the problem is a hardware fault, or caused by incorrect use of the system. If this is the case, the appropriate result code is stored in the task header, and no further work is required.

5.2.2.3. Changes Header

The third form to be filled is the changes header form. This is created when the actual changes to the system have been decided upon. This contains references to the change details forms (see later) for this change. It also contains information about the state of the software system at the time the change is implemented.

In an ideal maintenance environment, the change, once designed, will be incorporated and tested on a version of the software reserved for use by the maintenance team. The change will not actually be incorporated into a release

of the system until such time as a new release is decided upon. At this stage, change specifications will be taken from a library of changes and incorporated into a new base line of the system that will form the new release.

Based on this scenario, it is important that the actual state of the software at the time the change is specified, is recorded. This is important in the case where a piece of code, or specific functionality, is affected by a change, then a subsequent change affects the same piece of code, or functionality. When the second change is designed we must record whether the system being changed includes the first change, or does not. In the case where it does include the first change, at the time of creating a new release, the first change must be implemented before the second for the resulting system to be correct. If the second change was defined on a system not including the first, then if both are to be implemented, an analysis of their effects on each other must be performed.

For these reasons, the change header records the baseline system to which the change applies, along with any other changes that either must be made to the baseline system before this one, or changes that are mutually exclusive with this one.

5.2.2.4. Change Details

Multiple copies of this form may exist for each maintenance task. Each form contains details of changes to only one unit of the system source. This unit may be a module, a program or a file depending on how the system is defined. Each form is referenced by the change header to relate these forms to the original task.

This form contains details of the unit to be changed and the changes to be made. It also contains a reference to a documentation file to be changed and the changes to be made to the documentation.

Although not actually incorporated into the prototype, the aim is to store the change as context sensitive editing commands. This form of storage would allow the engineer to record the changes easily - by just doing the required edits, or comparing the initial and final forms of the unit that had been changed. This form would allow the database to be used to actually perform the edits, and could be used to generate documentation about the change made, for example, in a release notification. Performing these tasks straight from the database store would ensure that there were no transcription errors introduced, once the change had been specified. This would also help if problems were encountered in the change, as the analysis could start at the change design, and not have to examine the possibilities of errors introduced later.

The reference to a documentation change is very environment specific. The SCIMM system is seen as providing a control framework in which the maintenance tasks are performed. The system stores references to the new and updated documentation related to this change ensuring these exist are are current. The other important feature of the documentation reference, is to allow quality assurance reviews to ensure that documentation has been updated. In the SCIMM prototype, this reference consists of a record of the documentation file changed and the changes made, as with the code.

5.2.2.5. Test Details

This is the fifth of the basic forms that record information about the change. It corresponds to the stage where the change is incorporated into a new release of the system. This, as described earlier, should ideally be a scheduled procedure, incorporating multiple changes and involving a complete system test. It is, however, equally applicable to the case where the change is incorporated into the system immediately.

This form is created when the corresponding change is implemented in a release system. This form is another example of an interface to other tools in the maintainers suite. This form should interface with a rigorous system of unit and system testing, and a full regression test system. In the prototype, however, it stores various information about the test of the change. This information includes, specification of the required regression test, results of the regression test, specification of new tests required to verify the change request requirements have been fulfilled, results of these new tests and a summary of results and required future action.

The test specifications come from the change request details, and the change design. The results show the comparison between expected and actual test outcomes, and the action specifies the outcome of the tests. If all tests are completed satisfactorily, the test details are recorded as passed, and the change is marked complete in the task header, with a reference to the new release in which it is incorporated.

If the test results are not acceptable, a feed back loop is produced.

5.2.2.6. Quality Assurance and Feedback

The five main stages of the change are described above, but this is not the whole story. Attached to each form is a quality assurance form. In the prototype, these are simple forms that record the date the form was reviewed, the reviewers and the action resulting.

The actual quality procedures, and at what stages they are performed are dependent on the environment and on work practice. This is another example of a possible interface to other tools during the evolution process. The use made of the quality form is primarily to allow easy identification of those stages that have not been reviewed, and to provide links to the feedback loops.

Feedback loops are the method by which work is redone. There are two major causes for feedback, these are, unsatisfactory test results or actions generated by a quality review. When feedback is required, work returns to a previous stage of the change process, and new versions of the required forms are generated, as work is redone. Once a form is completed, it is not allowed to be edited, but a new version is created that stores the changed details. In this way, a full history of the change, its rework and the reasons for the rework are stored.

An example of feedback would be where a quality review on a diagnosis form decides that the side effects of the proposed change have not fully been explored. In this case the review result would be that diagnosis rework was required. Once the work had been done, a new version (version 2) of the diagnosis form would be created and submitted for review. Once passed, the change would move on to the change details stage, etc.

If a problem is found in the change testing, this may cause a feedback to any stage. For example, it may be that a problem's cause was not correctly diagnosed, therefore a new diagnosis stage will be required. It may be that an actual code change is incorrect, in which case it is just a new change details form that is required. Whatever the result, a new form is created and the change process then proceeds from there.

The only form for which there is never a new version is the request form. This form defines the task being performed. If the requirements for the change alter, or testing highlights a new problem, a whole new task is created and queued. In this way, a single task responds to a single request.

5.2.3. Data Analysis

The data analysis part of the SCIMM system provides simple methods of accessing the data collected by the above scheme. The first important point to note is that the data is stored as a connected network of forms relating to a change, (see figure above), including any rework necessary, and so the information about a change, and therefore the change process itself, is completely traceable.

The information in the database can be accessed in two different forms.

5.2.3.1. Change Information Retrieval

The actual data collected about a specific change can be accessed. This access can be achieved by specifying a particular change reference, or by specifying various change criteria. These criteria can include:

- *Change State*: allows the selection of changes that are at a certain stage in the change process, such as in testing, or waiting to be implemented.

- *Request Age*: selection of changes based on the length of time the request has been awaiting action.

- *Request Keywords*: allow the selection of changes based on the users view of the software being maintained. For example, changes dealing with a specific user screen or functions, such as performing a dial operation on the exchange, can be retrieved. This type of selection relies on the definition of the keywords entered in the change request form.

- *Diagnosis Keywords*: allow selection of changes based on programmers view of the software. For example, changes involving requesting a long distance line connection on the exchange. This selection relies on the definition of the keywords entered in the change diagnosis form.

- *Unit Changes*: selection of changes based on which units they affect.

- *Quality or Testing Results*: selection of changes that have had specific problems in quality reviews or at the testing stage.

In fact, searches can be made on almost any feature of the data entered in order to access the data for that specific change. The change details can then be printed as a report or the group of changes selected can be listed and quantified.

5.2.3.2. Reports

SCIMM is primarily designed as a system to help the management of software maintenance. This it does by providing visibility of the project to the project manager. The main supporter of this visibility is the production of detailed and summary reports about the data being captured.

The reports produced by SCIMM can be at various levels of detail, from providing details of changes made to code, to high-level statistical summaries of the project state. The reports are based on change selection as described above. Changes fulfilling various change criteria can be selected, or all changes on the database can be included.

The amount of information included for each change can vary from a simple count of the number of changes, to a detailed description of all the fields stored. As has been described earlier, the most important feature of this stage of the data collection/analysis method is flexibility - the ability to analyse data in whatever way is desired, and the ability to change the way in which the data is analysed.

An example of the utility of the reports generated by SCIMM to the maintenance manager is the total number of requests received. This report, if produced at regular intervals, provides a history of the rate of request arrival.

This history can be examined to find time dependent patterns in the rate of requests coming in. These patterns can then help predict future request patterns, an allow forward planning.

Examples of reports produced by SCIMM are:

- Frequency of incoming requests;
- Total requests outstanding;
- Requests outstanding by time outstanding;
- Requests outstanding by engineer assigned;
- Total changes being processed by engineer;
- Average time for a request to be completed;
- Total changes awaiting implementation;
- Number of changes requiring rework;
- Time for each stage of a change;
- Total changes per module by time;

Examples of some of these reports are shown in Appendix B.

The actual reports possible are unlimited. The specific ones useful to a particular team must be decided on by that team. The important feature to note is how the visibility of what is going on in the team and the project is immediately raised, providing the ability to correct problems and generally manage and control the processes involved.

5.2.4. Maintenance Programmer Support

The collection of data described above causes an overhead in time, effort and resources allotted to the project. Maintenance programmers have to spent time collating information and filling in forms. Machine resources are expended, storing and manipulating the information and procedures must be incorporated that ensure the validity of the information entered. These have effects on the project, which have to be justified.

The preceding discussions present a strong case as to why this overhead is justified, but in a commercial environment it is still difficult to quantify the benefit to compare against the cost, in an often under-resourced field. For this reason, SCIMM provides extra functionality to help the maintenance programmers in their tasks, in order to reduce the overall effort required. Not only does this offset the cost of the system, it would also help the system to be accepted, and used to its fullest capacity.

5.2.4.1. Change Cross-referencing

The major feature that helps the maintenance programmer is the ability to cross-reference changes stored on the system. The first stage of this is the facility to search the database of changes, as with the analysis functions above,

to find changes that fulfill certain criteria. These criteria are generally searches for changes with similar request keywords, or diagnosis keywords or which affect a certain area of code.

This process of selecting certain changes has a number of advantages for the maintenance programmer while working on a change. The first is the ability to detect complete changes, or requests for change, that fulfill the requirements of a new change request. In this way, repeated work can be avoided, and customers with a change request that has already been dealt with can receive the fix immediately.

The second advantage is the ability to detect similar changes made in the past. These similar changes may well short circuit the analysis and design phases of the change process, so speeding the process and reducing the effort required. An example would be tax changes in a financial package. If a programmer can reference the changes made last time there was a tax change, the current task is a simple recoding job, instead of the need for analysis of the code and change design.

The maintenance programmer is also able to find changes in the change library that may conflict with the change being designed. This is particularly important where multiple versions of the system exist, or, in the case of British Telecom, where a library of unimplemented changes exist. By identifying those changes that may conflict, problems can be avoided, or documented, before implementation is started.

The fourth use of the search criteria, is the identification of ripple effect errors. It has been shown that maintenance activity itself is responsible for a high proportion of errors in a system. System degradation and decay is advanced by

maintenance tasks consisting of patches onto the original code, followed by the implementation of further patches when the original fix fails to work. By allowing searches for similar changes, and changes affecting the same area of code, the global picture of what has been done to a piece of code can be learnt with little effort. There is also access to the original reasons the changes were made. From this information, if a previous change introduced an error, the original change can be corrected, instead of adding a new patch. If a previous change was not responsible, there is still the option to redesign the whole set of changes to make them better and more compatible as a group. In this way, system degradation is slowed and the maintenance effort is better directed and, therefore, less wasteful.

A feature that was found to be useful was that of permanent stored links between changes. These links could be put in place for a variety of reasons, such as cause of problem and fix. The task of searching for changes and manually assessing the results is then further shortened. The links also provide documentation about the changes being implemented, and provide a further facet of information about the processes involved.

5.2.4.2. Change Search Criteria

These searches are all based on the three features of a change stored for this specific reason. These are:

- The keyword description of the request (users perspective): provides the ability to distinguish changes to the outside, users view of the software. This often defined an area of the program or a

functional block. It is all there is to go on at the early stages of a change process.

- The keyword description of the diagnosis (system perspective): distinguishes changes at the system, functional level of the software. This defines a functional unit of the code, the users perspective provides information about the timing, or context of the use of the functional unit.

- The area of effect of the code changes: defined in terms of actual code statements or units changed.

The searching algorithm can then be a simple pattern matching search, from a set of required keywords or statements effected, to the keywords and area of effect associated with each change in the database. These three features have been shown to provide a useful, and easy to implement, searching method for particular changes, or groups of changes fulfilling certain criteria. In general, these search criteria are not exact, in other words, manual intervention is required once the automatic selections have been made, to isolate the changes that are of interest. This does, however, provide considerable help to both the maintainer and the manager which is available in no other way.

5.3. Evolution of SCIMM

The ongoing method application requires the ability of the system to evolve. This can be demonstrated by a number of examples.

The major implemented evolution step is that of the keyword search algorithm. From the initial collection and analysis system, the usefulness of the collected data to the maintenance programmer, as well as the maintenance manager, was identified. This usefulness, however, depended on a system for accessing the required data both easily and quickly.

To satisfy this requirement, the keyword coding system of both the user view and the system view was developed. This was added to the SCIMM prototype and shown to satisfy the selection requirement. This also added an important feature to the management use of the system by allowing certain types of change to be isolated.

The second evolution step identified was an evolution of the data set collected. A requirement of the management process is for information about time related features, such as the average time for responses to requests, or total time spent implementing changes. Accurate information of this type was not retrievable from the initial set of data due to the lack of control over when forms were filled and the dates entered on forms.

For these reasons, time information was added to the data set collected, and the date information was specified more clearly as the '*date of form completion*'. This date could now be completed automatically. Although no actual data of this type was available from the British Telecom data set, sample data shows the usefulness of these new fields.

The evolution described above requires that the SCIMM system be implemented using a flexible implementation strategy, for example, a 4GL database management development system, so that data can be added and removed, and analysis and reports can be easily developed and used.

5.4. Summary

SCIMM is a computer based system, designed using a flexible database implementation system, that stores information about requests for changes and changes made to software systems. This information is then presented in a form to help the maintenance manager in his task of managing the project, and also provides facilities to help the maintenance programmer, thus reducing the effective overhead of the data collection.

The system demonstrates a method whereby a data collection and analysis system can be incorporated into a maintenance environment without large amounts of effort or expenditure. It also demonstrates how flexibility of the data collection method, and the analysis performed can be included to allow use to be made of the system immediately and without prior knowledge, and how it can quickly produce useful results and still provide for evolution of the method as knowledge, experience and requirements for the system grow.

Chapter 6

Evaluation and Conclusions

6.1. Comparison to Criteria for Success

The basic premise of this thesis is that measurement of software maintenance products and processes produces visibility and understanding, leading to better management of the software maintenance environment at both line and corporate level. This has been shown by a number of studies, including the Goal/Question/Metric paradigm [ROMBACH87].

The hierarchy presented in the Goal/Question/Metric paradigm of collection of data providing answers to higher level questions is valid and useful. This thesis, therefore, addresses the shortfalls of the Goal/Question/Paradigm by applying a bottom-up design approach. This approach will help target the hierarchy, overcome the overhead problems which are a major consideration in an industrial environment, and provide for an evolution of the system to take account of goal changes and gathered experience.

This thesis presents a method whereby a measurement system can be developed and introduced into a maintenance environment, and, once introduced, can evolve to better meet the requirements for such a measurement system and to allow for changes in those requirements as knowledge and experience grow in

that environment. The method produces a system that addresses both the collection and use of data, in a way that is specifically tailored to the particular environment and working practices in which it is to be used.

Previous examples of work in the maintenance field have demonstrated how the measurement of the processes and products in a maintenance environment lead to greater visibility and, therefore, greater potential for management control [ROMBACH87, GRADY87, GRADY87a]. It also provides the required starting point for communication with corporate level management. This result is supported by the initial application of the method to the British Telecom maintenance data. Using these results we can infer that the measurement system developed as a result of the method presented here will also provide this increased visibility and the potential for management control.

The basic requirements, identified for an industrial maintenance setting, for the measurement system to have low initial investment levels and quick feedback into the management cycle have been demonstrated with the British Telecom example. The initial planning stage is kept to a minimum by simply requiring the identification of the five different types of data available in the particular environment and a decision about which data items will be collected. The initial impact of the collection system can be minimised by concentrating on the *available* data for the first implementation. The feedback to the management cycle and for communication to higher management depends, initially, on the presentation of the collected data. Thus the feedback can be immediate, but is still shown to be useful by the British Telecom examples presented previously (see Chapter 4).

The method presented here produces a measurement system that is tailored specifically to the environment in which it is to be used, and has all the benefits

of a measurement system produced from the Goal/Question/Metric paradigm, without some of the significant drawbacks as identified earlier (see Chapter 1), and expanded on below.

The work presented here, therefore, has been shown to satisfy all the criteria for success identified in Chapter 1 and evaluated above.

6.2. Comparison to the Goal/Question/Metric Paradigm

Two important results of the work on the Goal/Question/Metric paradigm are important in this context, and can be applied to the work presented here. The first is that measurement of the products and processes involved in the working of a maintenance environment increases the visibility of that environment to both line management and to higher level management. This increase in visibility leads to improved potential for management of that environment which, in turn, should lead to improved performance and reduced costs - important targets in any industrial setting.

The method presented in Chapter 3 produces a system that collects data about the products and processes in the particular environment to which it applies, thus leading to the improved visibility and, therefore, improved management potential as described above.

The second result of the Goal/Question/Metric paradigm that can be applied here is the applicability of a hierarchy of questions at different levels of abstraction, each level being answered by the questions in the level below. The bottom level of this hierarchy is the data that can be collected directly from the

environment. This structure is a valid interpretation of the use of data in most environments and provides a structure for the use of measured data. The method presented here supports this hierarchical structure during the analysis phase, but addresses it from a bottom-up point of view, thus supplying a number of advantages over the top-down approach of the Goal/Question/Metric paradigm approach.

The major shortfalls of the Goal/Question/Metric paradigm include the initial investment required for its application, and the lack of flexibility of the structure as it is created from the top, down (see Chapter 1). This structure can also be tenuous as it assumes an ordered hierarchy can be found from a defined top level to a required measurement level. Finding this order is often impossible. Once the bottom measurement level has been defined, it is rigid, and therefore often implies large overheads in the application of the measurement strategy, or even dramatic changes in working practices.

These shortfalls are addressed in the method presented here in two ways, firstly by the use of a bottom-up development method and secondly by allowing, explicitly, for the evolution of the measurement system.

The definition of the set of metrics, first, allows metrics to be chosen that fit into the environment without causing excessive overheads in planning or collection. This start point for the measurement system can, in many cases, be hampered by a lack of knowledge or experience of what is to be measured and how to go about it. The method presented here, by allowing a modest start, provides a basis for knowledge and experience to be gained, without costly mistakes and false starts.

Knowledge and experience is gained from the use of the data collected to answer higher level, more abstract questions, and the subsequent evaluation of these answers to identify pointless data, or areas where questions could be better answered, or other questions could be answered by the collection of further information. This also provides a platform for decisions about whether the extra data is worth the collection effort it requires.

In this way, the collection set evolves, along with the analysis of the data to generate the Goal/Question/Metric paradigm hierarchy, but always with a view towards its cost and impact as well as its practicality and maintaining its flexibility.

The bottom-up approach of measurement system development, along with the explicit support for system evolution, provide the major original contribution of the work presented here.

6.3. Evaluation of the Method

The method itself was applied to data collected by British Telecom during a large scale maintenance project (see Chapter 4). From this initial application, a prototype system was developed for the automatic collection and analysis of data (SCIMM, see Chapter 5) and the first stages of evolution were applied. Although the data used was of an historical form, valuable insight into the application of the method can be gained.

6.3.1. Phase 1 - Initial Data Set

The initial data set definition involves the identification of the five types of data present in a maintenance environment (see Chapter 3). The data set chosen in the British Telecom example consisted entirely of *available* data, that is, data that is a byproduct of the maintenance procedures in force during the project. This data set highlights immediately the advantages of this methods approach to data collection. The overheads involved in the collection of the identified set of data were negligible. This is evident from the fact that the data exists as a byproduct of the project, without any further effort being expended in its collection. It can be argued that, in fact, the data collection in a machine based form would have been directly beneficial to the project.

The identification of the data set for the first phase of the method also proved an easy task, when taken in the context of the environment to which it would be applied. The *available* data was a readily identifiable set and provided a good starting point for the measurement system. The data set used, however, can be seen to be very dependent on the particular working environment concerned. It proves a very difficult task to attempt to identify an optimum data set, independently from the working environment in which it will be used. The example of an initial data set from the British Telecom environment may represent a significant change in working practice and effort in another environment, even in the same field or company. This exemplifies the problems of applying theoretic requirements for the data set from an early stage, instead of the practical approach.

In an environment where the other types of data are present, such as attainable, collateral and other, these must also be addressed, with a view to their usefulness weighed against their cost to collect.

The identification of the initial data set was a straightforward task when based on a knowledge of the environment and the working practices involved.

6.3.2. Phase 2 - Collection Strategy

The data collection and storage schema required more effort to define and implement due to the requirement of making the system machine based. This overhead in design and implementation was, however, easily reconcilable with the benefits provided by the automatic system. These immediate benefits, including instant validation of data, easy correlation of the many parts of a single request for change and the reduced clerical effort involved in storing the information, are apparent even without any further use of the data.

The collection of data by the system was simulated by transcribing the paper based data into machine based form. This collection emphasized a number of aspects relevant to the measurement system. The first is the necessity for the collection system to hold a central place in the working methods of the environment. In the case of the British Telecom example, the collection system follows the normal working scheme and the data entered on the system is required for the change to progress toward completion. This is demonstrated by the review process required at each stage of the change. The review requires that the information about each stage be presented and thus must be entered in the system. The review stage completion is required before the change progresses to any new stage. It is also immediately apparent, using the data access facilities, when the system is not being used correctly. In this way, the collection system becomes part of the work scheme and is not just a peripheral device to use if there is time.

The second important requirement highlighted by the data collection is that of validation of data as it is entered. In the British Telecom example, this should be achieved by reviews of the data entered, however, the paper based data set did reveal a number of deficiencies and irregularities. These problems with the data lead to a general lack of confidence in that part of the data and so that part of the data becomes useless. An example of this was the date information entered on the forms. Some was missing and some was obviously incorrect. This leads to a lack of confidence in the whole of the date information.

An example of accurate, and therefore, high confidence information was that of the actual changes made to the code. These documented code changes where actually used to make the changes to the live system, (again, back to a central role in the work processes), so had to be complete and accurate. The result of this is the requirement for all data to be validated, at the time it is current, in order to maintain confidence in that data. This also becomes a driver for an evolution cycle, as described later.

6.3.3. Phase 3 - Collection and Analysis

At this stage the measurement system can be assessed. From the example reports shown earlier it is immediately obvious that the visibility of the project has been raised in a way that was certainly not previously possible. Ammunition is now available for the manager of the project to better control the project and to manage the environment. In examination of the data and reports it can be seen how the information can be used to increase control on the project. For example, using queries to ensure progress is being made, work load is evenly

distributed and to examine resourcing levels and make rough estimates of future resourcing requirements.

The sum total of the effort expended on implementing the system, and the overheads incurred in its use can also be readily seen as very low. Thus we have succeeded in the major goals of the method.

6.3.4. Phase 4 - Evolution

A number of areas for system enhancement were identified at an early stage. Date information, that is, the dates of stage completion, was seen as an important piece of information. This data would allow the tracking of time taken for stages to be completed, as well as time dependent variables to be monitored, such as the number of change requests over time. The overhead cost for this information was very small as it could be collected automatically by the system. This cost/benefit ratio allowed this change to the data set and collection strategy.

This evolution of the system demonstrates clearly the ease with which ideas about the data set can be evaluated and implemented without any large scale planning or rework effort. This must be a requirement in any commercial setting, allowing the system to remain usable and to become a useful management tool.

This evolution is, however, driven very specifically by the environment in which it is used and the knowledge and needs of those using the system. This evolution has been shown to be a simple task, not controlled by theoretic

guidelines, but by practical, on the ground experience, knowledge and local requirements.

The second evolution change in the system was brought about by the realisation of how useful the collected data could be to the maintenance programmers. This required a facility to select change data based on definable criteria. A method for this type of selection has not been available before.

The selection criteria facility, based on *user perspective* and *system perspective* keyword descriptions of the change, was developed to allow this selection. This facility permits maintenance programmers to access similar or contradictory changes. This reduces repeated work, reduces the degrading effect of ripple effect changes, and reduces the work involved in certain changes. This facility also proves useful to the maintenance manager for concentrating attention on certain changes and groups of changes (see Chapter 5).

This facility, however, mainly shows how the potential usefulness of the data collected to the tasks involved should not be overlooked.

6.4. Conclusions

A number of conclusions can be drawn from the preceding discussion.

Firstly, it has been shown that the method presented here allows a measurement system to be developed and implemented in a commercial environment, without undue cost in development effort or in overheads of use. The measurement system developed is directly applicable to the environment in which it is to be used, and incorporates the knowledge and requirements of those working in the

environment, in fact it relies on these. The measurement system produced has also been shown to be useful in the management of the environment, provided reasonable data is collected and used.

The method also provides for the evolution of the system without further large scale investment in planning or rework. This is an important feature of the method, and along with its bottom-up approach, distinguishes this method from others in the field. The measurement system can evolve to take account of changes in the environment and the increase in knowledge about the system that its use provides.

A major feature of the method is that it relies on the features of the specific environment in which it is to work. The *environment* in this context includes all the features of the application area, from work practices, to peoples knowledge and requirements, right through to the business rules of the company for which the maintenance role is a part. For this reason, no specific guidelines have been provided for the application of the various stages of the method, as the possible environments are too diverse to allow meaningful classification within the resource limits of this project. This diversity is characterised by the identification of the five data types defined in Chapter 3. These are available, attainable, collateral, other and inaccurate data. The identification of these five groups and the related costs of their collection are specific to the environment and form the important first step of the method. The application of the method within any specific environment has been shown in the case of the British Telecom project to be both straightforward and useful.

With the above in mind, it is useful to examine the British Telecom example further. The measurement system developed in this example case was shown to quantify a large number of the variables in the Foster Model Team Level (see

Chapter 5). This model has been presented as being representative of a large proportion of maintenance environments at the team level, and the quantification of the variables in the model have been shown to be a useful target for increased line management visibility and control and a basis for better corporate level communication. This leads to the conclusion that the measurement system and, therefore, the data set may have applicability to other environments.

For these reasons, and bearing in mind the ultimate uniqueness of any environment specific solution, the SCIMM system and the underlying data set are presented as a guideline, or template, from which to start.

Chapter 7

Further Work

This Chapter outlines some of the areas in which further work can be done, based on the contents of this thesis. Measurement is an important area of software engineering, and in order that software maintenance be brought under proper management control, more work in the fields of research, and practical application, is required.

7.1. Measurement System Development Method

The method presented here, in Chapter 3, has been shown to be applicable, and to produce useful results. This, however, is only a first stage.

The next stage must be to apply the method to a large commercial maintenance environment with on-going maintenance projects, over a long period of time. This sort of environment is important as it is the target of the method. Small scale experiments contradict the reasons behind the method and can, therefore, never produce useful results. This large scale application over a long time scale is required to assess the long term affects of the evolution process and the general commercial acceptability of the method and resulting measurement system. This is, however, outside the resources of this project and is, therefore, left to others.

Once the system has been applied in a number of environments, work can begin in assessing general similarities and differences in the measurement systems and environments. In this way, it may be possible to generate some general guidelines for the application of the method. This is only feasible with large amounts of information.

7.2. The SCIMM System

The Software Change Information for Maintenance Management (SCIMM) prototype was a system developed to demonstrate how data could be collected in a maintenance environment, and used to the benefit of the management of that environment. It also shows how the overheads involved in collecting data could be minimised by bottom-up selection of the data to be collected, simple analysis and by providing facilities to help the maintenance programmer based on the data collected.

The system was specifically developed to satisfy the requirements of the British Telecom environment, however, it has further reaching potential. The system demonstrates how a measurement system can be implemented in a central position in an environment, and provide useful facilities to both management and programmer. The system has potential for further development, to better implement these facilities. It also has the potential to become the core of a complete change control system.

Work has to be done on the interface of the SCIMM system to other common maintenance tools such as configuration management tools and documentation tools. By showing this interface is possible, it demonstrates the feasibility of a

data collection system as the heart of an integrated tool set. This reduces the importance of the overheads inherent in data collection and will make the idea of a data collection and analysis system more attractive to real organisations. It is only by persuading these organisations to collect data, that the data required to advance research in these areas will be made available. Some ideas of the interfaces that would be beneficial are to an Inverse Configuration Management tool [KENNING90] and a redocumentation tool [FLETTON88, FREEMAN90], both being developed at Durham.

As part of the system enhancement work, special interest should be placed on the report generation and analysis functions of the SCIMM system. The methodology described earlier proposes evolution of these functions to tailor a collection and analysis tool to a particular environment. The direction this evolution will take is dependent, very much, on the specific environment in which the system is to be used. The SCIMM system, however, provides a base for developing guide-lines as to the evolution by assessing its applicability to other environments. Real examples are certainly necessary if the system is to be adopted.

It is, realistically, the most effective way of introducing measurement into a commercial setting, by showing a system that does the job and is effective. The development of systems such as SCIMM is, therefore, an important step toward providing better management of maintenance and also improving the understanding of the fundamental concept of maintenance, and the processes involved.

7.3. British Telecom Maintenance Data

The data provided by British Telecom is a significant contribution to the store of data about software maintenance. The data is a very wide ranging set, covering and entire maintenance project in a commercial environment, and as such, is quite a rare commodity, not withstanding the deficiencies described above.

In this thesis, the data has been used to demonstrate the utility and usefulness of the measurement system development method. However, further analysis could be performed on this data as it has the potential to make a great contribution to knowledge about the software maintenance process itself.

The analysis of the data should progress in a number of areas. The first is that analysis of the data itself from a research perspective. Data from real software maintenance projects is a necessity if accurate and useful models of the processes involved in software maintenance are to be developed. This data is very hard to come by as it takes a long time to collect and requires commitment from a commercial organisation. The British Telecom data, therefore, has great importance in the research field, and should form valuable input into further software maintenance research, and maintenance model development.

Another area worthy of work is the assessment of the network of connections contained in the change information. As shown before, certain connections exist between changes made to a system. These connections include those between similar changes, and connections between fixes and the change that introduced the problem, i.e. ripple effect changes. The network of connections that can be built up from these base connections will give important

information about the nature of the system, the maintenance methods used and maintenance itself. This is therefore, another area of study.

The British Telecom data represents a valuable commodity in the maintenance field and its potential uses should not be underestimated.

7.4. Measurement for the Management of Software Maintenance

Only by measuring the products and processes involved in software maintenance, and the tasks performed can maintenance be brought under true management control. As has been shown, this measurement requires progress on two, interdependent fronts.

The first is on the research front, to develop understanding of the maintenance process, and identify models that describe the processes involved. This will allow identification of the important facets of an environment that must be measured, and also provide the analysis methods by which these measurements can be turned into useful information about the current state of the project, and the future states, allowing planning. This research requires real world data on which to work, and to allow validation of its models.

The second front, therefore, is the commercial maintenance environment. Systems for measurement have to be introduced now in order to collect the data required by research. These data collection systems also provide the basis for understanding the process in the commercial environment, and allow immediate progress toward bringing the maintenance environment under control.

Data must be collected over a wide spectrum of environments, and a wide spectrum of projects if a true picture of the maintenance process is to be gained. This also creates direct and immediate benefits in those environments and projects.

In this way we can head toward a real science of measurement of the maintenance process and bring software maintenance onto an equal footing with other commercial activities and also other areas of study. Only in this way can true management be brought to software maintenance.

Appendix A

SCIMM Data Collection

There follows a list of all the data fields collected by the SCIMM system. This data set is was produced by the application of phase 1 of the Meaurement System Development Method to the British Telecom maintenance environment..

Task Header

- Task identifier
- Current status of task
- Date of status
- Customer raising the request
- Staff the change is allocated to

- Result of change if complete

- Priority assigned when request received

- Current priority

- Expected time to complete

- Actual time to complete

- Date request received

- Actual date completed

- Date of change release

Change Request

- Keyword description of problem

- Text description of current performance

- Text description of required performance

Diagnosis

- Keyword description of diagnosis
- Text description of the causeof the problem,or an analysis of the required change
- Specification and design of change required
- Testing requirements
- Completed by
- Completed date
- Time taken to complete

Change Header

- List of any changes that must be made before this one is implemented
- The number of files that must be changed
- Completed by
- Completed date
- Time taken to complete

Change Details

(One of these forms is created for each file that must be changed.)

- File to be changed
- Documentation file to be changed
- Description of changes to the file
- Description of changes to the documentation
- Completed by
- Time taken to complete

Test Details

- Testing advice from by change designer
- Regression test advice from change designer
- Testing results
- Regression test results
- Results Summary and actions

- Completed date

Appendix B

SCIMM Example Reports

Example 1

Requests Outstanding .. more than 3 Weeks
by Time

Outstanding more than 9 weeks

Ref:	Description	Status	Recieved	Status Date
0101	T. Forge Ltd.	Test	25/11/88	08/01/89
0107	Dept. 5342	Diagnosis	02/12/88	03/12/88

Outstanding more than 6 weeks

Ref:	Description	Status	Recieved	Status Date
0111	SWC Plastics	Review	27/12/88	01/01/89
0115	RPZ Ind.	Diagnosis	29/12/88	29/12/88

Outstanding more than 3 weeks

Ref:	Description	Status	Recieved	Status Date
0156	SDC Ltd.	No Action	07/01/89	
0158	Dept. 5342	Diagnosis	12/01/89	13/01/89
0161	Internal	Hold	12/01/89	16/01/89

Example 2

Totals of Requests Outstanding .. by Time
by Engineer

Outstanding more than 9 weeks

Engineer	No.
ARC	2
SFN	1
Total	3

Outstanding more than 6 weeks

Engineer	No.
SFN	1
WRP	1
Total	2

Outstanding more than 3 weeks

Engineer	No.
DFH	2
Total	2

Outstanding less than 3 weeks

Engineer	No.
ARC	4
WRP	4
DFH	2
RLG	2
SFN	1
Not Allocated	8
Total	21

Total

Total	28

Example 3

Totals of File Changes .. Program 2
6 Months

File Changed	No.
arc.mod	2
command.mod	10
common01.mod	3
common02.mod	16
common03.mod	4
comms.mod	12
comp01.mod	6
comp02.mod	23
comp03.mod	8
comp04.mod	16
comp05.mod	3
output01.mod	5
output02.mod	0
output03.mod	1
readdata01.mod	0
readdata02.mod	1
scan.mod	18
user.mod	26
Total	158

References

[ALBRECHT79] A.J. Albrecht, "Measuring application development productivity", in *Proceedings of the IBM Applications Development Symposium*, Monterey, California, 1979.

[ARNOLD86] R.S. Arnold and D.A. Parker, "The dimensions of healthy maintenance", 1986.

[BAIRD78] J.C. Baird and E. Noma, *Fundementals of Scaling and Psychophysics*, John Wiley & Sons: New York, 1978.

[BASILI84] V.R. Basili and D.M. Weiss, "A methodology for collecting valid software engineering data", *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, pp. 728-738, 1984.

[BASILI85] V.R. Basili and R.W. Selby, "Calculation and use of an environments characteristic software metric set", in *Proceedings of the 8th International Conference on Software Engineering*, pp. 386-391, 1985.

[BELADY76] L.A. Belady and M.M. Lehman, "A model of large program development", *IBM Systems Journal*, vol. 15, no. 3, pp. 225-252, 1976.

[BERNS84] G.M. Berns, "Assessing software maintainability", *Communications of the ACM*, vol. 27, no. 1, pp. 134-143, January 1984.

[BOEHM76] B.W. Boehm, "Software engineering", *IEEE Transactions on Computing*, vol. 25, no. 12, pp. 1226-1242, December 1976.

[BOEHM84] B.W. Boehm, "Software Engineering Economics", *IEEE Transactions on Software Engineering*, vol. SE-10, no. 1, pp. 4-21, January 1984.

[BROWN80] P. Brown, "Why does software die ?", *Life-Cycle Management, Infotech State of the Art Report*, vol. 8, no. 7, 1980.

[CARD87] D.N. Card, D.V. Cotnoir and C.E. Goorevich, "Managing software maintenance cost and quality", in *Proceedings of the Conference on Software Maintenance*, Austin, Texas, pp. 145-152, 1987.

[CHAPIN88] N. Chapin, "Controlling the software maintenance process", in *Proceedings of the 6th International Conference on Software Maintenance and Management*, pp. 131-149, 1988.

[COLLOFELLO86] J.S. Collofello, "An analysis of the technical information necessary to perform effective software maintenance", 1986.

[COLLOFELLO87] J.S. Collofello and J.J. Buck, "Software quality assurance for maintenance", *IEEE Software*, pp. 46-51, September 1987.

[COLTER88] M. Colter, "The business of software maintenance", in *Second Software Maintenance Workshop Notes*, Centre for Software Maintenance, Durham, England, September 1988.

[COOK87] C.R. Cook and M. Nanja, "Prototype software complexity metrics tool", *ACM Software Engineering Notes*, vol. 12, no. 2, pp. 58-60, July 1987.

[COOPER89] S.D. Cooper and M. Munro, "Software change information for maintenance management", in *Proceedings of the Conference on Software Maintenance*, Miami, Florida, pp. 279-287, 1989.

[COTE88] V. Cote, P. Bourque, S. Oligny and N. Rivard, "Software metrics: an overview of recent results", *The Journal of Systems and Software*, vol. 8, no. 2, March 1988.

[DEMARCO82] T. DeMarco, *Controlling Software Projects*, Yourdon Press: New York, 1982.

[DRUCKER79] P.F. Drucker, *Management*, Pan Books: London, 1979.

[DUNSMORE84] H.E. Dunsmore, "Software metrics: an overview of an evolving methodology", *Information Processing & Management*, vol. 20, no. 1, pp. 183-192, 1984.

[EPICTETUS00] Epictetus, "The discourses of Epictetus", circa 100 A.D.

[FENTON86] N.E. Fenton and R.W. Whitty, "Axiomatic approach to software metrication through program decomposition", *The Computer Journal*, vol. 24, no. 4, pp. 330-339, August 1986.

[FLETTON88] N.T. Fletton and M. Munro, "Redocumenting software systems using hypertext technology", in *Proceedings of the Conference on Software Maintenance*, Pheonix, Arizona, pp. 54-59, October 1988.

[FOSTER89] J.R. Foster, A.E.P. Jolly and M.T. Norris, "An overview of software maintenance", *British Telecom Technology Journal*, vol. 7, no. 4, pp. 37-46, October 1989.

[FOSTER89a] J.R. Foster, "Priority control in software maintenance", in *Proceedings of 7th International Conference on Software Engineering for Telecomunication Switching Systems*, July 1989.

[FREEMAN90] R.M. Freeman and M. Munro, "Xebra - a Xerox based redocumentation aid", in *Proceedings of the Software Maintenance Association Conference*, Vancouver, pp. 4.35-4.47, 1990.

[GRADY87] R.B. Grady, "Measuring and managing software maintenance", *IEEE Software*, pp. 35-45, September 1987.

[GRADY87a] R.B. Grady and D.L. Caswell, *Software Metrics: Esablishing a Company Wide Program*, Prentice-Hall: Englewood Cliffs, 1987.

[GUNN88] C. Gunn and D. Jolly, "Commercial software -- development versus maintenance", in *Second Software Maintenance Workshop Notes*, Centre for Software Maintenance, Durham, England, September 1988.

[HALSTEAD77] M. Halstead, *Elements of Software Science*, Elsevier North-Holland: New York, 1977.

[HARRISON82] W. Harrison, K. Magel, R. Kluczny and A. DeKock, "Applying software complexity metrics to program maintenance", *IEEE Computer*, vol. 15, no. 9, pp. 65-79, 1982.

[HARRISON84] W. Harrison, "Software complexity metrics: a bibliography and category index", *SIGPLAN Notices*, vol. 19, no. 2, pp. 17-27, 1984.

[HUFF86] K.E. Huff, J.V. Sroka and D.D. Struble, "Quantitative models for managing software development processes", *Software Engineering Journal*, pp. 17-23, January 1986.

[IEEE84] *IEEE Software engineering standards*, IEEE, pp. 31-32, 1984.

[INCE88] D.C. Ince and S. Hekmatpour, "An approach to automated software design based on product metrics", *Software Engineering Journal*, pp. 53-56, March 1988.

[INCE90] D.C. Ince, "Software metrics: an introduction", in *Proceedings of the IEE Computing and Control Division Colloquium on Software Metrics*, January 1990.

[KAFURA87] D. Kafura and G.R. Reddy, "The use of software complexity metrics in software maintenance", *IEEE Transactions on Software Engineering*, vol. SE-13, no. 3, pp. 335-343, March 1987.

[KAPOSI87] A. Kaposi and B.A. Kitchenham, "The architecture of system quality", *Software Engineering Journal*, pp. 2-8, January 1987.

[KENNING90] R.J. Kenning and M. Munro, "Understanding the configurations of operational systems", in *Proceedings of the Conference on Software Maintenance*, San Diego, California, pp. 20-27, November 1990.

[KITCHENHAM84] B.A. Kitchenham, "Program history records: a system of software data collection and analysis", *ICL Technical Journal*, pp. 103-114, May 1984.

[KITCHENHAM84a] B.A. Kitchenham and N.R. Taylor, "Software cost models", *ICL Technical Journal*, vol. 4, no. 1, pp. 73-102, May 1984.

[KITCHENHAM86] B.A. Kitchenham and J.A. McDermid, "Software metrics and integrated project support environments", *Sofware Engineering Journal*, pp. 58-64, January 1986.

[LIENTZ79] B.P. Lientz and E.B. Swanson, "Software maintenance: a user/management tug-of-war", *Data Management*, April 1979.

[LIENTZ80] B.P. Lientz and E.B. Swanson, *Software Maintenance Management*, Addison-Wesley: Reading, MA, 1980.

[MARTIN83] J. Martin and C. McClure, *Software Maintenance: The Problem and Its Solutions*, Prentice-Hall: London, 1983.

[MCCABE76] T. McCabe, "A complexity measure", *IEEE Transactions on Software Engineering*, vol. SE-2, pp. 308-320, December 1976.

[MORISSEY79] J.H. Morissey and L.S.Y. Wu, "Software engineering: an economic perspective", in *Proceedings of the 4th International Conference on Software Engineering*, Munich, Germany, pp. 17-19, September 1979.

[PARIKH82] G. Parikh, "Some tips, techniques, and guidelines for program and system maintenance", in *Techniques of Program and System Maintenance*, Winthrop Publishers: Cambridge, MA, pp. 65-70, 1982.

[PATKAU83] B.H. Patkau, *A Foundation For Software Maintenance*, PhD. Thesis, Department of Computer Science, University of Toronto, December 1983.

[PRESSMAN87] R.S. Pressman, "A practitioners approach", *Software Engineering*, 2nd Edition, 1987.

[ROMBACH87] H.D. Rombach and V.R. Basili, "Quantitative assessment of maintenance: an industrial case study", *Proceedings of the Conference on Software Maintenance*, Austin, Texas, pp. 1134-144, September 1987.

[ROMBACH89] H.D. Rombach and B.T. Ulery, "Improving software maintenance through measurement", *Proceedings of the IEEE*, vol. 77, no. 4, pp. 581-595, April 1989.

[ROOK86] P. Rook, "Controlling software projects", *Software Engineering Journal*, pp. 7-16, January 1986.

[RT312185] RT3121, , "Bug control system", *Technical Report*, British Telecom Research Laboratories, 1985.

[SCHAEFER85] H. Schaefer, "Metrics for optimal maintenance management", in *Proceedings of the Conference on Software Maintenance*, pp. 114-119, 1985.

[SCHNEIDEWIND87] N.F. Schneidewind, "The state of software maintenance", *IEEE Transactions on Software Engineering*, vol. SE-13, no. 3, pp. 303-310, March 1987.

[SWANSON76] E.B. Swanson, "The dimensions of maintenance", in *Proceedings of the 2nd International Conference on Software Engineering*, San Francisco, California, pp. 492-497, 1976.

[WADE88] S. Wade, "Preventive maintenance, the neglected aspect", in *Second Software Maintenance Workshop Notes*, Centre for Software Maintenance, Durham, England, September 1988.

[WAGUESPACK87] L.J. Waguespack and S. Badlani, "Software complexity assessment: an introduction and annotated bibliography", *ACM Software Engineering Notes*, vol. 12, no. 4, pp. 52-71, October 1987.

[WINGROVE86] A. Wingrove, "The problems of managing software projects", *Software Engineering Journal*, vol. 1, no. 1, pp. 3-6, January 1986.

[YAU80] S.S. Yau and J.S. Collofello, "Some stability measures for software maintenance", *IEEE Transactions in Software Engineering*, vol. SE-6, no. 6, pp. 545-552, November 1980.

[YAU85] S.S. Yau and J.S. Collofello, "Design stability measures for software maintenance", *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 849-856, 1985.

Bibliography

[ARNOLD83] R.S. Arnold, *On the Generation and Use of Quantitative Criteria for Assessing Software Maintenance Quality*, PhD. Thesis, Department of Computer Science, University of Maryland, 1983.

[BASILI81] V.R. Basili, "Data collection, validation and analysis", in *Software Metrics: An Analysis and Evaluation* (A.Perlis et al. eds.), MIT Press:Cambridge, MA, 1981.

[BASILI83] V.R. Basili and D.H. Hutchens, "An empirical study of a syntactic complexity flamily", *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 664-672, 1983.

[BOEHM81] B.W. Boehm, *Software Engineering Economics*, Prentice-Hall: Englewood Cliffs, NJ, 1981.

[CALOW89] H. Calow, "The impact of managing maintenance: a case study", in *Third Software Maintenance Workshop Notes*, Centre for Software Maintenance, Durham, England, September 1989.

[CANNING72] R.G. Canning, "That maintenance iceberg", *EDP Analyzer*, vol. 10, no. 10, pp. 1-14, October 1972.

[CONTE86] S. Conte, H.E. Dunsmore and V.Y. Shen, *Software Engineering Metrics and Models*, Benjamin Cummings: Menlo Park, California, 1986.

[FENTON87] N.E. Fenton and A.A. Kaposi, "Metrics and software structure", *Centre for Software and System Engineering*, South Bank Polytechnic, 1987.

[FENTON90] N.E. Fenton, "Software metrics: theory, tools and validation", *Software Engineering Journal*, pp. 65-78, January 1990.

[FENTON91] N.E.Fenton, *Software Metrics*, Chapman & Hall, 1991.

[GUIMARAES83] T. Guimaraes, "Managing application program maintenance expenditures", *Communications of the ACM*, vol. 26, no. 10, pp. 739-746, October 1983.

[LEACH90] R.J. Leach, "Software metrics and software maintenance", *Software Maintenance: Research and Practice*, vol. 2, no. 2, pp. 133-142, June 1990.

[MESCON88] M.H. Mescon, M. Albert and F. Khedouri, *Management*, Harper & Row: New York.

[PETERS88] T. Peters, *Thriving on Chaos: Handbook for a Management Revolution*, Macmillan: London, 1988.

[TURVER91] R.J. Turver, *Metrication of the Reverse Engineering Process*, PhD. Thesis Proposal, Computer Science, School of Engineering and Applied Science, University of Durham, England, 1991.

[WARBURTON83] R.D. Warburton, "Managing and predicting the cost of real-time software", *IEEE Transactions on Software Engineering*, vol. SE-9, no. 5, pp. 562-569, 1983.