Mission-Oriented Legacy System Evolution Through Architectural Recovery and Evaluation

G. Abowd, A. Goel, M. McCracken, M. Moore, C. Potts, S. Rugaber, and L. Wills[1]
Georgia Institute of Technology, Atlanta, GA 30332

A. Introduction: MORALE

An important consideration in evolutionary design is to understand what drives the need for change. A common driver is a change in the purpose, or mission, of the system. The MORALE[2] project focuses on support for such *mission-oriented* evolution. There is an unexpected symmetry between requirements analysis, software architecture and reverse engineering that can be exploited to improve the process of mission-driven system evolution. Effective system evolution requires understanding of both the way that an existing system accomplishes its tasks and also the mission-oriented rationale for any changes that feed the evolution. Understanding the higher level structuring, or architecture, of an existing system aids in predicting the impact of change that is mandated by new mission-oriented requirements. We use requirements analysis techniques to suggest what concepts are most useful in understanding how an existing system works and how it should be evolved. We use reverse engineering techniques to extract high level architecture, both in terms of static and dynamic behavior, of legacy systems. The MORALE suite of tools and techniques will harness this symmetry by growing a common model of the architecture for multiple versions of a system or system family. The common model is a basis for assessing the effects of proposed changes and the extent to which legacy code can be reused.

The MORALE acronym summarizes our goals:

- **M**ission **OR**iented: We want the legacy system enhancement process to be driven by the mission to be accomplished rather than by purely technical criteria. Moreover, we want to ascertain the applicability of the old system to the new requirements, thereby avoiding unnecessary work.
- **A**rchitectural: The most time-consuming and costly alterations to software are those that distort architecture, by which we mean its structure and patterns of component interaction. We want to provide a mechanism for predicting the impact of architectural changes so that the risks of making those changes can be ascertained early in the evolution cycle.
- **L**egacy **E**volution: We are concerned with the evolution of legacy systems. The most expensive phase of the software development life cycle is maintenance/enhancement, and the most time consuming part of these activities is analyzing and understanding existing software. We want to provide a cost effective way of analyzing existing software, and once analyzed, extracting those parts of it which can be used in the new version.

To accomplish these goals, MORALE is integrating the following innovative technologies.

- A mission-directed requirements determination process that, through a systematic process of inquiry and refinement, turns mission-oriented goals into specifications of the desired behav-

1. Point of contact: linda.wills@ece.gatech.edu.

2. Effort sponsored by the Defense Advanced Research Projects Agency, and Rome 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. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

iors of architectural components. The MORALE requirements determination strategy is extending our previous work on Inquiry-Based Requirements Analysis (IBRA) [1][2][13] and the use of schematic scenarios [11] [15] to understand requirements.

- An architectural assessment process that can ascertain the impact of new requirements on an existing system's architecture. Our interest here is in the analysis of architectural descriptions in order to determine how well they satisfy a large class of quality criteria. We are extending our scenario-based evaluation technique, the Software Architecture Analysis Method (SAAM) [3][7][8], to perform this analysis.
- A reverse engineering process for extracting architectural information from an existing system. Beyond the traditional structural analyses provided by existing commercial and research tools, MORALE will extract behavior information derived from dynamic analysis of the message flow among architectural components. This work builds on our experience with cliche recognition (detection of common occurring patterns in code) [14][17], interleaving detection (determination of situations in which code implements more than one design goal or requirement) [16], and visualization of the dynamic behavior of software [6].
- An incremental adaptation process which supports the adaptation of a system after the analyses are made by the three processes above. Control techniques are needed for interleaving the processes of generating and evaluating architectural changes so that the evaluation guides the generation as well as the requirements analysis [4].
- A user interface evolution process to support UI migration. Using knowledge-based techniques, we are automating the process of reverse engineering user interfaces to detect, isolate, and extract user interface components and model them as abstract, technology- and language-independent forms [9]. This will significantly simplify and reduce the cost of user interface maintenance and migration.

B. The MORALE Process

The MORALE process integrates requirements analysis, architectural evaluation, and reverse engineering. The relationships between these activities and the artifacts relied upon and produced are depicted in the data flow diagram shown in Figure 1. The initial inputs to evolution are the existing system in the form of legacy source code and test data, a statement of the new requirements that the system must meet in the form of mission-oriented goals, and (desirably) an

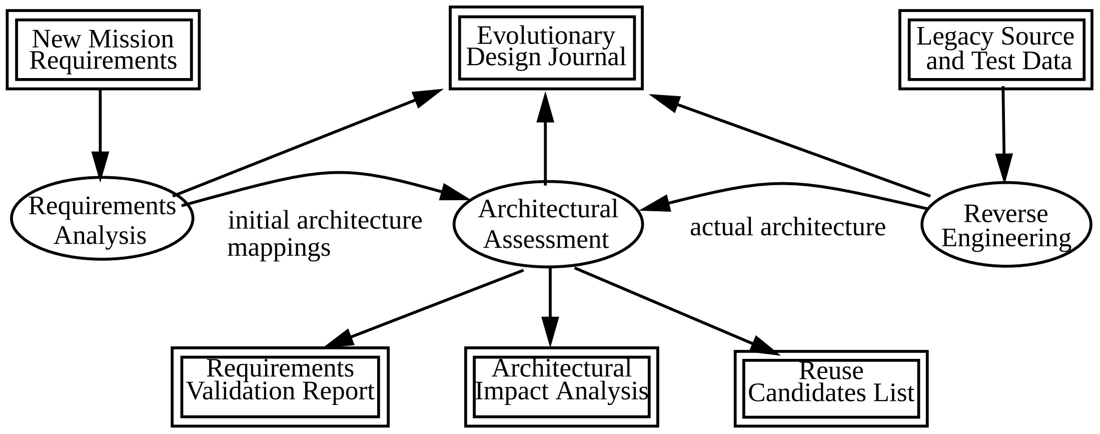
New Mission Requirements
Evolutionary Design Journal
Legacy Source and Test Data
Requirements Analysis
initial architecture mappings
Architectural Assessment
actual architecture
Reverse Engineering
Requirements Validation Report
Architectural Impact Analysis
Reuse Candidates List

Figure 1 MORALE inputs, activities and outputs

architectural representation of the existing system. Requirements analysis uses the mission-oriented statement to suggest concepts of importance in the overall structuring of the system; that is, it provides a suggested taxonomy for an initial architectural description of the system in the case where no such architecture exists. Requirements analysis also provides a collection of scenarios that represent a complete description of the new system requirements. These will serve as the basis for the architectural impact analysis. Architectural assessment includes the application of the SAAM methodology for architectural evaluation to predict the impact a set of scenarios will have on an architecture. Reverse engineering provides techniques for assuring that the architectural representation, the basis for the impact analysis, is an accurate reflection of the actual system under scrutiny. The three pieces work together to predict the extent to which the architecture of the old system can be adapted to meet the new requirements.

Besides this assessment, MORALE produces and uses several other artifacts in planning system evolution: a list of candidate code components from the old version that may be reused in the new version and the ability to enhance impact analysis by considering requirements on the old version of a system that must still be supported in the new system (a sort of regression suite for architectural evaluation). The entire MORALE suite provides an evolutionary design journal that incorporates items such as system goals and problems, traceability information between requirements and code, design alternatives and rationale. The result of the MORALE process is the development of instantiated architectural components together with a record of design decisions and the mission-oriented goals that were met or traded off to develop the new system.

C. The Role of Architecture Recovery and Assessment

Most systems built today do not have a well-documented software architecture. More likely, the architectural information that exists for these systems is in the form of glossy presentation slides with box-and-line diagrams that bear little resemblance to the code. Or the architecture may simply be a collection of organization principles that sits in the head of a few senior designers, many of whom may no longer be available.

Before any impact assessment of the architecture can proceed, MORALE will have to generate high-level models of the systems that are an accurate reflection of the actual systems. The MORALE method proceeds by iterative refinement that engages a designer in approximating the architecture of the existing system. An architectural approximation identifies active or passive components and the data and control relationships between those components. The initial approximation could come from the glossy presentation slides, or from goal-refinement techniques used to predict major domain concepts that will drive an architectural description.

Simple rules (such as suggesting how to map code units and files to the various components in the suggested architecture) are fed into the MORALE reverse engineering tools and used to compare the actual system structure to the input approximation. This activity extracts static structural and more dynamic behavioral information from source code using static and dynamic analysis techniques and visualization techniques. The extracted architecture is then compared to the input approximation. The feedback is in the form of a confidence measure, indicating the extent to which MORALE could map all of the source into the various suggested components and whether the suggested data and control relationships were detected. A high confidence measure from MORALE means that the actual system is structured as suggested by the architectural representation and further assessment can proceed. A low confidence measure can result for a number of reasons. For instance, MORALE may be unable to assign parts of the legacy code to any architectural component, or MORALE may uncover data and control relationships between architectural components that differ from those suggested. The feedback is used to suggest alternate structuring strategies for understanding the actual architecture, providing impetus for iterative discovery of the actual architecture.

When confidence is high that the architectural representation is an accurate reflection of the actual system, then that architecture can be evaluated to determine the extent to which it will support the new mission-oriented requirements. Our belief at present is that there are no simple (scalar) absolute measurements for evaluating abstract quality attributes, with the possible exception of metrics for maintainability. In the absence of absolute metrics, we suggest that there are only context-dependent measures that we can represent as *scenarios* of expected uses of a system, representable as natural language event flows similar to use-cases [5]. Scenarios are concrete, so they are appropriate gauges for evaluating a system (similar to benchmarks) Another advantage of scenarios is that they implicitly express overlapping concerns between abstract software qualities. Often what is most important in evaluating a system would be to show for example that the system was scalable while remaining portable.

The analyst using MORALE breaks down high-level mission-oriented requirements into structured scenarios. The MORALE architecture assessment tool then uses these scenarios to evaluate the fitness of the suggested architecture (that is, whether the architecture or components *directly* support the scenario or could do so *indirectly* after stated modifications). The MORALE architecture tools simulate the effect of direct scenarios on the architecture, collecting coverage statistics to determine how much of the architecture is being exercised by the scenarios, and compile the results from indirect analyses to predict the cost of transforming the architecture to meet the new requirements.

D. Case Study

Much of our previous work in requirements analysis, architectural analysis, and reverse engineering has been carried out by conducting large scale case studies. We have articulated our approach to conducting research as one in which Industry (or Government) is our laboratory[12]. An important tenet of this approach is that the case study is a first-class research vehicle, not merely a sanity check or validation exercise. Our experience shows that the conceptual complexity of evolving software systems cannot be reproduced in toy examples and that the feedback provided by early case studies is well worth the practical difficulties. For the short term, we have started a case study in which we explore several evolutionary episodes of the Mosaic World Wide Web browser [10]. Longer-term case studies are currently being planned in collaboration with industry and government partners, for example, in the areas of advanced distributed simulation and downloader/verifiers for embedded systems. We have placed a high priority on choosing an application area that has representative architectural problems and challenges.

E. Conclusion: Integrating Principles

The MORALE framework portrayed in Figure 1 shows several processes as if they were disjoint, only coming together through their influence on architecture models, but this is a simplification. Two integrating principles pervade the processes but are not revealed explicitly in the figure:

- Design decision support. The MORALE approach to decision support extends IBRA and Synchronized Refinement and integrates the representation of decisions with the artifacts (models, scenarios, requirements or code fragments) that they affect. A common hypertextual annotation mechanism supports both decision making and finding the rationale for earlier decisions.
- Scenario analysis. Scenarios play a major role in architectural evaluation and in allocating

behavior to architectural components. In MORALE, we manage scenarios at varying levels of abstraction (from mission-oriented narratives to architectural event traces). Scenarios are an important adjunct to the architectural model and are managed accordingly.

The tight integration of requirements analysis, architectural assessment and reverse engineering is the major contribution of MORALE. Ultimately, all of the research and tools developed in MORALE will contribute to a much richer account of evolutionary design than has previously been possible. We will be able to ground the evolution in a set of common goals that have pervaded the design, made concrete by scenario fragments. Recurrent issues in the design will be documented together with candidates for resolving them. Over time, and across different versions of the same system, canonical architectural patterns will emerge and we will be able to capture the rationale behind those patterns, documenting what requirements motivate the structure and behavior and how resilient the architecture is to changing requirements.

References

[1] Anton, A., McCracken, M. & Potts, C. (1994) "Goal Decomposition and Scenario Analysis in Business Process Engineering". *Proceedings CAiSE: Conference on Applied Information Systems Engineering*, Utrecht, Netherlands, Springer-Verlag.

[2] Anton, A. *Identifying and Analyzing Goals for Requirements Elaboration*, submitted for publication, 1995.

[3] Clements, P., Bass, L., Kazman, R., and Abowd, G. Predicting software quality by architecture-level evaluation. In *Proceedings of the Fifth International Conference on Software Quality*, Austin, Texas, October 1995.

[4] Goel, A. and Prabhakar, S. A Control Architecture for Model-Based Redesign Problem Solving. In Pr*oc. IJCAI-1991 Workshop on AI in Design*, Sydney, Australia, August 1991.

[5] Jacobson, I. *Object-Oriented Software Engineering: A Use Case Oriented Approach*, Addison Wesley, 1992.

[6] Jerding, D. and Stasko, J. Using visualization to foster object-oriented program understanding. Technical Report GIT-GVU-94-41, Graphics, Visualization and Usability Center, Georgia Institute of Technology, July 1994.

[7] Kazman, R., et al. "Scenario-based Analysis of Software Architecture" *IEEE Software*, to appear.

[8] Kazman, R., Bass, L., Abowd, G., Webb, S.M., "SAAM: A Method for Analyzing the Properties of Software Architectures", *Proceedings of ICSE 16*, Sorrento, Italy, May 1994, 81-90.

[9] Moore, M. "Rule-Based Detection for Reverse Engineering User Interfaces," *Proc. 3rd Working Conference on Reverse Engineering*, Monterey, CA., November, 1996.

[10] "Mosaic" home page. http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/.

[11] Potts, C. Using Schematic Scenarios to Understand User Requirements. *Proceedings DIS'95: Symposium on Designing Interactive Systems*, Ann Arbor, MI: August 23-25 1995, ACM.

[12] Potts, Colin, "Software Engineering Research Revisited." *IEEE Software*, September 1993, pp. 19-28.

[13] Potts C. "Supporting Software Design: Integrating Design Methods and Design Rationale", in T.P. Moran & J.M. Carroll (Eds.) *Design Rationale: Concepts, Techniques and Use*. Lawrence Erlbaum Associates (in press) 1995.

[14] Rich, C., and Wills, L. "Recognizing a Program's Design: a Graph-Parsing Approach," *IEEE Software*, Vol 7, No 1, Jan 1990.

[15] Rosson, M.B. & Carroll, J.M. *Integrating Scenario Evolution with Application Development*, IBM Research Report RC 19290 (82428), April 29, 1993.

[16] Rugaber, S. Stirewalt, K. and Wills, L. "Understanding Interleaved Code" *Automated Software Engineering*, Vol. 3, No. 1-2, pp. 47-76, June 1996.

[17] Wills, L. "Using Attributed Flow Graph Parsing to Recognize Cliches in Programs" in *Graph Grammars and Their Application to Computer Science* (edited by J. Cuny, H. Ehrig, G. Engles, G. Rozenberg), *Lecture Notes in Computer Science*, No. 1073, pp. 170-184, Springer, 1996.