Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development

Richard W. Selby, Adam A. Porter, Doug C. Schmidt, and Jim Berney

Department of Information and Computer Science*, University of California
Irvine, California 92717, 714-856-6326, selby@ics.uci.edu

Abstract

Metric-driven analysis and feedback systems enable developers to define *empirically guided processes* for software development and maintenance. These systems enable developers to achieve S.E.I. process maturity levels 4 and 5, "managed" and "optimizing" processes. We define a set of architectural principles and abstract interfaces for designing metric-driven analysis and feedback systems. A fundamental principle is to *make measurement active* by integrating measurement and process, which contrasts with the primarily passive use of measurement in the past. A second principle is that these systems should provide capabilities for user-specifiable interpretation of process events, object state changes, and calendar time abstractions in order to allow a representative basis for project analysis. Another principle is that these systems should provide an extensible integration framework that supports the addition of new techniques and integrates approaches for synergistic application.

These and other architectural principles are embodied in a prototype analysis and feedback system, called *Amadeus*. This prototype system is intended to demonstrate the feasibility and merit of these systems and the principles for designing them. An initial version of the system is running, and it has been used in several evaluative studies. We define the abstract interfaces in *Amadeus* and describe its operation.Metric-driven analysis and feedback systems influence the design of process modeling and programming languages and software environment architectures. Accordingly, *Amadeus* is integrated with and leverages the *Arcadia* process-centered software environment architecture.

*This work was supported in part by the National Science Foundation under grant CCR-8704311 with cooperation from the Defense Advanced Research Projects Agency under Arpa order 6108, program code 7T10; National Aeronautics and Space Administration under grant NSG-5123; National Science Foundation under grant DCR-8521398; University of California under the MICRO program; Computer Sciences Corporation; Hughes Aircraft; and TRW.

1 Introduction

Measurement and empirical analysis systems enable the systematic evaluation and improvement of large-scale software. The most advanced levels of the S.E.I. process maturity framework, "managed" and "optimizing" processes (levels 4 and 5), require the use of measurement techniques and highlight the need for these systems [8]. We have specified a measurement and empirical analysis system, called *Amadeus*, which enables a software developer to integrate measurement and empirical feedback mechanisms into development and maintenance processes. These *empirically guided processes* can be used, for example, to focus tool and technique application on high-payoff areas. The underlying principle is to *make measurement active* by integrating measurement and process, which contrasts with the primarily passive use of measurement in the past.

Developers and maintainers can use metric-driven analysis systems to guide the application of synthesis, analysis, and evaluation techniques and tools. The "80:20 rule" states that approximately 20 percent of a software system is responsible for 80 percent of its errors, costs, and rework [5]. Boehm and Papaccio [4] conclude:

> "The major implication of this distribution is that software verification and validation activities should focus on identifying and eliminating the specific *high-risk* problems to be encountered by a software project, rather than spreading their available early-problem-elimination effort uniformly across trivial and severe problems."

Empirically guided processes leverage the feedback

Recommended by: Watts Humphrey

provided by metric-based analysis systems to focus development efforts on the high-payoff areas, i.e., the "troublesome 20 percent."

Metric-driven analysis systems for enabling empirically guided processes influence many aspects of software research and practice:

- software environment architectures;
- process modeling and programming formalisms; and
- techniques for definition, collection, and analysis of empirical data.

This paper summarizes the goals of metric-driven analysis and feedback systems and describes a prototype system, *Amadeus*, which defines abstract interfaces and embodies architectural principles for these types of systems.

2 Metric-Driven Analysis and Feedback Systems

Several metric-based analysis systems have been proposed or are under development: *TAME* [1], *Ginger* [22], *SME* [6], and *AMS* [10], among others. These systems support empirically based techniques for using measurements to describe, analyze, and control software systems and their development processes. These modeling techniques leverage past experience and have many desirable properties, including being scalable to large systems, integratable, and calibratable to new projects. The underlying capabilities of these systems facilitate the definition, collection, analysis, and feedback of empirical metric data — software *metrics* are numeric and symbolic abstractions of software artifacts, e.g., components, processes, systems. There are also several general paradigms for empirically based software development and evaluation: Basili's improvement paradigm [1] [3], Humphrey/SEI's process maturity levels 4 and 5 [8], McCall/RADC's factor-criteria-metric paradigm [10], Basili/Weiss' goal-question-metric (GQM) paradigm [2], Selby/Porter's classification paradigm [18], and Weiderman/SEI's environment evaluation methodology [23], among others. In this paper, we describe the *Amadeus* system which provides a flexible and extensible framework for supporting the approaches in these systems and paradigms.

The goals of the *Amadeus* system are to:

- Provide measurement and empirical feedback mechanisms,
- Define software environment architecture mechanisms for enabling empirically guided development and maintenance processes (level 5),
- Formulate an extensible integration framework for empirically based analysis techniques, and
- Define abstract interfaces for measurement and empirical analysis systems.

Amadeus focuses on evaluation and analysis capabilities for *large-scale* software systems and processes (e.g., >100,000 lines). Measurement-based techniques can be used effectively on very large systems, and they can guide toward high-payoff areas fine-grain analysis techniques that are more tractable on smaller portions of systems. Early foundations of the *Amadeus* system have been described in [16] [17].

3 *Amadeus* System Operation

This section provides an overview of the *Amadeus* system operation, including an example empirically guided process, a description of the system characteristics, an explanation of the system conceptual operation, and a summary of the users' view of the system.

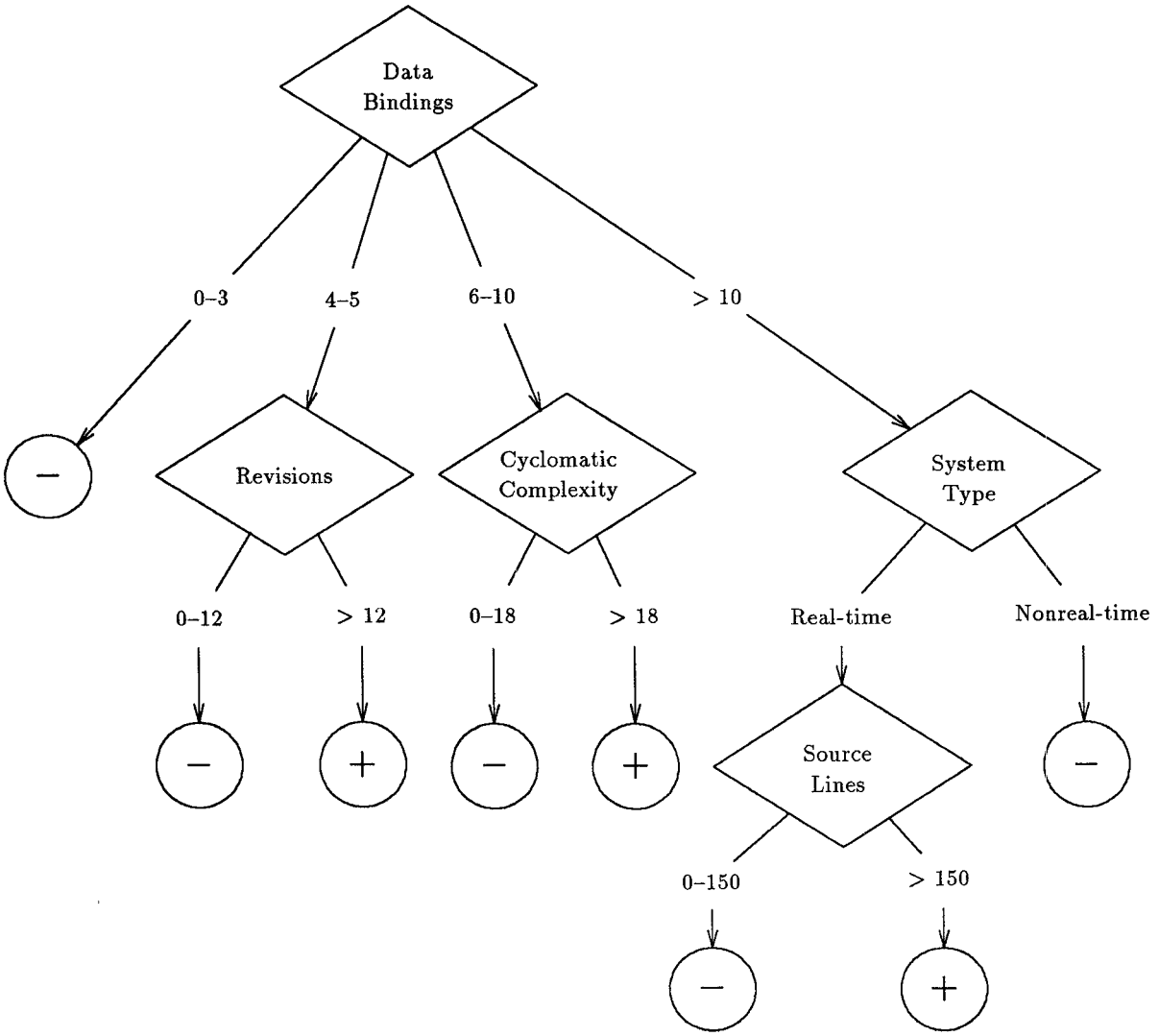
3.1 Example Empirically Guided Process

Figure 1 shows an example fragment of an empirically guided software maintenance process. This simple, hypothetical example indicates that a software maintainer will check-out a software object (e.g., component) that needs to be modified, modify it, and then check it back into the object store. This maintenance process has been statically instrumented with *Amadeus* data collection and analysis scripts. After the maintainer has modified the object, an empirical analysis is conducted on the object to calculate whether it is likely to have interface or control faults. If it is likely (based on past data) to have either of these types of faults, appropriate analysis and testing tools are executed on the object.

In this example, the empirical process guidance is derived from the use of metric-based classification trees and underlying metric collection mechanisms. An example classification tree appears in Figure 2, and Section 5 describes this analysis technique more completely. The classification tree generation and metric collection processes are embodied as *Amadeus* "agents," and they are triggered via "process events" represented as procedure-calls in this example. This example scenario illustrates one type of *Amadeus*

```
begin
...
        -- as part of a software maintenance activity, a maintainer decides that
        -- object_i in system_k needs to be modified

        -- object_i is checked out from the object store
check_out (object_i, system_k);
        -- maintainer modifies object_i

        -- tools generate metric-based classification trees for identifying components
        -- likely to have interface faults or control faults
classification_tree interface_faults, k :=
    generate_classification_tree ("interface_faults", system_k, metric_data_k, calibration_parameters);
classification_tree control_faults, k :=
    generate_classification_tree ("control_faults", system_k, metric_data_k, calibration_parameters);

        -- tools collect the metric data on object_i required by the classification trees
metric_data_k :=
    collect_metric_data ( classification_tree interface_faults, k , object_i, system_k, metric_data_k);
metric_data_k :=
    collect_metric_data ( classification_tree control_faults, k , object_i, system_k, metric_data_k);

        -- metric-based classification trees guide tool and technique application
if  apply_classification_tree ( classification_tree interface_faults, k , object_i, metric_data_k)
            > prob_j
    then
        -- apply to object_i the analysis and testing tools designed
        -- to detect or isolate interface faults
fi
if  apply_classification_tree ( classification_tree control_faults, k , object_i, metric_data_k)
            > prob_m
    then
        -- apply to object_i the analysis and testing tools designed
        -- to detect or isolate control faults
fi

        -- object_i is checked back into the object store
check_in (object_i, system_k);
...
end
```

Figure 1: Example fragment of empirically guided process that uses classification analysis and a straightforward branching form of feedback.



"+" = Classified as likely to have interface faults

"−" = Classified as unlikely to have interface faults

Figure 2: Example (hypothetical) metric-based classification tree. There is one metric at each diamond-shaped decision node. Each decision outcome corresponds to a range of possible metric values. Leaf nodes indicate whether or not an object is likely to have some property, such as high error-proneness or errors in a certain class.

event, a statically interpreted process event, and the following sections explain a more comprehensive view of the system.

3.2 System Characteristics

The *Amadeus* system provides an interactive script language that encapsulates environment mechanisms for static and dynamic interpretation of process events, object state changes, and calendar time abstractions. These three event types can be combined to form compound events, e.g., object changes and time abstractions can be combined in an event defined as "every time an object changes but no more frequently than daily." The system is composed of software environment architecture components, such as event monitors, data integration frameworks, and a language interpreter, as well as capabilities for specifying empirical analyses, collecting the underlying data, and feeding the results back into development processes. The *Amadeus* system embodies:

- scalable, calibratable, empirically based evaluation and analysis,
- triggering based on events from process model or program executions, object state changes, or calendar time abstractions,
- static or dynamic event interpretation,
- separation of event and agent specification,
- transparent, concurrent data collection, and
- low entry barrier through script reuse.

The *Amadeus* system can provide services to several different types of users. Software developers may use the system to localize components likely to be error-prone or to identify components likely to be reusable. Project managers may use the system to monitor human resource expenditure or system progress. Analysts and experimenters may use the system to compare the effectiveness of different processes or tools or to generate error frequency profiles.

3.3 System Conceptual Operation

The conceptual operation of the *Amadeus* system appears in Figure 3. This is a complicated figure but it summarizes many system characteristics and interactions. A software developer or analyst activates *Amadeus* scripts that monitor events from process model/program executions, object state changes, and system clock abstractions [see upper right of Figure 3]. These events trigger user-specifiable "agents" (embodied as processes) that analyze the state of the environment and its constituent processes and objects [see far left of Figure 3]. These agents collect, analyze, integrate, or display data, but may also be used for other purposes such as process coordination. Process programs or human users may interactively query the collected information for guidance about how the data affect the continued execution of processes [see upper left of Figure 3]. The collected information is used to guide and adapt processes and, for example, to focus tool application on high-payoff areas. The measurement and empirical guidance capabilities enable "managed" and "optimizing" (level 4 and 5) processes. The reusable scripts enable the system to provide users with a low entry barrier.

Amadeus separates event specification from agent specification, which allows users the option of deferring the specification of agents corresponding to events until after process execution has commenced. In Figure 3, static interpretation is depicted on the left half and dynamic interpretation is on the lower right. A user specifies the conditions for an event to occur (based on either process, object, or clock representations) and then either specifies a corresponding agent or leaves it unspecified (which becomes the null agent). If a corresponding agent is specified, it can be interpreted statically and incorporated directly into the process, object, or clock representation (as was done in Figure 1). Whether or not an agent has already been specified, additional agents may be specified either before, during, or after the execution of a process. *Amadeus* dynamically interprets agents specified during process execution. So when an event occurs, multiple agents may be triggered some of which may have been specified after the commencement of the process execution. This capability is especially important for long-running processes that may need to be changed while they are executing, because the appropriate agents will often not be able to be foreseen until the process is underway.

3.4 Users' View of the System

An *Amadeus* user interacts with the user-customizable goal palette that provides a concise summary of the analysis processes and services available. A user may select a particular analysis process, such as one for classification, interconnectivity, sensitivity, or descriptive analysis. The system displays and executes the process, and it guides a user through the technique. See Section 5 for a description of classification and interconnectivity analysis. The processes selected from the

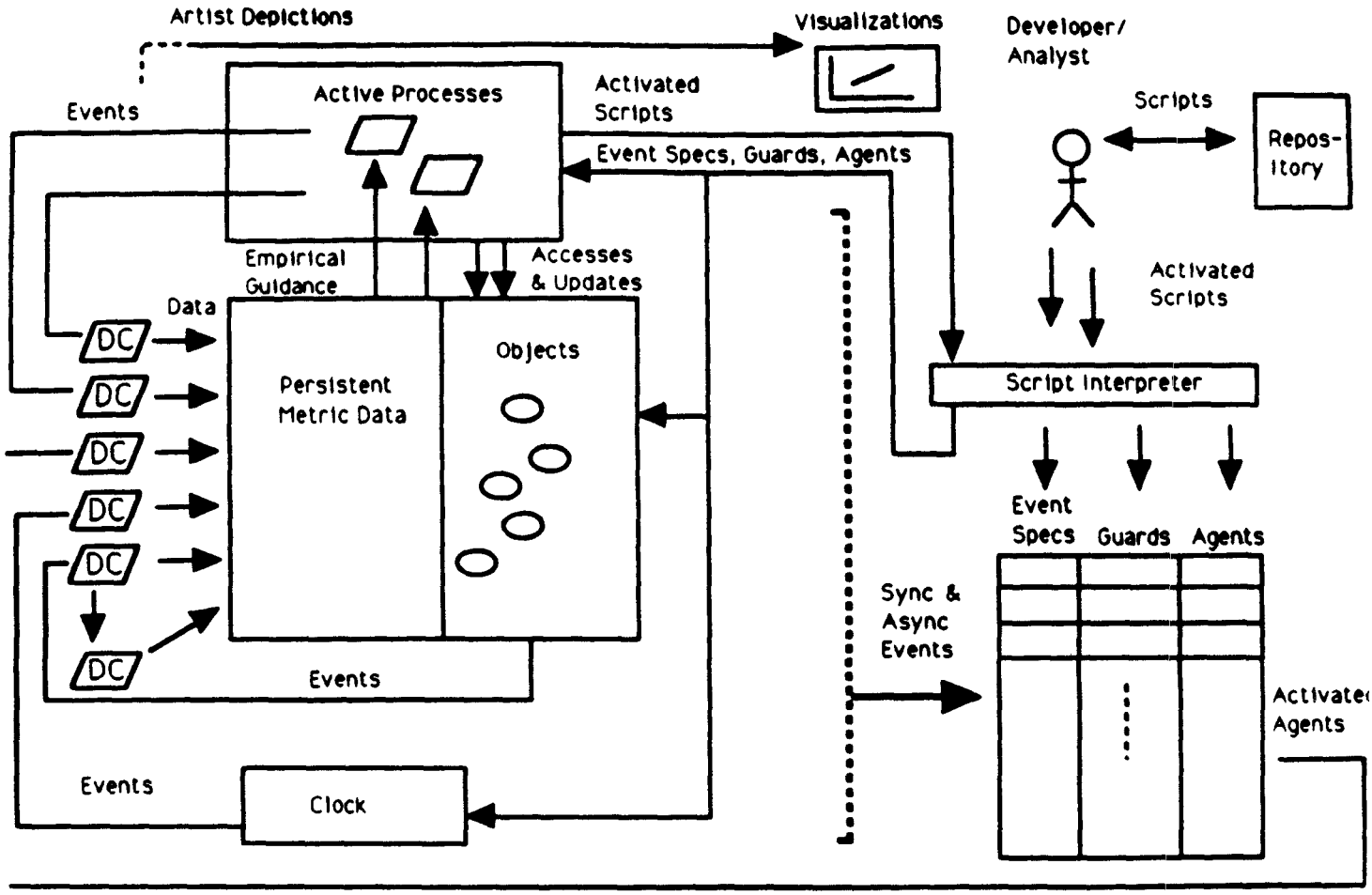
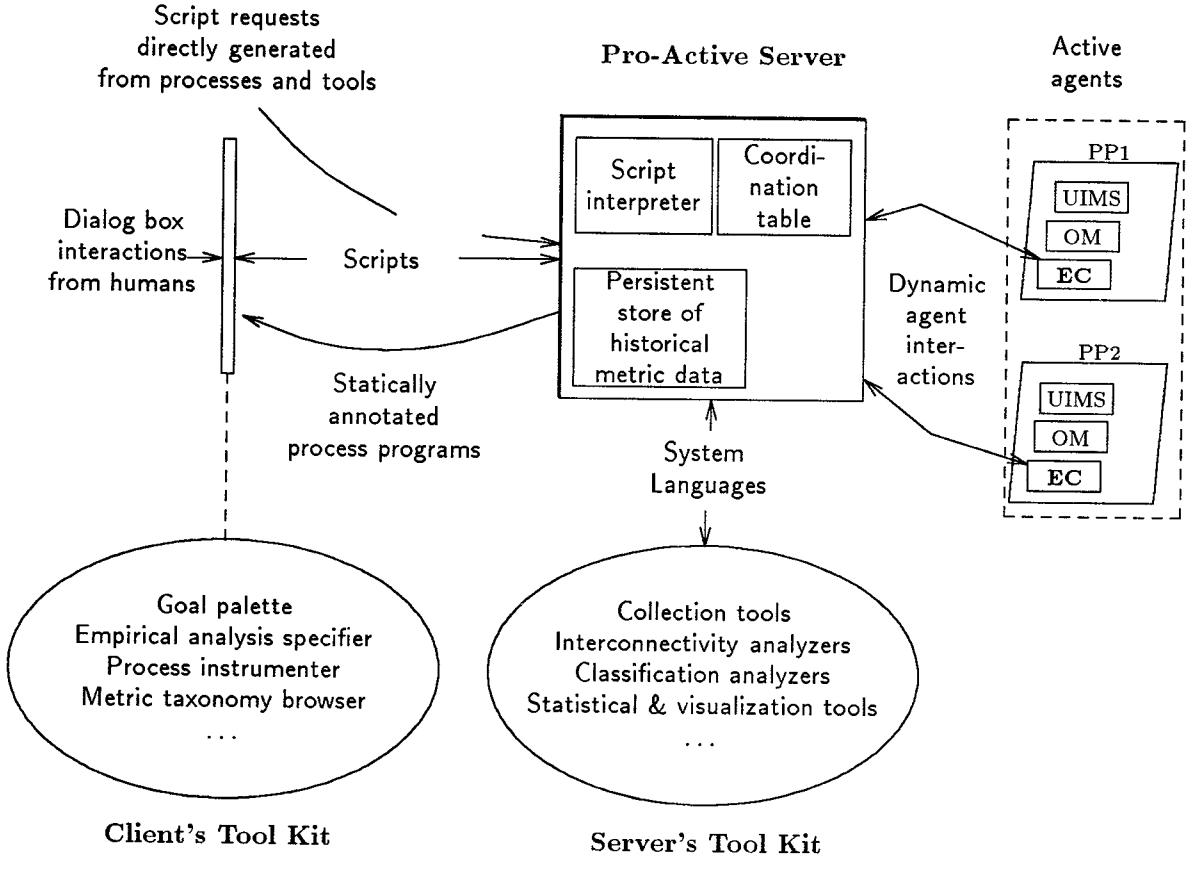


Figure 3: Conceptual operation of the *Amadeus* metric-driven analysis and feedback system.

goal palette coordinate the activation of scripts and tools that collect the necessary data, integrate it, and feed the resulting empirical guidance back into development processes. Measurement tools are examples of agents that may be triggered by various scripts that monitor events from process, objects, and clocks (see Figure 3). Advanced users can directly access some constituent tools to experiment with specialized needs, define defaults, etc.The functions for managing scripts include browsing existing scripts, creating new ones, and showing those that have been activated. Users may also add new capabilities to the system and invoke assistance features.

4 *Amadeus* System Architecture

The conceptual architecture of the *Amadeus* system appears in Figure 4. The centerpiece of the system is a pro-active server, which interprets scripts and coordinates event monitoring and agent activation. Multiple servers can be running at one time, according to functional specialization or workload. The collected data is persistent across projects, and various transformation tools access and modify the data. Extensible data-integration frameworks, which are represented as trees and networks, are defined that enable the integration of symbolic and numeric data from any source (e.g., products, processes, personnel). The desirable properties of attribute integration frameworks are being scalable, automatable, and extensible — both tree- and network-based frameworks possess these proper-



Scripts	=	Condition-action pairs. Conditions: event-based, object-based, or time-based. Actions: processes or tools.
Server	=	Pro-active server interprets scripts, delegates dynamic collection to individual EC's, and coordinates analysis across multiple EC's. Server is PPL, UIMS, and OM independent.
EC	=	Evaluation component and monitor in active agent. EC is PPL, UIMS, and OM dependent.

Figure 4: Conceptual architecture of the *Amadeus* metric-driven analysis and feedback system. PPL is "process programming language," UIMS is "user interface management system," and OM is "object manager."

ties. Scripts may be defined and activated by human users or they may originate directly from processes or tools. Since a common script language is used to represent these requests, the server's script interpreter does not care if a request is from a human, a process, or another server. This symmetry of interaction results in increased flexibility and interoperability. The classification process mentioned in Section 3.4 is an example process that would generate script requests.

The "client toolkit" and "server toolkit" in Figure 4 are both extensible collections of system capabilities. Users can add particular data specification, collection, analysis, and feedback tools to these toolkits. The client toolkit contains capabilities that human users may access directly, and these tools are primarily for assistance in the script definition and management processes. The server toolkit contains capabilities that are used by the server as part of script interpretation.

The *Amadeus* system is integrated with the *Arcadia* software environment architecture [21]. *Amadeus* leverages the process interpretation [20], object triggering [7], and graphical depiction [24] components in *Arcadia*.

5 Support for Empirically Based Analysis Techniques

Amadeus scripts encapsulate data collection, analysis, and display techniques and define when they should be executed (see Figures 3 and 4). One of the goals of the system is to allow users the flexibility of specifying agents appropriate for their particular goals and circumstances. It is important for evaluation and feedback systems to be extensible in terms of allowing users to add new techniques to the system. *Amadeus* provides an extensible framework for adding new empirically based analysis techniques, which is represented as the server's toolkit in Figure 4. In order to validate the framework, we are populating it with techniques that we are developing as part of the project and ones developed externally. The following sections describe two techniques that we are developing: classification analysis and interconnectivity analysis.

5.1 Classification Analysis

According to the "80:20 rule," 20 percent of a software system is responsible for 80 percent of its errors, costs, and rework. We are developing classification analysis techniques for identifying the high-risk components (i.e., the "troublesome 20 percent") throughout the lifecycle [18]. These techniques classify software components according to their likelihood, based on past systems, of having user-specified properties such as high error-proneness, certain types of errors, or high development effort. Developers can define their own "target classes" of components to identify, based on their particular goals and constraints. The classification models are scalable, extensible, automatable, and calibratable, and they serve as a framework for integrating symbolic and numeric information from software products, processes, and personnel.

We conducted extensive validation studies using tree- and network-based classification models on NASA [18] and Hughes system data [19]. An example tree-based classification model appears in Figure 2. We defined a classification methodology [11] and developed distribution-sensitive data partition methods [12].

5.2 Interconnectivity Analysis

The interconnections among software components constitute a central feature of system structure. We are developing interconnectivity analysis techniques for characterizing, evaluating, and visualizing system structure and its evolution [14]. Interconnectivity analysis methods facilitate the understanding of complex component interrelationships in large-scale systems and enable developers to evaluate potential modifications. The analysis technique incorporates multiple interconnection criteria, calculates the interconnections automatically, and derives multiple views of system structure and its evolution. The technique is scalable to large systems and can be used, for example, to localize error-prone structure and to evaluate system flexibility. An example hierarchical system view derived from component interconnections appears in Figure 5.

We applied the interconnectivity analysis technique in validation studies where it detected the most error-prone (7:1) parts of a 148,000-line system [15]. We analyzed the component interconnections through non-local data objects to characterize flexible and inflexible sub-structures in existing systems, using concepts such as "producer" and "consumer" components (which define and reference non-local data values, respectively). Our assertion is that appropriate system structure, in terms of interconnectivity measures, is a necessary but not sufficient condition for system flexibility. This application of interconnectivity analysis can be viewed as "proving a small theorem about a large program" (i.e., showing the presense or absence of system sub-structures known to be flexible or inflexible), as opposed to the research theme of "proving a large theorem about a small program" (e.g., proving correctness of a GCD algorithm).

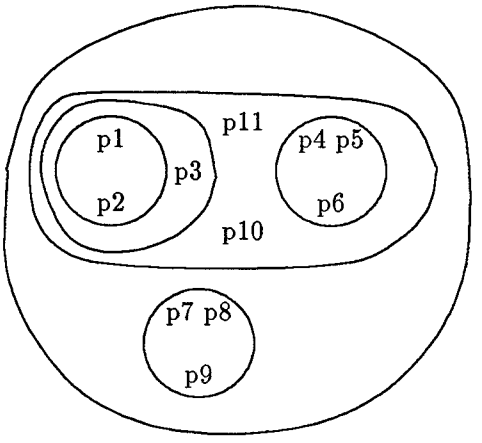


Figure 5: Example hierarchical system view derived from component interconnectivity analysis. Routines (e.g., procedures, functions) are denoted by p_i, and clusters are denoted by circles. The smaller clusters are relatively tighter in terms of degree of interconnection (and form earlier), while the larger clusters are relatively looser (and form later). The clusters define a system hierarchy in the form of a tree: the smaller clusters at the leaf nodes and the largest cluster at the root node.

5.3 External Tools

Empirically based analysis techniques from other research organizations, including USC, University of Maryland, Virginia Tech, and TRW, may also be integrated into *Amadeus*. Incorporating these tools into the *Amadeus* system validates its extensibility and enables synergistic interaction among techniques.

6 Development Status

Validation studies showed that *Amadeus'* empirical guidance features correctly categorized 89.6% of the components on the average, according to whether or not they were within a particular target class (such as having high errors or high development effort) [18] [9]. The guidance had 79.5% consistency and 69.1% completeness on the average, which are measures of the intersection between the set of components predicted to be within the target class and those actually within it.

Amadeus has a broad spectrum of applications, including system evaluation and improvement, process guidance, software understanding, and experimentation. An initial version of *Amadeus* is up and running. Several organizations have already committed to or expressed serious interest in using all or part of it.

7 Conclusions

An important goal in our work is to develop a framework that supports multiple empirical evaluation and feedback paradigms, such as Basili's improvement paradigm, Humphrey/SEI's process maturity levels 4 and 5, McCall/RADC's factor-criteria-metric paradigm, Basili/Weiss' goal-question-metric (GQM) paradigm, Selby/Porter's classification paradigm, and Weiderman/SEI's environment evaluation methodology, among others. Our examination and analysis of these paradigms led to the identification of many underlying principles for metric-driven analysis and feedback systems.

These architectural principles are embodied in the *Amadeus* system. One principle, for example, is that the three types of events that *Amadeus* supports — process events, object state changes, and calendar time abstractions — are sufficient to provide a representative basis for monitoring and analysis of software projects and systems. Another principle is that developers need the flexibility of separating the specification of events from the specification of actions that should be taken when those events occur. These actions should be user-specifiable so that they reflect the particular user's goals and circumstances. Also, these systems should provide extensible frameworks for integrating numeric and symbolic data, which enables the analysis of a wide spectrum of data from products, processes, and personnel. *Amadeus* serves as an extensible integration framework for empirically based analysis techniques, and therefore, it should be viewed as a complementary project to these other systems and paradigms.

We have outlined architectural principles and abstract interfaces for metric-driven analysis and feedback systems. In order to refine these principles and demonstrate their feasibility and merit, a prototype system, called *Amadeus*, has been built based on them. The *Amadeus* system can be viewed as providing a set of services that enable empirically guided software development and maintenance. The results from the initial version of *Amadeus* have convinced us that it provides a good foundation for further research. For example, we propose the expansion of the script repository to support a wide spectrum of data analysis and feedback templates. We also need to limit the operating system dependence and to plan for a reimplementation on top of Mach [13]. We plan to investigate the use of *Amadeus* as a general process modeling/programming and coordination system, as opposed to serving as primarily a measurement and analysis system.

8 Acknowledgements

The authors are grateful to many persons who assisted in the design and development of *Amadeus*: Ken Anderson, Tom Asouksamlane, Jose Duarte, Dennis Heimbigner, Greg James, Kent Madsen, Lee Osterweil, Allyn Randall, Debra Richardson, Craig Snider, Richard Taylor, and Harry Yessayan. The authors are also grateful to the many collaborators in the Arcadia project.

References

[1] V. R. Basili and H. D. Rombach. The TAME project: Towards improvement-oriented software environments. *IEEE Trans. Software Engr.*, SE-14(6):758–773, June 1988.

[2] V. R. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6):728–738, November 1984.

[3] Victor R. Basili. Quantitative evaluation of software methodology. In *Proceedings of the First Pan Pacific Computer Conference*, Melbourne, Australia, September 1985.

[4] B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, SE-14(10):1462–1477, October 1988.

[5] Barry Boehm. Industrial software metrics top 10 list. *IEEE Software*, 4(5):84–85, September 1987.

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

[7] D. Heimbigner. Triton reference manual. Arcadia technical report, University of Colorado, 1990.

[8] W. S. Humphrey. Characterizing the software process: A maturity framework. *IEEE Software*, 5(2):73–79, March 1988.

[9] R. Kent Madsen and Richard W. Selby. Metric-driven classification networks for identifying high-risk software components. In *Proceedings of the International Conference on Applications of Software Measurement*, San Diego, CA, November 1990.

[10] J. A. McCall, P. Richards, and G. Walters. Factors in software quality. Technical Report RADC-TR-77-369, Rome Air Development Center, Griffiss Air Force Base, NY, November 1977.

[11] Adam A. Porter and Richard W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, 7(2):46–54, March 1990.

[12] Adam A. Porter and Richard W. Selby. Evaluating techniques for generating metric-based classification trees. *Journal of Systems and Software*, 12(3):209–218, July 1990.

[13] Richard F. Rashid. From RIG to Accent to Mach: The evolution of a network operating system. In *Proceedings of the Fall Joint Computer Conference*, pages 1128–1137, November 1986.

[14] Richard W. Selby. Generating hierarchical system descriptions for software error localization. In *Proceedings of the Second Workshop on Software Testing, Analysis, and Verification*, pages 89–96, Banff, Alberta, Canada, July 1988.

[15] Richard W. Selby and Victor R. Basili. Error localization during software maintenance: Generating hierarchical system descriptions from the source code alone. In *Proceedings of the Conference on Software Maintenance*, Phoenix, AZ, October 1988.

[16] Richard W. Selby, Greg James, Kent Madsen, Joan Mahoney, Adam Porter, and Doug Schmidt. Classification tree analysis using the Amadeus measurement and empirical analysis system. In *Proceedings of the Fourteenth Annual Software Engineering Workshop*, NASA/GSFC, Greenbelt, MD, November 1989.

[17] Richard W. Selby, Greg James, Kent Madsen, Joan Mahoney, Adam Porter, and Doug Schmidt. *Amadeus*, An automated measurement and empirical analysis system: Conceptual architecture overview. Arcadia Technical Report UCI–89–21, University of California, June 1989.

[18] Richard W. Selby and Adam A. Porter. Learning from examples: Generation and evaluation of decision trees for software resource analysis. *IEEE Transactions on Software Engineering*, SE-14(12):1743–1757, December 1988.

[19] Richard W. Selby and Adam A. Porter. Software metric classification trees help guide the maintenance of large-scale systems. In *Proceedings of the*

Conference on Software Maintenance, pages 116–123, Miami, FL, October 1989.

[20] Stanley M. Sutton, Jr., Dennis Heimbigner, and Leon J. Osterweil. Language constructs for managing change in process-centered environments. In *Proc. of the Fourth Symposium on Practical Software Development Environments*, 1990. Irvine, California.

[21] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michal Young. Foundations for the Arcadia environment architecture. In *Proceedings of ACM SIGSOFT '88: Third Symposium on Software Development Environments*, pages 1–13, Boston, November 1988. Appeared as *Sigplan Notices 24*(2) and *Software Engineering Notes 13*(5).

[22] Koji Torii, Tohru Kikuno, Ken ichi Matsumoto, and Shinji Kusumoto. A data collection and analysis system Ginger to improve programmer productivity on software development. Technical report, Osaka University, Osaka, Japan, 1989.

[23] N. H. Weiderman, A. N. Habermann, M. W. Borger, and M. H. Klein. A methodology for evaluating environments. In *Proceedings of the First ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 199 – 207, Palo Alto, California, December 1986.

[24] Michal Young, Richard N. Taylor, and Dennis B. Troup. Software environment architectures and user interface facilities. *IEEE Transactions on Software Engineering*, 14(6):697–708, June 1988.