(Research Paper)

A Tool Suite for Evolving Legacy Software

Spencer Rugaber
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
spencer@cc.gatech.edu

Abstract

Evolving an existing software system is fundamentally different from developing one from scratch. Consequently, tools to support evolution must go beyond traditional development tools. This paper describes the Esprit de Corps Suite (EDCS) of software evolution tools. EDCS supports the Mission Oriented Architectural Legacy Evolution (MORALE) software reengineering process. The paper briefly describes MORALE before presenting the individual tools and how they interoperate to support legacy system evolution.

Keywords: software tools, software evolution, reengineering

1. The Problem

Software evolution is the process of adapting an existing software system to conform to an enhanced set of requirements. *Software reengineering* is software evolution performed in a systematic way. In particular, Chikofsky and Cross define *reengineering* to be "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form" [6]. Altering existing systems comprises the majority of all software development time and expense, and evolution comprises the majority of system alteration (maintenance) activities [4].

The major difference between initial development and evolution is, of course, having to take into account the existing version of the system. Among the important concerns are making sure that the new requirements are consistent with those of the existing version, trying to maintain control of the architecture of the system, understanding the code of the current version, and suggesting how the enhancement might be made while maintaining the conceptual integrity of the design. MORALE is a software development method specifically designed to address these issues.

2. MORALE

The goal of the MORALE project [1] is to facilitate the evolution of legacy software systems. Facilitation takes the form of improved quality by requirements validation, reduced risk via architectural evaluation and assessment, and increased productivity from maintenance and access to design rationale and from high-level reuse of architectural components.

MORALE is an acronym standing for *Mission Oriented Architectural Legacy Evolution*. MORALE assumes that evolution is a process that takes as inputs an existing system, a set of high-level (mission-oriented) change requirements, and, possibly, other traditional development documents such as requirements describing the existing version of the system, a description of the current architecture, test cases, and program documentation. Evolution produces as output a new version of the system, preferably including updates for the related documents. This high-level view of software evolution is depicted in the dataflow diagram in *Figure 1*.

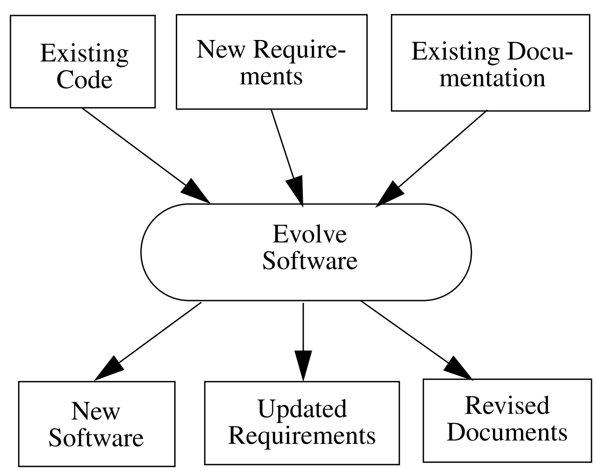

Figure 1: Software evolution process

To support the evolution process, MORALE provides a set of methods as illustrated in *Figure 2.* ScenIC is responsible for eliciting updated requirements; SAAM evaluates an architecture to determine the impact of the new requirements; Synchronized Refinement analyzes the existing software to extract architectural information; MESA suggests redesign strategies for functional enhancements; Architectural Reconciliation combines architectural views provided by the other methods, reconciling any inconsistencies; and MORPH is specifically aimed at extracting user-interface components in the situation where evolution of a system's user interface includes migration to a new graphical toolkit.

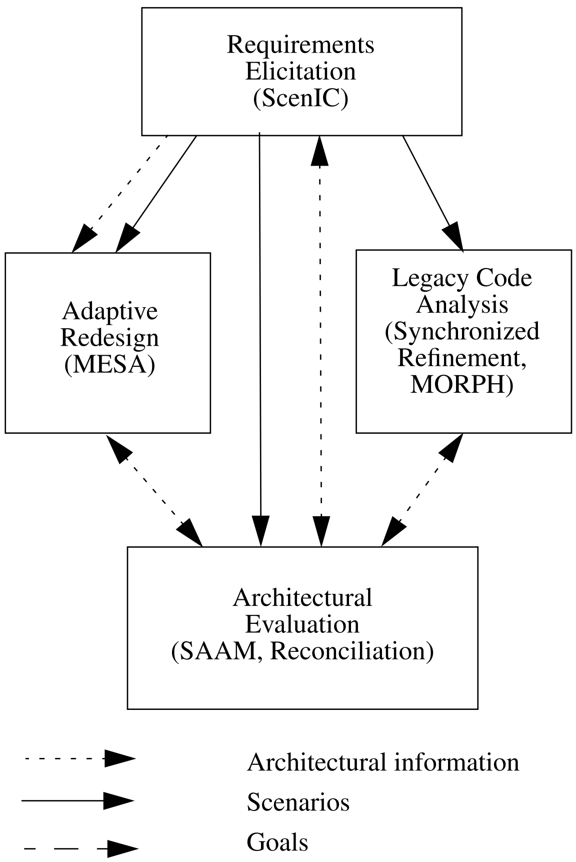

Figure 2: MORALE methods

The MORALE methods share three kinds of information. The most central of these are architectural descriptions—consisting of *components* that comprise the system and *connectors* that indicate their interactions. The second kind of information shared by the methods is a set of *scenarios* describing behavioral episodes. These may take the form of usage scenarios for ScenIC, high-level, internal architectural scenarios for SAAM, or typical test cases for Synchronized Refinement. The third form of information shared by the tools describes system goals. This information is used to suggest redesign strategies in MESA.

We now provide brief descriptions of the individual MORALE methods.

2.1 ScenIC

The MORALE evolution process begins with ScenIC, a requirements-elicitation technique. *ScenIC* is an acronym standing for *Scenario-based Inquiry Cycle* [17]. ScenIC supports the evolution of legacy software emphasizing the engineering of requirements for new systems or new versions of existing systems. ScenIC builds on a generic model of requirements and design activities known as the *Inquiry Cycle* [18]. The Inquiry Cycle is a process model and an associated prescriptive framework for acquiring, analyzing and refining requirements descriptions and design specifications for systems. ScenIC, as the acronym suggests, is a specialization of the Inquiry Cycle (IC) that emphasizes the invention of scenarios and the refinement of requirements and early designs based on an analysis of these scenarios. In addition to driving the requirements elicitation process, ScenIC produces as output a description of system goals, a set of usage scenarios, and an initial architectural description for use by the other MORALE methods.

2.2 SAAM

The Software Architectural Analysis Method, or SAAM [3][10][12], was originally developed at the Software Engineering Institute (SEI) to enable software developers to compare different proposed architectures based on how they would be impacted by current and future requirements of a system. The SAAM method revolves around group discussions by the various stakeholders in the system, including designers, customers, and users. SAAM comprises the following steps.

- Construct a description of the architecture of the existing version.
- Develop scenarios that illustrate the kinds of activities the evolved system must support and the kinds of quality concerns that the participants find important.
- For each scenario not supported by the existing version, list the changes to the architecture that are necessary for it to support the scenario and estimate the cost of performing the change.
- Summarize and prioritize the gathered information.

The results of a SAAM analysis can serve as input to both MESA and Synchronized Refinement. Moreover, as the system evolves, SAAM analysis can provide a jumpstart to the ScenIC elicitation of new requirements

2.3 Synchronized Refinement

Synchronized Refinement (SR) is a method for reverse engineering software for the purpose of documentation, reengineering, or reuse [7][19]. SR takes as input the source code of a software system and a description of the application domain that the system supports. The output consists of three parts: an elaborated and instantiated domain description, an abstracted program description, and a series of annotations detailing how the code realizes the application. The SR process consists of two parallel activities: analysis of the program and synthesis of the application description. Analysis proceeds by detecting design decisions and replacing the related code by an abbreviated version. In this way, the program description continues to become smaller and more abstract. Synthesis begins with the domain description which is elaborated as more is learned from the source code. Synchronized Refinement has been specialized for MORALE by concentrating on design decisions at the architectural level. To support architectural extraction, examination of the source code is driven by dynamic analysis using program executions of usage scenarios. In this way, use of particular features can be mapped to internal architectural components. The resulting architectural information can be used by SAAM and MESA to initiate their activities.

2.4 MESA

MESA is a method for the adaptive redesign of evolving software systems [16]. It is an acronym that stands for Model-based Evolution in Software Agents. The MESA method has two parts: (1) modeling a system's information-processing architecture and domain knowledge and (2) using the model for architectural and knowledge redesign. A software system is viewed as teleological artifact, i.e., as an artifact that has particular functions and specific behaviors that arise from the structure of the artifact and result in the accomplishment of its functions.

A system is modeled in MESA using the Task-Method-Knowledge (TMK) notation, which specifies both the functional architecture of the system in terms of its task-method structure, and its domain knowledge. The TMK model of a software artifact is organized as a hierarchy of functional and compositional abstractions and explicitly relates the behaviors of the artifact to the domain knowledge. The TMK model of a software artifact explicitly represents what can be potentially modified, which ranges from the tasks, methods, domain knowledge, and flow of data and control. The TMK notation for functional modeling provides a set of functional, behavioral and compositional abstractions of software artifacts. These abstractions can help guide the process of restructuring the legacy systems.

MESA takes as input a set of system goals (from ScenIC) and an initial functional architecture (from SAAM or SR). It produces a redesigned architecture suitable for use by the other MORALE methods or by the actual designers responsible for producing the new version of the system.

2.5 Architecture Reconciliation

Architectural Reconciliation [2] deals with the problem of integrating different architectural perspectives. An architectural *perspective* is a potentially incomplete representation of an architecture generated either by an automated tool or a human analyst. Reconciled architectural perspectives provide a more complete representation of a real system. Reconciliation uncovers inconsistencies pointing to areas needing further analysis or explanation. Architectural reconciliation consists of a cycle of activities that synthesize perspectives into a relevant set of views that together form a complete architectural representation. A *view* is a representation of an architecture used to describe a particular aspect of a system's structure or behavior and which provides coverage of the entire system. Reconciliation consists of the following four steps:

- **Generation**: Obtain the perspectives to be synthesized. These perspectives may come from existing documentation, source code analysis, domain analysis, or interviews with human experts.
- **Classification**: Group perspectives into their respective views. This helps an analyst to focus initially on reconciliation of perspectives that are intended to describe the same aspects of a system.
- **Union**: Analyze all perspectives that represent a specific view and develop a single view from them that incorporates structural information from all of them.
- **Fusion**: Analyze multiple views to check for commonality, consistency and to create compositions of different views.

The other MORALE methods are all capable of generating architectural perspectives. It is the responsibility of Architectural Reconciliation to check these for consistency and to encourage convergence of names and abstractions.

2.6 MORPH

MORPH is a process for reengineering the user interfaces of text-based legacy systems to graphical user interfaces (GUIs) [13][14][15]. MORPH begins by extracting the user interface from the computational legacy code, using program understanding techniques to build an abstraction, or model, of the existing interface. The legacy system model is then transformed to a model structured into support a GUI interface. Input from the human analyst is added at this stage, to define presentation details and to enhance the model. Once the model has been restructured, a forward-engineering tool can be used to automatically generate a new graphical interface for the system.

User interfaces are both essential and superficial. That is, systems always have them, but often the UI can be thought of independently from the systems' functionality. On one level, therefore, by being particularly concerned with the UI, MORPH can reduce effort by the other MORALE methods. Furthermore and on a deeper level, MORPH is concerned with isolating and extracting program components for which a well-defined application programming interface (API) exists. The techniques it uses can be applied to other well-defined and separable architectural layers, such as the code used to implement interprocess communication in a multiple-process program. Hence MORPH is one instance of a general method for supporting the evolution of reusable components.

3. The Esprit de Corp Suite of Evolution Tools

The Esprit de Corp Suite of software evolution tools has been developed to support the MORALE methods. Several of the tools (ScenIC View, ISVis, MORPH, SIRRINE, SAAMPad, and REMORA) provide direct support for an individual method. Others (VisEd and ACME Server) are intended to support the interoperation of all of the tools. The overall architecture of EDCS is given in *Figure 3.*

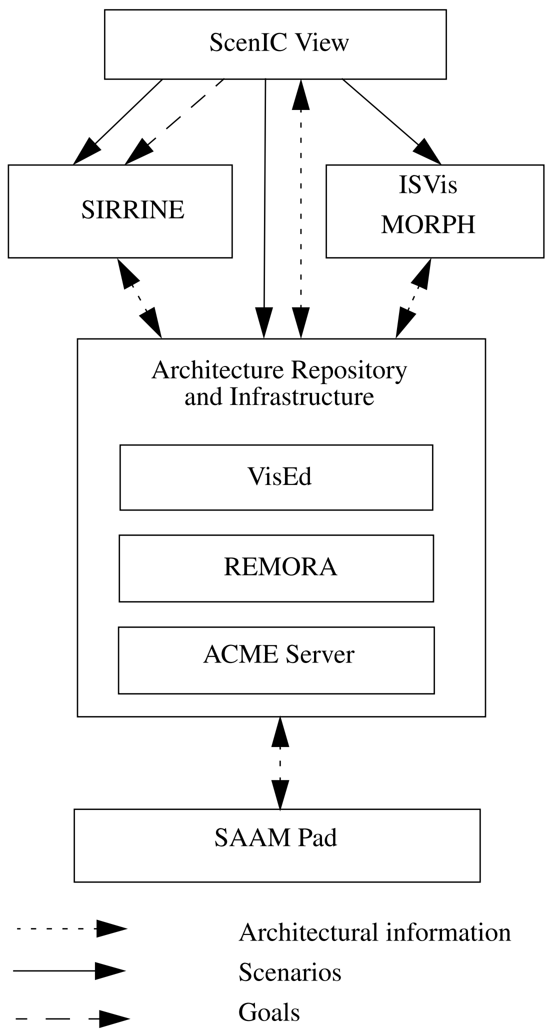

Figure 3: EDCS Architecture

Information to be communicated among the components is expressed in the ACME architectural description language (ADL) [9]. The communication takes place in real time supported by the ACME Server infrastructure software. We will first present the individual tools and then describe VisEd and ACME Server.

3.1 ScenICView

ScenIC View is an interactive scenario editor for use by software engineers evolving large-scale legacy software systems. ScenIC View supports the ScenIC requirements determination method.

ScenIC View is a way for software engineers to build ScenIC scenarios and manipulate them in a user-friendly and intuitive environment. ScenIC View allows software engineers to add scenarios, remove scenarios, and annotate scenarios.

Underlying ScenIC View is a data model and repository capable of managing information about scenarios, goals, and high-level architectural concepts. Included in the data model are an extensive set of relationships interconnecting these entities. For example, scenarios are composed of episodes, each of which has a point. The point of an episode illustrates the accomplishing or thwarting of a specific system goal. Episodes are expressed in terms of actors and the activities that they are capable of performing. Actors and activities, in turn, provide initial input to the creation of a system architecture.

3.2 ISVis

ISVis [11] is an architectural visualization and extraction tool for analyzing legacy software in order to acquire behavioral understanding, recover original design rationale, locate change sites, and validate design and implementation decisions.

ISVis consists of two primary views using which an software maintainer can analyze a software system. The main view allows the analyst to define components from combinations of files, classes and functions. This helps build higher-level abstractions to represent major elements of a software architecture. The scenario view allows the grouping of interactions into scenarios to provide higher-level behavioral abstractions of dynamic program execution.

ISVis has a wide range of features including:

- Analysis of program event traces numbering over 1,000,000 events.
- *Informational Mural* visualization techniques to portray global overviews of scenarios.
- Abstraction of actors and interactions through containment hierarchies and analyst-defined components and scenarios, respectively.
- Selective filtering of individual or multiple occurrences of a particular interaction.
- Use of patterns for locating the same or similar scenarios from interaction lists.
- Saving and restoring of analysis sessions.

3.3 MORPH

The MORPH toolset supports a process for reengineering the user interfaces of character-oriented interactive legacy systems to WIMP-style (Windows, Icons, Menus, and Pointers) graphical user interfaces. MORPH identifies basic user interaction tasks and associated attributes from legacy code by applying static program understanding techniques, including control and data flow analysis, and pattern matching. The resulting model can then be used to transform the abstractions in the model to a specific graphical widget toolkit.

MORPH supports three activities:

1. Detection—Analyzing source code to identify the user interface components in the legacy system.

2. Representation—Building a model of the existing user interface from the detection step.
3. Transformation—Manipulating, augmenting, and restructuring the resulting model to allow forward engineering to a graphical environment.

Input from the human analyst is supported in order to refine the model and make judgment decisions in the representation and transformation stages. Additionally, the transformation stage allows specific graphical implementations to be chosen and integrated into the legacy code.

3.4 SAAMPad

The purpose of SAAMPad is to provide automated capture support for SAAM. SAAM uses scenario evaluations to extract non-functional properties from a system. SAAMPad allows the creation of architectural descriptions required by SAAM, which can be evaluated. These evaluations are then captured. An evaluation generally consists of several scenarios, which can lead to changes in the architectural description. During the evaluations, audio and whiteboard notations are recorded to provide information during the evaluation and as an historical record.

SAAM is a useful way of extracting non-functional system properties. However, it relies on its users to record a lot of information. This has consequences for the way SAAM sessions are performed and limits both the amount and level of detail of the information that can be extracted. Users also need to create session summaries after scenarios have been performed. Together with the architectural diagrams, the summaries can be used to provide information about a scenario at a later point in time. However, this is often a hard task to do, because it is not known what information of the session will be needed at a certain point in time. SAAMPad provides a way to automatically capture SAAM sessions. This relieves the users and allows the scenarios to be more detailed. The capture tool also generates session summaries, which can point to potential problem areas. Components show a time line that indicates when they were created, focussed upon and altered. This information can be used to search for a particular part in the scenario that is of interest.

The system supports the use of "boxes and arrows" diagrams commonly used for architectural descriptions. The diagrams are drawn freehand by the user on an electronic whiteboard (or less desirably, a computer display) and then recognized by the system as components and connectors, after which an auto-layout feature is used to make cosmetic improvements.

The following features are currently available:

- Support for creating architectural descriptions. Diagrams can be drawn freehand and are then recognized by the capture tool.
- Automatic use of color. Color is used as a visualization technique both during and after the performing of a scenario.
- Support for scenario evaluations. It is possible to manage multiple versions of a single system and base a new scenario on one of them. During an evaluation new components can be added or existing ones can be marked as changed. These marks are later used in the access phase.
- Creation of session summaries. The capture tool can generate a summary of any scenario. It shows the affect the scenarios had on the system both in a diagram and in a summary table.
- Support for detailed scenario information retrieval. The user can search for a particular SAAM session, performed on an architecture, and SAAMPad shows the scenarios that were evaluated. Visualization techniques are used to show how often each component was the focus of attention during the scenario, and colors are used to indicate in what phase the component was created or annotated on. Thus, these techniques give an overview of the scenario. Furthermore, for each component, a time line is provided showing when in the scenario it was discussed and altered. Comparing time lines from different component can point the user to parts in the discussion that are of interest.

3.5 SIRRINE

SIRRINE is a tool that supports the MESA method. In particular, SIRRINE provides a graphical display of a TMK model of an evolving system. The display contains representations for system goals and the methods in the software available to effect them. In addition, SIRRINE takes as input a description of desirable new system behavior. The description is in the form of a supplemental goal tree. Users of SIRRINE can then "execute" the model. The tool steps through the TMK model indicating the subgoals and methods used to accomplish the new behavior. The analyst can then note the ways in which the existing system fails to accomplish the new goal. MESA can then suggest ways in which the TMK model can be reconfigured (manipulated and augmented) to adapt to the desired new behavior.

3.6 REMORA

The MORALE tool suite creates many different architectural perspectives of the legacy system being evolved. These perspectives exist at varying levels of abstraction and may model many different aspects of the same system. REMORA (Reconciler of MORALE Architectures) is a tool which allows an analyst to combine and manipulate these representations into a more complete and consistent representation of the architecture under evolution.

Each MORALE tool creates different types of information. The information which is architecturally significant is stored in an ACME description. An analyst wishing to use REMORA, interacts with the VisEd application for visualization and user-interface services. The actual computational and analysis activities make decisions about combining and manipulating the different perspectives created by the other MORALE tool suite applications.

REMORA also uses the services provided by other packages (both commercial and research) to do specific processing related to key areas. A Dowser [8] allows the analyst to automatically develop key domain concepts from various textual artifacts pertaining to the legacy system under evolution. A text analyzer provides word matching services such as determining domain synonyms and searching for root words within composite word forms. Finally, a concept analysis package takes a formal context and creates a concept lattice from it. The concept lattice is used to identify components and connectors provided by the various EDCS tools. These services all operate in a synergistic fashion to allow the analyst to semi-automatically perform architectural reconciliation activities while reducing the opportunity for error.

3.7 VisEd

VisEd is a graphical tool for displaying and manipulating architectural drawings. Drawings are made up of boxes denoting architectural components and arrows denoting connectors. Components may themselves have internal structure, and VisEd allows users to dynamically control the visible level of detail. Additionally, users can ask to see the attributes of the individual components and connectors, and a property sheet is displayed providing these details. VisEd is not just a passive display tool. Users can manipulate architectural drawings and the underlying properties. This includes improving layout, adding features, and editing properties. VisEd displays can be dynamically updated by other MORALE tools. For example, an analyst using ISVis can propose a high-level component made up of existing lower-level features. The resulting architecture can be exported and displayed by VisEd in real time.

3.8 ACME Server

All of the EDCS tools are concerned with descriptions of software architectures. To promote their interoperation, EDCS has made use of the ACME architectural description language [9]. The intent of ACME is to provide a common interchange format for architecture design tools. It consists of a small set of basic language constructs and an extension mechanism whereby individual tools can add properties to an architectural description that other tools can use.

In addition to the language definition, the authors of ACME have developed a library of useful functions (ACMELib) that tools can use to read and write ACME architectural descriptions.

We have developed a tool called ACME Server to augment these capabilities. ACME Server provides an interface between multiple running applications and an underlying ACME representation. The communications interface is provided via PBIO (portable binary IO streams) and Dataexchange. Dataexchange supports redirection and forwarding to allow multiple applications to access a single ACME representation.

That is, ACME Server can be thought of as a run-time repository for architectural descriptions. Multiple tools can access the descriptions simultaneously. Moreover, the tools can be running on different hardware platforms. ACME Server also has the ability to serve multiple architectural descriptions simultaneously.

4. Validation and Status

The EDCS tools are in various stages of maturity. Table 1 describes the current status of each of the tools.

Tool	Status
ScenIC View	Prototype
SAAM Pad	Prototype
ISVis	Public Release
MORPH	Prototype
SIRRINE	Public Release
REMORA	Being Designed
VisEd	Public Release
ACME Server	Public Release

Table 1: EDCS Tool Status

Most of the EDCS tools work on a variety of platforms. However, several of the tools have dependencies on commercial software. Details of the dependencies are provided on the tools' web pages[1].

We are in the process of evaluating the EDCS tools. For example, ISVis has been installed at several external sites, and we have received feedback on its portability and applicability. SAAMPad sessions have been run at an industrial site and videotaped. Analysis of these sessions is providing valuable feedback to direct future tool evolution.

5. Future Directions

A collection of individual methods does not necessarily serve as an overall evolution process. Likewise a set of tools need not work well together to support software maintainers. What provides cohesion to MORALE is a common vocabulary of concepts including architectural representations, scenarios, and goals. These sources of information are all directly supported by EDCS. Moreover, we have been able to use ACME and its support libraries to provide an interoperation infrastructure in which the tools can cooperate. This said, there are still areas that we would like to enhance.

Scenario Mark-Up Language: While ACME serves well to communicate architectural information, and, even though it is extensible, it is not ideally suited to communicate all forms of information. Consequently, we are developing SCML (Scenario Mark-Up Language) as a notation for communicating schematic scenarios. SCML is an XML variant. It is therefore straightforward to build parser front-ends so that other tools can access and manipulate MORALE scenarios.

1. The tools themselves may be obtain on the world wide web at URL `http://www.cc.gatech.edu/morale/tools`.

ACME and UML: Although the use of ACME has become widespread in the academic research community, the use of UML [5] is growing even faster. Even though UML was not designed as an architecture description language, its popularity dictates that EDCS should provide UML import-export capabilities.

Design Rationale: Design rationale has long been thought of as the silver bullet of effective design reuse. However, consensus has not been reached on exactly what it is and how it should be used. Currently, the MORALE method defines rationale as all of the information collected from the various constituent methods. Further clouding the picture is the need to carefully track version information in the evolving system under study. MORALE needs to address these issues to provide more comprehensive support for design evolution.

Acknowledgment

The work described in this paper was performed by the members of the MORALE project: Gregory Abowd, Ashok Goel, Mike McCracken, Bill Murdock, Colin Potts, Heather Richter, Spencer Rugaber, and Bob Waters of the College of Computing at Georgia Tech, Linda Wills of the School of Electrical and Computer Engineering at Georgia Tech, and Melody Moore of the Computer Information Systems Department of the College of Business Administration at Georgia State University. In addition, Dean Jerding, a former student at Georgia Tech developed the initial version of ISVis, Pascal Schuchhard, a visiting student from the Netherlands, constructed the initial prototype of SAAMPad, and Mary Lefferson a former student at Georgia State contributed to the development of MORPH.

Effort sponsored by the Defense Advanced Research Projects Agency, and the United States Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-2-0229. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon.

References

[1] Gregory Abowd, Ashok Goel, Dean F. Jerding, Michael McCracken, Melody Moore, J. William Murdock, Colin Potts, Spencer Rugaber and Linda Wills. “MORALE—Mission Oriented Architectural Legacy Evolution.” *Proceedings International Conference on Software Maintenance’97,* Bari, Italy, September 29-October 3, 1997, pp. 150-159.

[2] G. Abowd, S. Rugaber, and R. Waters. “Using the Architectural Synthesis Process to Analyze the ISVis System–A Case Study.” Georgia Tech Technical Report GIT-CC-98-22, August 1998.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 1998.

[4] Barry W. Boehm. *Software Engineering Economics.* Prentice Hall, 1981.

[5] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison Wesley, 1999.

[6] Elliot J. Chikofsky and James H. Cross II. “Reverse Engineering and Design Recovery: A Taxonomy.” *IEEE Software*, 1(7):13-17. January 1990.

[7] Richard Clayton, Spencer Rugaber, Lyman Taylor, and Linda Wills. “A Case Study of Domain-based Program Understanding.” *5th International Workshop on Program Comprehension*, Dearborn, Michigan, May 28-30, 1997.

[8] Richard Clayton, Spencer Rugaber, and Linda Wills. “Dowsing: A Tools Framework for Domain-Oriented Browsing of Software Artifacts.” *Automated Software Engineering 1998*, Honolulu, Hawaii.

[9] D. Garlan, R. T. Monroe, and D. Wile. “ACME: An Architecture Description Interchange Language.” *Proceedings of CASCON 97*, November 1997, pp. 169-183.

[10] R. Kazman, G. Abowd, L. Bass, and P. Clements. “Scenario-Based Analysis of Software Architecture.” *IEEE Software*, 13(6):47-56, 1996.

[11] D. Jerding and S. Rugaber. “Using Visualization for Architectural Localization and Extraction.” *Proceedings of the Fourth Working Conference on Reverse Engineering,* Amsterdam, the Netherlands, October 6-8, 1997, pp. 56-65.

[12] P. Kazman, L. Bass, G. Abowd, and S. M. Webb. “SAAM: A Method for Analyzing the Properties of Software Architectures.” *Proceedings of the International Conference on Software Engineering (ICSE 16)*, pp. 81-90, 1994.

[13] Melody Moore. “Rule-Based Detection for Reengineering User Interfaces.” *Proceedings of the Third Working Conference on Reverse Engineering,* Monterey, California, November 8-10, 1996, pp. 42-49.

[14] Melody Moore and Spencer Rugaber. “Domain Analysis for Transformational Reuse.” *Proceedings of the Fourth Working Conference on Reverse Engineering.* October 6-8, 1997.

[15] Melody Moore and Spencer Rugaber. “Using Knowledge Representation to Understand Interactive Systems.” *Proceedings of the Fifth International Workshop on Program Comprehension,* Amsterdam, the Netherlands, May 28-30, 1997, pp. 156-163.

[16] J. Murdock and A. Goel. “A Functional Modeling Architecture for Reflective Agents.” *Proceedings of the AAAI-98 Workshop on Functional Modeling and Teleological Reasoning*, Madison, Wisconsin, July, 1998.

[17] Colin Potts. “Using Schematic Scenarios to Understand User Needs.”

[18] Colin Potts, Kenji Takahashi, and Annie I. Anton. “Inquiry-Based Requirements Analysis.” *IEEE Software*, 11(2):21-32, March 1994.

[19] Spencer Rugaber, Stephen B. Ornburn, and Richard J. LeBlanc, Jr. ”Recognizing Design Decisions in Programs.” *IEEE Software,* 7(1):46-54, January 1990.