One common goal is to evaluate the maturity of a software organization and its process in order to improve it. We shall see how the Goal-Question-Metric paradigm is applied to do such an evaluation.

3.2.1 Goal-Question-Metric paradigm

Many metrics programs begin by measuring what is convenient or easy to measure, rather than by measuring what is needed. Such programs often fail because the resulting data is not useful to the developers and maintainers of the software. A measurement program can be more successful if it is designed with the goals of the project in mind. The GQM approach provides a framework involving three steps:

1. List the major goals of the development or maintenance project.

2. Derive from each goal the questions that must be answered to determine if the goals are being met.

3. Decide what must be measured in order to be able to answer the questions adequately.

By deriving the measurements in this way, it becomes clear how to use the resulting data. As an example, Figure 3.2 illustrates how several metrics might be generated from a single goal.

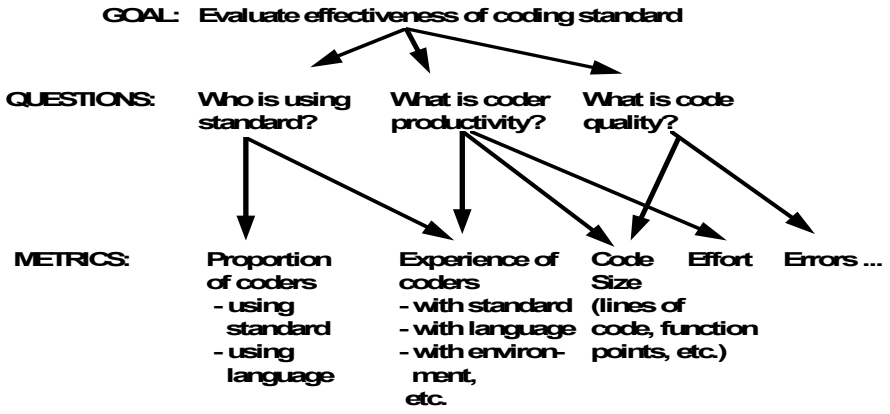

Figure 3.2: Example of deriving metrics from goals and questions

Suppose your overall goal is to evaluate the effectiveness of using a coding standard, as shown in the figure. That is, you want to know if code produced by following the standard is superior

in some way to code produced without it. To decide if the standard is effective, you must ask several key questions. First, it is important to know who is using the standard, so that you can compare the productivity of the coders who use the standard with the productivity of those who do not. Likewise, you probably want to compare the quality of the code produced with the standard with the quality of non-standard code.

Once these questions are identified, you must analyze each question to determine what must be measured in order to answer the question. For example, to understand who is using the standard, it is necessary to know what proportion of coders is using the standard. However, it is also important to have an experience profile of the coders, explaining how long they have worked with the standard, the environment, the language, and other factors that will help to evaluate the effectiveness of the standard. The productivity question requires a definition of productivity, which is usually some measure of effort divided by some measure of product size. As shown in the figure, the metric can be in terms of lines of code, function points, or any other metric that will be useful to you. Similarly, quality may be measured in terms of the number of errors found in the code, plus any other quality measures that you would like to use.

In this way, you generate only those measures that are related to the goal. Notice that, in many cases, several measurements may be needed to answer a single question. Likewise, a single measurement may apply to more than one question. The goal provides the purpose for collecting the data, and the questions tell you and your project how to use the data.

Example 3.8: AT&T used GQM to help determine which metrics were appropriate for assessing their inspection process. [Barnard and Price 1994] Their goals, with the questions and metrics derived, are shown in Table 3.3 below.

Table 3.3: Examples of AT&T goals, questions and metrics

Goal	*Questions*	*Metrics*
Plan	How much does the inspection process cost?	Average effort per KLOC Percentage of reinspections
	How much calendar time does the inspection process take?	Average effort per KLOC Total KLOC inspected
Monitor and control	What is the quality of the inspected software?	Average faults detected per KLOC Average inspection rate Average preparation rate
	To what degree did the staff conform to the procedures?	Average inspection rate Average preparation rate Average lines of code inspected Percentage of reinspections
	What is the status of the inspection process?	Total KLOC inspected
Improve	How effective is the inspection process?	Defect removal efficiency Average faults detected per KLOC Average inspection rate Average preparation rate Average lines of code inspected
	What is the productivity of the inspection process?	Average effort per fault detected Average inspection rate Average preparation rate Average lines of code inspected

What is not evident from the GQM tree or table is the model needed to combine the measurements in a sensible way so that the questions can be answered. For example, the tree in Figure 3.2 suggests measuring coder productivity; this attribute may be measured in terms of effort per line of code, but that relationship is not explicit in the tree. Thus, the GQM approach must be supplemented by one or more models that express the relationships among the metrics.

> **Example 3.9:** Once AT&T researchers and developers generated the list of metrics in Example 3.8, they specified metrics equations and the data items that describe what the metrics really mean. For instance, the average preparation rate is a function of the total number of lines of code inspected, the preparation time for each inspection, and the number of inspectors. A model of the metric expresses the average preparation rate as an equation. First, the preparation time for each inspection is divided by the number of inspectors; then, the sum over all inspections is calculated and used to normalize the total number of lines of code.

Even when the metrics are expressed as an equation or relationship, the definition is not always clear and unambiguous. The tree does not tell you how to measure a line of code or a function point, only that some measure of code size is needed to answer a question about productivity. Additional work is needed to define each metric. In cases where no objective measure is available, subjective measures must be identified.

In general, typical goals are expressed in terms of productivity, quality, risk, customer satisfaction and the like, coupled with verbs expressing the need to assess, evaluate, improve or understand. It is important that the goals and questions be understood in terms of their audience: a productivity goal for a project manager may be different from that for a department

manager or corporate director. To aid in generating the goals, questions and metrics, Basili and Rombach provided a series of templates:

Templates for goal definition:

- *Purpose:* To (characterize, evaluate, predict, motivate etc.) the (process, product, model, metric etc.) in order to (understand, assess, manage, engineer, learn, improve etc.) it.

  Example: To *evaluate* the *maintenance process* in order to *improve* it.

- *Perspective:* Examine the (cost, effectiveness, correctness, defects, changes, product measures etc.) from the viewpoint of the (developer, manager, customer etc.)

  Example: Examine the *cost* from the viewpoint of the *manager*.

- *Environment:* The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints etc.

  Example: The maintenance staff are poorly motivated programmers who have limited access to tools.

A wide range of goals have been derived along with questions and associated metrics to develop measurement programs in industry.

> **Example 3.10:** Fuggetta et al. report on the use of GQM on a software process at the Digital Software Engineering Center in Gallarate, Italy. (Fuggetta et al. 1998) The group identified five goals including the following goal, which has a format consistent with our templates:
>
> > "Analyze the design and qualification phases of the development process for the purpose of evaluating failure detection effectiveness from the viewpoint of the management and the development team."
>
> The measurement group derived 35 questions that could be answered using data from approximately 50 metrics.

Basili and Rombach as well as van Soligen and Berghout (1998) provide separate guidelines for defining product-related questions and process-related questions. Steps involve defining the process or product, defining the relevant attributes, and obtaining feedback related to the attributes. What constitutes a goal or question may be vague, and several levels of refinement may be required for certain goals and questions; that is, a goal may first have to be related to a series of subgoals before questions can be derived.

We can relate the GQM templates to the attribute framework introduced earlier in this chapter. A goal or question can be associated with at least one pair of entities and attributes. Thus, a goal is stated, leading to a question that should be answered so that we can tell if we have met

our goal; the answer to the question requires that we measure some attribute of an entity (and possibly several attributes of an entity, or several attributes of several entities). The use of the measure is determined by the goals and questions, so that assessment, prediction and motivation are tightly-linked to the data analysis and reporting.

Thus, GQM complements the entity-attribute measurement framework. The results of a GQM analysis are a collection of measurements related by goal tree and overall model. However, GQM does not address issues of measurement scale, objectivity, or feasibility. So the GQM measures should be used with care, remembering the overall goal of providing useful data that can help to improve our processes, products and resources.

3.2.2 Measurement for process improvement

One common goal in the software industry is process improvement. Software processes can range from chaotic and ad hoc, to well-defined and well-managed. A more mature process is more likely to develop software that is reliable, adaptable, delivered on time, and within budget. Measurement quantifies the relationhips between the processes, products, resources, methods and technologies of software development. Thus, measurement plays a key role in evaluating and improving software processes. We will look at a popular process evaluation technique, the Software Engineering Institute's (SEI's) Capability Maturity Model Integrated (CMMI®) for Development, from the perspective of GQM. (CMMI Product Team, 2010)

The CMMI for Development provides an ordinal ranking of development organizations from *initial* (the least predictable and controllable, and least understood) to *optimizing* (the most predictable and controllable), which is described by the CMMI Product Team as follows:

1. Initial: Level 1 processes are ad hoc and "success depends on the competence and heroics of the people in the organization".
2. Managed: Level 2 processes are planned; "the projects employ skilled people … have adequate resources ... involve relevant stakeholders; are monitored, controlled, and reviewed …".
3. Defined: Level 3 "processes are well characterized and understood, and are described in standards procedures, tools and methods."
4. Quantitatively Managed: A Level 4 "organization and projects establish quantative objectives for quality and process performance and use them as criteria in managing projects."
5. Optimizing: A Level 5 "organization continually improves its processes based on a quantitative understanding of its business objectives and performance needs."

(CMMI Product Team, 2010)

The SEI CMMI distinguishes one level from another in terms of key process activities going on at each level. Although it appears that measurement is only important at Level 4, actually measurement is used in evaluation at each level. Specific goals, questions, and metrics are developed to assess whether an organization has reached a particular level. Generally, to reach

a particular level, an organization must have measurement values that give the desired answer to each question at that level.

To reach CMMI-Development version 1.3 Level 2 Managed, a process must satisfy fifteen goals in seven process areas:

1. Configuration management goals: establish baselines, track and control changes, establish integrity.
2. Measurement and analysis goals: align measurement and analysis activities, provide measurement results.
3. Project monitoring and control goals: monitor project against plan, manage corrective actions to closure.
4. Project planning goals: establish estimates, develp a project plan, obtain commitment to the plan.
5. Process and quality assurance goals: objectively evaluate processes and work products, provide objective insight.
6. Requirements management goals: manage requirements.
7. Supplier agreement management goals: establish supplier agreements, satisfy supplier agreements.

Answers to questions related to the goals in each process area determine whether or not a goal is achieved. These questions tend to be measured by yes or no answers—either a process performs an activity at a required level or it does not. For example, answers to the following questions determine whether or not configuration management process area goals are met:

Does the development process

1. Identify configuration items?
2. Establish a configuration management system?
3. Create or release baselines?
4. Track change requests?
5. Control changes to configuration items?
6. Establish configuration management records?
7. Perform configuration audits?

Trained evaluators determine whether each question is answered yes or no based on interviews with process participants and evaluations of process documents. Evaluators determine whether the process performs each activity as required for Level 2 certification. To achieve a Level 2 rating, the answer must be yes to questions concerning 65 practices related to 17 goals.

To achieve Level 3 Managed, an organization must satisfy all Level 2 goals, plus 27 additional goals in 11 process areas. The answers to questions concerning 88 different practices determine whether or not the goals are met. Levels 4 and 5 add additionl process areas, goals, and questions about practices.

The CMMI and other models, such as ISO-9000, SPICE, and Bootstrap, share a common goal and approach, namely that they use process visibility as a key discriminator among a set of "maturity levels." That is, the more visibility into the overall development process, the higher the maturity and the better managers and developers can understand and control their

development and maintenance activities. At the lowest levels of maturity, the process is not well understood at all; as maturity increases, the process is better-understood and better-defined. At each maturity level, measurement and visibility are closely related: a developer can measure only what is visible in the process, and measurement helps to enable and increase visibility. Thus, the five-level maturity scale such as the one employed by the CMMI is a convenient context for determining what to measure first and how to plan an increasingly comprehensive measurement program.

Successful metrics programs start small and grow according to the goals and needs of a particular project. [Rifkin and Cox 1991] To be successful, a metrics program should be planned in context with an organizations "processes, structures, climate, and power". [Frederiksen and Mathiassen, 2005] A metrics program should begin by addressing the critical problems or goals of the project, viewed in terms of what is meaningful or realistic at the project's maturity level. The process maturity framework then acts as a guideline for how to expand and build a metrics program that not only takes advantage of visibility and maturity but also enhances process improvement activities.

Next, we explain how to use the process maturity framework, coupled with understanding of goals, to build a comprehensive measurement program.

3.2.3 Combining GQM with process maturity

Suppose you are using the Goal-Question-Metric paradigm to decide what your project should measure. You may have identified at least one of the following high-level goals:

- Improving productivity
- Improving quality
- Reducing risk

Within each category, you can represent the goal's satisfaction as a set of subgoals, each of which can be examined for its implications for resources, products and process. For example, the goal of improving productivity can be interpreted as several subgoals affecting resources:

- Assuring adequate staff skills
- Assuring adequate managerial skills
- Assuring adequate host software engineering technology

Similarly, improving productivity with products can mean

- Identifying problems early in the life cycle
- Using appropriate technology
- Reusing previously-built products

Next, for each subgoal, you generate questions that reflect the areas of deepest concern. For example, if

Improving productivity

is your primary goal, and then

Assuring adequate staff skills

is an important subgoal, you create a list of questions that you may be interested in having answered, such as:

1. Does project staffing have the right assortment of skills?
2. Do the people on the project have adequate experience?

Similarly, if you have chosen

Improving quality

with a subgoal of improving the quality of the requirements, then the related questions might include:

1. Is the set of requirements clear and understandable?
2. Is the set of requirements testable?
3. Is the set of requirements reusable?
4. Is the set of requirements maintainable?
5. Is the set of requirements correct?

However, before you identify particular measurements to help you answer these questions, it is useful to determine your process maturity level. Since process maturity suggests that you measure only what is visible, the incorporation of process maturity with GQM paints a more comprehensive picture of what measures will be most useful to you.

For example, suppose you want to answer the question:

Is the set of requirements maintainable?

If you have specified a process at level 1, then the project is likely to have ill-defined requirements. Measuring requirements characteristics is difficult at this level, so you may choose to count the number of requirements and changes to those requirements to establish a baseline. If your process is at level 2, the requirements are well-defined and you can collect additional information: the type of each requirement (database requirements, interface requirements, performance requirements, and so on) and the number of changes to each type. At level 3, your visibility into the process is improved, and intermediate activities are defined, with entry and exit criteria for each activity. For this level, you can collect a richer type of measurement: measuring the traceability of each requirement to the corresponding design, code

and test components, for example, and noting the effects of each change on the related components. Thus, the G-Q analysis is the same, but the M recommendations vary with maturity. The more mature your process, the richer your measurements. In other words, the more mature your process, the more mature your measurements.

Moreover, maturity and measurement work hand-in-hand. As the measurements provide additional visibility, aiding you in solving project problems, you can use this approach again to expand the measurement set and address other pressing problems. Thus, your measurement program can start small and grow carefully, driven by process and project needs.

Using goals to suggest a metrics program has been successful in many organizations and is well-documented in the literature. (For examples, see [Grady and Caswell 1987], [Rifkin 1991], [Pfleeger 1993]., [Mendonca and Basili 2000], and [van Solingen and Berghout 1999]) The GQM paradigm, in concert with process maturity, has been used as the basis for several tools that assist managers in designing measurement programs. For example, the ami project, funded by the European Community under its ESPRIT program, has a handbook and tool that suggest the use of both GQM and process maturity. [Pulford 1995]

GQM has helped us to understand why we measure an attribute, and process maturity suggests whether we are capable of measuring it in a meaningful way. Together, they provide a context for measurement. Without such a context, measurements can be used improperly or inappropriately, giving us a false sense of comfort with our processes and products. In the next section, we look at the need for care in capturing and evaluating measures.

3.3. Applying the framework

In Chapter 1, we introduced several diverse topics involving software measurement. In particular, we saw how the topics relate to essential software engineering practices. When divorced from any conceptual high-level view of software measurement and its objectives, many of the activities may have seemed unrelated. In this section, we revisit the topics to see how each fits into our unifying framework and to describe some of the issues that will be addressed in later chapters of this book. That is, we look at which processes, products, and resources are relevant in each case, which attributes (and whether these are internal or external) we are measuring, and whether the focus is on assessment or prediction. Using the same topic headings before, we describe these activities briefly; details will be provided in subsequent chapters.

3.3.1 Cost and effort estimation

Cost and effort estimation focuses on predicting the attributes of cost or effort for the development process. Here, the process includes activities from detailed specification through implementation. Most of the estimation techniques present a model; we examine a popular approach to see how cost and effort estimation are usually done.