Using the Internet to Combine and Analyze Distributed Software Engineering Data

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

Abstract

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 quite complex. 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 display and analysis of development data in distributed, heterogeneous environments. This paper provides an overview of the system and its capabilities.

KEYWORDS
Collaborative software development, Distributed project management, Measurement, 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.

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 [10, 8, 13], problem tracking [15, 3]). 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.

Within the NASA Goddard Software Engineering Laboratory (SEL), data collection is a major component of the Quality Improvement Paradigm (QIP) [2] and is part of the Software Engineering Institute's Capability Maturity Model (CMM)[9]. However, neither activity gives much detail on how the 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? How does a distributed development environment affect the data gathering and analysis problem?

The NASA SEL had developed a tool, the Software Management Environment (SME) [5], 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 database 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.

While the SME system was not designed to be used in a distributed, cooperative environment, we believed it provided a basis for a more effective tool. The remainder of this paper discusses our system, called the Web Measurement Environment (WebME), which extends the basic functionality of SME to a web-based development environment.

2 SYSTEM ARCHITECTURE

The WebME system uses a World Wide Web browser interface to provide users access to the system and the data. The current WebME prototype places no restrictions on access; however, it could be used within the boundaries of a corporate intranet with appropriate security measures in place.

The WebME system is based on a mediator architecture [14]. A mediated architecture horizontally partitions the architecture into three layers: end-user applications, mediating information servers, and information resources. In the WebME context (see Figure 1), the distributed databases are the information resources. The *data wrappers* interface with 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.

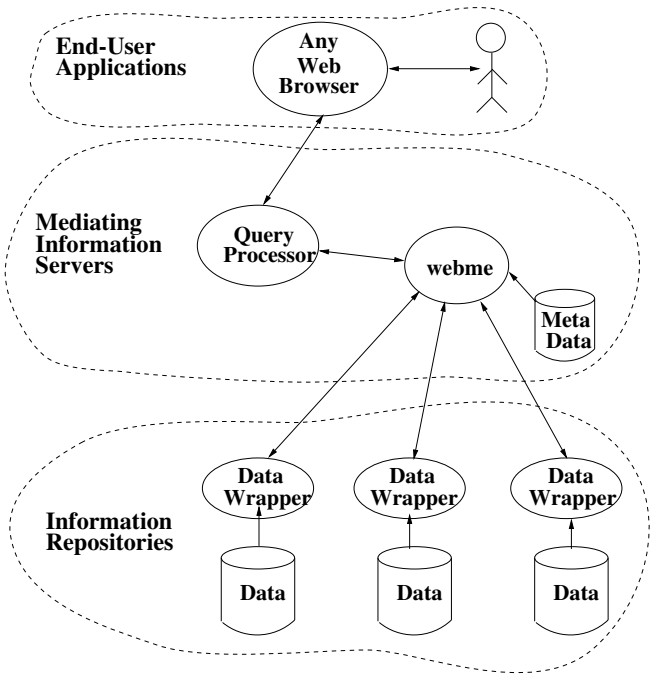

Figure 1: WebME System Architecture.

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[4, 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 a WebME mediator as shown in Figure 2. A WebME mediator is a continuously running process which receives requests from the script processor or end-user. When a request comes from the script processor, the class and interface definitions are updated.

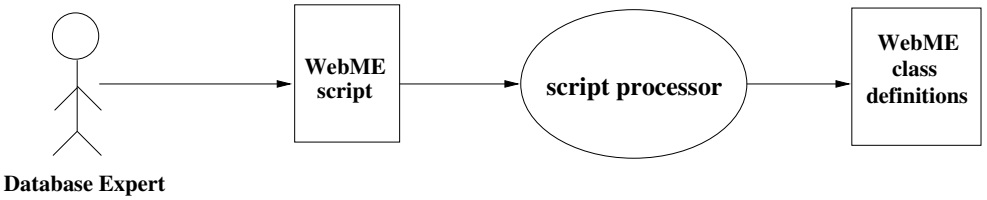

Figure 2: WebME data and interface definition process.

When an end-user makes a request, the class and interface definitions are used by a WebME mediator to gather and process the necessary data. When a request comes from an end-user, a process is started which passes the query request to the appropriate mediator. The mediator processes the query and sends the results to the requesting process. The results are incorporated into an HTML page and sent back to the requesting Web browser. This process is illustrated in Figure 3.

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

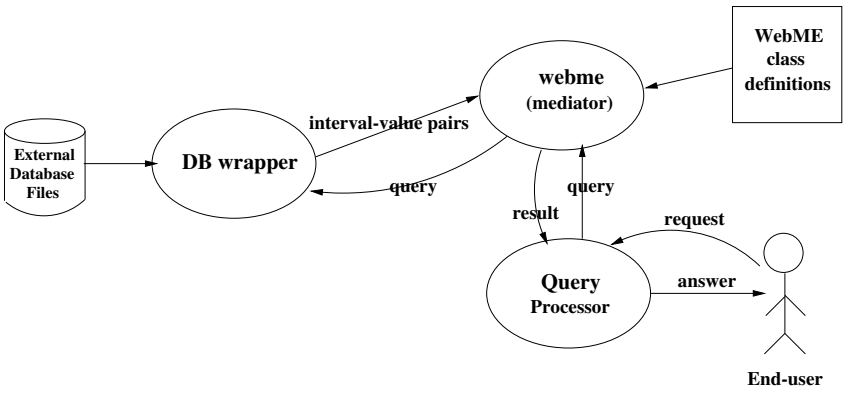

Figure 3: Using class and interface definitions.

The use of a mediated architecture provides flexibility by allowing changes in the structure of the architecture. The wrappers at the information repositories allow the owners of the data to limit access to the data. Each site controls the viewable information with the access methods provided to the repository. This provides the basic security measures inherent in current large-scale development projects. Subcontract management is becoming of increasing concern as subcontracted organizations don't wish to reveal proprietary data whereas the prime contractor needs such data for proper project control and risk management. The mediator architecture allows the subcontractor to control what information is provided to the prime contractor based upon a mutual agreement between the parties. However, as currently implemented, the system provides no explicit security measures for transporting data. By using a Web browser at the end-user level, access to the data is provided on a wide variety of platforms. Updates to the data are available immediately.

Figure 4 is a screen shot from the WebME system. In this figure, the user has requested a growth plot of the cumulative number of reported errors for a single project, COBEAGSS. In addition to the actual data, the guidelines show the normal range of projects similar to the current project.

WebME screen shot goes here (file = webme-screen.tif)

Figure 4: Example WebME screen shot showing errors on COBEAGSS compared to baseline errors (3 line band surrounding actual errors).

3 DESCRIBING DATA

Distributed developments require the combination of data from various locations. In order to assess and monitor the progress of the entire project, data collected from each location must be combined. When similar data sets are collected from two different environments, it would be useful to be able to display the data on the same graph for comparison purposes.

In order to describe the data attributes for each site, the structural model of measurement[6] is used. This framework identifies units and values as properties of attributes. We have added a temporal property to those attributes (e.g., weekly, monthly) as a third *interval* property. A measurement instrument uses the units and the interval to supply the correct value for an attribute. The attribute definition represents the wrapper used for extracting data from the external database in WebME's mediator architecture.

All data to be displayed will be sequenced data with ratio scale type. For direct measures, the measurement *instrument* is an executable program that will extract measured values at the desired interval from a database. For indirect measures, the measurement instrument is an *equation* whose execution computes the attribute's values. The allowable operations in the equations are 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).

The compatibility of attributes used in the equations of indirect attributes must be validated to provide consistency in the combination of the data and for display of data on the same set of axes. We use name equivalence to define compatibility. 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. Attributes that are compatible (i.e., have equivalent units) may be plotted on the same graph.

The WebME data definition scripting language facilitates the combination of data. 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.

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. This allows for different, but related data that are collected and stored separately to be viewed consistently.

The scripting language allows for flexibility in the types of data that can be defined. If measurements are collected for a new attribute, the scripting language can be used to provide access to that data. Although consistency checks for the units and intervals are in place, there are still difficulties that may be encountered. For example, lines of code (LOC) is a unit that takes on different meanings. If LOC is used as a unit, it is the responsibility of the database expert to make sure that the units at each location mean the same thing. There may be other properties of attributes that should be checked to ensure consistency. As we gain more experience with the scripting language, more properties may be added.

A potential weakness with the scripting language as currently specified is the descriptive power of the access method definitions. Access methods are defined by specifying an executable to be used as an instrument. Although comments may be included in the script, the executable name, host, and the parameters to the executable are the only description of the instrument stored in the meta-data.

4 MODEL BUILDING

The consistent combination of data from multiple sources is one part of the problem that the WebME system attempts to address as extensions to the original SME concept. Building additional models from those present in SME for the combined data for the purposes of process control and improvement is another.

The modeling technique used in the SME system is defined and used to build baseline and predictive models of growth data. Those models built into SME were static. For example, a set of similar projects were averaged together to form a baseline attribute (e.g., the average number of reported errors per week), and then a new project could be compared to this average to see how the new project performed relative to the older projects in the SME database (e.g., Figure 4).

WebME uses a more dynamic evolving set of models. In 1993, a clustering algorithm using Euclidean distance was investigated [7] as an alternative to the existing static SME growth models. Only those projects with the same `shape` (i.e., were in the same cluster as the new project) were included in the baseline average.

The current growth modeling algorithms seem 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. We have developed an algorithm which attempts to uncover significant trend changes in noisy data[12]. Our algorithm uses statistical techniques used to develop a class of economic forecasting models [1]. Once the major trend changes have been identified, we can draw what we call a *characteristic curve*. It is hoped that this algorithm and the resulting characteristic curve will give a clearer picture about the underlying process from which the measurement was taken.

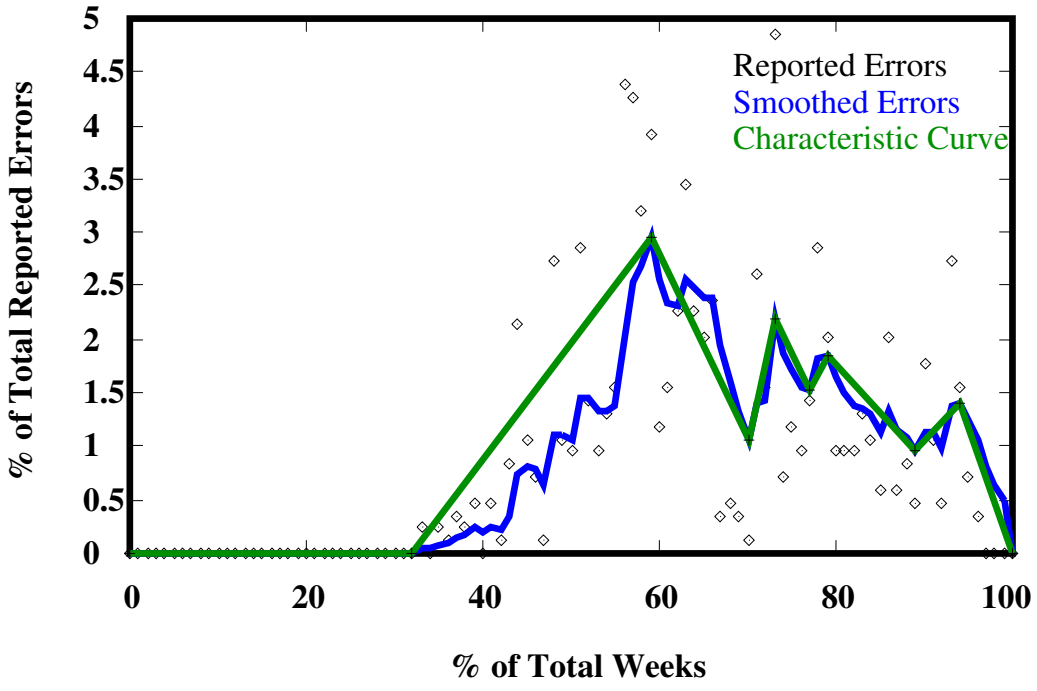

Figure 5: Raw data with its characteristic curve

The scatterplot of Figure 5 represents the weekly number of error reports filed for a single NASA project.[2]

[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.

This data is quite noisy and it is hard to see any trend or underlying model in the data.

Figure 5 shows the characteristic curve computed by our algorithm along with the original data. The characteristic curve exposes an underlying structure to the data that is not apparent from the scatterplot alone.

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 Example description

To illustrate how the scripting language is used to combine data and to provide feedback, we will use bug report data collected from the Guidance Navigation and Control Rapid Development Laboratory (GN&C RDL) of NASA's Johnson Space Center. The GN&C RDL uses the ClearDDTS tool [3] to collect information about their bug reports. One project using the ClearDDTS tool, the Orbiter Upgrade Deorbit Flight Software Demonstration Project (Deorbit) is divided into two major subsystems, the flight software (FSW) and simulator (Sim) subsystems. Suppose project management wants to view the data from the tool to get a better understanding of the testing process in order to build a baseline for future projects. More specifically, the differences between major and minor bug reports are to be examined.

In order to incorporate the data collected by the ClearDDTS tool into the WebME system, the following steps would be taken. First, the locations of the machines where the data are stored must be described. Next, the interfaces to the information repositories have to be described. Finally, a class of attributes must be defined to describe the types of data that will be viewable in the system. In the following subsections, the details of these steps are described.

5.2 Definitions for location

The data for this project are actually stored in a single database. However, for illustration purposes, we assume the data are stored on different machines as shown in Figure 6.

To specify the physical locations of the information repositories, hosts are defined with the WebME scripting language by specifying the host name and port number. At each host, a data wrapper listens to the specified port for WebME data requests. In the current example, there are two hosts. The hosts are *fsw.node* (listening to port 2000) and *sim.node* (listening to port 2010).

The following host definition statements would appear in the WebME script:

create host fsw.node **port**=2000;
create host sim.node **port**=2010;

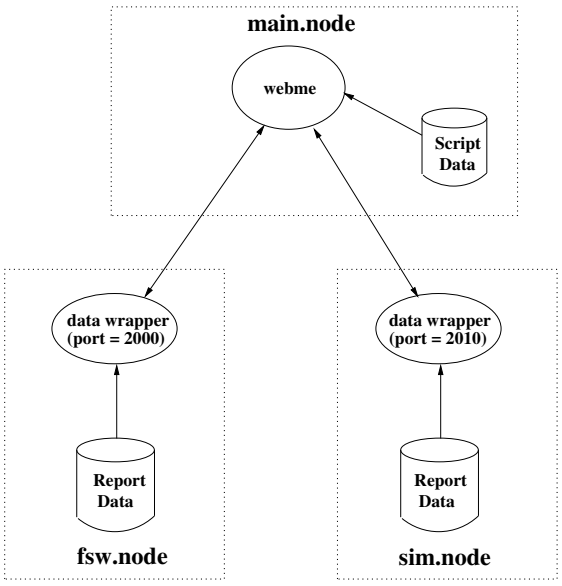

Figure 6: Example architectural schema for Deorbit project.

5.3 Definitions for access

In addition to identifying the physical locations of the information repositories, the *webme* mediator needs the definitions of the access methods to the information repositories. In the WebME scripting language, this is done by defining instruments. In an instrument definition, a host name, a path to an executable, and any parameters to the executable are specified.

In the current example, we need to define two instruments, *FSWaccess* and *SIMaccess* for each information repository. The executable *getDDTSattr* will be used to extract the data from the Report Data repository. The *getDDTSattr* takes two parameters, the entity name and the attribute name. These parameters will be identified at runtime.

The following instrument definition statements would appear in the WebME script:

create instrument FSWaccess
host=fsw.node, **path**=/usr/bin/getDDTSattr,
parameters=$entity $attribute;

create instrument SIMaccess
host=sim.node, **path**=/usr/bin/getDDTSattr
parameters=$entity $attribute;

5.4 Definitions for data

To describe the type of data that will be viewed with the WebME system, classes of attributes are defined with the scripting language. Entities possessing the described attributes are assigned to classes.

In the current example, we are interested in the severity and the analysis time required for each bug

report. We would also like to know how severity impacts the rate of analysis time per bug report. The severity and analysis time for each bug report can be obtained directly from the database of each subsystem. Direct attributes are defined to extract the number of major and minor bug reports submitted (fswMajor, simMajor, fswMinor, simMinor) and the analysis time for major and minor bug reports (fswMajorAnalysis, simMajorAnalysis, fswMinorAnalysis, simMinorAnalysis) for each subsystem. Indirect attributes are defined to compute the total number of bug reports (Major, Minor), the total amount of analysis time (MajorAnalysis, MinorAnalysis), and the rate of analysis time per bug report (MajorRate, MinorRate). These attributes are defined for a class called RDL with one entity, the Deorbit project.

The following statements would appear in the WebME script:

```
create class RDL;
assign Deorbit to RDL;

/* number of bug reports for flight software */
create attribute direct RDL.fswMajor
    with units Reports, interval Week, instrument FSWaccess;
create attribute direct RDL.fswMinor
    with units Reports, interval Week, instrument FSWaccess;
/* number of bug reports for simulator */
create attribute direct RDL.simMajor
    with units Reports, interval Week, instrument SIMaccess;
create attribute direct RDL.simMinor
    with units Reports, interval Week, instrument SIMaccess;
/* analysis time for flight software */
create attribute direct RDL.fswMajorAnalysis
    with units Reports, interval Week, instrument FSWaccess;
create attribute direct RDL.fswMinorAnalysis
    with units Reports, interval Week, instrument FSWaccess;
/* analysis for simulator */
create attribute direct RDL.simMajorAnalysis
    with units Reports, interval Week, instrument SIMaccess;
create attribute direct RDL.simMinorAnalysis
    with units Reports, interval Week, instrument SIMaccess;
/* indirect attributes */
/* number of bug reports */
create attribute indirect RDL.Major using fswMajor + simMajor;
create attribute indirect RDL.Minor using fswMinor + simMinor;
/* analysis time */
create attribute indirect RDL.MajorAnalysis using fswMajorAnalysis + simMajorAnalysis;
create attribute indirect RDL.MinorAnalysis using fswMinorAnalysis + simMinorAnalysis;
/* rate of analysis time per bug report */
create attribute indirect RDL.MajorRate using MajorAnalysis/Major;
create attribute indirect RDL.MinorRate using MinorAnalysis/Minor;
```

5.5 Viewing compatible data

Once the script has been parsed, the definitions will be made available to the *webme* mediator and compatible attributes can be viewed through any Web browser. Figure 7 shows the growth in the total number of Major and Minor error reports for the Deorbit project.

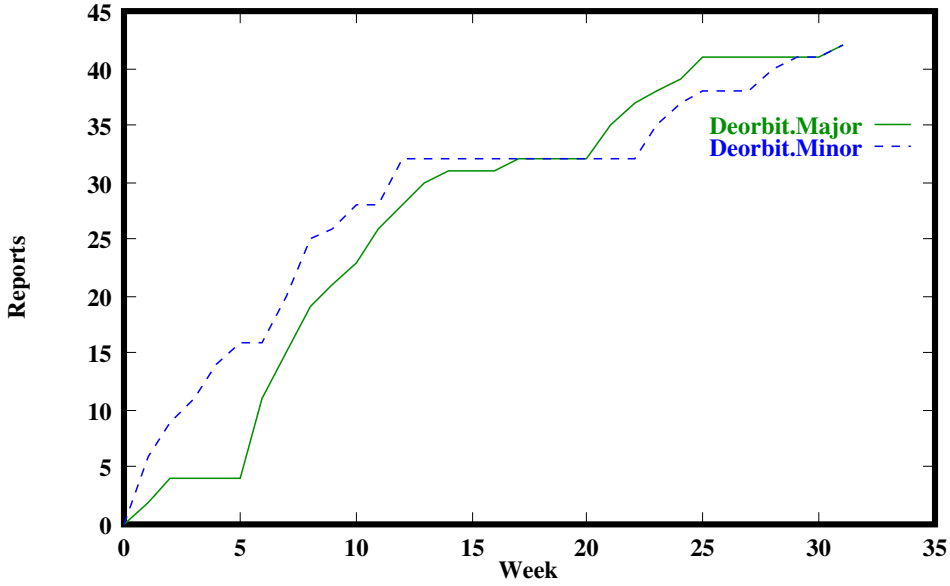

Figure 7: Growth in total number of Major and Minor reports.

Figure 8 shows the growth in the rate of analysis time per error report separated into the Major and Minor categories. This graph illustrates that severity does have an impact on the total analysis time. Over time, the difference between the two curves becomes greater. As expected, the more severe errors appear to take longer to analyze. However, it is a common belief that software errors found later in the development process are more time-consuming and difficult to fix. Figure 8 shows that this is only true for severe errors. Minor errors do not take appreciably more time to analyze regardless of when they are found. If data such as this can be confirmed from other projects, it could have implications in the appropriate process model needed to find minor and major errors.

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. Approaches like the Experience Factory have been proposed as a means to organize such developmental practices. However, means must be found for passing information among such environments and 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. We have had a limited amount of experience using the prototype WebME system. The system has been used with data from two environments. We have used the scripting language to define the SEL data that was originally available in SME. The scripting language has also been used with data from the GN&C RDL and we expect to follow through with future GN&C RDL projects. We have been able to uncover interesting information (e.g., structure of SEL reported errors, GN&C RDL major and minor analysis rates) from these datasets. While the initial results are encouraging, we need to conduct studies with additional

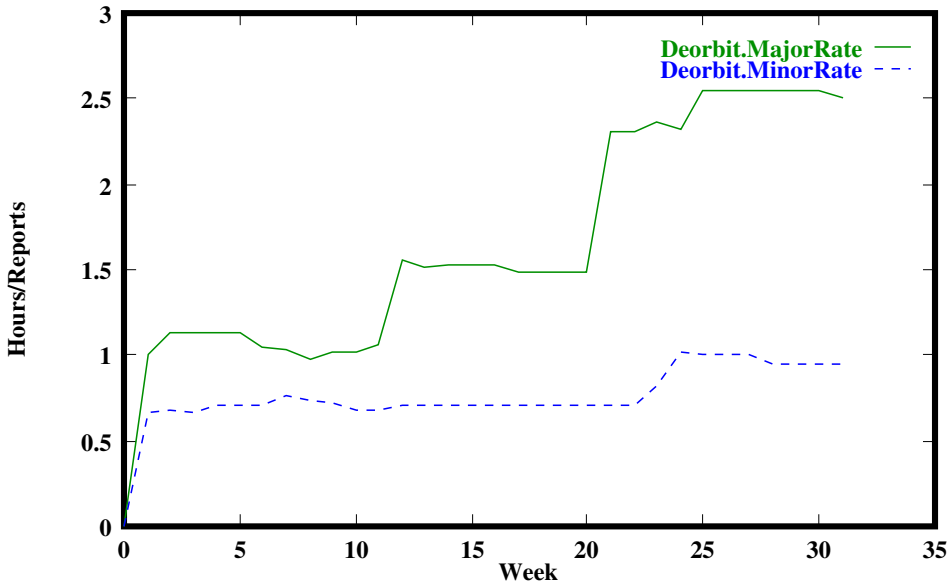

Figure 8: Rate of growth in analysis time per error report.

datasets in order to more fully evaluate the system and to determine the effectiveness of our system in building indirect attributes across a wide range of application domains. 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.

ACKNOWLEDGMENTS

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. We acknowledge the help of David Petri of the JSC GN&C RDL for providing us with the data from the Johnson Space Center.

References

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

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

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

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

[5] R. Hendrick, D. Kistler, and J. Valett. Software management environment (SME) concepts and architecture (revision 1). Technical Report SEL-89-103, SEL, September 1992.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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* 19^{th} *International Conference on Software Engineering*, pages 14–21, May 1997.

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

[12] R. Tesoriero and M.V. Zelkowitz. A model of noisy software engineering data (status report). In *Proc. of the 20th International Conference on Software Engineering*, April 1998.

[13] 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.

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

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