

## Metrics

**John D. McGregor**, Clemson University and Luminary Software LLC, U.S.A.

### Abstract

Strategic goals are of no use if you can't tell whether they have been accomplished. A measurement program is an essential element in the strategic arsenal. In this month's issue of Strategic Software Engineering, I will explore some issues about measurement and effective strategic metrics. I will discuss some specific metrics but also how to setup a measurement program that addresses your needs.

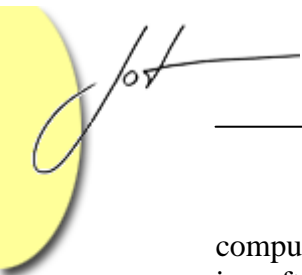
## 1 INTRODUCTION

I once sat through a presentation of the metrics group for a rather large project. By the time the report was completed I knew how many lines of code had been written so far, the number of defects detected, the number of use cases written, and I had mentally drafted a column (for another journal at that time). I was very grateful when the project manager looked at the presenter and said, "yes, but how are we doing?" No doubt, the report had given an accurate description of the project but it had not provided any information that could be used to affirm the present direction of the project or to make corrections to that direction.

Descriptive metrics, which simply summarize a set of measurements, are intended to describe "what is." Descriptive metrics can be useful to quickly give an overview of the current situation. They can be made even more useful when used as a basis for comparison, discovering trends, and summarizing large quantities of more detailed measures. Descriptive metrics must usually be interpreted in the context of some model of the ideal domain such as "All variables will be private."

Prescriptive metrics provide guidance for decision making by producing results which convey an explicit value judgement. Where a descriptive metric might say "The current value is 12 lines of code per day", a prescriptive metric says "Programmer productivity improved 12%." Prescriptive metrics are sometimes computed by combining several measurements of descriptive metrics according to some formula. Programmer productivity improvement would be computed by comparing current productivity with some previous value.

One of the frustrations for project managers is the need for historic data – benchmarks – that can be used to interpret the meaning of descriptive measures or to



compute prescriptive metrics. Industry standard benchmarks have not been widely used in software engineering due to the complex nature of the context that affects the value that is produced. However, some models, such as COCOMO [Boehm 00] are accompanied by default values. There are so many factors that affect the qualities of a software development process that the most effective metrics are ones that measure your process now against your process previously. Most metric programs begin by collecting descriptive measures and eventually use these values to compute metrics that are useful in decision making.

Having given Metrics 101, I will now focus on the strategic use of metrics in software product production. I will discuss ways to define effective metrics and to use the information provided by these measures to improve product production. While I will specify a few metrics, the most important contribution of this column will be to raise some questions that lead you to define your own metrics.

## 2 QM

GQM stands for the Goal-Question-Metric technique developed by Basili [Basili 99] for defining metrics. Several brief industrial strength examples of the technique are given at [Informit 05]. At a strategic level, goal satisfaction is a primary driving force. Every strategic metric begins with the need to track progress toward a specific goal.

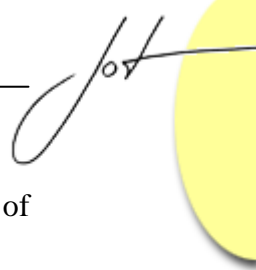
### Strategic Goals

*Strategic* is in the eye of the beholder. That is, *strategic* refers to encompassing some conceptual whole relative to the definer's responsibilities. For a CEO, strategic goals are those that address the entire organization. For a project manager, strategic goals are those that address an entire project, but these are *tactical*, narrower in scope, from the perspective of the CEO.

*Strategic* by its nature also refers to a relatively longer time period than *tactical*, which encompasses some subset of the whole addressed by a strategic view. A time period that is considered long for the project manager will be viewed as a much shorter time period for the CEO. Strategic measures are less sensitive to short-term fluctuations and expose trends rather than incidents.

Tactical objectives often address some implementation of the strategy intended to satisfy the strategic goals. Since a tactical objective addresses a portion of a strategic goal it naturally has a much shorter time horizon than the strategic goal. Tactical objectives provide the short term reaction to longer term directions.

Tactical measures are gathered from a more constrained population and over a shorter time period than strategic measures. Progress on the *reduce time to market* goal can not be measured accurately until several products have been delivered. The tactical objective *reduce time to productivity* can be measured after a few programmers have been



trained which would usually be a much shorter timeframe than delivering a set of products.

Every manager should use a blend of long term, wide reaching strategic goals and a second level of shorter term, more narrowly focused tactical objectives, to guide their organization. Some of the strategic goals we have presented in previous columns include **reduce time-to-market, improved productivity** and **improved quality**. Likewise, the manager should use a blend of strategic and tactical metrics to monitor the organization.

The *reduce time to market* strategic goal could lead a manager to adopt a strategy such as *hire more programmers*. Tactical objectives could then be chosen to *reduce time-to-productivity* achieved by speeding training and using mentoring programs and to *increase automation*, achieved by purchasing or crafting additional software tools. Corresponding metrics should be defined to measure progress on these goals.

### Applying GQM

The fundamental idea behind GQM is to maintain traceability from a goal to the metrics that are intended to measure progress toward that goal. This results in metrics that are more relevant, and more accurate. Essentially, the process begins by taking each goal and identifying questions that, when asked, would reveal what progress is being made toward accomplishing the goal. Each question is then translated into attributes that, if measured, would allow the manager to answer those questions. In many cases the results of the measurements are filtered through a computation resulting in a metric that answers the question. **Figure 1** shows the chain from goal to question to metric for the *reduce time-to-market* strategic goal.

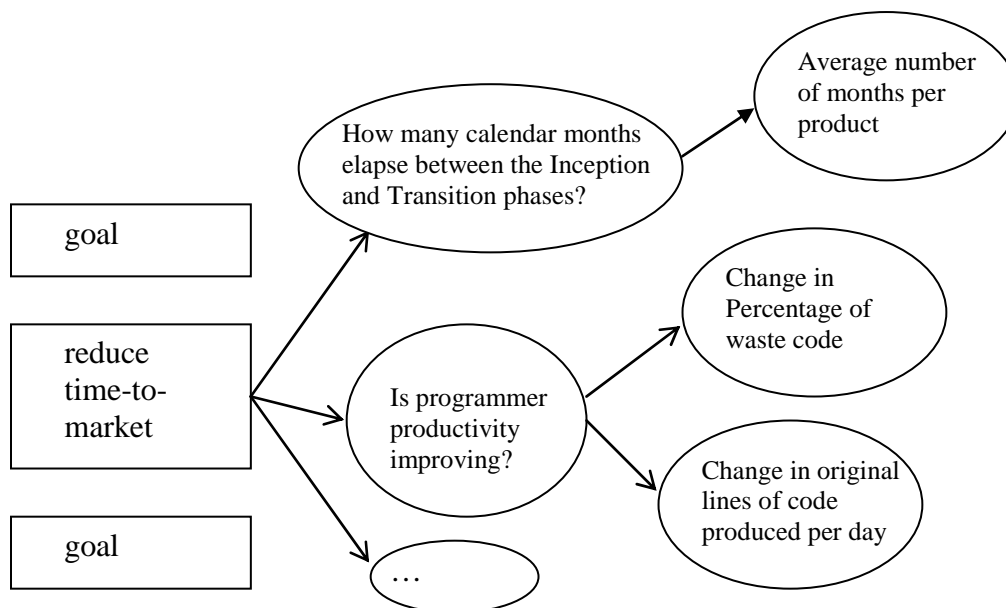


Figure 1 - GQM chain of dependency

## Metric Context

As with any measurement, metrics that will be compared must come from a common context. A personal example, when on a diet you should weigh at the same time of the day wearing similar clothes in order to compare weight over several days. For a software development organization the strategic context is defined by several attributes including:

- The development process – When the activities in the development process are changed, the context changes. Previous measurements that relate to the activities can no longer be compared to new measurements that are related to a different set of activities. For example, many of the metrics used to evaluate a traditional development process are not appropriate for an agile development process. This affects process metrics.
- The design method – Structured design, object-oriented design, aspectual design have different objectives. Even some very fundamental notions such as modularity are viewed differently in these various approaches. This certainly affects the definition of design metrics and some process metrics.
- The type of system – For example, in developing real-time software a design that would be rated very highly by the metrics for an interactive data entry system would be rated very low by the design metrics for real-time software. This affects at least some product metrics.
- The personnel – Experienced developers produce more lines of code with fewer defects than novices. Domain-experience is one part of that experience factor. Loss of several domain experts will, if they are replaced by less domain experienced personnel, render previous time estimates useless. This affects process metrics.

The scope of strategic metrics make them vulnerable to very complex and changing contexts.

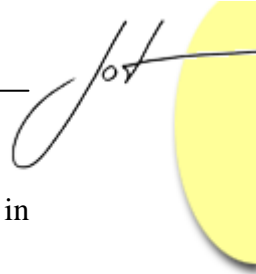
## Strategic Metrics Do's and Don'ts

There are various lists of how to define effective metrics [Lermusiaux 05]. I will just touch on a few points.

**Do update the context, and consequently the metrics, periodically.** Goals change and so does the organizational context. Include a review, and, if necessary, a revision of the metrics with every review of the strategic goals of the organization.

Consider some of the popular metrics about object-oriented software development. One is *depth of the inheritance hierarchy*. This metric was born early in the life of object technology. It is descriptive. For a particular design you get a 4 if there are three levels of children under an abstract class. In certain design philosophies this was treated as not as good a design as one with a hierarchy of depth 2.

Some languages did, and maybe still do, impose a performance penalty on deep hierarchies. However, recent implementations of most languages have eliminated this penalty and some newer languages never had a penalty at all. As a result we see, in the



Java libraries for example, very deep hierarchies that work very well. Changes in language implementation change the need for a particular metric and its interpretation.

If one of the goals is reuse, the deeper the hierarchy the better. However, if one of the goals is more flexible and dynamically modifiable software inheritance is not the best approach and the deeper the hierarchy the less appropriate the design. The design philosophy is part of the metric context.

**Do not rely on a single measure.** Strategic issues are complex. There is usually no single attribute that will give a complete view of progress to the goal. Be certain that the questions asked cover all facets of the goal and that measures provide complete answers to questions.

**Do recognize the limits of the accuracy of the measurements.** Attributes that describe strategic goals are usually very high level. The accuracy of the measurements will be affected by the clarity and concreteness of the definition of the quantity to be measured. Where possible encode the definition of the attribute in a tool that automatically performs the measurement.

**Do account for the variation among measurements.** A single value is more representative of a group of measures if there is little difference among the values. Simply reporting an average is not useful. Because almost none of the actual measurements may have been close to that value. Reporting the average and either a range of values or a variance allows more meaningful interpretation. Strategic attributes are sufficiently encompassing that large variations in values may be more of an issue than exactly what the values are.

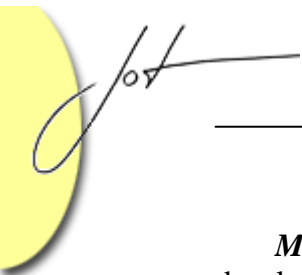
### 3 STRATEGIC METRICS

We now have a technique for defining metrics and some idea of what makes them strategic, so what are some strategic metrics? According to the earlier discussion, a metric is strategic if it covers the width of a manager's interest and takes a long term view relative to the life of the activity. It is effective if it provides unambiguous guidance for decision making. Since *strategic* is relative I will address managers and developers separately. It is not my intention to prescribe a complete set of metrics. Use these as examples and use GQM to define the precise set that is appropriate for your organization.

#### Managers

Managers are usually interested in the two traditional categories of metrics: process and product. I have already mentioned some goals that managers have. Let's consider two: *improve product quality* and *make product production more efficient*.

In order to evaluate *improve product quality*, I will use the definition that quality is satisfaction of the requirements, but with the caveat that those requirements include non-functional quality attributes [Bass 03].



*Make product production more efficient* is very much dependent upon the software development process used by the organization. The Software Engineering Institute (SEI) has provided a strategic model for this in the Capability Maturity Model (CMM) [SEI 05]. I will defer to the large body of literature in this area.

I will describe one type of strategic metric that actually addresses both these areas. In this case the goal under consideration is *improve the quality process*. A strategic metric in this area would be the *defect live range*. This metric is not an easy one to compute but it certainly has strategic importance.

This metric measures the length of time, in terms of the product development life cycle, from a defect being injected into the product to the time it is detected and repaired. Where the defect is detected can usually be determined very accurately. It is sometimes less clear where the defect was injected. Typically some level of analysis will be required to determine this. Table 1 shows a matrix in which the row signifies when a defect was injected and the column indicates when the defect was detected and repaired. Figure 2 shows a graphical representation of the same information.

These values can be reduced to ranges by subtracting the index of the row from that for the column. So the 20 defects injected in phase  $p_1$  and detected in phase  $p_1$  have a live range of zero phases because they did not escape the phase where they were injected. We can then multiply by the number in the cell. In this way we could produce the average defect live range for design defects or requirements defects. For  $p_1$  the computation is:

$$\frac{(1-1)*20 + (2-1)*5 + (3-1)*3 + (4-1)*0 + (5-1)*1 + (6-1)*1}{20 + 5 + 3 + 0 + 1 + 1} = \frac{20}{30} = .67$$

So the average requirements defect lives into the next phase, but most are detected in the requirements phase. The assumption is that the smaller this number, with a lower limit of zero, the better the quality assurance process since the longer a defect lives the more costly it is to repair.

**Table 1 - Inject to Detect Defect Ranges**

	P1	P2	P3	...	...	Pn
P1	20	5	3	0	1	<b>1</b>
P2		30	5	2	0	<b>2</b>
P3			10	4	1	<b>0</b>
...				5	1	<b>1</b>
...					1	<b>0</b>
Pn						<b>0</b>

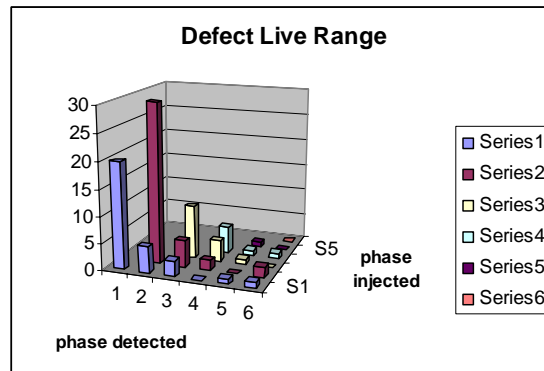


Figure 2 Defect Live Range

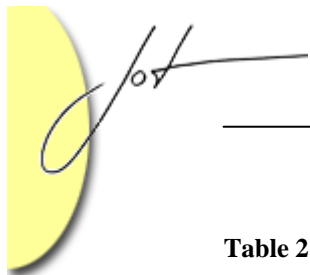
A second type of strategic metric for managers tracks actuals to predictions. For example, the manager may have used some cost-benefit prediction such as the SIMPLE model [Boeckle 04]. At specific milestones the actual costs and benefits to date are computed. These values are easy to compare if the milestones represent either a percentage of the time available for the project or an assumed percentage of the product is completed. Some measures such as return on investment can not be computed at milestones so stay with those measures that can be, such as costs.

## Developers

For developers there are two types of strategic metrics. The first addresses the products they produce and their responsibility to their employer. The second addresses their professional growth and their responsibility to themselves.

I assume that over a strategic horizon every developer has a goal of improving their productivity and quality. The Personal Software Process (PSP) provides the developer with a forest of values that can be measured, see Table 2 [Humphrey 95]. Many of these are very detailed and address very specific actions taken by the developer. These tactical metrics allow the developer to identify specific improvements to make. I suggest that a strategic measure for a developer is a more encompassing, less specific metric: *iterations required to achieve a build*. This is similar to the process yield metric of PSP but I measure through to product build while PSP stops at compilation and PSP measures number of defects while I simply care about iterations. That is, how many cycles does it take to get a finished piece of code? And is that number decreasing over time?



**Table 2 – Sample of PSP metrics**

Defect density	Time per phase
Review rate	Size of base code
Development time ratios	Size of added code
Yield	Size of deleted code
Defect removal leverage	Size of new code intended for reuse

My second assumption is that every developer has the long term goal of remaining professionally current. GQM applies here as well. Most developers at least set yearly goals as part of the company evaluation scheme. Some of us set even longer term goals than that. When a developer sets their strategic professional goals, they should apply GQM to define metrics that will measure satisfaction of those goals.

*Learn a new technology* or *attain a new certification* are typical strategic goals for personal growth. Lets consider *learn a new technology*. One possible question is “How many examples have you studied?” This leads to a descriptive metric which is indicative but not definitive. A more productive question might be “Can I develop a product using the same *number of iterations to build* as with the existing technology?” When the answer is yes, I feel that I have fully learned the new technology.

## 4 MEASUREMENT PROGRAMS

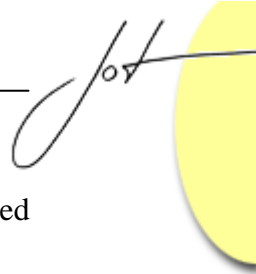
A measurement program collects and disseminates quantitative data for decision making. An organization that operates a measurement program develops a culture that expects decision making to be based on data. The program should produce a suite of metrics that addresses all of the goals, processes and organizational structures. Some of the metrics contribute to the computation of wider-ranging metrics at higher levels of the organization and some are intended for immediate consumption.

Why does CMM not include quantitative process management until Level 4? Because it takes a certain maturity to consistently take best advantage of this type of specific input. But that does not mean that organizations should not begin to collect, review and react to quantitative data until then. By the time you reach level 4 you should have sufficient historic data to understand its importance and to be able to do historic comparisons.

When do I measure? Measures should be collected periodically. There is no one period that makes sense in all cases. I will give two answers.

- Some measures are applied at project milestones such as the end of the first iteration or increment. Certain product-based measures only make sense at times





- when a certain point in development is reached. Development process-oriented measures also are applied at milestone.
- Some management process-based measures are applied at regular calendar intervals such as every quarter. Measures that are to be contrasted to annual budgets or stockholder meetings are applied at these intervals.

How do I measure? Measurements need to be made in a structured manner that ensures that the measures are giving correct answers and consistent answers.

- Take measurements at regular intervals but avoid periods of chaos. I once asked an industrial training class, “How is it going?”, as a casual opening remark. It turns out many of them had been given 30 days notice the previous day. Measuring their productivity that day would not have been a good idea. (Nor was asking how things were going!)
- Use a standard measuring instrument. At one time the yard was defined as the distance from the nose to the tip of the finger of a member of British royalty. Rather, use standardized questionnaires, formulas, or industry standard data definitions so that the measures are repeatable.

Do I need a metrics program if I am doing agile development? I think every organization should have a metrics program even if the organization is currently happy with their processes and products. Obviously an agile development project needs an agile metrics program. It needs metrics that are oriented toward individuals and that have a short time horizon. GQM provides a natural progression to determine what should be measured regardless of the process model being used.

## 5 SUMMARY

The most important, and most difficult, issue regarding a measurement program is establishing a culture that respects and utilizes data in decision making. The most relevant metrics will not provide any value if they are not consulted. Ironically, the best way to establish this culture is to define relevant, effective metrics that provide useful information. In my view this means carefully choosing a small set of strategic metrics that can summarize information that is too distributed for one person to experience personally. Periodically measurements should be used to measure the costs and benefits of the measurement program.

A Project DashBoard is a summary of current values for an essential set of metrics the manager should be watching to understand the health of their organization. This device works because it limits the manager to a small set of metrics. It is effective because it makes it easy for the manager to scan the values. And, since it is easy the manager might actually use it. The DashBoard should contain strategic and tactical metrics and both descriptive and prescriptive metrics. This blend provides comprehensive coverage that provides useful information for decision making.

## REFERENCES

- [Basili 99] V. R. Basili, L.C. Briand, and W. L. Melo. "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *IEEE Trans. on Software Engineering*, vol. 25, No. 1, (1999) 91-121.
- [Bass 03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*, Second Edition, Addison-Wesley, 2003.
- [Boeckle 04] Guenter Boeckle, Paul Clements, John D. McGregor, Dirk Muthig and Klaus Schmid. "Computing Return on Investment for Software Product Lines", *IEEE Software*, vol. 21, No. 3, May/June 2004.
- [Boehm 00] Barry W. Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with COCOMO II*, Prentice-Hall, 2000.
- [Humphrey 95] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, 1995.
- [Informit 05] <http://www.informit.com/articles/article.asp?p=30306&seqNum=4>
- [Lermusiaux 05] Yves Lermusiaux. Characteristics of a Good Metric. [http://www.ilogos.com/en/expertviews/articles/strategic/20030709\\_YL.html](http://www.ilogos.com/en/expertviews/articles/strategic/20030709_YL.html).
- [SEI 05] Capability Maturity Model, <http://www.sei.cmu.edu/cmm/>.

## About the author

**Dr. John D. McGregor** is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests include software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at [johnmc@lumsoft.com](mailto:johnmc@lumsoft.com).