1

Software Metrics: What and Why?

Key Points

Software Metrics defined

Potential areas where measurement can be effectively applied

Some underlying principles to always keep in mind

In this chapter I would like to define what it is I mean by the term *Software Metrics*; to discuss in general terms the domain covered by that term according to my definition, and to discuss some of the reasons why an organization may consider the use of Software Metrics.

My introduction to Software Metrics came with a job move from a normal, commercial software engineering environment. The work had also involved some project management. The move required me to change location and I arrived for my first days work fully expecting to be put in charge of a development team only to be offered something completely different. The work they wanted me to get involved in seemed to be dealing with some rather ill defined problems and I also had to figure out how to stop a spreadsheet package beeping at me every few minutes as I made yet another mistake with this new tool.

The first problem we considered concerned the question of change requests that went to make up an enhancement project. Given that the scope of the release might have to be reduced, I had to identify those requests that could most easily be removed from the total package. My most abiding memory of those early days is of other people in the office walking up and asking what I was doing. My answer was invariably "I don't really know but it's good fun." And it was! I was making use of mathematical skills I thought had been consigned to the waste bin, I was meeting people and I had a good level of job satisfaction. What I did not have was a convenient handle by which I could describe my work.

However, we must have been doing something right because the team grew quite quickly from two, myself and my manager, to the grand size of five staff. As we grew we also started to hear of teams in other organizations who were involved in a new fad called *Software Metrics*. Well not that new nor, perhaps, that much of a fad. The ideas we were hearing about had been around for twenty years or more and some organizations had a significant investment in their use. Even more importantly, a few of these organizations seemed to be getting good returns on that investment. Yet it is true to say that the current widespread interest in Software Metrics only really started in about 1984. Since then, like Topsy, it has 'growed and growed!'

This scenario, of my initial involvement with Software Metrics starting from the organization's almost total naiveté regarding the subject to the point where the topic becomes recognized as being important to the business, is not uncommon. It typifies the way in which Software Metrics programs progress and it is worth remembering that a good way to learn about any subject is to start working in that area.

So, our team now had a name that we could associate with our work, "Software Metrics." But what are Software Metrics?

1.1 DEFINITION OF SOFTWARE METRICS

Software Metrics can be defined as:

> *"the continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products."*

As you will see, this definition covers quite a lot.

Software Metrics is all about measurement which in turn involves numbers; the use of numbers to make things better, to improve the process of developing software and to improve all aspects of the management of that process. Software Metrics are applicable to the whole development lifecycle from initiation, when costs must be estimated, to monitoring the reliability of the end product in the field and the way that product changes over time with enhancement. It covers engineers or programmers using techniques to spot error-prone components before they get as far as coding and controlling a project as it progresses so that the fact that it is going to be six months late is recognized as early as possible rather than the day before delivery is due. It even covers organizations determining which of its software products are the cash cows and which are the dogs.

1.2 AREAS OF APPLICATION

There are many different ways in which Software Metrics can be used, some of which are almost specialties in their own right. There are also many ways in which the domain of Software Metrics can be divided. The approach I prefer is to consider specific areas of application of Software Metrics.

The most established area of Software Metrics has to be cost and size estimation techniques. There are many proprietary packages on the market that will provide estimates of software system size, cost to develop a system and the duration of a development or enhancement project. These packages are based on estimation models, the best known of these being the COnstructive COst Model (COCOMO), developed by Barry Boehm, Boehm(1)and subsequently updated based on the experiences of many companies and individuals, Boehm et al (1). Various techniques, that do not require the use of tools are also available.

There has been a great deal of research carried out in this area and this research continues in the United States, Europe and elsewhere. The Department of Defense in the United States, various governments around the world and the European Economic Community sponsor much of it. One thing that does come across strongly from the results of this research work is that organizations cannot rely, solely, on the use of proprietary packages.

Controlling software development projects through measurement is an area that is generating a great deal of interest, both in Europe and the United States. This has become much more relevant with the increase in fixed price contracts and the use of penalty clauses by customers who deal with software developers, not to mention outsourcing, facilities management or "partnership" arrangements that are so prevalent today.

The prediction of quality levels for software, often in terms of reliability, is another area where Software Metrics has an important role to play. Again, there are proprietary models on the market that can assist with this but debate continues about the accuracy of these. The requirement is there, both from the customers point of view and that of the developer who needs to control testing and proving costs. Various techniques can be used now, and this area will become more and more important in the future.

The use of Software Metrics to provide quantitative checks on software designs is also a well established area. Much research has been carried out, and some organizations have used such techniques to very good effect. This area of Software Metrics is also being used to control software products that are in place and that are subject to enhancement.

Other applications of Software Metrics include research into the effect of soft or environmental factors on the effectiveness of the development process. Some years ago, this prompted one large organization to build a development complex specifically designed with the needs of engineers or programmers in mind, (McCue 1978). This option is not open to most organizations but there is usually a great deal that can be done to improve the development process by making changes to the environment that process operates in.

Measurement can be used to identify where change should be concentrated. Just starting to measure soft factors can often lead to useful insights regarding the way in which a process operates and this can lead to benefits to a business by improving performance in key areas such as lead time to market (Ahlgren 1992).

Using measured quantities to compare your own organization with others is an extremely popular area of Software Metrics, especially for senior managers. This is most commonly referred to as "Benchmarking" and indeed, it is often why a measurement program starts in the first place. Benchmarking does, however, involve effort on the part of the organization, so the benefits must be weighed against the costs. One result of using

such an approach is that you can actually discover that you were as bad as you thought but that most other organizations are also as bad! This can be very useful information, but even more importantly such a service can help you identify who is "best in class." Once you have this information you can learn a great deal from it.

Finally, we come to the most common use of Software Metrics: the provision of management information. This includes information about productivity, quality and process effectiveness. It is important to realize that this should be seen as an on going activity. Snapshots of the current situation have their place, but the most valuable information comes when you can see trends in data. Is productivity or quality getting better or worse over time? If so, then why is this happening? What can management do to improve things? The provision of management information is as much an art as a science. Statistical analysis is part of it but the information must be presented in a way that managers can make use of, at the right time and for the right reasons.

All this shows that Software Metrics is a big field! Recognizing this fact is an important step in that it presents a choice. You can probably see many of the areas identified above as having relevance to your business. Do you try to implement Software Metrics in these many forms or do you adopt a "softly-softly" approach tackling one or perhaps two areas first? Both approaches have benefits and dangers associated with them. Resolving this issue within an organization brings us to the first principle of Software Metrics implementation: pragmatism and compromise.

At this point I would like to make a number of points regarding certain principles that, while I will attempt to justify them, I suggest are treated as axioms as far as this book is concerned. These are some of the concepts that form the foundation for the core of this book. I will also take the opportunity to discuss some of the dangers inherent if the approach taken to software engineering and its management excludes Software Metrics.

1.3 PRINCIPLE NUMBER ONE — PRAGMATISM AND COMPROMISE

The first principle of Software Metrics is that of pragmatism and compromise. Implementing Software Metrics in an organization of any size can be difficult. There are many problems that must be overcome and decisions which have to made, often with limited information to hand. One of the first decisions you will face concerns the scope of the work. There is a rule in software development that you do not try something new on a large or critical system and this translates, in the Software Metrics area, to "don't try to do too much." On the other hand, there is evidence that concentrating on too small an area can result in such a limited payback as to invalidate Software Metrics in an organization. As Darlene Brown, a long term metrics activist succinctly puts it: "don't bet your career on a single metric."

Metrics programs that work seem to be a pragmatic compromise between these two extremes. Identifying the key requirements of the organization and satisfying these, while avoiding truly esoteric areas such as predicting the portability of new systems (if this is not a key business requirement) is one approach that can be used to define the scope of a Software Metrics program. The work of Basili and Rombach, among others, has been instrumental in formalizing this strategy (Rombach, 1990). Organizations such as Hewlett Packard and Du Pont de Nemours who are recognized by many as having beneficial metrics programs can show a portfolio of applied measurement techniques that form a multifaceted program. Organizations that introduce one metric, for example the software system sizing technique known as Function Point Analysis (which will be discussed later), and nothing else, often end up without a Software Metrics program. This is truly unfortunate because Function Point Analysis can form the backbone of a successful program but alone it is

not enough! So, help your program to succeed by linking areas of work within the program to specific business requirements in the organization.

This principle of pragmatism and compromise runs through successful Software Metrics program implementations.

1.4 PRINCIPLE NUMBER TWO — MEASURING PEOPLE — DON'T!

Measuring the performance of individuals is extremely dangerous. One organization I know of used individual productivity, in terms of functionality divided by effort, as a major determinant of salary increases. While this may appear attractive to some managers, the organization later stated that this was one of the worst mistakes it ever made. Using measurement in this way is counterproductive, divisive and simply ensures that engineers will rig the data they supply.

An associate of mine often says that management has one chance and one chance only. The first time that a manager uses data supplied by an individual against that individual is the last time that the manager will get accurate or true data from that person. The reasoning behind such a statement is simple — employees do not like upsetting the boss!

I do know of one organization that uses Software Metrics to measure individuals and who seem to do this effectively. A single measure is not used, but instead productivity is combined with the quality of the items produced and the environment of the individual is also considered. This information is used to benefit the individual by identifying training needs and ways that the environmental processes can be improved. Essentially, the information is used constructively rather than destructively.

Two points are worth noting about this second organization:

> It has found that individuals with high productivity levels also tend to produce better quality products than individuals with lower productivity; and,
>
> It is a Japanese organization.

My personal view is that most Western organizations are not mature enough to use the measurement of individuals constructively. Remember that, in the East, measurement is an integral part of life. In Japan, the kyu and dan grades are used to rank or measure many aspects of artistic life. I also believe that the culture in the West is such that any manager should be capable of assessing an individuals worth through knowledge of that individual. A good manager should not need measurement to know if an engineer is pulling his or her weight and it is dangerous to use measurement in this way.

1.5 PRINCIPLE NUMBER THREE — MODELING = SIMPLIFICATION

There is a great temptation in Software Metrics to look for the "silver bullet," the single measure that tells all about the software development process. Currently this does not exist.

Software Metrics depend upon the use of modeling and by definition this involves simplification. We may, for example, model reliability through *mean time to failure* or *defect density*, (these are discussed later), but nobody really believes that these measures truly tell the whole story about reliability. Yet they can still be used to assess this attribute of a software system. In this way, they are a pragmatic attempt to satisfy a real requirement for information.

Models that simplify reality have a large role to play in any Software Metrics program but recognizing and accepting the limitations of these models is another major step forward towards making practical use of Software Metrics.

1.6 PRINCIPLE NUMBER FOUR — ASK NOT FOR WHOM THE BELL TOLLS — ASK WHY?

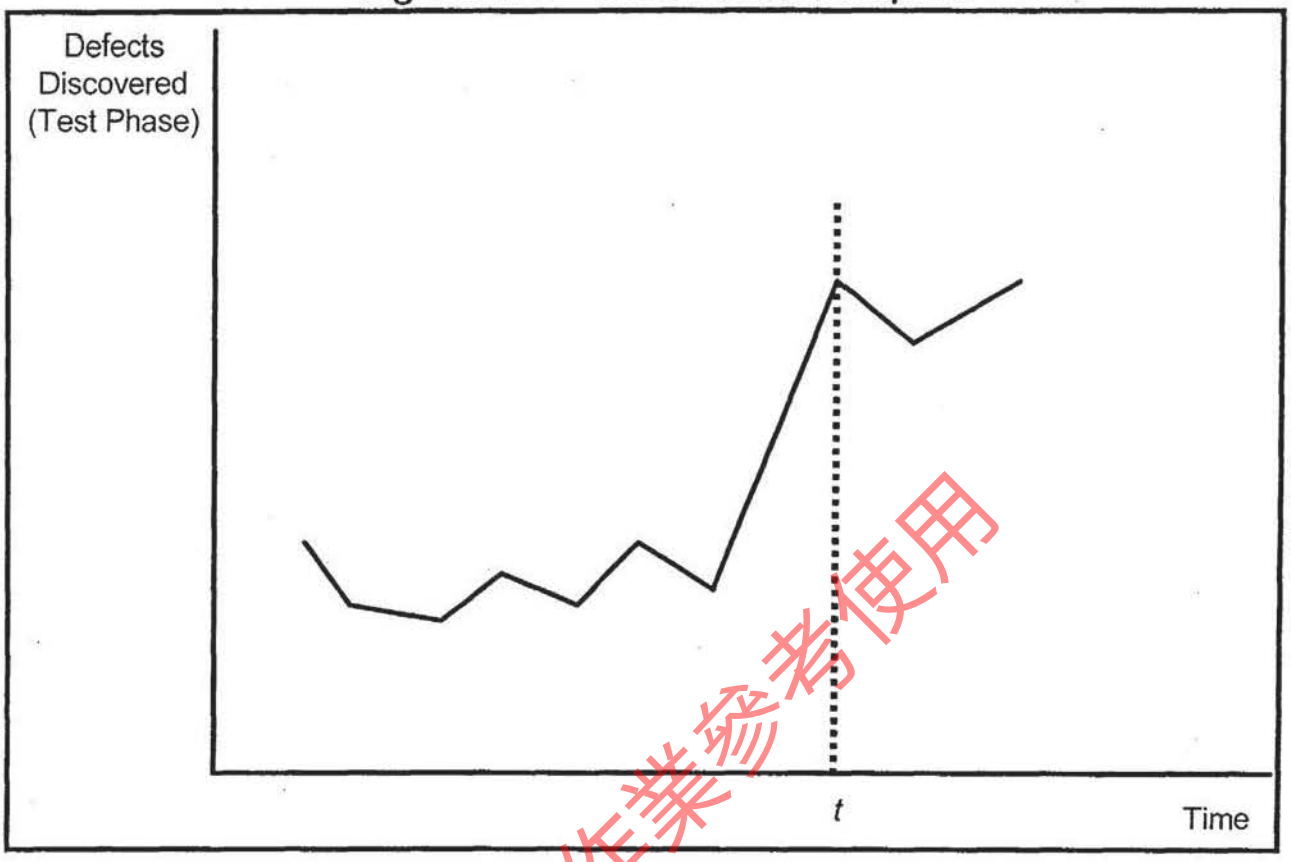
If you attend conferences and seminars that address the topic of Software Metrics, the question often posed is: *"what do I do with all this data now that I've collected it?"*

One glib answer with a great deal of truth behind it is to give it to a statistician. There is a very good reason for this. Statisticians are trained to interpret data, which is an art in itself, but more importantly they realize that, for example, recognizing the presence of a trend is only the first step. The next thing to do is to find out why that trend is present. Why is something happening? Answer that question and you have useful information.

Of course, the answer to the question "why" may not be obvious from the data itself. Usually that data needs to be related back to the environment it came from. To illustrate this, imagine the trivial case where the number of defects found during testing suddenly rises from some norm. Let us also assume that the items being tested are essentially the same.

In ***Figure 1.1***, each point could be an enhancement build on a generic software product where each build is of similar size and complexity.

Figure 1.1 Test Defects Report



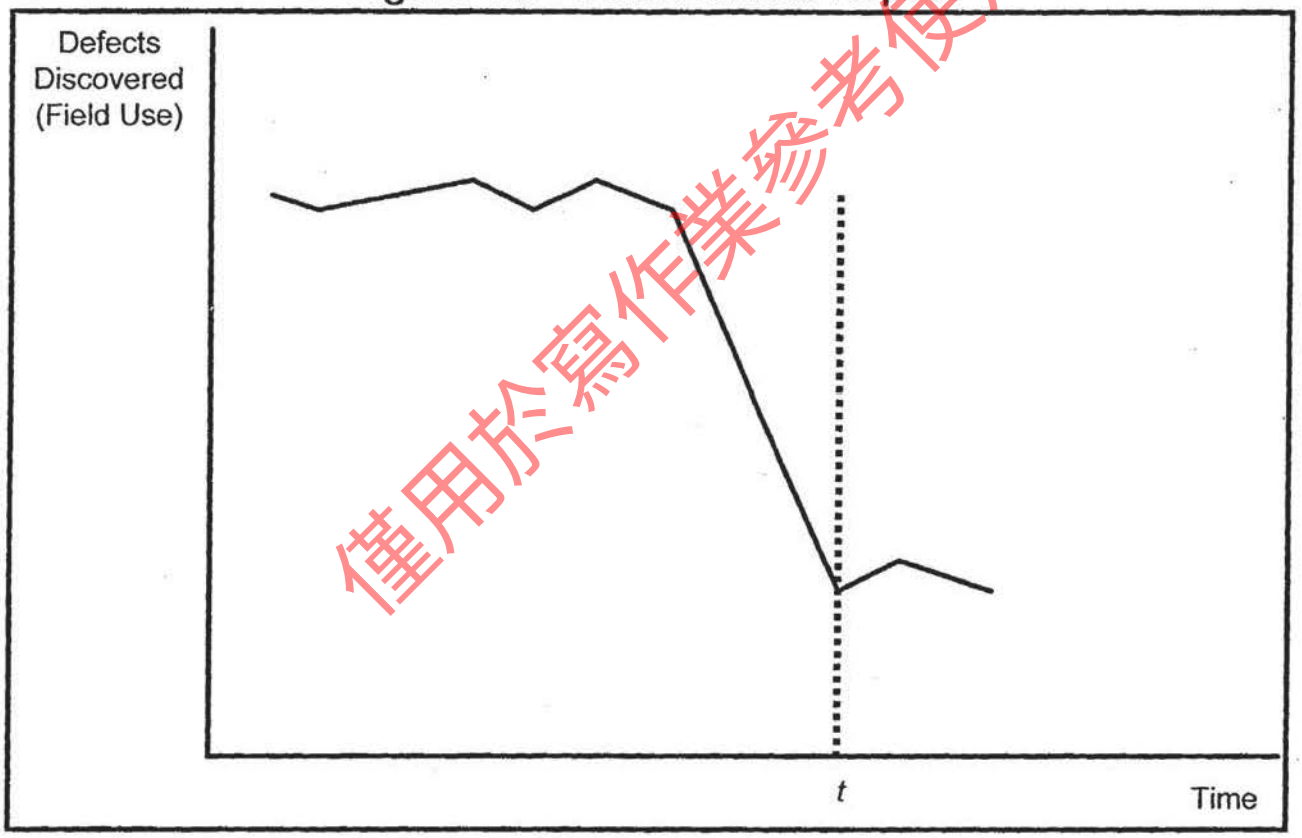
What does our data tell us? Unless we relate it back to the environment, very little. Looking at that testing environment, we may find that at time *t* we changed our test strategy, perhaps introducing a new tool that automatically generates test cases, thus freeing up effort so that more testing could be carried out. An initial knee jerk reaction that quality (in terms of the development deliverables to testing) had gone down would have been false.

Learning to ask "why" is yet another important step towards making practical use of Software Metrics.

1.7 PRINCIPLE NUMBER FIVE — "THE SUM OF THE WHOLE IS GREATER THAN THE CONSTITUENT PARTS"

This well-known axiom is often forgotten or ignored, yet it follows on from the previous principle. Consider again the situation outlined above. Also, imagine that you have the additional information to that in *Figure 1.1* showing field defects decreasing at time *t* as in ***Figure 1.2.***

Figure 1.2 Field Defects Report



Putting the two pieces of information together may show that the new test strategy is discovering more bugs in testing before they are delivered to the customer. This information could be extremely useful, and may lead to the tool being made available to all test teams because of a demonstrable benefit.

Looking at the two items of information has given us much more than looking at each in isolation.

1.8 PRINCIPLE NUMBER SIX — CULTURE SHOCK!

This is more a fact of life than a principle. We must realize that implementing Software Metrics is about changing the way in which people work and think.

The software engineering industry is maturing. Customers are no longer willing to accept poor quality and late deliveries. Management is no longer willing to pour money into the black hole of IT, and more management control is now a key business requirement. Competition is growing. We, as IT professionals, have to change and mature as well, and this can be a painful experience as we find tenets of our beliefs being challenged and destroyed. Some simple statements that could be made by many in our industry together with my interpretation of current management trends will illustrate the point:

> *Software development is so complex it cannot be managed like other parts of the organization.* 'Forget it, it will be managed or the business will find developers and managers who will manage that development!'
>
> *I am only six months late with this project.* 'Fine, you are only out of a job!'
>
> *But you can't put reliability constraints in the contract.* 'Then you don't get the contract!'

The list is almost endless and the message is clear. We will grow up or we will cease to grow. Like any organism, no growth means no life.

Software Metrics cannot solve all our problems but they can enable managers to improve their processes, to improve productivity and quality, to improve the probability of survival. But this is not an easy option. Many metrics programs fail. One reason for this is that organizations and individuals who would never dream of introducing a new computer system without a structured approach ignore the problems of introducing change which is inherent in Software Metrics implementation. Only by treating the implementation of Software Metrics as a project or program in its own right with plans, budgets, resources and management commitment can such an implementation succeed.

Some organizations have already bitten the bullet and now have Software Metrics programs running successfully. From their experiences and my own, I have formed the opinion that the introduction of Software Metrics has a lifecycle not unlike that of other projects. It is that lifecycle model that we will investigate further in section 2 of this book. Before that we will look at some specific applications of Software Metrics in more detail.

1.9 SUMMARY

Software Metrics is simply the application of measurement-based techniques in a software environment. There are many, many ways in which such techniques can be applied and I like to simplify things by considering four main application areas:

Cost estimation

Project control

The use of metrics in the design process

Management information.

Some individuals still ask if the use of Software Metrics is justified. I would say to these people that not only is the use justified, but that it is actually inescapable. The reality today is that business managers are no longer willing to allow their IT departments or their software suppliers to operate without normal business controls. This is a new situation for the IT industry — the honeymoon is over! Either we will have such controls forced upon us, or we will grasp the nettle and start to put our house in order before the iron fist lands. Something of a mixed metaphor but it makes the point!

We used to hear it said that the role of Software Metrics was to enable software engineers to understand the process that they were involved in. I claim today that the role of Software Metrics is to enable engineers and managers to survive in today's business environment. Of course that means a greater understanding but it also means that the pressure is on to start to get it right now.

This chapter is intended to give some meaning to the term Software Metrics. We have looked briefly at various areas where the techniques we collectively call Software Metrics can be applied and we have looked at certain basic principles which, I would suggest, should underlie any measurement initiative. Finally, we have discussed some of the reasons why an organization may wish to invest in a Software Metrics program and in this respect the message is simple. *The use of Software Metrics does not ensure survival but it improves the probability of survival.*