18 Metrics

Metrics

Measurement in Software Development

This theme is about measurement in software development, a field that is known as metrics or software metrics. Before we start (p.18-2) collecting data we must ask questions like

- Why are we collecting these Data? (p.18-2)
- How shall we collect these Data? (p.18-5)
- How will we analyze the Data? (p.18-6)

A method that focuses on this is GQM - The Goal Question Metrics approach (p.18-4).

We also discuss How to define Useful Metrics (p.18-11), and present a few Individual metrics (p.18-14).

This theme is closely related to Process Improvement.

Before we start

It is easy to collect software metrics, either manually or automatically. The real problem starts when we want to use these data. Many of the software metrics activities that have been started, both in industry and academia, have resulted in large data repositories which quickly have turned into data cemeteries, complete with ghouls (software metrics researchers) and the un-dead that always come back to haunt us - like Adam's data on software MTTF.

The reason why software metrics collection ends up in this way is that the work has been started without a clear view of what the people participating in the work want to achieve. In order to make software metrics activities succeed, the following questions must be answered first:

1. Why are we collecting these Data? (p.18-2) This must be decided first since all later decisions will depend on the answer to this first question.
2. How shall we collect these Data? (p.18-5) When shall they be collected and by whom? Here it is important to give a complete description. If we fail here, all our conclusion will be marred by discussions on whether two data sets are comparable, what they mean, whether they are relevant and so on.
3. How will we analyze the Data? (p.18-6) The analysis must enable the company to fulfil the why from point 1.

The following sections will discuss each of the above points in some detail.

Why are we collecting these Data?

Strange as it may seem, this question is seldom asked. The most popular approach is to give somebody a large collection of data and then ask "What do these data mean?". The only reasonable answer to this rather awkward question is "Well, what do you want them to mean?". In reality, the question cannot be answered.

A practical solution to this problem is the GQM - The Goal Question Metrics approach (p.18-4), which we will describe shortly below.

GQM - The Goal Question Metrics approach

As the name implies, GQM consists of defining The Goals (p.18-4), The Questions (p.18-4) and The Metrics (p.18-5) in an orderly fashion. GQM is a metrics method that is closely related to Process Improvement, especially to Measurement based process improvement.

The Goals

In order to answer the why-question, we first need to decide what we want to achieve - i.e. we need to have one or more goals. The goals can be related to the process or the product or to both of these entities.

Examples of goals that we have seen in companies that develop software are:

- Reduce the development time by 30%.
- Reduce the density of after-delivery errors to less than 1 error per 10 000 lines of code.
- Reduce the calendar time for handling customer change requests by 50%.

The Questions

Each stated goal will raise a series of questions. These questions are concerned with such information as:

- What is the current status concerning the goals?
- Which external and internal factors influence the goals?
- How can we understand and describe the relations between the factors that influence the goals?

When the questions are clear, then - and only then - are we ready to start to discuss measurement. The measures that we collect must enable us to answer the questions. This has two big advantages:

1. It helps us to focus our data collection. Since we have a clear goal - answer one of the questions - we will be able to search for, discuss and select metrics in an orderly manner.
2. It conveys the reason for the data collection to all personnel that are involved in the process. Since everybody knows why we collect the data, everybody can contribute to the process with their knowledge. In addition, everybody will be more interested in the results and be more motivated to collect the data in the best possible manner.

The Metrics

The GQM approach does not guaranty that it will be easy to find the data to collect. On the contrary, having clarified the purpose will show that much of the information that we need is not contained in the Standard Metrics (p.18-11). Instead we might find that a substantial portion of the measures can not be collected automatically or that they cannot even be collected in an objective manner at all.

However, the clear purpose that we have been able to describe through the goal-question-metric approach will help us to work through the decisions until we have a set of goals, questions and metrics that will be of value to the participants in the measurement process. The most important is to be sure that the needed metrics really can be collected and that it can be done in an efficient manner.

How shall we collect these Data?

When we - at long last - have decided which data we want to collect, we will have to decide and describe how we shall collect them. The important things to decide on are:

- *Which data shall be collected*? Here we must be precise, so that the data collection activity is repeatable.
- *How shall the data be collected*? Here we must bear in mind that the data we are looking for not always will be objective. In some cases collecting the data is simple - like counting the number of failure reports or the number of lines of code. In other cases, the data may be subjective - like how much time have you spent looking for a solution before you started coding? If the data are subjective it is important to register the uncertainty in the data. If we, for instance, ask a person how long time it takes him to walk to work, the answer will seldom be say 30 minutes. Most likely he will answer something like “Usually between 20 and 35 minutes”. It is important that this uncertainty is preserved in the data registration so that it can be used during the data analysis.
- *Who is responsible*? This must not necessarily be a person - in many cases it is a role in a project or in the organization, for instance the project leader or a function in the company’s economics department.
- *What is the object of data collection*? Here we need to describe the type of document where the data can be found. Examples are the design document for subsystem X, the test report for the FAT, or the code for procedure Y.
- *When shall the data be collected*? Usually, the answer here is not a time or a date, but a situation or the fact that a certain state has been achieved. Examples are just before system test, when the document has passed a review without any remarks and so on.

All the information listed above is needed in order to collect useful data - software metrics - and must be described before any data collection starts. The information can be organized in a table. An example, taken from the ESPRIT-project PERFECT, is shown in Table 18-1 (p.18-6) below.

Table 18-1: Example of how data is to be collected

Information identifier	Information
Data type	real
Measurement unit	person-hours
Measurement scale	real
Measurement object	personal resource reports
Measurement responsible	project manager
Measurement trigger	project is finished
How to measure	accumulate person-hours related to meetings and discussion pertaining to choosing solutions
Expected value	
Observed value	

The fact that some data are collected in a subjective manner gives some people the creeps. In our opinion, however, it is better to collect important data with some uncertainty than to collect other data with great precision but which we do not really need. In addition, some data, which we collect in a subjective manner early in the process may later be collected in an objective manner if they turn out to be important. Since this usually will carry extra costs, it is important to check that the effort is worth the money.

It is a sobering thought that the most important data item that we can collect will for ever be subjective - namely “How satisfied are the customers with our products?”.

How will we analyze the Data?

The purpose of the analysis is to provide answers to the questions that spawned the data collection. Basically, there can be two reasons for analyzing the data, namely to Check for Improvement (p.18-9) in product or process, or to Understand the Process (p.18-7). Usually, we will do both - first we want to understand the process and then we want to change it in order to improve the product or the process itself. We will discuss both approaches in the following sections.

The Role of the Environment

It is not enough to collect data pertaining to the process or product itself. In addition, we need to collect data that describes the environment. This is necessary in order to only compare or pool data that are comparable.

As an example, consider a productivity improvement activity. We have observed a productivity of A function points per day in a project. Then, for the next development project, we introduce a new tool that promises to boost productivity by at least 50%. In this project we observe a productivity of 2A function points per day. The question is now - did the new tool live up to its promises? The answer is that we will never know if we do not collect data concerning the environment.

Let us assume that we, in the example above, have the following additional information:

- In the first project we had high requirements on reliability and safety and had to implement a large MMI-package
- In the second project we had nominal reliability and safety requirements and no MMI-software

It is still possible that the tool boosted the productivity with 50%, but to be reasonably sure we need to be able to assess how much of the productivity increase that comes from the new tool, and how much that comes from the fact that we developed a simpler product.

It is important to bear in mind that if the two projects had been done in the reverse sequence, then we might have observed no productivity increase at all, even if the new tool really increased the productivity.

Understand the Process

The first requirement here is that the organization has a process that is documented, communicated, agreed on and used during development. In addition, the process must be kept stable for two or more development projects. Otherwise, it makes little sense to use resources to understand the process.

When we want to understand the process, we are particularly interested in relationships between “what we do” in the process and the effect that “what we do” has on the product. Process knowledge will usually be of the form:

> When we have a project that can be described by environment description X and the customer’s product requirements can be described by description Y, then an increase in A will reduce the project’s lead time.

In the GQM-approach it is usual to organize this knowledge in a GQM work sheet. The outline of this sheet is shown in Figure 18-1 (p.18-8).

QF This contains a description of what we want to find out, i.e. it is pertaining to ones of our goals	VF Information on factors pertaining to either the project environment or the process used for development
BH This is our baseline hypothesis - what we believe at the present	IM Information on how the VF influence the BH

QF: Quality Focus

VF: Variation Factors

IM: IMpact on baseline hypothesis

BH: Baseline Hypothesis

Figure 18-1: GQM abstraction sheet or work sheet

What we really want to analyze is the baseline hypothesis. The rest of the information is there in order to describe the state of things that must hold in order to make the hypothesis - and thus the knowledge - relevant.

Decision Rules

The last part of information needed is the decision rules. These are rules which decide under which circumstances we will accept, reject or hold our judgement concerning the baseline hypothesis. A simple example is:

> We will believe that code reading is beneficial if we observe a 30% reduction in post-release failures in two projects of type X.

The process of belief adjustment can be described in Figure 18-2 (p.18-9).

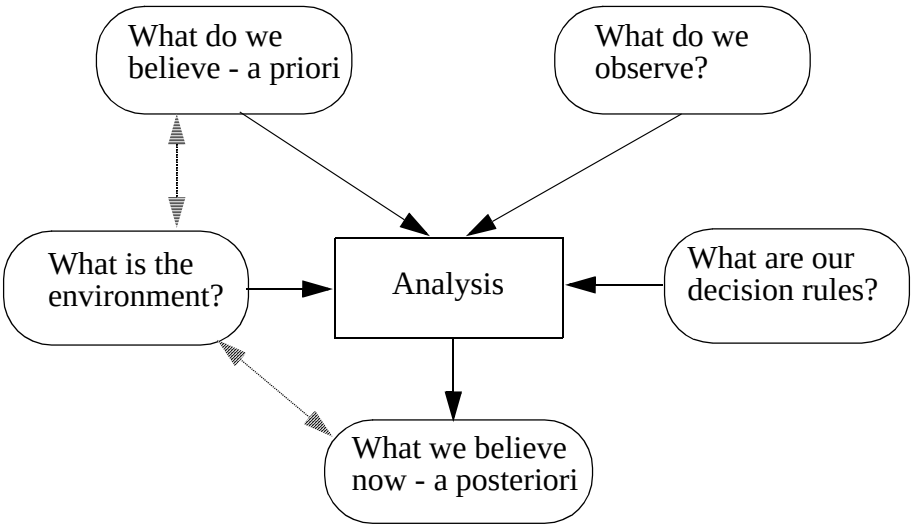

Figure 18-2: How do we change our beliefs

Note the dotted connections between belief and environment in Figure 18-2 (p.18-9). This is inserted into the diagram to keep in mind that all belief or knowledge depends on the environment.

Check for Improvement

When we have changed the process or its environment, we want to find out whether the change has had any - hopefully beneficial - effect on one or more process or product parameters. In order to perform a sensible analysis, we need the following information items:

1. What is the status of the environment for the changed process? This is important when we select data for computation of the baseline. If we forget this, we run the risk of making wrong decision, simply because the data that we compare are not comparable at all.
2. What was the status of the parameter that we intended to improve, before we made the changes? Usually, it is not enough to supply a single number. We must also supply the normal variation, preferably as a confidence interval. If we, for instance, are trying to reduce the failure density, it is not enough to give the mean value. There is a 50% probability of obtaining a failure density that is better than the mean value, even without making any changes. Instead, we could state that the failure density is in the interval {0.1, 0.9} with a 95% probability. Thus, if we observe a failure density of 0.08 there is a 2.5% probability of being wrong when we claim that the failure density is reduced.
3. How large a risk are we willing to take? This information is needed because it will influence the amount of data that we need before making a decision. The less risk we are willing to accept, the more data we will need. In our case, where the risk is con-

nected to process change or changes in tools and methods, the risk can be computed as the probability of being wrong multiplied by the cost of being wrong. This approach is easiest to use when we have quantitative values for probability and cost, but it is also possible to perform a risk assessment in an qualitative manner.

The decision process for a change or improvement can be illustrated in Figure 18-3 (p.18-10).

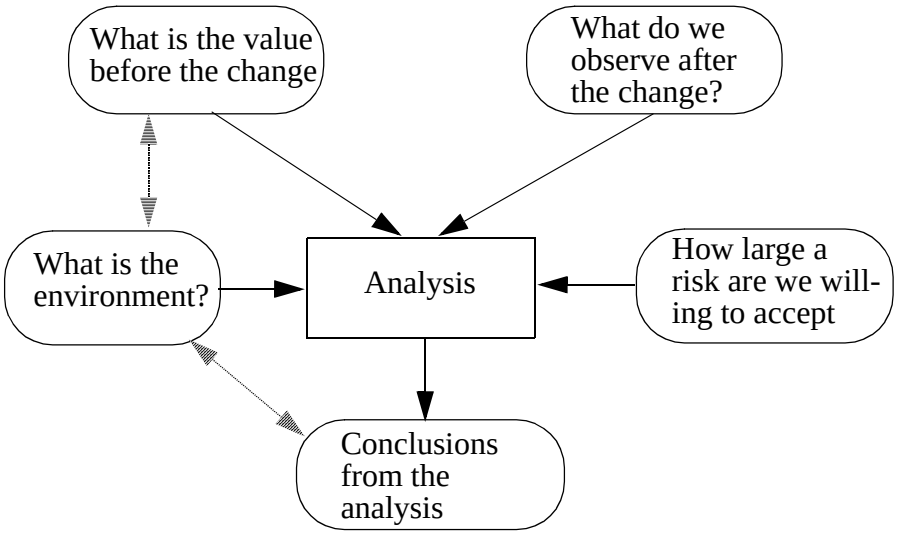

Figure 18-3: Change / improvement analysis

How to define Useful Metrics

The use of metrics related to the product and its development process are important if we shall be able to follow up on process improvement. This section will discuss:

- useful Standard Metrics (p.18-11)
- how to make a complete Definition and Description of Metrics (p.18-12)

How to derive a set of useful metrics for process improvement is detailed in GQM - The Goal Question Metrics approach (p.18-4).

Standard Metrics

There exists a set of metrics that are generally applicable for all software projects. They must, however, be defined anew for each company, since different companies usually will have different definitions. We suggest that you define the following standard metrics for your organization:

- Product (system) size (p.18-11)
- product (system) development costs
- Productivity (p.18-11)
- Reliability (p.18-12)
- Development time (p.18-12)

It is straight forward to see that for instance productivity and product size will depend on the language that is used - for instance assembler versus SDL. Similar dependencies will be relevant for the rest of these metrics. Thus, what follows below are just suggestions.

Product (system) size

Size measures that have been used includes several measures. Examples are lines of code, number of SDL symbols (p.18-14), number of Function points (p.18-14), number of assembler instructions and total volume measured in bytes.

Productivity

In general, productivity is defined as produced value divided by the value of the consumed resources. Usually, we can equate produced value with sales price. A useful alternative is to use the definition that productivity is equal to produced volume divide by the production - or in out case development - cost. How we define the produced value depends on our development methods.

Reliability

In some sense this is easy since Reliability has a standard definition. On the other hand, many other definitions are also used, such as failure density, MTTF and others. Failure density is the easiest to measure and follow up. It is defined as number of failures observed divided by product volume. For product volume, see the discussion under Productivity (p.18-11).

Development time

Usually, development time is used as the calendar time duration of the project. Each organization must, however, decide when a project starts and when it is finished. There are several ways to do this, but we have decided to look at just two of them:

- development for a customer:
  Start is when the company gives a go-ahead to the development project. Finish is when the customer has accepted the product.
- development for a general market, usually started by the marketing department

Definition and Description of Metrics

If we want to collect high quality data, it is important to define and describe the data in a precise way - so also for software metrics. The table below is taken from the ESPRIT-project PERFECT. The description shown in the table is generic definition of the minimum information that must be collected.

Table 18-2: Collected data

Information identifier	Information
Data type	Integer, real, text or boolean
Measurement unit	
Measurement scale	
Measurement object	The object is our data source such as a piece of code, a document or a person working in the project
Measurement responsible	The person who shall collect the data is identified. Even if we uses a tool for the actual data collection, a person must always be responsible for the data collection
Measurement trigger	What event will trigger the data collection. This can be either a specified point in time, a project event or an external event
How to measure	This is a short description of how the data shall be collected

Table 18-2: Collected data

Information identifier	Information
Expected value	This is the value that we have either estimated based on other information, or it is a value that we would expect to see based on a set of hypothesis
Observed value	This is the result that comes out of the measurement activity

Individual metrics

Here we present a number of standard metrics that are often collected: Function points (p.18-14), SDL/PR (p.18-14) and SDL symbols (p.18-14).

SDL/PR

A common metric for SDL size is simply to count the number of lines of SDL/PR, the phrase representation of SDL. A standard line counting tool can be applied on the SDL/PR output from the design tool.

Note that each SDL tool has its own way of formatting the SDL/PR in terms of lines (the standard gives room for this), thus one cannot immediately compare the output of different tools.

One can collect SDL/PR metrics for various SDL components, e.g. procedures, processes, blocks and systems, either by instructing the SDL tool to output SDL/PR for the wanted component, or by using e.g. standard unix *lex* to analyze the SDL/PR.

A future alternative to SDL/PR is CIF, the Common Interchange Format, which is equivalent to SDL/PR, but has (amongst other things) added graphical layout data, and is therefor more voluminous.

SDL symbols

A common metric for SDL systems is to count the number of SDL symbols in a complete system, or in any component of it (block, process, procedure etc.).

SDL symbols can be output from some SDL tools directly, or can be preformed by e.g. unix scripts (*lex/yacc*) of the SDL/PR (p.18-14). This requires more advanced parsing of the SDL/PR, but enables you to calculate a complexity metric by weighing some symbols more than others, or by performing e.g. multiplication of some as apposed to simple addition (e.g. the product of the number of state symbols with the number of input signals could be an element of a complexity metric).

Not enough measurement work as been done in the SDL community for any particular metric to have won general acceptance, and counting lines of SDL/PR (p.18-14) seems to be the most common.

Note that some vendors of metrics tools also support analysis of SDL systems (e.g. the Dutch vendor TechForce).

Function points

Function points were introduced by A.J. Albrecht from IBM. We will, however, use an improved model presented by C.R. Symons. It should be noted that there are several ways to compute functions points and ISO are currently trying to standardize this method.

We will start by introducing some notation:

Θ: adjusted function points

X: unadjusted function points

D: the degree of influence

The adjusted function points are then computed as follows:

$$\Theta = X(0{,}65 + 0{,}01D)$$

X and D are computed from the following tables

Table 18-3: Compute unadjusted function points

	Level of Information Processing Function			
Description	Simple	Average	Complex	Total
External input	x 3	x 4	x 6	
External output	x 4	x 5	x 7	
Logical internal file	x 7	x 10	x 15	
External interface file	x 5	x 7	x 10	
External inquiry	x 3	x 4	x 6	
Total unadjusted function points				

The computation is best shown by an example, see Table 18-6 "Example of how to compute unadjusted function points" (p.18-17).

The degree of influence - D - is computed as shown below. The D-values in Table 18-4 "Compute degree of influence" (p.18-15) have to be inserted for each new product. The appropriate values can be selected from the values show in Table 18-5 "Assigning D-values" (p.18-16):

Table 18-4: Compute degree of influence

ID	Characteristic	D	ID	Characteristic	D
C1	Data communications		C8	On-line update	
C2	Distributed functions		C9	Complex processing	

Table 18-4: Compute degree of influence

ID	Characteristic	D	ID	Characteristic	D
C3	Performance		C10	Reusability	
C4	Heavily used configuration		C11	Installation ease	
C5	Transaction rate		C12	Operational ease	
C6	On-line data entry		C13	Multiple sites	
C7	End user efficiency		C14	Facilitate changes	
Total degree of influence					

The score for each characteristic is assigned as follows:

Table 18-5: Assigning D-values

Influence	D-value
Not present or No influence	0
Insignificant influence	1
Moderate influence	2
Average influence	3
Significant influence	4
Strong influence	5

For an example see Table 18-7 "Example on how to compute degree of influence" (p.18-17).

Examples of metrics

Function points

This is an example that illustrates the use of Function points (p.18-14).

Assume that we have a system with 2 simple and 4 complex external inputs, 5 average outputs, 1 simple logical internal file and 10 simple external inquiries. This will give the following unadjusted function points:

Table 18-6: Example of how to compute unadjusted function points

	Level of Information Processing Function			
Description	Simple	Average	Complex	Total
External input	2 x 3		4 x 6	30
External output		5 x 5		25
Logical internal file	1 x 7			7
External interface file				0
External inquiry	10 x 3			30
Total unadjusted function points				92

The degree of influence - D - is obtained as in Table 18-4 "Compute degree of influence" (p.18-15), and the score for each characteristic is assigned as in Table 18-5 "Assigning D-values" (p.18-16).

Here we assume that we have a system with strong influence from data communications and distributed functions, while there is no requirements on performance. There is a average influence from on-line update, while the rest of the influence are small or non-existent. This gives the following table:

Table 18-7: Example on how to compute degree of influence

ID	Characteristic	D	ID	Characteristic	D
C1	Data communications	5	C8	On-line update	3
C2	Distributed functions	5	C9	Complex processing	0
C3	Performance	0	C10	Reusability	0

Table 18-7: Example on how to compute degree of influence

ID	Characteristic	D	ID	Characteristic	D
C4	Heavily used configuration	0	C11	Installation ease	0
C5	Transaction rate	0	C12	Operational ease	0
C6	On-line data entry	0	C13	Multiple sites	0
C7	End user efficiency	0	C14	Facilitate changes	0
Total degree of influence					13

The total number of function points is thus:

$$\Theta = 92(0{,}65 + 0{,}01 \times 13) = 71{,}76$$

List of figures

List of definitions

MTTF

MTTF is an acronym for Mean Time To Failure. The definition is as follows:

$$\text{MTTF} = \frac{\text{Toal execution tme}}{\text{Number of failures observed}}$$

Total execution time is the execution time accumulated over all installations that run the same product. Thus, if one site has run the product for two months on two computers and another site has run the product for one month on one computer, the total execution time is 2 x 2 + 1 x 1 = 5 months of execution time.

A simple example will show how it works:

$$\text{MTTF} = \frac{40 \text{ months}}{10} = 4 \text{ months}$$

One should avoid the use of MTTF as a reliability measure if the failure rate has varied much over the total execution time.

Reliability

According to IEEE, reliability for software is defined as follows:

> The probability that the software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and the use of the system as well as a function of the existence of faults in the software.

There exists, however, an alternative version which is also used:

> The ability of a program to perform a required function under stated conditions for a stated period of time.