Measurement Modeling Technology

Jim Lawler, *Predicate Logic*
Barbara Kitchenham, *Keele University*

Several recent software engineering trends are pushing software companies to adopt formal measurement programs:

- Organizations that have reached Level 2 of the Software Engineering Institute's Capability Maturity Model are working toward implementing Level 3 processes, which require a solid, meaningful measurement program.
- More software companies are outsourcing and subcontracting pieces of development work to distant locations to gain access to inexpensive and skilled labor. Although advances in communication technology have made geographically dispersed development possible, effectively monitoring the progress of a team halfway around the world is still difficult.
- Many projects, particularly government defense developments, use integrated product teams. Collecting and collating data from a mixed team of government and contractor personnel, each with different development and reporting systems, poses a significant challenge.

Software measurement programs aren't helpful if the data they need is missing, invalid, or delayed. This measurement modeling technology supports measurement automation, thus avoiding these problems and enabling reliable metrics comparison across projects, departments, and companies.

Despite this need for good measurement programs, existing programs seem unable to deliver the required capabilities. In addressing the topic of benchmarking, a recent issue of *IEEE Software* clearly identified the need for better measurement technology. John McGarry suggested that measurement technology was insufficiently mature to support benchmarking.[1] He proposed that the measurement community "adopt consistent measurement conventions and definitions for corporate-controlled processes." He also remarked on the need for consistent measurement models and standards.

Other articles in the special issue also provided insight into the problems of existing measurement programs. Katrina Maxwell dis-

0740-7459/03/$17.00 © 2003 IEEE

cussed data collection problems, pointing out that it is difficult to ensure consistent data collection within a single company, let alone define the cross-company standards for measures and metrics necessary for benchmarking.[2] In describing a benchmarking project, James Heires illustrated the problems that plague measurement programs.[3] He was unable to analyze 47 of the 75 completed projects because of "incomplete or unobtainable core metrics, unavailable knowledgeable project team members, and prematurely canceled projects." Preparing the data for inclusion in the benchmarking database required "validating the collected data for correctness and consistency." Even so, questions about data validity surfaced during the analysis phase.

These comments indicate that current measurement programs suffer both from a lack of metrics standards that reduces data comparability, and from invalid and missing data that cause delays in data analysis and reduce confidence in any reports' validity. The only way to avoid these problems is for companies to fully automate their measurement programs and integrate them directly with their software development support tools. This measurement modeling technology (MMT)—implemented as the commercial tool TychoMetrics (www.tychometrics.com)—addresses the need for fully automated software measurement.

What the MMT does

Although in this article we concentrate on the measurement technology itself, that technology could not work effectively without being embedded in an automated tool such as TychoMetrics. Together, the MMT and TychoMetrics address the following components of a measurement program:

- Rigorous definition both of the atomic measurements to be collected and of the metrics (that is, functions of atomic measures) to be calculated and reported. Rigorous definition is an essential element of standardization.
- Definition of the links between the measures and the development process support tools that extract or generate atomic measures (project management tools, defect logs, configuration management tools, and so on).
- Automatic interrogation of development process tools at predetermined intervals (for example, every night at 2:00 a.m.). TychoMetrics does not itself extract data from software artifacts; it provides an interface to the data output facilities of a company's software support tools.
- Data collection, collation, and storage. TychoMetrics views data collection as a sampling process. Each time the tool obtains data from a data source or (sources) and stores it in the data repository, it provides a new sample at a new time. Thus, it never overwrites data values. By default, it doesn't delete data, but it can do so if required. Our experience has shown that keeping past data stabilizes the measurement process and yields insights into the development process's stability or maturity.
- Integrated mechanisms for the measurement system's evolution.
- Model-driven definition of the database structure. Measurement schemas and measurement object models are the primary components defining the database's storage structure. The database design uses a general structure (class) that stores each measurement schema and its many instantiations using the same class. (In practice, the database has additional storage elements to support other features such as security and user management.)
- Generation of specified reports as required.

Benefits of the MMT and automation

As an example of how these capabilities translate into real situations—and to provide a context for features that we'll introduce later in the article—let's look at a recent implementation of the MMT in a Fortune 500 company working on a large US Defense Department project. The company's established metrics program had given management little assistance. Monthly manual data collection and processing took four weeks, with each development group—project control, development, and testing—controlling its own data. This made metric comparison difficult at best.

After installing TychoMetrics and implementing the MMT, data collection and processing was automated and standardized, and management could create new metric reports in real

Current measurement programs suffer both from a lack of metrics standards and from invalid and missing data.

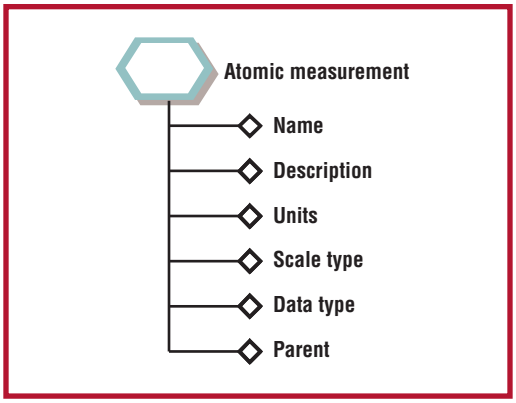

Figure 1. Atomic measurement properties. An atomic measurement diagram encapsulates a software measurement with its fundamental measurement properties.

time. Subsequently, the company changed its report frequency from monthly with a one-month delay to weekly with no processing delay.

Furthermore, the new technology integrated measurement data between the project control, development, and testing groups. Managers could quickly introduce metrics with new levels of granularity—this revealed, for example, that trouble reports were being opened at a faster rate than they were being closed. In addition, the MMT let managers link software component lines-of-code measurements to tasks identified in project control tools, which let them generate integrated size and effort metric reports.

The automation and standardization provided by the TychoMetrics tool and its embedded MMT resulted in timely, valid reports—which in turn supported proactive project management. In contrast, the previous, manual system had delivered late reports of dubious validity, despite having access to all the source data the TychoMetrics tool uses.

Measurement protocols

A critical element of the MMT is the concept of a measurement protocol, which we define in the following manner:

> *A measurement protocol defines and thereby controls its software environment so as to hold as much as possible constant. A measurement protocol ensures the repeatability of a measurement so that any difference from one measurement to the next is due to the factor under measurement and not to other confounding variables.*

The objective of using a measurement protocol is to document the measurement process sufficiently to prevent ambiguities, incompleteness, and contradictions and thereby allow an independent determination of the measured quantity's value. Our MMT's objective is to create such measurement protocols.

Several researchers have identified the need for a measurement protocol in addition to definitions of measurement entities, attributes, and units.[4,5] However, they have viewed the measurement protocol only as a textual description of the conditions under which measurement should take place. We will describe how to define a measurement protocol to allow full automation of the measurement process. This ensures that the measurement process and resulting measures and metrics are completely trustworthy and can be extracted and reported on without delay.

Measures and measurement schemas

In earlier work, Barbara Kitchenham, Shari L. Pfleeger, and Norman E. Fenton identified the need to define the object being measured.[4] However, they took the view that a measurement source was a single software entity (resource, product, process, or event) that exhibited the properties requiring measurement. To automate data extraction from complex software environments, it is better to think in terms of *measurement sources*. These could be single entities but could also be configuration management files and databases or project management tools. Data sources include one or more *atomic measures*—for example, a project management tool might maintain data pertaining to both projects and tasks. The MMT defines atomic measures in the format shown in Figure 1. This definition standard supports measurement consistency.

Kitchenham, Robert T. Hughes, and Stephen G. Linkman described a similar standard for defining atomic measurements.[5] However, they did not consider the relationships among different entities, such as the relationship between the project and its constituent tasks. Nor did they model the construction of metrics—that is, functions of one or more atomic measures. Models for entity relationships and metrics are crucial elements for an automated measurement system that provides useful and timely information.

We identify these atomic measurements at the lowest level of granularity to support a wide range of metric calculations. Examples of atomic measurements in a project control data source include (among hundreds) `PlannedWork`, `ActualWork`, and `PlannedDuration`.

For each measurement source, we organize atomic measurements into measurement groups, each of which corresponds to a single

entity within a measurement source that has one or more atomic measurements. Within a particular measurement source, we model the relationship between measurement groups in *measurement schemas*. Figure 2 shows a measurement schema relationship, in which the Elementary Task and Resource Task Assignment measurement groups are children to Summary Task, which in turn is child to Project. In addition, a measurement group can be a child to itself, creating a recursive link. For example, Summary Task could contain Summary Tasks. These relationships, called *internal links*, represent links within the measurement source. Although Figure 2 shows several sample atomic measurements attached to the Elementary Task measurement group, in practice, there exist many more; we have omitted some here for simplicity.

Some measurement sources also have *external links*, which connect one measurement group to a measurement group outside the measurement source. Many project management tools have external links indicating that a project has a subproject or summary task contained within another project management file (see Figure 3). Measurement schemas must obey several modeling rules to ensure consistency and data integrity. For example, within the same measurement schema, measurement groups must be unique, and there can only be one top-level or root measurement group.

Measurement object models and metrics

Reporting on measurements from a single measurement source is of limited value. The MMT's power comes from its ability to combine measurement schemas from different tools in a way that ensures measurement integrity and consistency. The result of combining two or more measurement schemas or portions of measurement schemas hierarchically is called a *measurement object model*. MOMs are the mechanism we use to

- Define metrics
- Specify how metrics are calculated
- Perform required calculations
- Specify report contents
- Construct required reports

The piece of a grafted measurement schema is called a schema *subtree*, with the topmost node being a measurement group. This is important when establishing the dynamic measurement protocol (discussed later in the article).

When we graft a schema subtree onto a piece of another measurement schema, we establish a *tunnel link*. The tunnel link must refer to the entire subtree to maintain measurement integrity. For example, a tunnel link to a source code counter lets the user report on size measurements in accordance with a work breakdown structure obtained from a project management tool (see Figure 4). Tunnel links give the MOM great power and flexibility.

We can associate a metric with any node in the MOM. In Figure 4, the metric `Productivity` is attached to the topmost Project node. A

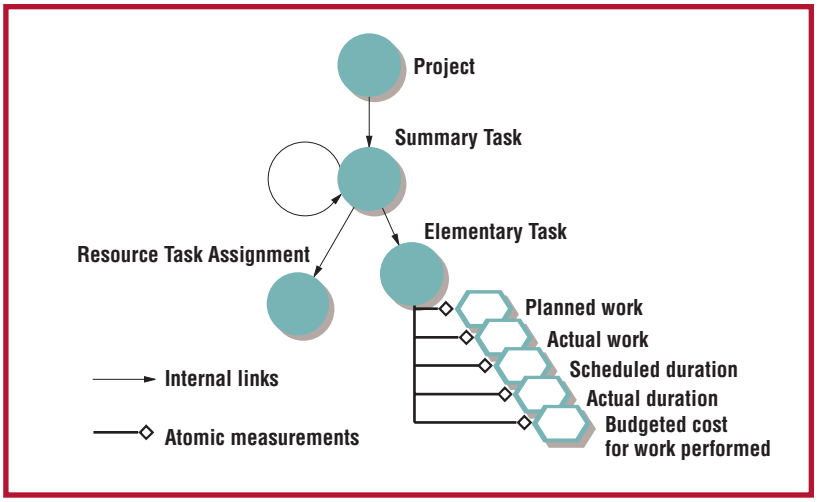

Figure 2. Measurement schema. A measurement schema diagram captures the hierarchical relationships between the measurement groups and their atomic measurements.

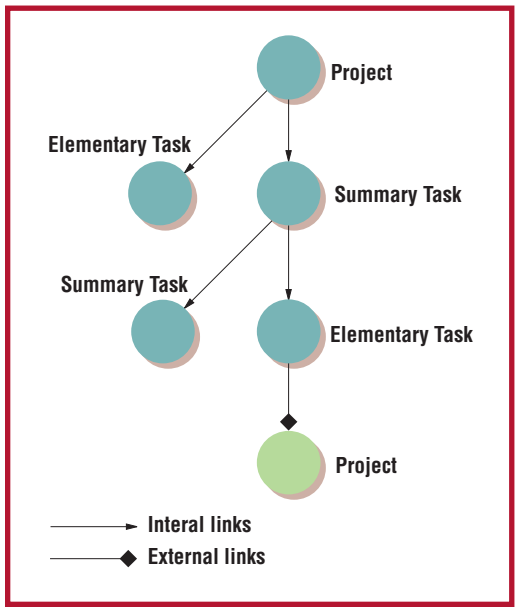

Figure 3. An external link in a project control measurement schema.

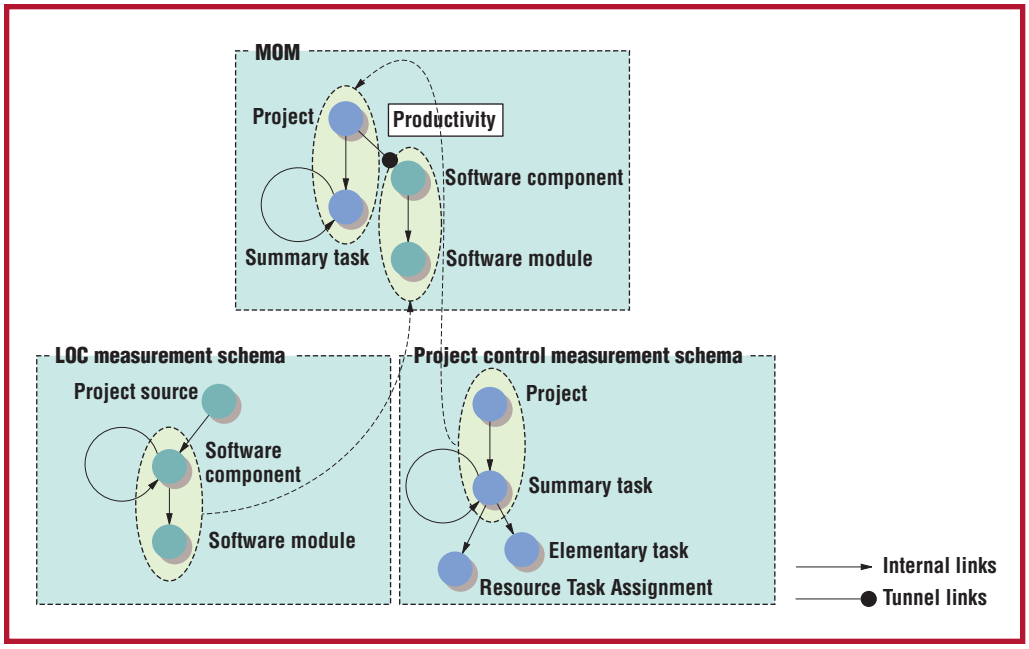

Figure 4. Project Control Size measurement object model (atomic measurements not shown for simplicity). This MOM is created from project control and source code size measurement schemas.

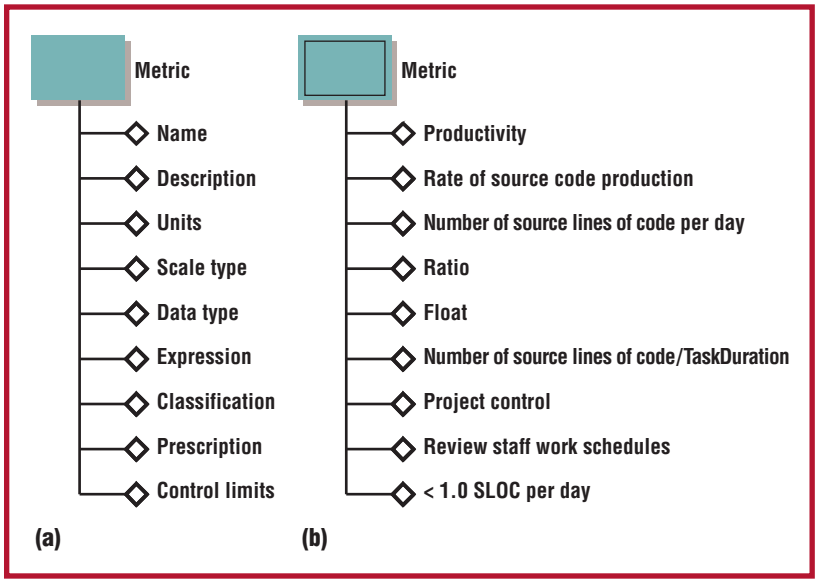

Figure 5. Instantiating a productivity software metric from its properties: (a) metric properties and (b) an example instantiation.

metric is a mathematical expression composed of one or more atomic measurements or other metrics that can include conditional selection statements. Additionally, a metric is an efficient method for computing various statistical quantities of any measurement, ranging from simple counts to statistical moments.

Figure 5 shows a partial set of properties for a metric. Every metric must have a name and description. We derive the metric's units, scale type, and data type from the atomic measurements included in the metric expression. In addition, we can classify a metric by items from one or more measurement categories, such as the software lifecycle phase or software key process area. For example, we could categorize a lines-of-code metric as falling in the implementation phase of the software lifecycle category.

The metric's upper and lower control limits, which support statistical process control, provide another useful property. We can use these control limits (which are themselves metrics) together with a property called the *prescription* to extend the metric's usefulness. The prescription is a text description (or one or more hyperlinks to the appropriate reference) of the actions to take if the measurement system detects a metric value outside the control limits. This provides a straightforward method for institutionalizing an organization's policies and procedures with regard to software development.

Measurement protocol dynamics

So far, we've discussed the measurement protocol's static elements. Its dynamic elements let us fully automate the processes of data extraction, instantiation, storage, collation, and reporting.

A major element of the dynamic protocol is a definition of when the system collects data and when it generates data reports. We refer to data collection and storage as *harvesting*. More specifically, harvesting is the process of combining data from a measurement source with its measurement schema and transferring this object to its destination, usually the TychoMetrics data repository. As shown in Figure 6, the data generation schedule does not usually map directly to the data harvesting or data reporting schedules. Thus, an MMT must allow for the measurement source being unavailable at the scheduled harvest time.

Once a harvest has occurred for each type of measurement source referenced in a given MOM, the TychoMetrics tool instantiates the MOM with actual data values. At that time, we can automate the calculation of related metrics

and generate reports. We define a report in terms of a set of metrics values that it will output.

Instantiation is a subtle but nonetheless important aspect of the dynamic protocol. To emphasize the distinction between a MOM and the instantiated MOM, we call the instantiated MOM a *measurement breakdown structure,* or MBS, because of its similarity to a work breakdown structure. Consider the simple Project Control Size MOM in Figure 4. When we construct the MBS by instantiating the MOM with specific measurement sources, data integrity rules mandate that all external links be unique to a measurement source so that double counting does not occur. For the same reason, we permit multiple tunnel links to the same measurement source if and only if the schema subtrees they reference are disjoint, thereby ensuring data integrity.

Upon harvesting measurements, the TychoMetrics tool instantiates the summary task and other measurement groups. For large projects, the resulting MBS could contain thousands of summary tasks. Moreover, as indicated in Figure 2, summary tasks can contain summary tasks. If we use this same MOM on another project, it produces a different MBS. If researchers wanted to compare productivity rates at the summary task level, how should they choose the instantiated summary tasks from the two different projects? One approach is to force the use of a standardized naming convention. This is the approach that Kitchenham and Linkman take with their MiniSquid tool (www.keele.ac.uk/depts/cs/se/e&m/minisquid.htm). However, this is only possible because the tool takes a simple approach to measurement sources (that is, equating a source to a single software entity) and does not support automatic metrics construction. For more complex measurement sources, this approach meets with less success.

An alternative is to consider the level at which the instantiated measurement group appears, as counted from the parent node of a recursive measurement group. This level, which we call the *level of insight,* provides a consistent and repeatable method for comparison. For example, Figure 7 shows a sample MBS that resulted from instantiating the project control measurement schema from Figure 4. This measurement schema contains two levels of summary tasks. Without using a naming convention, we can request reports at a specific level of insight—for example, Summary Task Level 2. Then, we can compare metrics from different projects at the same level of insight.

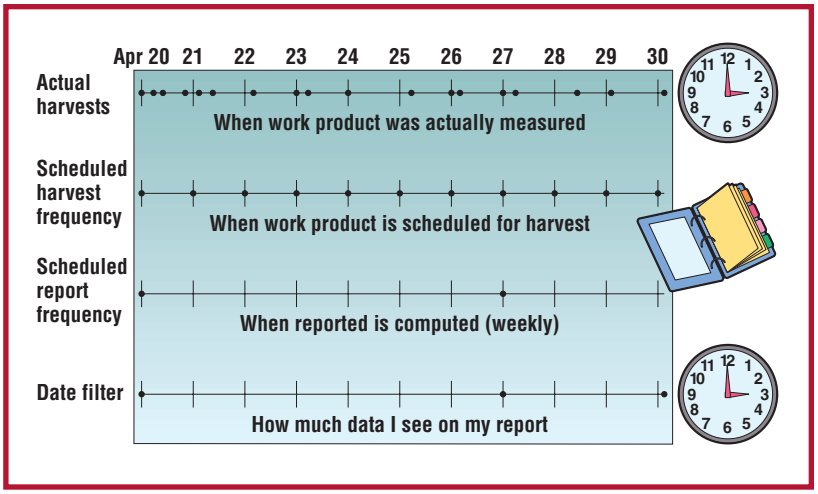

Figure 6. Data generation, collection, and reporting schedules. These schedules are usually different yet complementary to support the desired report data intervals and generation frequency.

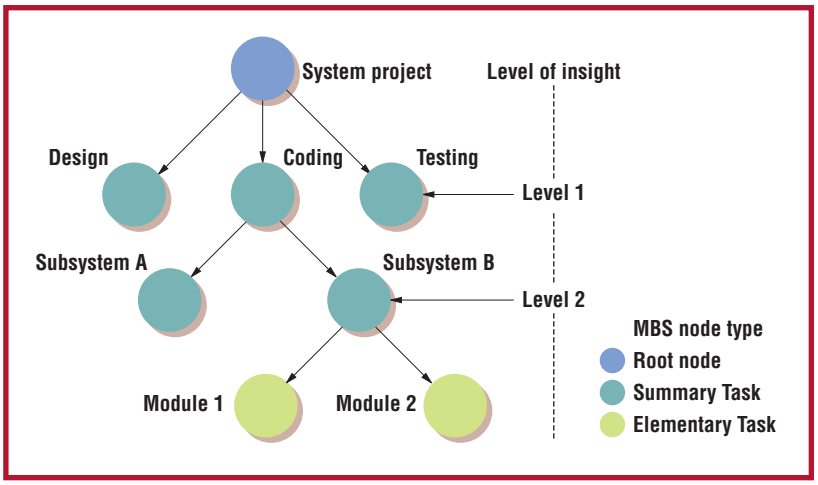

Figure 7. Instantiating a MOM results in a metric breakdown structure, such as this one, with identifiable levels of insight.

Metric calculation

Computing a value for a metric is a four-step process:

1. Simplify all elements in the metric expression to their atomic measurements.
2. For each atomic measurement, perform a dynamic tree search (*drill-down*) along all paths in the subtree from the initial node to the first terminal node.
3. For each type of atomic measurement, combine (*roll-up*) all atomic measurement values from all paths in the subtree.
4. Evaluate the metric using the defined mathematical expression.

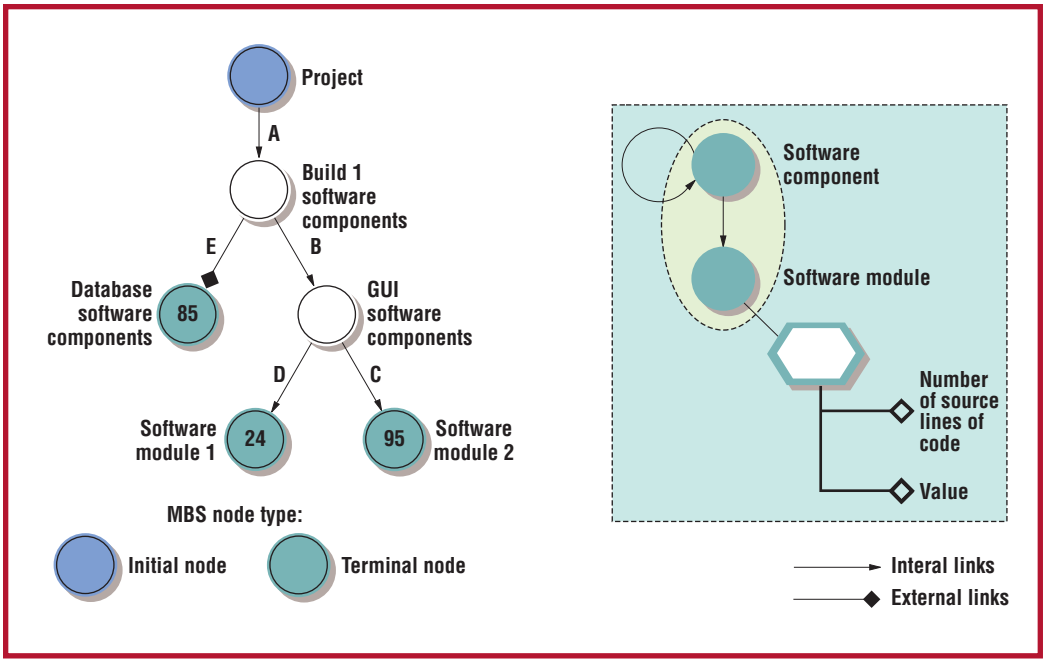

Figure 8. Drilling down paths in an MBS to compute a productivity metric. This illustration shows the lines of code portion of an MBS.

A terminal node is a node for which a value of the atomic measurement is assigned after a harvest. Any measurements below the terminal node roll-up to the value of the terminal node atomic measurement. If no measurements are found along a drill-down path, then the tool returns a null for the roll-up value. By this definition, all nodes below a drill-down path's first terminal node are terminal nodes themselves.

As a simple example, suppose we want to measure a project's productivity. Using the MOM in Figure 4 and the productivity metric shown in Figure 5, we follow the four computation steps. Because the metric, which is composed of two atomic measurements, `NSLOC` (number of source lines of code) and `TotalDuration`, is located at the MOM node named Project, that is where we start the drill-down. For simplicity, Figure 8 shows only those portions of the MBS whose nodes contain the `NSLOC`. As the drill-down proceeds along path AB, it encounters no measurements until it reaches the terminal nodes at the software module level, resulting in a subtotal of 119 (24 + 95). In a similar fashion, another

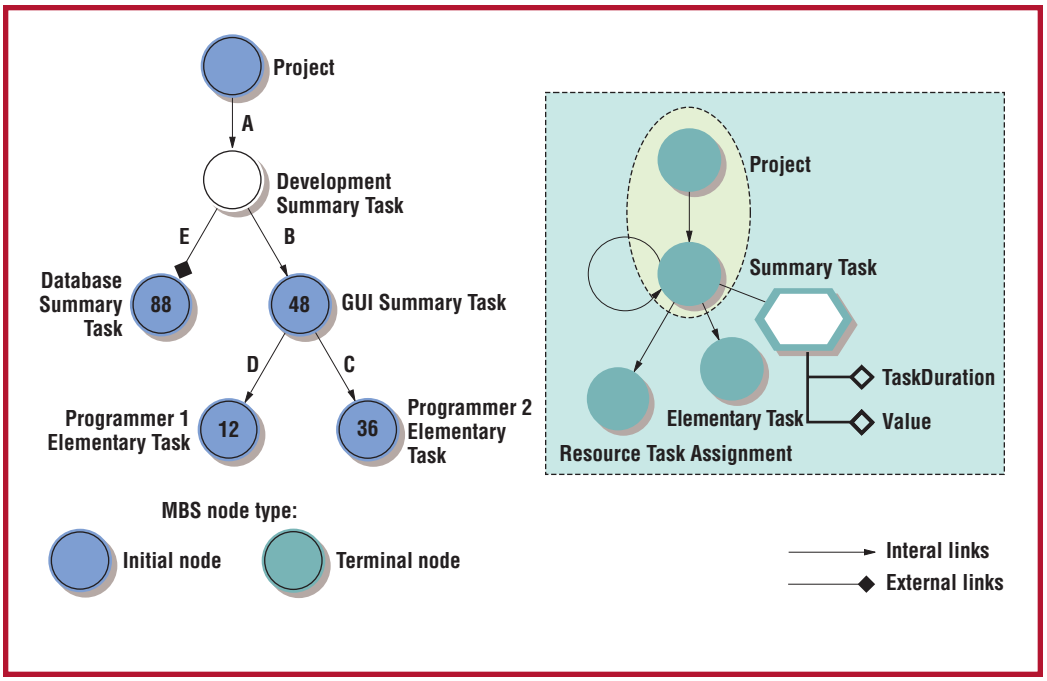

Figure 9. This portion of the MBS shows project control drill-down paths for rolling up the `TotalDuration` measurements.

drill-down must also follow the AE path, where it picks up the database software components measurement of 85, resulting in a roll-up of `NSLOC` of 204 (119 + 85).

Likewise, Figure 9 shows a portion of the MBS whose nodes contain `TotalDuration`. In this case, the GUI Summary Task is reached through the AB drill-down path and found to contain a `TotalDuration` value of 48; that terminates the drill-down for that path and avoids double counting. Continuing the drill-down from the Project root node along the AE path, the tool reaches the Database Summary Task to pick up a measurement of 88. Thus, for `TotalDuration` we compute a value of 136 hours (48 + 88), or 17 days.

Now we are in a position to compute the productivity metric,

```
Productivity = NSLOC /
               TotalDuration.
```

The calculation yields a productivity of 12 NSLOC per day.

More complex metrics, such as those specified using tunnel links, require additional rules to ensure data integrity. For example, if we want to report on defect rates per module, we must collate the defect counts with module size counts for the same module. TychoMetrics does this by ensuring each atomic measurement has a unique global identifier.

Our automated MMT provides a standard method for defining measures and metrics. It minimizes the familiar problems of invalid, incomplete, and late data by automating data extraction, collation, storage, and reporting. Nonetheless, this measurement approach demands significant work by the users, who must decide exactly what they want to measure and then specify the measures correctly. Technologies such as the Goal Question Metric[6] are often helpful when deciding what to measure.

The choice is clear. Organizations can continue to collect data in an ad hoc fashion, with the likely result that they will waste time analyzing invalid data, manually update old reports to reflect current project status, fail to recognize problems, and react to problems that don't exist. Alternatively, they can invest in a more rigorous approach to data collection, such as ours, to ensure that they gain real value from their metrics programs.

Acknowledgments

TychoMetrics is a commercial tool protected under US patent 5,930,798.

References

1. J. McGarry, "When It Comes to Measuring Software, Every Project Is Unique," *IEEE Software*, vol. 19, no. 5, Sept./Oct. 2001, p. 19.
2. K. Maxwell, "Collecting Data for Comparability: Benchmarking Software Development Productivity," *IEEE Software*, vol. 19, no. 5, Sept./Oct. 2001, p. 22.
3. J.T. Heires, "What I Did Last Summer: A Software Development Benchmarking Case Study," *IEEE Software*, vol. 19, no. 5, Sept./Oct. 2001, p. 33.
4. B.A. Kitchenham, S.L. Pfleeger, and N.E. Fenton, "Towards a Framework for Software Measurement Validation," *IEEE Trans. Software Eng.*, vol. 21, no. 12, Dec. 1995, pp. 929–944.
5. B.A. Kitchenham, R.T. Hughes, and S.G. Linkman, "Modeling Software Measurement Data," *IEEE Trans. Software Eng.*, vol. 27, no. 9, Sept. 2000, pp. 788–804.
6. V.R. Basili and H.D. Rombach, "The TAME Project: Towards Improvement Oriented Software Environments," *IEEE Trans. Software Eng.*, vol. SE-14, no. 6, June 1988, pp. 758–773.

About the Authors

Jim Lawler is the founder of Predicate Logic, a software and systems engineering service company. He is also a cofounder of the San Diego Defense and Space Technology Consortium, a high-tech incubator for defense and space technology. His research interests, which include software metrics and measurement modeling, led to the development of TychoMetrics, an automated measurement and reporting tool. Lawler holds a BS in mathematics from State University of New York at Albany and an MS in astrophysics from Virginia Polytechnic Institute and State University. Contact him at Predicate Logic, 9619 Chesapeake Dr., Ste. 200, San Diego, CA 92123; jlawler@predicate.com.

Barbara Kitchenham is a professor of quantitative software engineering at Keele University. She is also a visiting professor at both the University of Bournemouth and the University of Ulster. Her main research interest is software metrics and its application to project management, quality control, risk management, and evaluation of software technologies. She is a Chartered Mathematician, a fellow of the Institute of Mathematics and Its Applications, and a fellow of the Royal Statistical Society. Contact her at the Dept. of Computer Science, Keele Univ., Staffordshire, ST5 5BG, England; barbara@cs.keele.ac.uk.