The Web Measurement Environment(WebME): A Tool for Combining and Modeling Distributed Data

Roseanne Tesoriero and Marvin Zelkowitz
Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland USA
{roseanne, mvz}@cs.umd.edu

ABSTRACT
Many organizations have incorporated data collection into their software processes for the purpose of process improvement. However, in order to improve, interpreting the data is just as important as the collection of data. With the increased presence of the Internet and the ubiquity of the World Wide Web, the potential for software processes being distributed among several physically separated locations has also grown. Because project data may be stored in multiple locations and in differing formats, obtaining and interpreting data from this type of environment becomes even more complicated. The Web Measurement Environment (WebME), a Web-based data visualization tool, is being developed to facilitate the understanding of collected data in a distributed environment. The WebME system will permit the analysis of development data in distributed, heterogeneous environments. This paper provides an overview of the system and its capabilities.

KEYWORDS
Measurement, Software development, Meta-analysis, Empirical modeling

1 INTRODUCTION

Measurement has been emphasized as an effective method for gaining control and insight into software activities. Because of this, many organizations have incorporated data collection into their software processes. However, just as important as the collection of data is the presentation, understanding, and resulting actions that accompany the data collection process. Data collection must be an active component in the development cycle of a project and not simply a passive task that results in large, mostly unused, data files.

Collection of data is inherent in the NASA Goddard Software Engineering Laboratory (SEL) [2] as part of the Quality Improvement Paradigm (QIP) [3] and as part of the Software Engineering Institute's Capability Maturity Model (CMM) [13]. However, neither activity gives much detail on how this data should be displayed. How does one view such collected data in order to present information that would be most effective to the project manager in order to aid in real-time decision making? Can we compare a new project to previously completed projects in order to determine trends and deviations from expected behavior? What do we even mean by expected behavior?

The NASA SEL had developed a tool, the Software Management Environment (SME) [6, 8], that did provide a quasi-real-time feedback on project data. The SEL has been collecting data for over 20 years on NASA flight dynamics software. Data would be entered in a data base within two or three weeks of it being collected, and then a program could be run to summarize that data for SME. Management could then use SME to display growth rates of certain project attributes (e.g., lines of code, staff hours, errors found) and compare them to previous projects with similar characteristics. This would provide two major functions: (1) Baselining capabilities so management could understand the developing characteristics of a given project, and (2) Predictive capabilities by enabling management to compare this project with previously completed projects and with idealized models of growth built into the SME system. Knowledge of software development is built into the models of SME to allow for easier analysis of collected data in the software development domain.

With the increased presence of the Internet and the World Wide Web, the nature of software development has changed. The Internet and the Web are seen now as valuable tools to be used for cooperative development in distributed environments. Recent work in the CSCW area has addressed these new requirements. Several tools have been built to automate selected distributed software processes with Web technology (e.g., software inspections [14, 12, 17], problem tracking [19, 5]). Most of this work has been focused on automating the definition and enactment of a process model. Although data measurements usually are collected automatically with the CSCW tools, the analysis of the collected data is

still a mostly manual process. While the SME system was not designed to be used in a distributed, cooperative environment, we felt it provided a good basis for a more effective tool. The remainder of this paper discusses our system, called the Web Measurement Environment (WebME), which provides the same basic functions as SME, but, allowing for changing data in a distributed, cooperative environment.

2 SYSTEM ARCHITECTURE

The WebME system has a World Wide Web interface which provides a wide variety of users with access to the system and the data. For our instantiation of the WebME system, there are no restrictions to access to the system or data. However, a similar system architecture could be used within the boundaries of a corporate intranet with appropriate security measures in place.

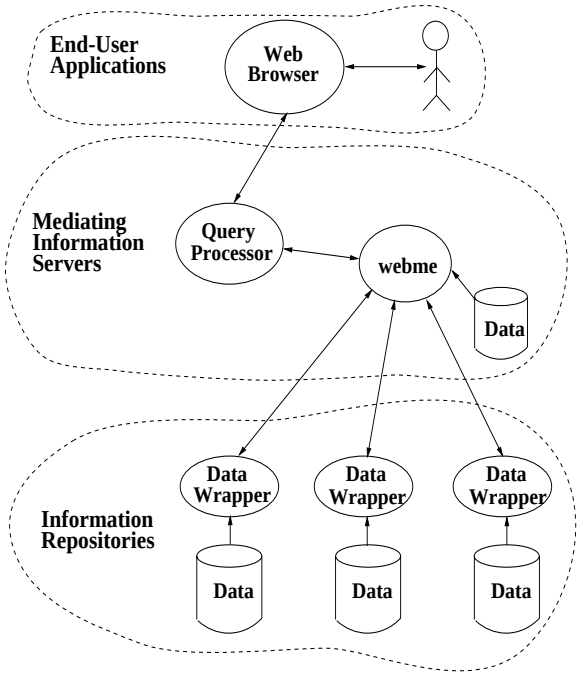

Figure 1: WebME System Architecture.

The WebME system is based on a mediator architecture [18]. A mediated architecture horizontally partitions the architecture into three layers: end-user applications, mediating information servers, and information resources. In the WebME context, the distributed databases with the software engineering data are the information resources. The *data wrappers* describe the interface between the information repositories and the mediating information server (i.e., *webme*). The Web browsers and the associated HTML forms represent the end-user application layer. The *webme* mediator is responsible for gathering and processing the data required to fulfill end-user requests and returning answers to the end-user.

In order to describe the system architecture, a schema of the required interfaces must be defined. Using a specialize language is a common technique used to describe a system architecture[7, 15, 11]. For WebME, we have defined a scripting language to describe the schema of the system architecture and the data definitions. In order to create the definitions for the interfaces and measurement types, an expert familiar with the development environment and databases will configure the system by creating a WebME script file using the scripting language.[1] The script will be processed into measurement class and interface definitions that will be accessible by the WebME mediator as shown in Figure 2.

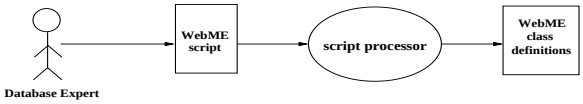

Figure 2: WebME data and interface definition process.

When an end-user makes a request, the class and interface definitions are used by the WebME mediator to gather and process the necessary data. This process is illustrated in Figure 3.

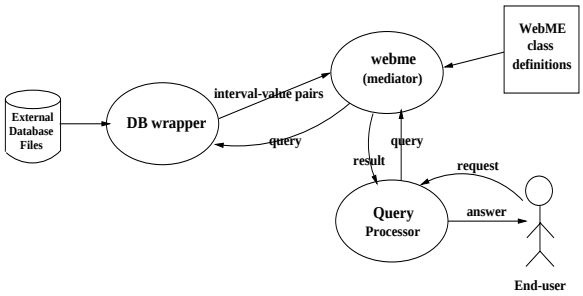

Figure 3: Using class and interface definitions.

3 COMBINING DATA

There are several cases where the need to combine data from various locations is necessary. For example, with a collaborative software development process, in order to assess and monitor the progress of the entire project, data collected from each location must be combined. Another way in which data might be combined is in the graphical display. When similar data is collected from two different environments, it might be useful to be able

[1] Creating the script file will be described in more detail in Section 5.

to display the data on the same graph for comparison purposes.

The WebME system will use a data definition language as part of the scripting language to facilitate the combination of data in these ways. WebME will allow the user to define *classes* of measurement types, where a class will represent a given development environment, such as the NASA SEL. The measurement types (*or attributes*) represent the data collected in the development environment.

3.1 Class Definitions

Each class consists of entities that possess dimensionality attributes (using the notation of Kitchenham et al [9]). Attributes may be direct or indirect. In our context, a direct attribute is one in which the measured value for the attribute can be extracted directly from an external database. An indirect attribute derives its value from a transformation applied to other attributes (e.g., an equation).

The structural model of measurement described in [9] identifies units and values as properties of attributes. We have added an *interval* (e.g., weekly, monthly) as an additional property of the unit. A measurement instrument uses the units and the interval to supply the correct value for the attribute. The attribute definition represents the format of the data stored in the repository and is used to extract data from the external database in WebME's mediator architecture.

All data that will be displayed in the WebME system will be sequenced data with ratio scale type. For direct measures, the measurement *instrument* is an executable that will extract measured values at the desired interval from a database. For indirect measures, the measurement instrument is an *equation*. The allowable operations in the equations are the arithmetic operations (addition, subtraction, multiplication and division). The *units* and *interval* properties of indirect attributes will be inferred dimensionally from the attributes used in the equation. These indirect attribute definitions will be validated to detect invalid operations (e.g., lines of code + hours of effort is dimensionally incorrect).

In WebME, attributes are grouped into classes. Entities (e.g., software projects) are assigned to a class of attributes. Any two entities possessing the same attribute can be displayed on the same graph as long as their units are equivalent (see Section 3.2). This allows for different, but related data that are collected and stored separately to be viewed consistently. In addition, any attributes that are compatible (i.e., have equivalent units) may be plotted on the same graph.

3.2 Attribute Compatibility

The *units* and *interval* properties of the attribute definition will be used to determine compatibility for viewing and to validate the equations of indirect attributes. Two attributes are *compatible* if the units and interval properties are name equivalent. Compatible attributes may be displayed on the same graph.

The compatibility of attributes used in the equations of indirect attributes must be validated. The allowed operators are +, −, * and /. For addition and subtraction, the units and interval properties of the operands must be name equivalent. For multiplication and division, this restriction is relaxed in that the units properties may be different, but the interval properties must be name equivalent.

4 MODEL BUILDING

The consistent combination of data is one part of the problem that the WebME system attempts to address. Building meaningful models from the combined data for the purposes of process control and improvement is another.

The modeling technique used in the SME system is used to build baseline and predictive models of growth data. In 1993, a clustering algorithm using Euclidean distance was investigated [10] as an alternative to the existing SME growth models. The current growth modeling algorithms appear to be a good starting point for growth data, however, we also wanted to build baseline models for the non-cumulative raw data. This type of data is highly variable and it is often difficult to uncover trends or patterns.

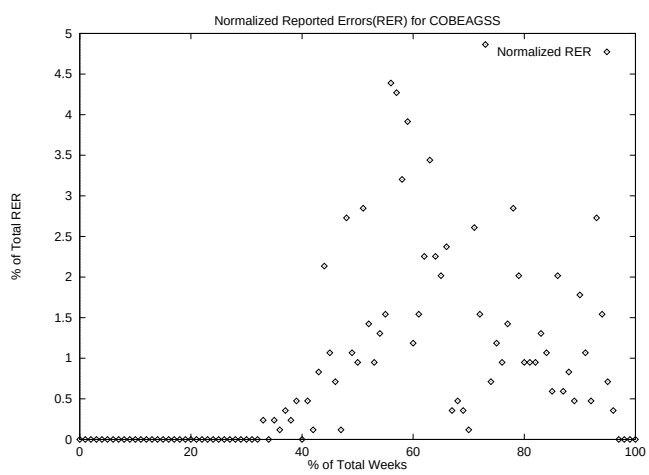

Figure 4: Scatter plot data of reported errors by week

Figure 4 represents the weekly number of error reports

filed for a single NASA project.[2] It is hard to see any trend or underlying model in the data. Is there any underlying process that determines how many errors are found each week? Can we make any reasonable models of this process?

4.1 Financial models

One way to partition the data is based on using trend changes as a signal for process changes. In the financial markets, the price of a stock or commodity is highly variable. An investor's objective is to buy at a minimum price and sell at a maximum price. However, because prices fluctuate frequently, an investor would not want to trade at every trend change in the market.

The problem of trend detection for financial data turns out to be similar to our problem. We have highly variable data and we want to detect major trend changes while ignoring minor fluctuations. Techniques used to detect trend changes with financial data should be applicable to our domain.

In particular, financial markets look at long term versus short term trends. Moving averages have long been used in this domain, where an N-day moving average is the average value of some feature over the past N days. If the long term average (i.e., using a large value of N) is greater than the short term average (i.e., using a small value of N), then a stock has a decreasing trend in value; otherwise it is increasing. Such trends eliminate the daily fluctuations inherent in this form of data. If the trend moves from negative to positive, then its price has presumably reached its minimum and should be bought. If the trend moves from positive to negative, then it has peaked and should be sold since waiting will only decrease its price.

The *Moving Average Convergence/Divergence* (MACD) trading system [1] [16] determines when the long term changes in a stock's value differs from the short term changes, which signals a decision to buy or sell the stock. When the trend crosses the *signal line* (i.e., the moving average of the long term average less the short term average) in a positive direction, the price is about to rise and a stock should be bought; if it crosses the signal line in the negative direction, a sell is indicated.

4.2 Modeling Algorithm

Based on the MACD examples, we have developed an algorithm for analyzing each data attribute. Given the raw scatter plot data for some attribute (such as given in Figure 4), we want to reduce it to several linear segments that best represent the governing processes during the period represented by each segment. We will call this the *characteristic curve* and our initial goal is to find the end points for each such linear segment, which we call the *pivot points* to this curve. Once we do that, we can apply more traditional curve fitting techniques to each segment in order to develop underlying models of each process.

The three steps we have developed are:

1. Use smoothing techniques to provide a rough envelope that represents the approximate behavior of the data. This process is not sufficient by itself. For example, the data of Figure 4 results in a smoothed curve (Figure 5) which still has 12 local maxima when using an 8 point moving average.

2. Determine which of the extreme points represent a significant event for these processes. Other local maxima (or minima) are assumed to be minor perturbations in the data and are to be ignored. We call these significant trend changes *pivot points*.

3. Connect the set of pivot points into a segmented line. This represents the *characteristic curve* for the original raw data.

We outline the algorithm in the following sections:

Data Smoothing. In order to remove day to day variability in the value of a stock, N-day moving averages are used. Often a short range moving average (e.g., 30 days) is compared with a longer range moving average (e.g., 150 days) in order to compare local changes to a stock's price compared with the longer range trend. The crossover points between the short and long term moving averages signal trend reversals.

This *simple moving average*, however, has a weakness. If a critical point is reached (e.g., the value reaches a maximum), the damping effects of the earlier points in the average delay the signaling of this phenomenon. That is, the moving average will continue to rise for several days after the peak is reached since all points are weighted equally in computing the average. In order to enhance the perception of such directional changes, the *exponential moving average* (EMA) is used for the MACD trading system described earlier. Rather than being the simple average of the last N points, the exponential moving average is given by the equation:

$$EMA_i = (1 - \frac{2}{N+1}) * EMA_{i-1} + \frac{2}{N+1} * v_i$$

[2] All data presented here is normalized from 0% to 100%. That allows us to compare multiple projects on the same graph. The time duration for the projects considered here range from 100 to 120 weeks – about 2 years.

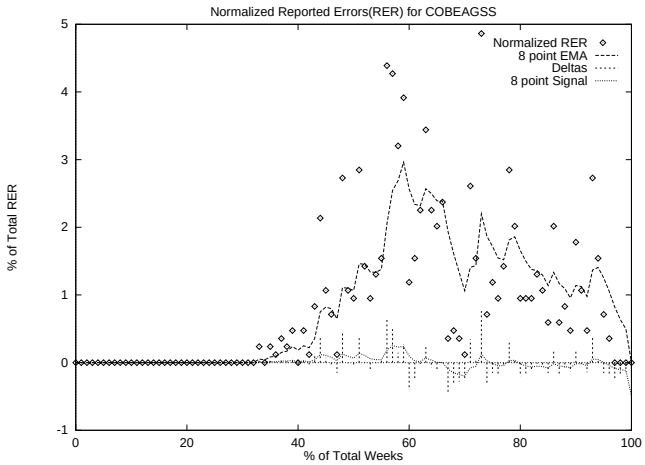

Figure 5: Smoothed data using moving averages

where:

EMA_i is the exponential moving average at time i

v_i is the new data value at time i

and $\frac{2}{N+1}$ is the smoothing constant where N is the number of points in the average.

For $N = 9$, $\frac{2}{N+1}$ has a value of .2 meaning each new point has about twice the "impact" (20% instead of 11%) that a simple moving average would have. Each successively older point has less of an effect on the total average, and the result is a moving average more sensitive to leading edge changes.

The higher curve in Figure 5 shows the effects of the EMA on the error data of Figure 4. From this EMA of the scatter plot data, we want to extract only those maxima and minima that represent significant changes in the underlying process.

Find significant trend changes. If we could simply take the derivative of this curve, we could solve for the derivative being zero in order to find the local maxima and minima. However, the actual (smoothed) data does not permit such computations. We can use the EMA to help again for this process. Between any two points we can compute the instantaneous derivative $\delta_i = \frac{\Delta v_i}{\Delta t}$. If we compute this for each time period t, and take the EMA for these delta values, we get what is called in the financial community the *signal line* (Figure 5). Where the signal line crosses the X-axis represents a zero EMA, or in other words, the average δ_i in the interval is 0, which represents an extreme value for the curve.[3] In our example, this signal line crosses the X-axis 7 times. Each of these represents a critical point in the original data.

What does this signal line represent? It is the average slope of the instantaneous derivatives for the past N points. If the signal line is 0, it means that the average delta between successive points is 0 and we have a local maximum or minimum. We simply have to go back over the last N points to determine which value of time t_i represents that extreme value. We call such values *pivot points*.

Computation of Characteristic Curve. Once we have identified the pivot points, we connect each segment with a straight line (Figure 6). This segmented line describes the general shape of the curve we are interested in. We have been able to eliminate minor hills and valleys from the curve and have left only the major features of the original data.

Figure 6 shows the characteristic curve computed by our algorithm along with the original data.

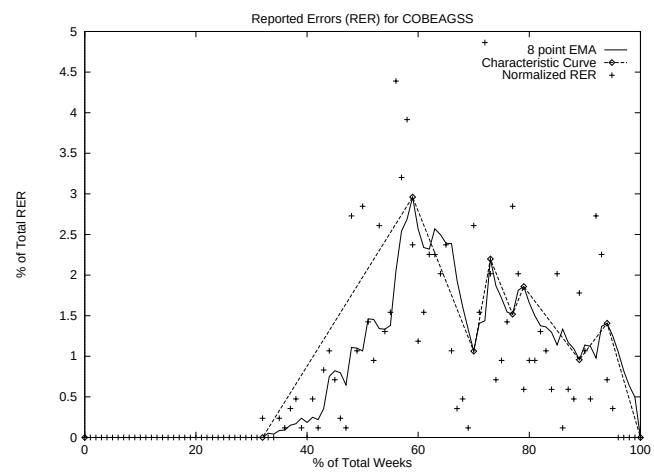

Figure 6: Raw data with its characteristic curve

One interest to us was the first dip noticed between 60% and 80% of project completion in the error data of Figure 6. Looking through old records (from 1988) we discovered that the minima point at 70% occurred just before the start of acceptance testing for the project, even though resource usage (i.e., hours worked) shows no such disruption in the process. This was also the time when the contracting organization that built the software moved into a new building. The identification of a milestone via the collected data seemed interesting, but the confounding influence of the building move concerned us. After looking at the characteristic curves of

[3] In the original MACD development, the signal line was the EMA of the difference between the long term and short term EMA. Here we are only concerned with the slope of the δ_i curve.

reported errors from other projects in this domain, we discovered similar behavior before acceptance testing.

While it appears that we can identify the start of acceptance testing (at least in this NASA environment) by the shape of the characteristic curve, we need to investigate further, the meaning of the characteristic curve. In addition, we are also working on the ability to catalog a project by the shape of the characteristic curve.

5 USING THE WEBME SYSTEM

In this section, we present examples of how the WebME scripting language is used. We only present the parts of the scripting language that are necessary here to illustrate our examples. The words in **boldface** fonts are keywords in the WebME scripting language. The words in the normal font are the parameters that are specific to the architecture being described.

5.1 Interface definitions

The interface definitions are used to describe the schema of the system architecture and the interfaces available in the architecture. Definitions of hosts (i.e., the physical location of an information repository) and wrappers (i.e., the interfaces available at the information repository) are created through the scripting language.

To define a host in WebME, the host name and port number are defined using the WebME scripting language. For example, if a *data wrapper* is listening to port number 8001 on a host named **aaron.cs.umd.edu** for WebME requests, the following statement would appear in the WebME script file:

create host aaron.cs.umd.edu **port=**8001;

To define a wrapper in the WebME scripting language, the host and path to the data wrapper must be identified. For example, if an executable called **getsize** which is used to extract size data from the host **aaron** is located in the **/aaron/webme/bin** directory, the following statement would be used:

create instrument getsize **host=**aaron.cs.umd.edu,
path=/aaron/webme/bin/getsize;

5.2 Combining data from multiple locations

To illustrate how the scripting language is used for the combination of data, consider the case of a fictitious collaborative software development process. Assume the development environment, called the Widget Development Division (WDD), has two locations where measurements of the development process for a project called *SmartWidget* are being collected and stored. For this example, the only attribute being collected is size measured in lines of code (LOC) developed each week.

If the hosts to be included in the system are *coffee.widgets.com* located in Seattle and *tea.widgets.com.uk* located in London, the hosts would be defined with the following statements:

create host coffee.widgets.com **port=**7000;
create host tea.widgets.com.uk **port=**8000;

To define the interfaces (UKsize and USsize) available at each location, the wrappers would be defined with the following statements:

create instrument USsize
host=coffee.widgets.com, **path=**/usr/bin/getattr;

create instrument UKsize
host=tea.widgets.com.uk, **path=**/usr/bin/getdata;

In addition, a class representing the WDD development environment and the attributes for size must be defined.

create class WDD;

/* size in lines of code for Seattle */
create attribute direct WDD.USsize
with units LOC, **interval** week,
instrument USsize;

/* size in lines of code for London */
create attribute direct WDD.UKsize
with units LOC, **interval** week,
instrument UKsize;

/* total size in lines of code (indirect attribute) */
create attribute indirect WDD.TotalSize
using USsize + UKsize;

Finally, the project *SmartWidget* would have to be assigned to the WDD class.

assign SmartWidget to WDD;

Now, the size of the project can be monitored at the individual site level (with the USsize or UKsize attributes) or at the entire project level (by viewing the TotalSize attribute).

5.3 Viewing compatible data

To demonstrate how the scripting language and the system can be used to display similar attributes on the same graph, suppose the attributes of reported errors (Reported), closed errors (i.e., errors that have been resolved) (Closed) and net errors (i.e., change in number of errors discovered in the current week) have been added to the WDD class with the following statements:

create attribute direct WDD.Reported
with units errors, **interval** week,
instrument getreported;

create attribute direct WDD.Closed
with units errors, **interval** week,
instrument getclosed;

create attribute indirect WDD.Net **using**
Reported - Closed;

Note that the units and interval for net errors (i.e., WDD.Net) would be inferred from the attributes in the equation. In this case, the units and interval would be errors and week, respectively.

Using the rules of compatibility described in Section 3.2, the attributes WDD.Reported, WDD.Closed and WDD.Net, could be displayed on the same graph in WebME as shown in Figure 7. However, WDD.Reported and WDD.size (as defined earlier) could not be displayed on the same graph because the units are not name equivalent.

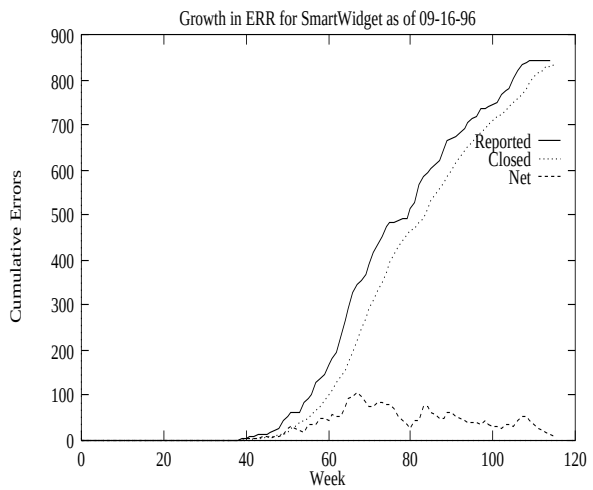

Figure 7: Compatible attributes.

Figure 7 is a graph showing the three compatible attributes using common axes. Although total errors and closed errors apparently track each other in a similar manner, the indirect attribute of open errors shows a clear bulge around week 65, which should cause management to further investigate its possible cause.

6 CURRENT STATUS AND CONCLUSIONS

The use of collected data on past projects as predictors of future project behavior is a growing phenomenon in software development. However, development environments vary widely. It is important that the baseline predictor projects have characteristics that are amenable to the new project being compared. Processes like the *Experience factory* [4] have been proposed as a means to organize such developmental practices. However, means must be found for passing information among such environments or for comparing results obtained in two different environments. A tool like WebME gives the analyst a mechanism for defining common characteristics across such domains.

At this time, the system architecture for WebME is operational allowing for access to WebME from anywhere on the WWW. The scripting language for defining interfaces and data types is being implemented. Additional data bases are under study in order to determine the effectiveness of our system in building indirect attributes across a wide range of application domains.

We still need additional experience with the proposed modeling technique before we can incorporate it into the WebME system. However, even without such additions, we have designed a system that aids software developers in accessing development data in various settings and obtaining visual feedback on the relative merits of a single project compared to a repository of related projects.

ACKNOWLEDGEMENTS

This research was supported in part by NASA grant NCC5170 to the University of Maryland. Jon Valett, formerly of NASA Goddard and now of Q-Labs, built the initial version of SME. N. Rorry Li, now of Oracle Corp., did the initial rehosting of SME to the SUN UNIX platform and added the clustering data model in 1993.

REFERENCES

[1] G. Appel and W. F. Hitschler. *Stock market trading systems.* Dow Jones-Irwin, Homewood, Illinois, 1980.

[2] V. Basili, M. Zelkowitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski. SEL's software process-improvement program. *IEEE Software*, 12(6):83–87, 1995.

[3] Victor R. Basili and H. Dieter Rombach. The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6), June 1988.

[4] V.R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page, and S. Waligora. The software engineering laboratory-an operational software experience factory. In *Proc. of the* 14th *International Conference on Software Engineering*, pages 370–381, Melbourne, Australia, May 1992.

[5] ClearDDTS.
URL: http://www.pureatria.com/ddts/ddts_main, Visited November 1997.

[6] W. Decker and J. Valett. Software management environment (SME) concepts and architecture. Technical Report SEL-89-003, SEL, January 1989.

[7] Object Management Group. The common object request broker: Architecture and specification. Technical Report 93-12-43, Object Management Group, December 1993.

[8] R. Hendrick, D. Kistler, and J. Valett. Software management environment (SME) components and algorithms. Technical Report SEL-94-001, SEL, February 1994.

[9] Barbara Kitchenham, Shari L. Pfleeger, and Norman Fenton. Towards a framework for software measurement validation. *IEEE Trans. on Software Engineering*, 21(12):929–944, Dec 1995.

[10] N. R. Li and M. V. Zelkowitz. An information model for use in software management estimation and prediction. In *Second International Conference on Information and Knowledge Management*, pages 481–489, Washington DC, November 1993. ACM.

[11] J. Magee, N. Dulay, and J.Kramer. A constructive development enviroment for parallel and distributed programs. In *Proceedings of the* 2nd *International Conference on Configurable Distributed Systems*. IEEE, 1994.

[12] V. Mashayekhi, B. Glamm, and J. Riedl. AISA:asynchronous inspector of software artifacts. Technical Report TR-96-022, University of Minnesota, Mar 1996.

[13] M.C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. The capability maturity model for software, version 1.1. Technical Report CMU/SEI-93-TR-24, CMU/SEI, 1993.

[14] J. M. Perpich, D. E. Perry, A. A. Porter, L. G. Votta, and M.W. Wade. Anywhere, anytime code inspections:using the web to remove inspection bottlenecks in large-scale software development. In *Proc. of the* 19th *International Conference on Software Engineering*, pages 14–21, May 1997.

[15] J. Purtilo. The polylith software bus. *Transactions on Programming Languages*, 16(1):151–174, Jan 1994. Also available as UMIACS-TR-90-65.

[16] E. Seykota. MACD: Sweet anticipation? *Futures*, 20(4):36+, Mar 1991.

[17] D. Tjahjono. *Exploring the effectiveness of formal technical review factors with CSRS, a collaborative software review system*. PhD thesis, Department of Information and Computer Sciences, University of Hawaii, 1996.

[18] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–48, March 1992.

[19] Web-Integrated Software Environment Home Page [1]. URL:
http://research.ivv.nasa.gov/projects/WISE/wise.html, Visited November 1997.