PROPOSAL

BAA #95-40

Evolutionary Design of Complex Software
ARPA Program Manager: John Salasin
Chief Scientist: Howard Shrobe

MORALE

Mission ORiented Architectural Legacy Evolution

Core Effort Three-Year Cost:	$891,344
Option 1 Three-Year Cost:	$127,691
Option 2 Three-Year Cost:	$207,073
Total Three-Year Cost:	$1,226,108

Technical Point of Contact:
Dr. Spencer Rugaber
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
voice: (404) 894-8450
fax: (404) 894-9442
spencer@cc.gatech.edu

Administrative Point of Contact:
Chris D’Urbano
Office of Contract Administration
Georgia Institute of Technology
400 Tenth St. N.W.
Atlanta, GA 30332-0240
(404) 894-4817

Type of Business: Other Educational

A. Innovative Claims

The MORALE project addresses the problem of designing and evolving complex software systems. 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.

MORALE will accomplish these goals by integrating three 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 behaviors of architectural components. The MORALE requirements determination strategy will extend our previous work on Inquiry-Based Requirements Analysis (IBRA) and the use of schematic scenarios 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 will extend our scenario-based evaluation technique, the Software Architecture Analysis Method (SAAM), 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. The work builds on our experience with cliche recognition (detection of common occurring patterns in code) and interleaving detection (determination of situations in which code implements more than one design goal or requirement).

To accomplish these goals, the MORALE team consists of experts in the areas of requirements analysis, architectural assessment, and reverse engineering. Also, the team has long experience in working with both industrial and military customers and fully understands the appropriate integration of research ideas for effective technology transition.

In addition to the core effort described above, MORALE offers two options. The first supports the adaptation of a system after the analyses are made by the core processes. The second investigates and supports evolution in a particular domain: user interface migration.

OPTION 1: Incremental Adaptation

Incremental adaptation is a central issue in evolutionary design. We want control techniques for interleaving the processes of generating and evaluating architectural changes so that the evaluation guides the generation. We also want to feed the evaluation results back to requirements analysis.

OPTION 2: User Interface Evolution

It has been estimated that half or more of the code of an interactive system is devoted to the user interface, which is usually platform-specific. Using knowledge-based techniques, we will automate the process of reverse engineering user interfaces and representing them with models, enabling analysis and rapid generation of new user interfaces for legacy systems.

B. Deliverables

MORALE is primarily a Technology Investigation Project. However, we feel that it is essential to perform the investigation in the context of an actual multi-version software development effort with an identified customer. Not only does this validate our efforts and provide valuable feedback, it also facilitates technology transfer and offers the opportunity for the software project to make direct use of our results. We have chosen the distributed simulation domain for investigation, not only because of its importance to ARPA, but because of our familiarity with it and our discussions with SAIC, as described in Section D.

The research we propose will develop, demonstrate, evaluate, and transition new technologies for evolving software. The deliverables of this effort will include the following.

- An integrated software evolution process that includes mission-oriented requirements capture, legacy code analysis, and architectural change assessment.
- A prototype toolset to support the process including tools to ascertain the impact of the new mission on an existing system. The tool suite comprises tools for capturing new system requirements scenarios, analyzing existing system code, capturing design rationale, storing and providing access to the integrated information, and performing architectural assessment. The tools are both graphical and textual, and they support incremental development so that engineers can perform "what if" analyses on mission variants.
- An analytical case study of an existing multi-version system. The case study will estimate the extent to which the design and implementation were or might have been reused between versions, and perform an analysis of requirements both before and after the changes, to assess the impact of the requirements modifications on the actual design and implementation. The case study will contrast the architectures of two versions, understand how the differences were manifested in code alterations, and compare the respective requirements specifications to understand how effective they were in anticipating issues that might reflect on the ultimate system architecture.

MORALE's process and tools, in turn, will support evolutionary software development by explicitly producing four products. The most important is an *architectural impact assessment*, indicating the extent to which the current architecture can be reused and what, if any, major accommodations must be made in order to satisfy the new version's requirements. Second, MORALE will suggest existing system components for *potential reuse*. Third, it will help validate *requirements completeness* by assuring that existing features are preserved (if needed) and that new requirements map to the architecture. It does this by producing a description of the previous requirements for comparison with the new ones. Finally, MORALE produces an *evolutionary design journal* capturing insights unearthed during the various analyses performed. If maintained over multiple system versions, the journal will record design decisions, alternatives, mission goals, and traceability data from requirements to code. These products are further described in the Technical Rationale section of this proposal.

OPTION 1 - Incremental Adaptation

We will deliver a prototype generic "shell" for incrementalizing adaptation of software systems. The shell will embody knowledge-based inference and control techniques for supporting two important functionalities. First, the tool will support dynamic interleaving of the processes for the generation and evaluation of architectural changes to meet new task requirements. This will enable the two processes to inform each other so that the generation of proposals for new changes is guided by the evaluation of earlier proposals. Second, the tool will enable feedback of the evaluation results to the process of requirements analysis.

OPTION 2 - User Interface Evolution

We will develop and deliver a prototype toolset to support reengineering of legacy user interfaces. This toolset will support static and dynamic analysis for the detection of legacy user interface components and modeling of the resulting user interface and its relationship to the overall system architecture. We will also produce an abstract, implementation-independent representation for legacy system user interfaces and a process for refinement of the model. The tools will be integrated both into the MORALE toolset and into a user interface generation tool in order to support the forward engineering portion of the user interface migration process.

C. Statement of Work

To develop and support the MORALE approach, we propose to engage in three primary activities:

- Conceptual process refinement: We will develop and refine the evolutionary process underlying MORALE which integrates mission-oriented requirements capture, architecture recovery from legacy systems, and architectural change assessment.
- Tool suite development: we will develop a suite of tools to support this integrated process and validate it with actual users.
- Case study and evaluation: perform an analytical case study of an existing multi-version system. This will help drive the development of MORALE process and will serve as a source of empirical data for validating the MORALE process and tools.

Our strategy is to start with an initial synthesis of the existing technologies and techniques we have developed for requirements capture and architectural recovery and assessment, and to iteratively refine the conceptual process underlying MORALE. At the same time, we will develop tools to support and experiment with this process. Our conceptual work and tool development will be informed by our case study. We describe first the tasks involved in conducting our case study and then the tasks we plan for process and tool development.

Case Study: Driving MORALE Development and Evaluation

We will perform a case study of an existing evolving system. We will concentrate on two versions of the system: an original version and an evolved version resulting from changed requirements, specified by an accompanying requirements documents.

- CS1: Requirements Analysis: We will examine the requirements of the original version and the changed version, identify the goal structure and how it evolves, and derive scenarios for describing the desired use of the changed system.
- CS2: Architectural Recovery: Using our initial synthesis of MORALE tools and strategies we will conduct an architectural extraction of the original system.
- CS3: Assessment/Prediction: The extracted architecture and the results of our requirements analysis (e.g., scenarios and changed goal structures) will then be used to assess the impact of the requirements changes on the original version. This will result in predictions for what can be reused between the two versions and the amount of perturbation that will occur in making the changes.
- CS4: Evaluation/Comparison to actual evolution: We will then compare the predictions made to what actually happened in the existing changed version. We will contrast the architectures of the two versions to measure the perturbation that occurred and compare it to the amount predicted. We will estimate the extent to which reuse occurred in the ways predicted by MORALE as well as whether MORALE identified reuse opportunities that were missed when the change was made without the support of MORALE.

The results of this comparison will guide the refinement of MORALE's software evolution process and will drive the development and integration of tools to support it. It will also provide baseline empirical measures that will serve as a basis of comparison for judging the impact of using MORALE (e.g., in increasing productivity, shortening the change cycle time, and reducing the risk of making significant requirements changes) during the evaluation phase.

MORALE Process and Tool Development

In each of the three aspects of the MORALE approach (requirements capture, legacy code analysis, and architectural change assessment) there are specific tasks that we intend to perform as part of conceptual process refinement and tool development.

To specify new requirements, the MORALE user must operationally define goals and allocate responsibilities to system components. These goals are refined and responsibilities are reallocated as requirements evolve. To support this requirements capture process, we will do the following.

- RQ1: Requirements Elaboration Strategy and Tools: We will develop a strategy and support tools for converting a textual mission statement into a network of goals, refining those goals into operational requirements for a system (i.e. desired system behaviors and information processing), elaborating those behaviors to accommodate goal-thwarting obstacles, and mapping the operations onto abstract architectural components. Included in the strategy will be a vocabulary for expressing goals and architectural responsibilities and a modeling notation for expressing goal dependencies and obstacles.

- RQ2: Scenario Analysis Strategy and Tools: We will develop a strategy and support tools for deriving scenarios of proposed system use from the goal/obstacle model and the abstract architecture. We will develop metrics for evaluating the coverage and overlap among scenarios, and heuristics for identifying and deriving the most salient scenarios.
- RQ3: Requirements Decision Making and Rationale Model: We will develop a process model for incorporating decision making discussions, commitments and rationale into the requirements elaboration and scenario analysis strategies, and tools to support the recording, collation and presentation of the rationale for requirements and architectural allocation decisions.

Given a new set of requirements, the MORALE user needs to be able to assess the impact these new requirements will have on an existing system's architecture. The tasks we propose to do to effect this are the following.

- AE1: Criteria Identification: We will identify quality criteria that should be used to evaluate the architecture.
- AE2: Scenario Adaptation: We will adapt the scenario statements from RQ2 as concrete realizations of the quality criteria.
- AE3: Scenario Selection: We will support the identification of those scenarios that focus the evaluation on properties relevant to the changed requirements.
- AE4: Assessment/Visualization: We will develop tools to support the application of SAAM and to visualize the affect of indirect scenarios on the architecture.
- AE5: Impact Analysis: We will develop tools to use the results of SAAM scenario analysis to determine impact of requirement change on the existing system.

Evaluating the architecture of the existing system and determining what needs to change as requirements change and goals are refined requires extracting an accurate description of the system's architecture. We propose to use innovative reverse engineering techniques to extract architectural constraints from an existing system. We will use a hybrid of static and dynamic analysis techniques to iteratively refine an architectural description of the existing system (including both architectural elements and their behavioral interactions). In particular, we will do the following.

- RE1: Structural Architecture Extraction: We will develop reverse engineering tools for generating approximate structural architecture descriptions, based on static analysis.
- RE2: Dynamic Analysis/Event Trace Generation: We will develop a dynamic analysis tool which supports the instrumentation of an existing system so that event types relevant to a particular usage scenario can be monitored and traced.
- RE3: Behavioral Abstraction/Protocol Recognition: We will develop a language for expressing behavioral abstractions. We will provide an interaction protocol recognition technique which detects patterns of communication in event traces in order to derive abstract behavioral descriptions of element interactions.
- RE5: Visualization: We will develop a graphical tool to help visualize the results of the dynamic analysis and abstraction process. Through the visualization interface, the analyst provides queries, focus, and transformations to guide and refine the abstraction process.
- RE6: Model Refinement: We will develop tools to support the refinement of the architectural description, based on the results of the behavioral abstractions and the visualization.

OPTION 1: Incremental Adaptation

Incrementalizing changes to the architecture requires interleaving of the processes of generation and evaluation of changes. We will develop innovative inference techniques for evaluating proposed changes and control techniques for enabling the evaluation to guide the generation of architectural changes.

- OP1-1: Case Study: We will analyze whether and how the processes for generating and evaluating architectural changes were interleaved. We will also analyze whether and how the performance of the changed architecture led to further analysis of the task requirements. In addition, we will study whether the changes to the architecture were soft and situation specific or hard and general purpose.
- OP1-2: Reverse Engineering: We will combine the results of RE1 (structure abstraction) and RE3 (behavior abstraction) to form a structure-behavior-function (SBF) model of the architecture. We will use the SBF model to track proposed changes to the architecture and revise the model to reflect the changes. We will simulate the revised model for predicting whether a proposed change results in progress towards the satisfaction of the new task requirements.
- OP1-3: Architectural Change Generation and Evaluation: We will interleave the processes of generating and evaluating architectural changes so that information about the evaluation of specific changes guides the generation of subsequent changes. We will also feed this information back to requirements analysis so that the architectural evaluation may help further refine the requirements.

OPTION 2: User Interface Reengineering

Updating the user interface for an interactive legacy system can be a primary task of system evolution. We will develop techniques and automated support for migrating user interfaces, including:

- OP2-1: UI Component Detection: We will refine our existing technique for static and dynamic detection of user interface components within a legacy system.
- OP2-2: UI Model Representation: We will develop an implementation-independent representation for the abstract user interface models generated from the detection step.
- OP2-3: UI Reengineering Toolset: We will design and build a prototype toolset to support automated assistance for reengineering user interfaces including the following: a static analysis tool that creates an initial model of the interface in the legacy code; instrumentation and tools that support dynamic analysis of the interface to refine the static model; integration with a forward-engineering tool to produce a new user interface from the abstract model; and integration into the MORALE toolset.

Summary of Tasks

TASK (process/tools)	**DESCRIPTION**
Requirements Capture *(All RQ tasks)*	Scenario analysis, requirements elaboration
Architectural Assessment *(All AE tasks)*	Perturbation prediction, reuse candidate detection
Reverse Engineering *(All RE tasks)*	Architecture recovery, abstraction, visualization
Process Development *(All tasks, but especially CS)*	Synthesis/integration, rationale modeling, evolution design journal
Incremental Adaptation *(OP1 tasks)*	Generation of architectural changes, interleaved with the evaluation of changes
User Interface Reengineering *(OP2 tasks)*	User interface modeling and migration

D. Results and Technology Transfer

An important component of research in software engineering, particularly evolutionary systems, is in testing the technology in realistic environments. A major component of our research will be a case study involving a real legacy system with architectural requirements changes. From this empirical study, we will gain critical insights into the nature of the evolutionary process. As we develop our techniques and prototype tools to support evolution, we will continue to use real legacy systems as vehicles for research validation. We have placed a high priority on choosing an application area that has representative architectural problems and challenges.

Advanced Distributed Simulation

We plan to pursue Advanced Distributed Simulation (ADS) as a proving ground for our technology. ADS is an ideal testbed for this research because of its stringent requirements for rapidly evolvable, quickly integratable systems. ADS systems are often composed of many different simulations, including legacy systems and newly developed systems, requiring diverse, large-scale integration. The evolution of these systems is often under schedule duress, since being able to respond quickly to mission rehearsal requirements is paramount to the utility of the rehearsal. Since the benefit of simulation is derived from adapting battlefield rehearsal to a variety of conditions, the MORALE tools and techniques for architectural changes can make a significant impact on the amount of time required to set up a simulation. This enhances the training capabilities of the simulation and allows quick action in crisis situations. Since ADS software is constantly being adapted, evolvability is a prime software engineering goal, and since ADS is a prominent figure in ARPA's top ten technology priorities, it makes an ideal proving ground for the MORALE techniques and tools.

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[60]. 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. We plan to select a real ADS simulation system, through input from government and industry advisors or partners, that has evolutionary requirements. The new requirements for the system will be fed into the MORALE techniques and tools to drive the architectural analysis process.

Benefits

MORALE supports ARPA's "better/faster/cheaper" goals as they relate to legacy system evolution.

- Better: the MORALE evolution process produces better systems because the new version's requirements have been compared with those of the existing version. The resulting system has a better architecture because it reflects the new requirements and does not unreservedly retain the old architecture when doing so implies too great an adjustment. And the resulting system is better because it makes use of (presumably well-tested) components from the old system that have been identified as conforming to the new architecture.
- Faster: the MORALE evolution process is faster because of component reuse, because it detects potential problems early in the requirements stage where they require less effort to correct, and because the MORALE tools can automate a significant part of the software analyst's job.
- Cheaper: the MORALE evolution process reduce costs, both because faster development reduces costs and because architectural assessment reduces risk, thereby lowering life cycle development costs.

The requirements determination strategy has two specific benefits:

- Because the requirements determination strategy is based on a rational analysis of mission-oriented goals and the ability to trace detailed requirements back to mission-oriented concepts, users of the strategy can have increased confidence that the detailed requirements they produce (specifically, the specified behaviors of architectural components) will, if implemented, meet the customer's mission and that the risk of late, expensive, and defect-inducing changes will be lessened.

- Because the requirements strategy incorporates knowledge about the architecture of the current system in addition to customer needs, users of the strategy will be able to analyze mission goals for feasibility, rationally choose among alternative proposed mappings of these high-level requirements onto architectural component behaviors, and thereby reduce the risk of unforeseen architectural incompatibilities among the new and existing features.

The focus on architecture as a unifying link between requirements and reverse engineering has two benefits specifically relating to evolution of complex systems:

- The architectural recovery techniques in MORALE give the designer greater confidence that the architecture is a living high-level specification of the system. This increased confidence in the correspondence of architecture to artifact means that we can now track how a system drifts from its original intended structure and behavior.
- We can focus on how changed requirements impact the architecture itself. The structured analysis of an architectural representation based on scenarios of direct functionality and potential extensions informs a prediction of how changing requirements impacts the architecture. Since the architecture is a faithful representation of an existing system, we can extend our predictions on the representation to predictions about effort needed to modify the actual artifact. Increased prediction means decreased risk in evolutionary decisions.

The reverse engineering approach for the MORALE project will have several benefits:

- The most troublesome aspects of software maintenance and evolution relate to the implicit assumptions components make about each other. Our architectural extraction process, which discovers behavioral interaction patterns, will address this critical concern more directly than existing approaches.
- There is often a difference between the actual behavior and architecture of a system and that intended by its original developers or described by its documentation. Because our dynamic analysis approach is driven by the actual execution events, the fidelity of our results will exceed those generated by traditional static analyses.

We expect our research on incremental adaptation as described in Option 1 to result in the following benefits:

- The core elements of MORALE address the issues of why adapt and what to adapt in an evolutionary system. This option addresses the question of when to adapt. The addition of structure-behavior-function models will allow us to control the process of dynamically interleaving the generation and evaluation of architectural changes by using prior results of adaptation to guide anticipated changes.

The research in User Interface migration and reengineering of Option 2 will add several benefits to MORALE:

- User interfaces in legacy interactive systems can consume a large portion of the functionality of a system, which can result in costly migration to new interface techniques. The technology to detect, isolate, and extract user interface components, and model them as abstract, technology and language independent forms will significantly simplify and reduce the cost of user interface maintenance and migration.

Paths for Technology Transition

By virtue of the case study, a measure of Technology Transition is already built into the research process for MORALE. By using a realistic evolutionary scenario, the basis of the MORALE tools and techniques will address real-world problems. We will then additionally validate the resulting tools and techniques by further empirical studies in conjunction with government and industry partners (or "customers") in the ADS community. The joint case study would consist of choosing a candidate ADS system to evolve, under an aggressive schedule, perhaps as a parallel effort to a manual reengineering process for comparison. The first step in the evolutionary process is eliciting and refining the new requirements for the system, and then judging the impact of the new requirements and adjusting them according to the deadline for developing the simulation. To do this, we will elaborate the goals for the new simulation, refine these into alternative possible operational behaviors required of the architectural components of the system, and evaluate these mappings of mission-oriented requirements onto architectural components for feasibility. It may be the case, for example, that some of the proposed changes could be too extensive to be done before the required deadline, but that weakening some of the mission-oriented goals would lead to operational behaviors that could be implemented in time. The rationale and decisions made during the development process will be captured for later analysis. Finally, using the architectural information stored by the MORALE tools, we will investigate rapidly building the missing pieces of the simulations.

Partnering Possibilities

We have discussed technology insertion with several potential partners.

- **SAIC** - We have held extensive discussions with SAIC and have participated in planning for their "Virtual Consortium", which also includes MCC, ISI, Rational, and several other University partners. We have contributed to their vision statement, and agree with their views on the importance of ADS as a testbed. Their vision includes a common scenario established for a collective, collaborative research effort including industry, government, and academic participants. They have extensive expertise in the ADS technology area, and have offered their Semi-Automated Forces Laboratory (SAFLab) for independent evaluation and pre-established simulation scenarios. Their knowledge of common ADS simulation evolution requirements and access to real simulation systems would be a major asset to our empirical case study strategy. They have also expressed an interest in establishing common interface standards for interoperability with other research technologies, and a venue for integration at the SAFLab site. SAIC has provided us with an estimate of $100,000 per year to support their interaction with us. This amount is not included in our proposal cost estimate.
- **Motorola** - We have also presented our ideas to Motorola and are in the process of developing a working relationship. Motorola would be a strong industry partner that could provide many candidate legacy systems for experimentation and case studies. We have been collaborating with Motorola to study large-scale evolving systems issues for several years.
- **Gunter Air Force Base** - We have also made preliminary contact with the software development organization at Gunter AFB, which might be a potential customer, since one of their primary concern is in evolving DoD application programs.
- **Army Research Laboratory** (ARL) - The ARL has expressed interest in acting as coordinator and Contract Administrator for the work and to help us in finding a suitable DoD customer. We have enjoyed a long and productive relationship with ARL and have performed significant research under their guidance and support.

Review Board

In addition to our technology insertion partners, we propose a review board to assess and evaluate the MORALE research results. Included should be a commercial partner for productization concerns, DoD customers for using the tools and techniques, academic evaluators, defense contractors or consultants whose knowledge spans both the military and industrial worlds, and our Contract Administrator. Note that the costs associated with Review Board meetings have not been included in the budgets provided in this proposal.

Evaluation

We want MORALE to be of use to real developers working on real software systems. One way of promoting this is our initial case study that analyses an existing multi-version legacy system. Another way is to support actual customers from whom we can get feedback on the usability and value of our tools and the feasibility of our process. And the third way is to measure our progress via quantitative evaluation criteria.

We feel that four such criteria are relevant to MORALE—two to the MORALE process and two to the MORALE products. The former are *applicability* and *productivity*. Specifically, we intend that MORALE be applicable to a broad range of activities throughout the software life cycle. In particular, by targeting mission goals instead of technical requirements, we hope to introduce benefits earlier in the life cycle than other processes. And by including legacy system analysis, we can apply knowledge from later in the life cycle than comparable approaches. Applicability can be measured by the proportion of legacy system evolution activities it supports compared to other processes.

MORALE also supports improved life cycle productivity by enabling reuse and reducing risks, both of which can be measured using traditional techniques. In particular, reduced life-cycle effort is indicated by how much of the previous system's source code is retained in the new version, and the extent to which subsequent versions take advantage of the design journal. Furthermore, risk reduction will manifest itself in less wasted effort, ultimately reflected in early system delivery.

The MORALE tools make predictions and suggestions. The predictions and suggestion can be evaluated for their *accuracy* and *precision*. Accuracy denotes the extent to which an estimate is reflected in the delivered system. For example, if MORALE predicts that a certain proportion of the architecture can be reused then post facto measurement will either substantiate the prediction or not. Precision is concerned with the granularity of the predictions and suggestions. For example, it would not be of much value for MORALE to suggest that subsystem requirements for event logging need to be re-examined in the new version. But pointing out that the old version time stamped all events while the new system requirements make no mention of this fact, could be quite valuable. Precision can be measured relative to a particular process artifact, for example, a requirements document or a software component, the measure indicating how coarse the suggestion or prediction is to the ultimate item it addresses.

The following table relates the particular tasks of this project with the evaluation criteria we developed in the preceding paragraphs.

TASK (process/tools)	**DESCRIPTION**	**EVALUATION CRITERIA**
Requirements Capture	Scenario analysis, requirements elaboration	Applicability, Accuracy
Architectural Assessment	Perturbation prediction, reuse candidate detection	Accuracy, Precision, Productivity
Reverse Engineering	Architecture recovery, abstraction, visualization	Accuracy, Productivity
Process Development	Synthesis/integration, rationale modeling, evolution design journal	Productivity
Incremental Adaptation (option 1)	Generation of architectural changes, interleaved with evaluation of changes	Accuracy, Productivity
User Interface Reengineering (option 2)	User interface modeling and migration	Productivity

E. Cost, Schedule, and Milestones

Cost

The three year cost for the core project amounts to 891,344, divided roughly uniformly among the three years. This includes participation by five faculty members whose primary roles are the following.

- Abowd: architectural recovery and analysis
- McCracken: evaluation and technology transfer
- Potts: requirements analysis
- Rugaber: reverse engineering, architectural recovery and technical coordination
- Wills: reverse engineering

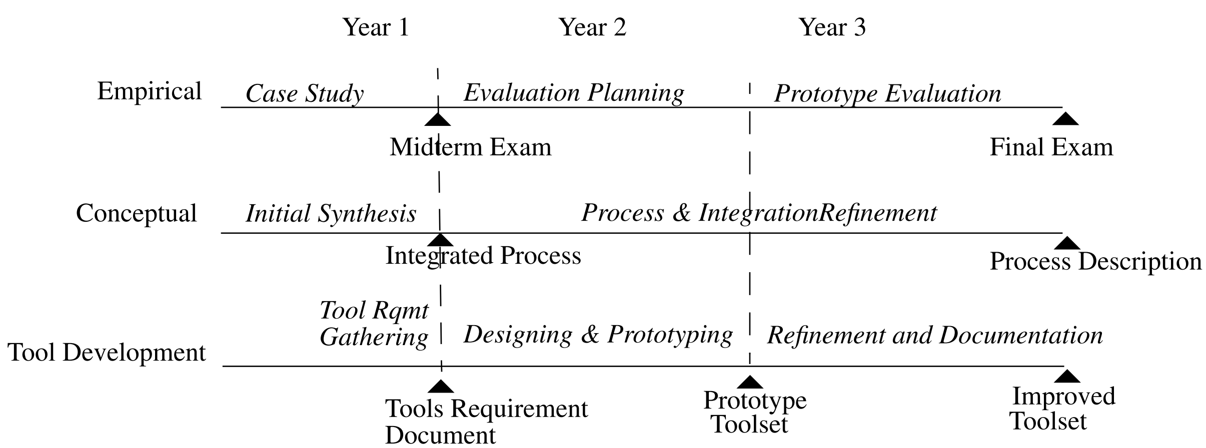
We have also included graduate student participation and significant travel money, particularly in support of customer contact and coordination with other ARPA groups.

OPTION 1 has a three year cost of $127,691, divided evenly among the three years, to support Dr. Goel and a graduate student.

Option 2 has a three year cost of $207,073, divided among the three years, to support Ms. Moore and a graduate student.

Schedule

Our schedule overlaps three main activities. The first year of the project is primarily concerned with performing a case study. This includes customer and system identification, requirements, architecture, and legacy system analyses, and evaluation of results. Tool design begins in the first year, but becomes the dominant effort during the second and third years. The third year is also heavily concerned with field trials of the tools with customers and tool tuning based on feedback received. The mid-term exam roughly coincides with the conclusion of the case study and will verify the process aspects of the project. The final exam corresponds to customer evaluations at the end of the project and to quantitative analyses based on the criteria described in Section D.



F. Technical Rationale

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. MORALE 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.

In this discussion of technical rationale, we will first motivate the research goals of MORALE with a concrete scenario that demonstrates the mission-oriented approach to evolution in the domain of distributed simulation. We will then outline the flow of activities in MORALE that link requirements analysis, architecture and reverse engineering before describing each of these three activities in more detail. We will then discuss two optional areas of research that represent natural extensions of MORALE's research activities — incremental adaptation of design and user interface migration.

F.1 A concrete scenario

We put forth our technical rationale for this proposal in the context of a likely mission scenario from the domain of distributed simulation. While the exact details of this scenario will probably not be the same for the actual research, we have carefully chosen a single scenario here that is in the expected domain of our research as well as being similar in complexity to a problem we will actually face. The scenario description begins with a mission-oriented statement followed by a description of how MORALE tools would fit in.

F.1.1 Mission-oriented description

An existing, integrated Advanced Distributed Simulation involves three distinct distributed simulations. Two of the simulations — a tank simulator and a sensor simulator for radar — are virtual, meaning they are realized entirely by software that simulates the behavior of some real-world entity (e.g., a tank or radar). The other simulation — representing the actions of a platoon of dismounted infantry — is live, meaning that at least part of the simulation is linked to an actual real-world entity (e.g., the soldier). The purpose of interfacing the virtual and live simulations is to feed real data for the positioning of live soldiers and troop movement to the virtual simulators to test sensor detection ranges.

A new technology has been added to the live simulation. Instead of relying on centralized data entry points reporting positions at the platoon level, each live soldier will now use a personal communication device to report information to the virtual world and to get information back from the virtual simulations. This means that now the individual soldier can be represented as an entity in the simulation rather than just a whole platoon. It also adds a requirement to support aggregation of the soldier entities to determine the location of a platoon. The simulation can also feed data back to the soldiers concerning position of enemy troops and tanks.

In order to incorporate the new requirements into the live simulation, the dismounted infantry simulation will require modifications to represent individual soldiers' positions and movements. The tank and sensor simulations will also require modification in order to handle the increased capabilities of the live simulation. The prototype sensors to be tested are scheduled to be deployed within one year. The task at hand is to determine whether the addition of the personal communication device in the hands of each soldier significantly affects the architecture of the existing simulations, and, if so, to assess the impact of the changes.

F.1.2 The MORALE tool suite

The MORALE suite of tools can be used to examine a set of existing systems to aid in the architectural evaluation in support of evolution. Like most systems built today, the simulators that will be the basis for the mission 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 bare 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 some high-level models of the simulators that are an accurate reflection of the actual systems. The MORALE method proceeds by iterative refinement that engages a designer in a series of approximations of 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 it could come from the MORALE Analyst tool using goal-refinement techniques 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. The analyst using MORALE breaks down high-level mission-oriented requirements into structured scenarios. The MORALE architecture evaluation 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.

There are other uses of the MORALE tool suite. As a system is modified, the MORALE Miner can highlight violations to the overall architectural structuring scheme. System drift from global structuring principles makes a system more and more difficult to comprehend. The previous paragraphs described how to use MORALE in situations in which the drift has gotten out of hand. But in a forward engineering situation, MORALE can also advise when system modifications violate the intended architecture of the system, increasing the likelihood that the architectural representation remains valid. Patterns of high-level structure, both within a single system and across system, can also be recognized by MORALE's Miner, suggesting places in which future design-level reuse would be appropriate.

F.1.3 The MORALE Process

As suggested by the three major parts of the MORALE tool suite (the Analyst, the Architect and the Miner), MORALE can naturally be broken up into three research activities, one each for 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

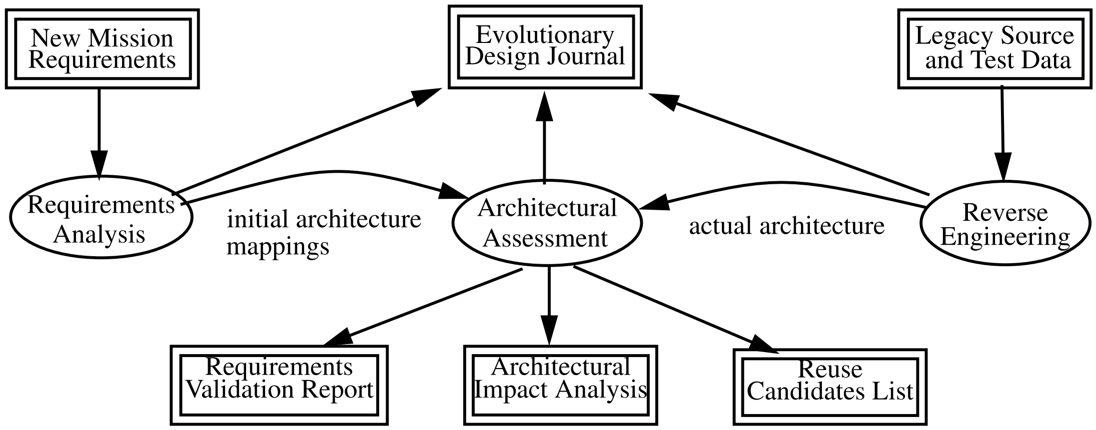


Figure 1 MORALE inputs, activities and outputs

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 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 a historical 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.

F.2 Mission-oriented requirements determination

Requirements for evolving systems involve an interaction among new mission needs and existing architectural constraints. We therefore view the process of requirements analysis as a combination of two processes: the elaboration of mission-oriented goals and the mapping of operational behaviors onto architectural components. A third activity helps direct these processes: the derivation and analysis of scenarios. Scenario analysis helps raise issues that must be addressed and therefore drives the elaboration of requirements and architectural mappings.

F.2.1 Mission-Oriented Goal Elaboration

Our strategy for elaborating requirements involves describing explicitly the mission in terms of goals and then refining the goals using several standard transformations. Transforming a set of mission-oriented goals into operational requirements involves the customer and developer in many decisions. High-level mission-oriented goals must be decomposed into more concrete sub-goals, and these must be allocated to abstract architectural components. The concrete goals must be transformed into goal-attaining and goal-preserving actions to be required of the system (or assumed of its environment). And at every step of the way, goal-thwarting obstacles must be identified, and secondary goals, allocations and required behaviors proposed to overcome them. Such obstacles may result from system component failures (and therefore can only be identified after early allocation decisions have been made), they may arise from operator behavior, or they may be undesirable environmental conditions.

We have developed a domain-independent classification scheme for goals and obstacles, which we propose to evaluate and extend [8, 9, 83]. There may be achievement, maintenance, avoidance, or improvement goals, depending on the type of condition or criterion against which they are judged. The point of classifying goals is to suggest elaboration heuristics for goal refinements and episodic fragments for scenario analysis. Obstacles differ according to their origin and the type of goal affected. Some are caused by actor failure (not necessarily equipment breakdown, but also operator unavailability) and therefore depend on allocation of responsibility to architectural components; others result from object confusions or contention (such as inability to discriminate among target signals); still others stem from goal interference (when two normally desirable goals interfere with each other and must be prioritized). The point of classifying obstacles is to suggest possible prevention or mitigation strategies in the architectural allocation (such as recovery, retry or check-before-execute) operations.

F.2.2 Mapping Requirements onto Architectural Components

The output of our requirements elaboration strategy is a specification of operations to be performed by abstract architectural components. Components are assumed to be autonomous, asynchronously interacting automated subsystems of the proposed system. They may not correspond precisely to implementation components, especially in real-time systems, because design optimizations may be necessary to break the modularity of the abstract components and interleave their implementations. However, we assume that the abstract model is a good approximation of the architecture. Our current work goes further and addresses the mapping of operationalized goals onto OOA (object-oriented analysis) classes by decomposing the identified operations further and attributing the resulting operations to object classes.

We pay particular attention to the location of information and intrinsically distributed aspects of the problem, because these are almost certain to lead to architectural separation of components. For example, in the simulation mission scenario, the platoon members' personal communication devices, the sensors, and the tank simulator are clearly separate components. For identifying collocated components, we are using ideas from the architectural styles and design patterns literature [20, 25, 26, 68]. For example, within the simulator, we may apply a model-view-controller style to suggest separate engagement model, simulated presentation, and operator manipulation interface components.

F.2.3 Mission-Oriented Scenario Analysis

A scenario is a description of a concrete use situation or sequence of events. Scenarios help in elucidating the exact operationalization of goals and mapping onto architectural components and in evaluating how satisfactory will be the proposed behavior.

Scenarios may be analyzed at varying levels of abstraction and formality. We consider three: mission-oriented narratives, goal-oriented episodes, and architecture-oriented event traces. The first has a purely textual representation; the other two are structured and tabular. Our aim is to work exclusively at the goal-oriented or architectural levels and, where appropriate, translate the resulting scenarios back to mission-oriented text for customer consumption. The translation is a manual process, but straightforward, given the traceability that exists from architectural responsibilities back through goals to mission statements.

Our current work is concentrating on integrating the use of goal-oriented episodic scenarios for requirements elaboration with the architecture-oriented scenario analysis of SAAM (see Section F.3.1), and the refinement of heuristics for identifying salient (or "right") scenarios using the goal model as a starting point. To identify salient scenarios, we are building on the work of cognitive scientists who have studied the structure of easily memorable and culturally salient stories. By analogy with story schemata, we have developed a scenario schema that we use as a template for deriving scenarios [63]. The resulting scenarios consist of common goal-seeking fragments, or episodes, plugged together in salient combinations. The episodes themselves may be represented as lower-level event traces involving interactions among the architectural components.

To prune the set of possible scenarios, which is unbounded, we have developed a preliminary set of domain-independent heuristics for identifying scenarios. These suggest which combinations of actors, goals and obstacles should be the “point” of the scenarios and are therefore likely to lead to useful discussions of the requirements and architectural mappings. Our heuristics especially emphasize obstacles and obstacle combinations, because these can easily be glossed over by conventional review practices.

We are also proposing to develop a scheme and automated support for the management of multiple scenarios that refer to alternative requirements elaborations and architecture mappings. In our experience, many scenarios contain common episodes, and we plan to exploit this commonality by using a multi-level versioning scheme. A change of an architecture mapping in one scenario will therefore be reflected by similar changes in others.

F.2.4 Working Rationale and Decision Support

?Our Inquiry-Based Requirements Analysis (IBRA) strategy considers requirements elaboration (and design) to consist of three concurrent and semantically interlinked activities: expression, refinement, and discussion [59, 64]. Expression is the description of requirements-related information in artifacts, including mission statements, goal models, scenarios and architecture models. Refinements are the decompositions, mappings and revisions that mediate between these artifacts or between versions of them. Discussion refers to the raising and resolution of issues about the artifacts, and it is discussion that drives the choice among alternative refinements. We are currently working on an instantiation of IBRAfor goal-oriented requirements elaboration [62, 65], depicted in Figure 2.

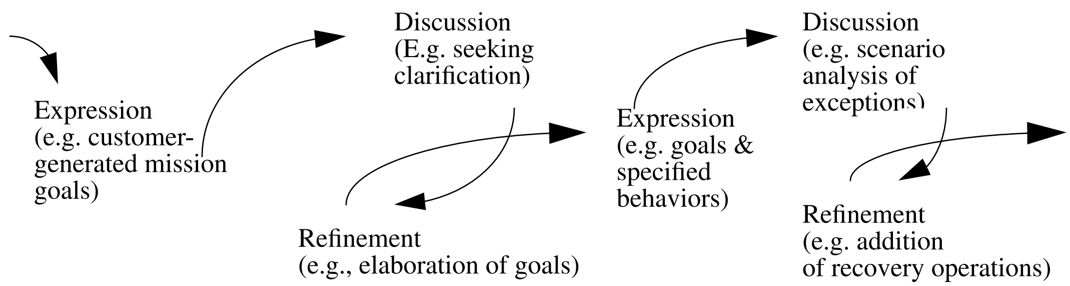


Figure 2 Inquiry-Based Requirements Analysis

In our process model, it is the discussion activity that embodies decision making and the rationale for decisions. For example, a discussion of a specific goal (which may, in turn appeal to a scenario as evidence) may lead to a decision to refine the goal in a particular way rather than another or to add a secondary obstacle-mitigating goal. When we ask later why the requirements include this obstacle avoiding behavior (for example, the specification of recovery operations), it is the discussion and the scenario that provide its rationale.

F.3 Architectural Evaluation

The MORALE approach to architectural representation and evaluation will extend the SAAM method [4, 15, 42, 43]. SAAM helps to focus attention on the critical details of the architecture, by which we mean those architectural details that most contribute or fail to contribute to specific mission-oriented requirements. We have previously applied SAAM to several software architectures, including several large industrial systems, in a variety of application domains. The adequacy of an architectural model depends on how well it represents the structure, behavior and functions of the system and reveals to the designer what components or interconnections will need to be changed to accommodate new requirements. In MORALE, we will first construct a representation of the architecture and then analyze the degree to which the architecture supports a set of defined scenarios. In doing so, we emphasize two factors: architectural changes that would be necessary to accommodate mission-oriented functions that are not yet fully supported and scenarios that interact and which therefore rely on common architectural components.

For many years, successful designers have known that it pays to be concerned with how the software in large systems is structured and coordinated. The academic discipline of software engineering does not provide any predictive theories behind the importance of structure and coordination at this gross level of organization of a software system —what is called the *software architecture*. The software architecture is a description of the high-level organization of computational elements and their important interactions. This aspect of MORALE is aimed at developing predictive techniques to analyze software architecture to support the specific needs of evolutionary design.

An architecture is often the first artifact in a design that represents decisions on how requirements of all types are to be achieved. As the manifestation of early design decisions, the architecture represents those design decisions that are hardest to change and hence are deserving of the most careful consideration. Analyzing an architectures in an evolutionary context leads to the most predictable and cost-effective modifications to the system.

In collaboration with a team of researchers at the Software Engineering Institute and the University of Waterloo, we have defined a scenario-based evaluation technique for software architecture, called the Software Architecture Analysis Method (or SAAM). The motivation for SAAM is to make it possible to predict certain qualities about a system by studying its architecture. The motivation for SAAM was the recognition that designers frequently make written claims about how the overall organization of their product leads to their system having desirable abstract properties (e.g., such as maintainability, portability, scalability, security) but do not have the means to validate such claims.

There were two problems in architectural evaluation that SAAM is intended to solve. First, there is no generally accepted architecture description language (ADL), so evaluation of competing architectures results in an apples-to-oranges comparison. We therefore defined SAAM independently of any ADL.

The second problem is that abstract quality attributes of software (maintainability, portability, etc.) are too abstract to form the basis of a meaningful evaluation. Suppose we can change the colors in a user interface by changing a resource file which is read in at run-time, but changing the fonts used in the interface requires a re-compilation. Is this system modifiable or not? The answer is, perhaps; yes with respect to changing colors, but no with respect to changing fonts. And whether the system is acceptable or not depends on whether the user interface is modifiable in a way that is important to its owner. If so, we can say that the system is *appropriately* modifiable. This notion of appropriateness applies to all quality attributes.

Our belief at present is that there are no simple (scalar) absolute measurements for 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 [46]. 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. An evaluation technique that focused on evaluating a single abstract quality at a time would miss this important overlap. Since we are not concerned at the outset with determining the abstract qualities of interest with scenarios, we do not fall into this trap.

We cannot provide a full explanation of the scenario-based technique of SAAM in this proposal. There are several published accounts of SAAM evaluations which can provide details on the method [4, 22, 56, 57]. The overall method can be described in five steps. Figure 3 shows the steps of SAAM and the dependency relationships between those stages.

- **Describe candidate architecture.** The candidate architecture or architectures should be described in a syntactic architectural notation that is well-understood by the parties involved in the analysis. It is therefore reliant on some ADL, but independent of any existing ADL. These architectural descriptions need to indicate the system's computation and data components, as well as all component relationships, sometimes called connectors. Accompanying this static representation of the architecture is a description of how the system behaves over time, or a more dynamic representation of the architecture.

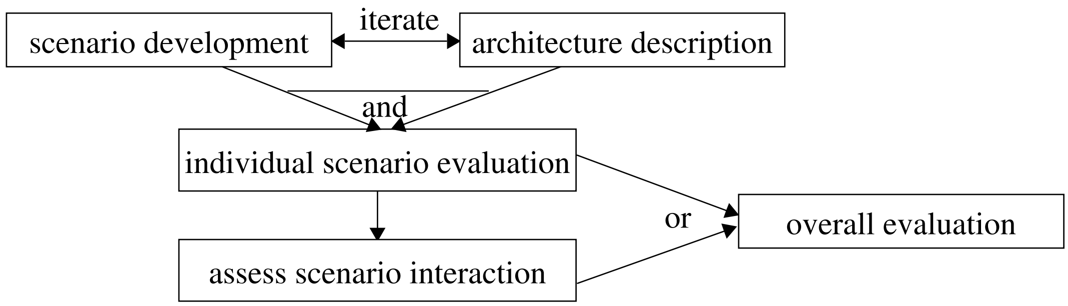


Figure 3 Activities in SAAM

- **Develop scenarios.** Develop scenarios that illustrate the kinds of activities the system must support (*direct* scenarios) and the kinds of changes which it is anticipated will be made to the system over time (*indirect* scenarios). Scenarios will represent tasks relevant to different stakeholders such as: end user/customer, marketing, system administrator, maintainer, and developer. Direct scenarios are useful in helping to understand and describe the dynamic behavior of a system, so we can see a relationship between steps 1 and 2, as indicated in Figure 3.
- **Perform scenario evaluations.** For each indirect task scenario, list the changes to the architecture that would be required in order to support the scenario. If possible, provide an estimate of the difficulty of performing such a change. A modification to the architecture means that either a new component or connection is introduced or an existing component or connection requires a change in its specification. By the end of this stage, there should be a summary table which lists all scenarios (direct and indirect). For each indirect scenario the impact, or set of changes, that scenario has on the architecture should be described.
- **Reveal scenario interaction.** Different indirect scenarios may necessitate changes to the same components or connections. In such a case we say that the scenarios *interact* in that component or connector. If two or more scenarios interact, then either those scenarios are specializations of a more general scenario or the architecture puts responsibility for more than one feature of a system in one place. In the first case, the concrete scenarios can be generalized towards more abstract quality considerations. In the second case, the scenario interaction indicates the need for further separation of concern at the architectural level.
- **Overall evaluation.** Finally, weight each scenario and the scenario interactions in terms of their relative importance and use the weighting to determine an overall ranking. This is a subjective process, involving all of the stakeholders in the system. The weighting chosen will reflect the relative importance of the quality factors that the scenarios manifest.

F.3.1 Extending SAAM

We have gained experience applying SAAM in eight case studies, both in industry and academia [4, 22, 56, 57]. There is now sufficient evidence to show that a structured method for analyzing a software architecture is an immediate practical contribution to software design. We have recently conducted a workshop which compared SAAM to existing architectural review practices in industry [53]. The conclusions from that workshop show that SAAM is the most structured technique for evaluating an architecture against a stated set of concrete measures, but that there is still a leap of faith between the stated architecture of a system and its actual high level structure as determined from source code and execution. Before addressing this concern for architectural confidence, there are several other weaknesses in the method that are short-term goals for this research.

Effective documentation of the SAAM evaluation requires tools to describe architectures and accumulate the results of scenario evaluations. Colleagues at the University of Waterloo, have developed a simple tool to construct architectural descriptions and tabulate scenario impact evaluations for SAAM. This allows us to quickly summarize the results of scenario evaluations for a large number of scenarios without any extra administrative effort. This tool will be extended and included within the MORALE suite to:

- support any of a number of architecture description languages;
- incorporate other commercial tools for describing dynamic behavior in terms of event flows for scenarios;

- use visualization techniques to summarize scenario interaction; and
- support regression testing of scenarios when an architecture evolves over time.

The main benefit of SAAM in its current form is people-oriented. We have defined a method that forces discussion of architectural decisions, resulting in consensus and concrete documentation of quality evaluations. There are, however, several technical hurdles that must be overcome in order for SAAM to support evolutionary design. In the evolutionary context, there is an existing system that requires change. For example, in our distributed simulation scenario, three actual systems already exist (two virtual simulations and one live simulation) and we are required to modify all of them to accommodate a higher granularity of positioning information from infantry carrying personal communications devices linked to some position detection system (such as GPS). In principle, SAAM can be used to understand how an existing architecture will be impacted by such a change inrequirements. But conclusions based on an architectural representation depend on the correctness of that representation with respect to the actual system. SAAM lacks any measure of confidence in the correctness of the architectural representation beyond input from domain experts.

The conclusion we draw from these and other experiences is that we need to provide ways to automatically recover the architecture of an existing software artifact, therefore increasing the confidence that the architectural description is a faithful representation of the system under scrutiny. This is particularly relevant in the case of evolutionary development in which there is an existing system which probably does not have a well-documented architecture and a design team needs to decide how to alter the system to accommodate new requirements without sacrificing any of the old and still relevant requirements.

F.4 Architecture recovery

There are many unexploited resources that an existing version of a software system can provide to the developers of a new version.

- There is the source code itself. The difficulty here is deciding exactly which pieces are directly usable, which can be readily adapted, and which are either no longer needed or not worth the trouble. Less obvious is that the existing system can provide requirements for the new version. The new version is expected to reproduce most of the old version's behavior, adding new features or modifying parts of the old version. It may also be the case that the existing version is in fact the only accurate source of information about itself, other sources of documentation being out of date or unfaithful. Hence, the old version can serve to augment and validate the requirement provided for the new version.
- Old versions of a system can also provide developers data about what worked and what didn't, particularly concerning architectural design decisions. If a particular aspect of the architecture has over time been proven brittle as evidenced by a history of structural alterations, then there is evidence that that architectural approach is not flexible enough to adapt to the kind of changes that this product family confronts.
- Old versions can also act as laboratories, providing a test environment for developers to learn more about how a system intended to solve a particular class of problems actually works. This includes information about what components are exercised and in what combination when confronted with particular classes of requests from users.
- Finally, in some cases other artifacts such as regression tests, source code control histories, and design documentation can provide insight into the relationship between original intent and actual use.

These resources are typically not very well exploited when constructing a new version of a system, and it is our intent to remedy this situation. Doing so requires understanding the existing version in ways that go beyond current reverse engineering technology.

We imagine a process like the following:

- An analyst is confronted with the task of preparing a plan for evolving an existing version of a system to satisfy new mission requirements. The analyst has a draft of the new requirements, access to the source code of the existing system, and possibly unreliable documentation, test sets, etc.
- Using existing commercial technology, the analyst produces an initial, rough approximation to the architecture of the current system. (This approximation will be refined in subsequent iterative phases of this process.) The architectural approximation describes a collection of architectural elements, each capable of

providing some functional service through an interface. The elements are organized in such a way that interface compatibility is enforced. Note that at this stage, the elements are identified with system functions and not with source code modules.

- The new requirements are expressed in terms of usage scenarios. The scenarios describe how the architectural elements (of the new version) interact to accomplish its mission.
- Using an appropriate scenario, significant event types are identified and the source code instrumented. Test data is obtained and the code executed. Raw event trace data is obtained.
- Using a visualization tool, the analyst abstracts over the event trace to obtain a model of significant element interactions expressed as path expressions/regular expressions/state machines. The visualization tool provides an interface to abstraction mechanisms which recognize behavioral patterns of communication (protocols) in the event trace. Through the visualization interface, the analyst provides queries, focus, and transformations to guide the abstraction mechanisms.
- The abstract results are fed back to the architectural analyst who refines and augments the architectural description with new architectural elements (or aggregations of existing ones) and with behavioral descriptions of element interactions. The process is iterated to increase the accuracy of the description.

This process results in several useful pieces of information. First, there is a description of the actual architecture of the existing version as it responds in a set of situations known to be relevant to the new version. This description includes both architectural elements and behavioral interactions (communication protocols). We can determine the differences between the architecture of the existing system and the proposed architecture of the new system. The differences can then be used to judge the amount of effort required to evolve the old architecture to its new form. Likewise, the differences can be used to suggest software components suitable for reuse.

The analysis process also has several other beneficial side effects. If during dynamic analysis, parts of the existing system are not exercised, then a discrepancy report can be generated. One kind of discrepancy is indicative of existing system behaviors that are not represented by any new requirements. Another indicates parts of the existing system that need not be retained in the new version. Of course, system architectures are often complex. Part of the complexity may be due to the inherent nature of the problem being solved, but part of it may be caused by structural deterioration over time. The analysis described above can indicate software components that should be examined for possible restructuring, either because they are contributing to accomplishing unrelated functions (interleaving) or because their interactions with the rest of the system violate expectations.

The approach described above makes use of existing program understanding technology, including the following:

- Ability to produce an initial approximation of system architectural elements and their interactions (from commercial tools);
- Ability to visualize event data [40];
- Ability to infer abstractions [20];

Our contributions extend this work in the following ways:

- Use of dynamic analysis, rather than purely static structural analysis;
- Different types of architectural abstractions (e.g., protocols);
- Alternative abstraction techniques which involve recognizing behavioral patterns;
- Detecting interleaving at the architectural level.

The benefits of the approach are the following:

- Validation of the new system requirements;
- Source code candidates for reuse;
- Architectural impact assessment;
- Restructuring recommendations.

For MORALE to combine information from the new requirements and the existing system and then predict the effects of the changes, there must be a common representation for the information from the two sources. Because of the central importance of scenarios to requirements elaboration and architectural evaluation, the representation used in architecture recovery should be structurally similar to the scenarios from which the information is derived. MORALE will supplement traditional structural analyses that are typical of commercial and existing research tools with the extraction of behavior information represented as event traces, patterns of messages among the interacting components. The architectural components of the legacy system will interact in many ways, and these interaction protocols must be understood if we are to evaluate the revised architecture against mission-oriented scenarios. Our work in trace extraction builds on our experience with cliche recognition [69, 87, 88, 89, 90], where the cliches are commonly recurring patterns in legacy code and interleaving detection [73, 74], where the patterns extracted represent situations in which the code implements more than one design goal.

F.5 Option 1: Incremental Adaptation

Adaptation in evolutionary design involves four interrelated questions: *why*, *what*, *how*, and *when*. The first question asks why adapt a system design? Potential answers include failure to meet task requirements and evolution of task requirements. The second question asks what needs to be adapted? Potential answers include changes to the system architecture and the system's knowledge. The third question asks how might the changes be realized in the system? Potential answers include making soft, situation-specific changes or hard, general-purpose changes. The fourth question asks when might the changes be realized in the system? Potential answers include executing the changes immediately or delaying the execution until after evaluation and verification. A robust toolset for supporting evolutionary design will need to address all four issues. The main body of our proposal describes how the MORALE project explores the first two: why and what to adapt. Option 1 on incremental adaptation will study the last two: how and when to adapt.

Consider the last question (when to adapt) first. If a formal model of the old architecture was available, if the new task requirements were known to be complete, and if the proposed architectural changes were provably correct, then the changes could be executed on the design with confidence and without delay. But, in general, none of these premises is valid for complex software. In practice, the proposed changes may need to be evaluated before execution. But this total ordering of the tasks of generation and evaluation of changes means that if the evaluation fails, then the generation process has to start begin all over again. This suggests a partial ordering of the two tasks so that architectural changes are generated and evaluated *incrementally*.

We propose to study this issue of control of processing in incremental architectural adaptation. For the last several years, we have been developing a theory of adaptation in which processes of generating and evaluating changes are *interleaved*[1] [27, 32, 34]. This interleaving enables the evaluation process to inform the generation process so that the generation of proposals for changes is guided by the evaluation of earlier proposals. This control technique itself is informed by a structure-behavior-function model (SBF) model of the design [28, 29, 30, 31]. The SBF model specifies how the software architecture implements its functions, how the internal behaviors of the software compose the functions of its design elements into the functions of the design as a whole. In recent work, we have found that SBF models also enable the process of verification of the new architecture as a whole to inform the process of requirement analysis of performance failures and failure-based detection of originally implicit task requirements [9, 66, 67].

Our work on incremental adaptation is linked to the three main elements in MORALE in the following way. The SBF model-based techniques complement the 'architectural evaluation' methods described in the main proposal: while the latter refer to the assessment of impact of requirement evolution on the architecture of a system, the former are more concerned with the evaluation and verification of the architectural changes and the changed architecture. We will use MORALE's reverse engineering methods to extract a SBF model of the old system, and we will use SBF model-based techniques to inform MORALE's method for requirements analysis. This linkage will result in a more robust toolset for supporting architectural adaptation.

Now consider the question of how to adapt. A proposed architectural change can be realized in software either as a soft, situation-specific modification or as a hard, general-purpose modification. In general, it is easier to make hard, general-purpose modifications. However, it is not easy to undo a hard modification. But

1. This use of the term *interleaving* should not be confused with its use in the discussion of reverse engineering.

as the task requirements continue to evolve, it may become necessary to undo it: an architectural change that is right for today's new task requirements may not be appropriate for tomorrow's. This suggests making the changes soft and situation specific so that they are retractable. But this would make the software bulky and more complex. One possible solution is to initially make only soft, situation-specific changes, but, once the architecture stabilizes, to then abstract the situation-specific changes to more general-purpose changes. We propose to use MORALE to experimentally study the trade-offs between soft and hard architectural changes in simulation software systems.

F.6 Option 2: User Interface Migration

A significant problem in the continued evolution of large systems is in adapting the user interface to a new environment. Often, drastic changes in the user interface are inevitable, as in migrating a command-based system to a workstation with graphical user interface capabilities. Many factors contribute to the difficulty of migrating user interfaces, including rapidly evolving display technologies, lack of standards for user interface API's, and varying degrees of integration of the user interface into the legacy system. Often, fundamental changes in the system architecture are required to update the user interface. For example, migrating to a callback-oriented graphical interface from a strictly menu-driven, procedural interface can require complete restructuring of the legacy system. In many instances the user interface is simply discarded and rewritten.

Since fifty percent or more of the code of an interactive system may be devoted to the user interface, technology for automating the user interface evolution process could significantly reduce the cost of maintenance and migration. The objectives of this research are to reduce the amount of time, cost, and effort required to evolve user interfaces to fit new system environments and technologies by adding automation to the reengineering process. Another benefit of this technology is in reducing overall maintenance costs by representing legacy user interface requirements as an abstract model that is implementation-independent, making migration and further evolution simpler and more automatable.

Our initial research has combined software engineering, reengineering, and human-computer interface technologies to create a technique for addressing user interface evolution from text-based systems to graphical systems [49]. Furthermore, the information extracted by the architectural analysis tools of the MORALE environment can be fed into the user interface migration tools, to help assess the level of effort required to evolve the user interface for a particular system.

Foundational work for this technology has included migration case studies and experiments, transitioning real legacy systems from text or menu-driven interfaces to interactive graphical interfaces. We identified three steps in the migration process:

- Detection - identifying the components of the user interface and the code that implements it within the legacy system,
- Representation - Describing the existing user interface model in an abstract, technology and language-independent manner, and
- Transformation - Generating the new user interface for the desired environment from the abstract model.

As a result of the case studies, we have developed a rule-based detection technique involving slicing, static dataflow analysis, and dynamic analysis to build a model of an existing user interface and to represent the relationship between the user interface and the other components of the legacy system. Currently, this research focuses on developing and refining a implementation-independent representation for user interface abstractions to use as the target of the detection process. We have also experimented with knowledge based techniques for graphical user interface representation and migration [56], which show promise of being applicable to MORALE.

This option proposes the development of a prototype toolset to further refine and automate the detection and representation tasks of the user interface migration process. A central component of this work will be the development of an abstract, technology-independent representation specifically tailored for the reengineering of legacy system user interfaces. The prototype user interface tools will be an integral part of the MORALE toolset and will make active use of architectural knowledge gained from the analysis tools in order to better understand the user interface. Tools such as these have the potential to greatly reduce the effort required to evolve interactive systems.

F.7 Integrating Principles

The MORALE framework portrayed in Section F.1.3 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 Figure 1:

- Design decision support. The MORALE approach to decision support extends IBRA and Synchronized Refinement [70] 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 in process 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 being presented in this proposal. We have described a suite of tools to support MORALE in the context of evolutionary design. 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.

G. Comparison with Related Ongoing Research

Inquiry-Based Requirements Analysis

In our work, we have emphasized a process of requirements elaboration that we believe is common to most styles of requirements expression. Because our approach to requirements elaboration is based on the identification and elaboration of mission-oriented goals, it is closely related to the work of several teams in the AI planning and transformational development traditions [17][96][95][100][101][97][95]. These researchers typically emphasize the formalization of requirements very early in the elaboration process and the role of formal analysis. While their work is conceptually valuable, we believe that requirements determination must start with informal theories, otherwise the problem of mission understanding is merely pushed further upstream. We have been influenced also by groups who are working on the derivation of nonfunctional requirements from mission objectives [52] [46][102].

Other writers have also shown the value of scenarios in software development—for example, in producing object-oriented analysis models [39], [71] and validating operational specifications and prototypes [38], [10]. Our approach most resembles that of researchers who have used scenarios to help user representatives visualize a proposed system [41], [70]. The problem with using scenarios in this way, at least for large systems, is that the scenarios do not clearly help the user representatives develop a better understanding of the system's subtleties. Our linkage of scenarios with the goal-based refinement approach is intended to provide strong guidelines about which scenarios to elaborate and analyze in most detail.

Our model for working rationale [58] [64] is similar to IBIS [16] except that the properties of the artifact trigger the raising of specific types of issue and suggest canonical resolutions. There has been much research into design rationale models (see, for example a forthcoming book by Moran and Carroll), and some of this work emphasizes the expressive adequacy of rationale representations. SYBIL [102], for example, extends our approach in several ways. Although we plan to incorporate ideas from REMAP [106], QOC [105], and QFD [104], our emphasis will remain on pragmatic support for the process of requirements elaboration, not the representational adequacy of rationale models. Empirical evidence strongly suggests that designers are able to use only the simplest representations while they are designing and that complex rationale networks or issue bases are impractical to maintain [103].

Despite the term 'rationale' some of the models used for recording the rationale for design decisions (e.g. IBIS [16], QOC [106]) were originally proposed as frameworks for discussing decisions during the process of making them. Our view of working rationale, and the role it plays in IBRA and architectural evaluation is closer to this tradition than that of research into archival rationale. Like IBRA, "Win-Win" [11] requirements discussions are tied to documents and help the participants elaborate their requirements. The emphasis in Win-Win, however, is more on "people" issues, such as risk management, project planning, and accommodation of multiple stakeholders, than on the technical elaboration of requirements. Our contribution will be to take our earlier IBRA work, and the contributions of Win-Win to the collaborative and planning aspects of requirements analysis, and to refine them for the goal-oriented requirements elaboration method, scenario analysis, and architectural evaluation. Our integration of decision making with architectural models is also similar to the work of Fischer et al. [19], except that they emphasize iconic design representations in domains such as kitchen design and network planning, in which the topology of design diagrams "talk back" to the designer, rather than in more abstract domains in which ideas are more likely to be expressed in text.

Most research in requirements engineering has emphasized tool building or language design, and has not concentrated on empirical analysis of project phenomena or evaluations of new technologies. What little empirical research that has been done concentrates on success factors obtained through surveys and interviews [92][44], analysis of the structure of requirements collaborations [96] and controlled experiments on small problems [101]. There is a dearth of deep analyses of project data (although see [107]). Early in the development of IBRA we started empirical studies of its effectiveness [65], and we are continuing these evaluations.

Architectural Assessment

Software architecture has been the subject of much research activity recently [25], having been the focus of international workshops [22] [21], a conference panel [23], a book [26], and a special issue of a journal

[24]. It was the writings of Shaw in the late 1980's that marked the emergence of a recognized subdiscipline of software architecture to emphasize the unique problems of representing large system structure [25][78][79]. In particular, Garlan & Shaw [31] define an architecture as a configuration of *components*, the location of important function or computation in the system, and *connectors*, the location of coordination between components. Their earliest efforts in software architecture have been distilled and disseminated in various forms (e.g., a book [80], and materials for a graduate course). Perry & Wolf [57], by analogy with other architectural disciplines, introduced the notion of architectural style as an idiomatic pattern of architectural elements.

Analysis of a software architecture has its roots in the software maintenance literature, where focus for many years has been on metrics to help predict the likely cost of modifications based on high level structuring. There are many examples of metrics devised to determine the connectedness (or coupling) of software to predict the ripple effect of modifications (e.g., [35]). These measures can be derived straight from statically or dynamically observed data or control dependencies, or from design-level descriptions, such as common data formats or shared high-level specification [45][75]. Selby & Reimer have shown how interconnection analysis techniques can be used to derive hierarchical descriptions of a large software system and reveal likely error-prone components [76].

Most of the recent research in software architecture has been directed at architecture description languages (ADLs), formal foundations of software architecture, architectural design methods or tool environments, and case studies of architectural design. ADLs and formal models of architecture are important to the work done in SAAM because we rely on some description of the architectures to be analyzed. Our previous SAAM work does not critically rely on specific ADLs or on advances in formal models, and so far have used a minimal ADL that we crafted ourselves. We intend to adopt an established ADL eventually, but the wide variety of formal models (component-centric [2], [50][45] versus connector-centric [6], and model theoretic [2] versus algebraic [51][39]), scope of application (general purpose [85], [47], [6] versus domain-specific (e.g., Honeywell's MetaH for avionics [116])), and presentation style (textual [85], [47] versus graphical [21]), as well as the lack of large case studies in which the full theoretical power of an ADL or model is demonstrated are all clear signals that much work remains in this area.

Several projects [26][89][10][11] are developing programming support environments at the architecture level. Though tools to support SAAM are important, architectural design environments are not a principal concern in this project.

In their introductory article, Garlan & Shaw demonstrate how a discipline can mature based on the existence of significant case studies. Much of the SAAM work has proceeded by means of case studies, and will continue to do so. However, there is little other academic/research work in the software architecture community that concerns itself with assessment of large systems. The only research that comes close to it is in the design patterns work in the OO community [20] [68], with the distinction that design patterns do not attempt to cover whole system behavior. There has been little other work on predicting properties of systems at the architectural level [3].

Reverse Engineering and Architectural Recovery

Architectural recovery has its roots in software maintenance, analysis of programs (both dynamic and static) and visualization. Harris, et al. [36][37] at MITRE have developed tools for automatically recognizing standard architectural features in legacy software. Using the Software Refinery[91], they define recognition rules that automatically extract concrete implementation details that are typically associated with commitments to particular architectural styles. The examples of reflexion models and style recognition were motivated by the work of Abowd, Allen & Garlan [2][3] and are very much in concert with the approach we will take initially in this project on automated architectural understanding because they seed the recovery process with hints above the low-level program details of control or data dependency about what to look for in the existing system. Because Harris, et al. use static analysis techniques, they are restricted to recovering only structural aspects of the architecture. Our work, based on dynamic event trace analysis and guided by scenarios, is aimed at extracting behavioral and functional descriptions of the architecture as well as its structural features. These richer types of descriptions more readily map to meaningful domain-specific concepts and requirements.

Murphy and her colleagues have demonstrated techniques and tools for describing intermediate and flexible models of the source code [50][51]. One technique they describe produces 'reflexion' models of how an extracted source code model compares to another high-level description (the architecture) of the same system.

A key difficulty that arises in architectural recovery - and software reverse engineering in general - is the problem of interleaving [82][83]. Plans for satisfying requirements and design goals are usually intertwined in the design of the software and must be disentangled to identify architectural components and to produce an abstract description of the software. Techniques for detecting and extracting interleaved plans are currently limited in scope.

When what is interleaved is familiar (i.e., stereotypical, frequently used plans), cliche recognition [44][49][53][57] [82][78][103][100][101][102]) is a useful detection mechanism. Many recognition systems deal explicitly with the recognition of cliches that are interleaved in specific ways with unrecognizable code or other cliches. However, existing cliche recognition systems tend to deal with interleaving involving only low-level data and control mechanisms and they rely solely on static analysis and structural pattern matching.

When what is interleaved is unfamiliar (i.e., novel, idiosyncratic, not repeatedly used plans), other, non-recognition-based methods of delineation are needed. For example, slicing [105][65] is a widely-used technique for localizing functional components by tracing through data dependencies within the procedural scope. Cluster analysis [14][47][84][93] is used to group related sections of code, based on the detection of shared uses of global data, control paths, and names. However, clustering techniques can only provide limited assistance by roughly delineating possible locations of functionally cohesive components.

Research into automating data encapsulation has recently provided mechanisms for hypothesizing possible locations of data plans at the level of objects. For example, Bowdidge and Griswold [16] use an extended data flow graph representation to help human users see all the uses of a data structure and to detect frequently occurring computations that are candidates for abstract functions. Techniques have also been developed within the RE^2 project [17][21] for identifying candidate abstract data types and their associated modules, based on call graphs and dominance relations.

Interleaving at the scope of objects and architectures and involving global protocol mechanisms is not yet well understood. Consequently, few mechanisms for detection and extraction currently exist in these areas. This is an open and fertile area of research, which our proposed work will allow us to explore.

OPTION 1 - Incremental Adaptation

Our work on structure-behavior-function (SBF) models builds on AI research on functional representations (FR) of complex devices. Chandrasekaran [12][13] surveys recent FR research. Allemang [5] showed how FR enables the diagnosis of faulty program structures such as interactive loops. Weintraub [86] showed how FR enables the diagnosis of knowledge faults (e.g., incorrect domain knowledge) in a problem solver. We have extended FR into SBF models for exploring evolutionary design. In a recently completed project called Autognostic [81], we have used SBF models for representing problem solving in software systems, including both architecture of problem solving and organization of domain knowledge. We showed how, given a performance failure, the SBF model of the problem-solving architecture and knowledge enables diagnosis of architectural and knowledge faults and redesign of the problem solving to fix the faults and improve the performance [81][82][83][84]. The proposed research seeks to build on this work, with the goal of incrementally adapting the software architecture to satisfy evolving task requirements.

OPTION 2 - User Interface Evolution

The foundations of work in reverse engineering user interfaces were established by Merlo [47] and the Macroscope project, modeling a Cobol legacy user interface in the AUIDL representation language. A rule-based technique for detection is described in [49]. In [48] a knowledge representation language was used to find best mappings between widgets in different GUIs in order to transition a legacy system. Toolsets such as Chiron [85] only support "wrappers" to separate the user interface code from the application code, making evolution more problematic.

H. Key Personnel and Qualifications

Dr. Gregory Abowd is an Assistant Professor in the College of Computing at Georgia Tech. His principal research focuses on the application of formal specification and software architecture for the design and analysis of interactive systems. Dr. Abowd received a B.S. in Mathematics and Physics from the University of Notre Dame in 1986. He attended Oxford University as a Rhodes Scholar and received the degrees of M.Sc. (1987) and D.Phil. (1991) in Computation. From 1989-1992, he was a Research Associate with the Human-Computer Interaction Group at the University of York. From 1992-1994, he was a postdoctoral Research Scientist with the Software Engineering Institute and the School of Computer Science at Carnegie Mellon University. He has been at Georgia Tech since September 1994. He currently holds an honorary appointment as a Visiting Scientist for the Software Engineering Institute. He has been on the program committee for numerous European and American HCI and Software Engineering conferences and workshops, including CHI'94 and CHI'96, SIGSOFT'96, BCS FAHCI, and DSV-IS.He is now serving as Principal Investigator (PI) on grants from Motorola and BNR for work on software architectures. He is also PI for grants working on the prototyping of future computing environments, sponsored by Bell South Intelliventures and In Focus Systems. Dr. Abowd will be supported for two months per year on this grant, and has pending support from NSF and Motorola.

Dr. Ashok K. Goel is an Associate Professor of Computer Science at Georgia Institute of Technology. His primary area of interest is artificial intelligence and design, and his research investigates evolutionary design of complex systems including software systems. He received his Ph.D. in Computer and Information Science from the Ohio State University in 1989. He has served on many conference committees including the 1992 National Conference on Artificial Intelligence (AAAI-92), and was a Vice-Chair of the 1994 International Conference on AI in Design. During 1991-92, he was a technical consultant on a software interface design project at NCR's Human Interface Technology Center, and during 1993-95 he was a member of the Technical Advisory Board of the Educational Technology Institute. He was a Lilly Teaching Fellow during 1992-93, and received both College of Computing's Outstanding Faculty Teaching Award and Georgia Institute of Technology's Junior Faculty Teaching Excellence Award in 1994. His research has attracted funding from NSF, ARPA, ONR, NCR, Northern Telecom, and the Lilly Endowment Foundation. Goel is a PI or a Co-PI on currently active grants from NSF (through Knowledge Systems and Cognitive Models, and Engineering Design and Manufacturing), ARPA (under the Intelligent Integration of Information Program), and ONR (under the case-based reasoning program). Dr. Goel will be supported under Option 1 for one month per year on this grant, and has pending support from NSF.

W. Michael McCracken is a Principal Research Scientist in the College of Computing and the Director of the Center for Information Management Research, and is the Associate Director of the Software Research Center at the Georgia Institute of Technology in Atlanta, Georgia. In his current position he teaches and conducts research in computer science and software engineering and is the coordinator of the Masters program for the College. Mr. McCracken's research interests include software testing and evaluation. Mr. McCracken has over seventeen years experience in the systems engineering, design, implementation, testing, and management of large-scale data processing systems for the U.S. Government. McCracken is the PI on grants from the NSF (Center for Information Management Research), and also has support from the EduTech Institute. Mr. McCracken will be supported for one and one half months per year on this grant, and has no other current pending proposals.

Melody M. Moore is a Research Scientist in the College of Computing at the Georgia Institute of Technology, where she is also pursuing a Ph.D. degree in computer science. Her research focuses on software transitionality, particularly user interface migration and open systems transition technologies. She is director of the Open Systems Laboratory in the College of Computing, whose mission is to develop techniques for transitioning information systems applications to open systems technology. Her work has been supported by the Army Research Laboratory, and she has recently received support from the Georgia Tech Research Corporation and a grant from Schlumberger. Ms. Moore will be supported by Option 2 for two months in the first year of this grant and six months in year two and three. Ms. Moore has no other pending support.

Dr. Colin Potts is Associate Professor of Computer Science at Georgia Institute of Technology, E-Systems Fellow, and Associate Director of the Graphics, Visualization and Usability Center. He received his Ph.D.in cognitive psychology from Sheffield University (UK) in 1982. Dr. Potts's research interests include information system requirements definition and design, user-centered design of interactive systems, and computer-supported cooperative work. He is Co-Editor in Chief of the journal *Requirements Engineering* (Springer), has served on many editorial boards and conference committees, and is a former program chair and steering committee chair for the International Workshop on Software Specification and Design. Potts is currently PI on projects supported by Motorola, NTT Software Laboratory, and E-Systems. Dr. Potts will be supported two months per year on this grant and has pending support from Motorola.

Dr. Spencer Rugaber is a Senior Research Scientist in the College of Computing at the Georgia Institute of Technology. His research interests are in the area of software engineering, specifically software evolution and program comprehension. Dr. Rugaber received his Ph. D. in Computer Science from Yale University in 1978. He is Vice-Chairman of the IEEE Committee on Revere Engineering and past Program Co-chair of the Workshop on Program Comprehension. He is also the author of the article on Program Understanding included in the Encyclopedia of *Encyclopedia of Computer Science and Technology.* Rugaber is currently PI on an ARL project exploring the applicability of domain analysis to reverse engineering and is C0-PI on a project investigating interleaving that is supported by ARPA's environments program. Dr. Rugaber will be supported six moths per year on this grant and has pending support from BNR.

Dr. Linda M. Wills is a Research Scientist at the Georgia Institute of Technology. She received her S.B., M.S., and Ph.D. degrees from the Massachusetts Institute of Technology. She completed her graduate work in the Programmer's Apprentice group at MIT. Her research interests are in automated software understanding, design recovery and reuse, as well as reverse engineering, testing, and optimizing hardware systems. Wills served as program co-chair of the Second Working Conference on Reverse Engineering, held July 1995 in Toronto and is guest editing a special issue of the journal Automated Software Engineering on reverse engineering. Dr. Wills is Co-PI with Dr. Rugaber on the Interleaving project funded by ARPA's environments program. Dr. Wills will be supported one month per year on this grant, and has no other pending support.

I. Previous Accomplishments

The research we are proposing links together requirements analysis, reverse engineering, and architectural assessment. Not only are we the first to point out this link, but we are particularly well-constituted to leverage it. We have a unique blend of talent in these areas which is well-matched to this problem.

Gregory Abowd is a pioneer in the area of software architectural evaluation, and his experience and insights in software architecture evaluation are his principal contributions to this proposal.. He has given tutorials at major HCI and Software Engineering conferences and at leading industrial centers on the topic of software architecture and evaluation. He has recently organized a workshop at the Software Engineering Institute on architecture evaluation in industry as an attempt to see how SAAM stacks up against existing architecture review procedures in place today. While a researcher at the SEI, Dr. Abowd worked on the real-time flight simulators project which has provided a fundamental contribution to software architectures in that domain [4]. His paper on a formal framework for the description and analysis of software architectural style was recommended as one of the top papers at the SIGSOFT'93 Foundations of Software Engineering Symposium [3]. Dr. Abowd is co-author of a leading textbook on Human-Computer Interaction.

Ashok K. Goel has pioneered AI research on structure-behavior-function (SBF) models and model-based theories of adaptation processes in evolutionary design of complex systems. His 1989 Ph.D. dissertation at the Ohio State University was among the first projects to show how SBF models of engineering devices provide the vocabulary and the strategies for automating the retrieval, simulation and adaptation of existing designs. Since joining the College of Computing faculty at Georgia Institute of Technology in 1989, he has continued developing the SBF ontology, language, and modeling techniques for design representation. A recent Ph.D. dissertation in his research group showed how SBF models of problem solving in software systems enable the system to reflect on its performance, and modify the functional and strategic organization of the problem solving in order to improve the performance.

W. Michael McCracken is a recognized authority in the area of software testing and evaluation, including being the co-author of a text on software testing. He has participated in the evaluation of numerous DoD systems for the Director Defense Test and Evaluation, including the IFFN simulation system and the Patriot Air Defense System. While with the General Electric Co, he was the chief software architect on the Conus OTH Experimental Radar System, and the A/N FPS-117. He has conducted research in requirements engineering wtih Potts, and in collaboration technologies with Nunamaker at the University of Arizona. In his role as Co-Director of NSF's Center for Information Management Research, he has been responsible for transition of supported research to numerous of the Center's industrial members.

Melody Moore has worked extensively in transition technologies, including migration to open systems and client-server architectures. She has pioneered a rule-based technique for detecting user interface components in legacy systems and has been studying user interface representation models for reengineering. Ms. Moore was PI for the Knowledge Worker Platform Analysis Project, which utilized a knowledge representation language to perform GUI-to-GUI migration. She was also PI for the TRANSOPEN project, supported by the Army Research Laboratory (ARL), managing a team of sixteen researchers investigating open systems transition issues. She has also served on an ARPA advisory board for Advanced Distributed Simulation.

Colin Potts and his colleagues have pioneered the goal-based requirements refinement approach [7]. To ensure that these concepts are applicable in complex systems, they have concentrated on goal identification [8] and requirements elaboration and validation through scenario exploration [125][72]. In addition to these more analytical approaches, they have also investigated the integration of goal-based models with workplace interviews and observation [124]. Most of Potts' research in design decision support involves refining the IBIS model of design rationale and integrating it with the ongoing elaboration of design models. That approach is generic and has been instantiated for data abstraction design [67][73], JSD [68][73] and VERDI distributed simulation models [73]. The Inquiry Cycle model [70][71] extends this approach to support refinement and method-specific strategies, especially the incorporation of scenario-based refinement. The Inquiry Cycle originates from method development work conducted in the 1980s, in which the goal was to derive formal specifications of real-time systems by systematically analyzing and formalizing requirements by considering recurring method-specific issues [121][122][123]. Potts has collaborated in the design of tools to support the Inquiry Cycle, both for individual designers [128] and collaborative teams [129], and has developed technology for integrating video and audio records of meetings with outlines and hypertextual notes [127]. The models and tools have been applied to a commercial product development [74]. At Georgia Tech, Potts has developed a framework for integrating ongoing research with technology evaluation and transfer [69]. The aim is to select detailed research topics only through investigating empirically the problems that occur in large development projects [120][51], and to conduct much of the research work in the field by participating in industrial strength case studies [8][126][74].

Spencer Rugaber has proposed and developed the Synchronized Refinement process for reverse engineering, which is based on the systematic detection of design decisions in programs [81]. Rugaber is currently PI in two research projects in this area: the first, supported by ARL, explores the application of domain analysis to the problem of reverse engineering; that is, how can domain knowledge be used to improve the process of understanding a program, and how can understanding a program enhance models of its application domain. The second project (with Dr. Wills) investigates a programming phenomenon called interleaving in which several program plans intertwine in a single segment of code. In order to understand the code, the interleaved segments must be isolated and extracted. A recent paper on the topic by Drs. Rugaber and Wills was selected as best paper at the 1995 International Conference on Software Maintenance.

Linda M. Wills developed one of the first cliche recognition systems for automating program understanding. The recognition technique detects and identifies standard, stereotypical computational structures, called cliches, in existing programs. Wills demonstrated the scalability and applicability of this technique to large programs (e.g., message-passing architectural simulators).

J. Facilities, Equipment and Other Resources

The College of Computing at the Georgia Institute of Technology provides an ideal environment for the proposed research because of the extensive computational facilities available and the concentration of colleagues working in Software Engineering. The existence of the Software Research Center (SRC) within the Institute provides strong evidence of the major commitment the Institute has made towards furthering research and education in Software Engineering.

The Software Research Center within the College serves as a focal point of research designed to enhance both the state-or-the-art and the state-of-the-practice of software engineering through industrial liaisons, education and research. Its staff includes nine faculty and eight graduate research assistants. The SRC runs the Open Systems Lab which was established in 1993 to study problems in technology and systems migration and evolution. The mission of the lab is to identify, track and participate in the development of relevant Department of Defense and industry standards and to assess the availability and maturity of open systems software. Areas of current research include standardized operating systems, open systems communications and networking, user interface migration, business process reengineering, reverse engineering and distributed databases.

The College of Computing maintains a variety of computer systems in support of academic and research activities. These include Sun systems used as file and compute servers, eight of which are quad-processor machines, and an Epoch file server with a read-write optical library unit. In addition, there are more than 200 workstation class machines from Sun, Digital, Hewlett-Packard, Silicon Graphics, and IBM, as well as numerous smaller systems and over 125 X Window terminals.

A number of specialized facilities augment the College's general-purpose computing capabilities. These include the Open Systems Lab, as well as the High-Performance and Parallel Computation Experimentation Laboratory (HPPCEL) and the Graphics, Visualization and Usability (GVU) lab. All of the College's facilities are connected via local area networks which, in turn, are linked to a campus-wide network, providing access to systems throughout the Georgia Tech community. As a NSFNET and SURAnet backbone site, the campus networking facilities provide a direct connection to the Internet, offering a means of high-speed communication with the scientific community worldwide.

Additional computing facilities are provided to the Georgia Tech campus by the Office of Information Technology, including several public-access clusters of Apple, IBM, Dell, and Sun workstations, a Sun SPARCcenter 2000 with 12 superscalar RISC processors, CDC Cyber 990 and IBM ES9000 mainframes, and public-access interactive terminals. All are connected to the campus network, which provides direct communication with the Internet, as well as with BITNET and Usenet.

The Army Research Laboratory is also located on the Georgia Tech campus. In the past they have funded some of the earlier work referred to in this proposal, as well as the Open Systems Laboratory described above. They have indicated an interest in this research area and a willingness to be considered as Contract Administrator, should ARPA concur.

K. Cost

See following pages for cost breakdowns for the core and options.

Bibliography

[1] Abowd, G., Allen, R., and Garlan, D. Formalizing style to understand descriptions of software architecture. *ACM Transactions on Software Engineering and Methodology*. To appear.

[2] Abowd. G., Allen, R., and Garlan, G. Using Style to Understand Descriptions of Software Architectures. In *SIGSOFT '93 Symposium on the Foundations of Software Engineering*, ACM Press, December, 1993.

[3] Abowd, G.D., Bass, L., Howard, L. and Northrop, L. Structural modeling: an application framework and development process for flight simulators. Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-93-TR-14, ESC-TR93-192, August 1993.

[4] Abowd, G., Pitkow, J. and Kazman, R.. Analyzing differences between internet information system software architectures. Technical Report GIT-GVU-95-34, Graphics, Visualization and Usability Center, Georgia Institute of Technology, September 1995.

[5] Allemang, D. *Understanding Programs as Devices*. Ph.D. Dissertation. The Ohio State University, Columbus, Ohio, 1990.

[6] Allen, R. and Garlan, D. Formalizing architectural connection. In *Proceedings of the 16th International Conference on Software Engineering*, pages 71–80, Sorrento, Italy, May 1994.

[7] 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.

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

[9] Bass, L., Abowd, G. and Kazman, R.. Issues in the evaluation of user interface tools. In *Software Engineering and Human-Computer Interaction*, volume 896 of *Lecture Notes in Computer Science*, pages 17–27. Springer-Verlag, 1995. Proceedings of the ICSE'94 Workshop on SE-HCI: Joint research issues.

[10] Batory,D. and Sean O'Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology*, 1(4):355–398, October 1992.

[11] Batory, D., Vivek Singhal, Marty Sirkin, and Jeff Thomas. Scalable software libraries. In *Proceedings of SIGSOFT'93*, pages 191–199, December 1993.

[12] Benner, K., Feather, M., Johnson, W.L. & Zorman, L. (1992) Utilizing Scenarios in the Software Development Process. *Proceedings IFIP Working Group 8.1 Working Conference on Information Systems Design Process*, December, North-Holland.

[13] Bhatta, S. Goel, A. and Prabhakar, S. Innovation in Analogical Design: A Model-Based Approach. In *Proc. Third International Conference on Artificial Intelligence in Design*, Lausanne, Switzerland, August 1994, J. Gero and F. Sudweeks (editors), pp. 57-74, Boston, MA: Kluwer Academic Press.

[14] Biggerstaff, T., Mitbander, B., and Webster, D. Program understanding and the concept assignment problem. *Communications of the ACM*, 37(5):72--83, May 1994.

[15] Boehm, B., Bose, P., Horowitz, E. & Lee, M.-J. (1994) Software Requirements as Negotiated Win Conditions. *Proceedings International Conference on Requirements Engineering*, Colorado Springs, CO (April 18-22), IEEE Computer Society Press. pp.74-83.

[16] Bowdidge, R. and Griswold, W. Automated support for encapsulating abstract data types. In Proc. 2nd ACM SIGSOFT *Symp. on Foundations of Software Engineering*, pages 97--110, New Orleans, Dec. 1994.

[17] Canfora, G., Cimitile, A. and Munro, M. A reverse engineering method for identifying reusable abstract data types. In *Proc. of the First Working Conference on Reverse Engineerin*g, pages 73--82, Baltimore, Maryland, May 1993. IEEE Computer Society Press.

[18] Chandrasekaran, B. "Functional representation and causal processes," in *Advances in Computers,* Vol. 38, Academic Press, 1994, pp. 73-143.

[19] Chandrasekaran B. "Functional representations: A brief historical perspective," *Applied Artificial Intelligence*, special issue on functional reasoning, 8:173-197, 1994.

[20] Chandrasekaran, B., Ashok Goel and Yumi Iwasaki. "Functional Representation as a Basis for Design Rationale". *IEEE Computer*, 26(1):48-56, January 1993.

[21] Cimitile, A., Tortorella, M., and Munro, M. Program comprehension through the identification of abstract data types. In *Proc. 3rd Workshop on Program Comprehension*, pages 12--19, Washington, D.C., November 1994. IEEE Computer Society Press.

[22] 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.

[23] Conklin, E. J. & Begeman, M. (1989). 'gIBIS: a tool for all reasons', *J. Am. Soc. Inf. Sci. 40:* 200-213

[24] Cook, Jonathan E. and Wolf, Alexander L. "Toward Metrics for Process Validation." *Proceedings Third International Conference on the Software Process,* Reston, Virginia, October 10-11, 1994, 33-44.

[25] Dardenne, A. Van Lamsweerde A. & Fickas, S. (1993) Goal-Directed Requirements Acquisition. *Science of Computer Programming, 20(1)*, pp. 3-50.

[26] Dean, T. R., and James R. Cordy. A syntactic theory of software architecture. *IEEE Transactions on Software Engineering, Special Issue on Software Architecture*, 21(4), April 1995.

[27] Dijkstra. E.W.. The structure of the "THE"-multiprogramming system. *Communications of the ACM*, 11(5):341–346, 1968.

[28] Fischer, G., Lemke, A. C., McCall, R., & Morch, A. I. (1991). Making argumentation serve design. Human-Computer Interaction, 6, 393-419.

[29] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.. Design Patterns: Abstraction and Reuse of Object-Oriented Software. Addison-Wesley, 1995.

[30] Garlan, D. ICSE-17 Software Architecture Workshop Summary, *Software Engineering Notes*, Vol. 20, No. 3, pages 84–89.

[31] Garlan, D., Robert Allen, and John Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of SIGSOFT'94: The Second ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 179–185. ACM Press, December 1994.

[32] Garlan, D., Tichy, W. and Paulisch, F. Daghstul Software Architecture Workshop Summary, *Software Engineering Notes*, Vol. 20, No. 3, pages 63–83.

[33] Garlan, D. and Perry, D. Software Architecture: Practice, Potential, and Pitfalls. *Proceedings of 16th International Conference on Software Engineering,* Sorrento, Italy, May 16-21, 1994, pages 363-364.

[34] Garlan, D. and Perry, D., guest editors, *IEEE Transactions on Software Engineering*, Vol. 21, No. 4, special issue on software architecture, April 1995.

[35] Garlan, D., Shaw, M. "An Introduction to Software Architecture". *Advances in Software Engineering and Knowledge Engineering*, Volume I, World Scientific Publishing, 1993.

[36] Garlan, D. and Shaw, M. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice-Hall, 1995.

[37] Goel, A. . Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving. Ph.D. Dissertation. The Ohio State University, Columbus, Ohio,December, 1989.

[38] Goel, A. "A Model-Based Approach to Case Adaptation". In *Proc. Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, August 1991, pp. 143-148, Hillsdale, NJ: Lawrence Erlbaum.

[39] Goel, A. "Model Revision: A Theory of Incremental Model Learning." In *Proc. Eighth International Conference on Machine Learning*, Chicago, June 1991, pp. 605-609, Los Altos, CA: Morgan Kaufmann.

[40] Goel, A. "Integrating Case-Based and Model-Based Reasoning: A Computational Model of Design Problem Solving". *AI Magazine*, 13(2):50-54, Summer 1992.

[41] Goel, A. "Representation of Design Functions in Experience-Based Design". In *Intelligent Computer Aided Design*, D. Brown, M. Waldron, and H. Yoshikawa (editors), pp. 283-308, Amsterdam, Netherlands: North-Holland, 1992.

[42] Goel, A and Chandrasekaran, B.. "Functional Representation of Designs and Redesign Problem Solving". In *Proc. Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, Michigan, August, 1989, pp. 1388-1394, Los Altos, California: Morgan Kaufmann Publishers.

[43] Goel, A and Chandrasekaran, B.. "Case-Based Design: A Task Analysis". In *Artificial Intelligence Approaches to Engineering Design,* Volume II: Innovative Design, C. Tong and D. Sriram (editors), pp. 165-184, San Diego: Academic Press, 1992.

[44] 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.

[45] Haney, F.M.. Module connection analysis. In *Proceedings of the AFIPS 1972 Fall Joint Computer Conference*, volume 41, pages 173–179, 1972.

[46] Harris, D, Rubenstein, H, Yeh, A. "Reverse Engineering to the Architectural Level", Proceedings of ICSE-17, April 1995.

[47] Harris, D. Rubenstein, H., Yeh, A. "Recognizers for Extracting Architectural Features from Source Code", *Second Working Conference on Reverse Engineering*, Toronto, Ontario, Canada, July 1995.

[48] Hartman, J. Automatic control understanding for natural programs. Technical Report AI 91-161, University of Texas at Austin, 1991.

[49] Hsia, P., Samuel, J., Gao, J., Kung, D., Toyoshima, Y. & Chen, C. (1994) Formal Approach to Scenario Analysis. *IEEE Software*, March. pp. 33-41.

[50] Hutchens, D. and Basili, V. System structure analysis: Clustering with data bindings. *IEEE Trans. on Software Engineering*, 11(8), August 1985.

[51] Inverardi, P. and Wolf, A. Formal specification and analysis of software architectures using the chemical, abstract machine model. *IEEE Transactions on Software Engineering, Special Issue on Software Architecture*, 21(4):373–386, April 1995.

[52] Jacobson, I. (1992) *Object-Oriented Software Engineering: A Use Case Oriented Approach*, Addison Wesley, Wokingham, UK.

[53] 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.

[54] Johnson, W.L. *Intention-Based Diagnosis of Novice Programming Errors*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986

[55] Karat, J. & Bennett, J.L. (1991) Using Scenarios in Design Meetings: A Case Study. In J. Karat (ed.) *Taking Design Seriously: Practical Techniques for Human-Computer Interaction Design*, Academic Press.

[56] Kazman, R., Abowd, G. Bass, L. and Clements, P. "Scenario-based Analysis of Software Architecture" Journal submission. July 1995.

[57] 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

[58] Kozaczynski, W. and Ning, J.Q. Automated program understanding by concept recognition. *Automated Software Engineering*, 1(1):61--78, March 1994.

[59] Lubars, M., Potts, C. & Richter, C. (1993) A Review of the State of the Practice in Requirements Modeling, *Proceedings RE'93: International Symposium on Requirements Engineering*, San Diego, CA: January 4-6, IEEE Computer Society Press, pp. 2-14.

[60] Luckham, D. C., Augustin, L. M., Kenney, J. J. , Vera, J., Bryan, J. and Mann, W. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering, Special Issue on Software Architecture*, 21(4):336–355, April 1995.

[61] Lange, R. and Schwanke, R.W.. Software architecture analysis: A case study. In *Third International Workshop on Software Configuration Management*. ACM Press, June 1991.

[62] Letovsky, S. Plan analysis of programs. Research Report 662, Yale University, December 1988.

[63] McDermid, J.A. (1994) Requirements Analysis: Orthodoxy, Fundamentalism and Heresy, in M. Jirotka & J. Goguen (Eds.) *Requirements Engineering: Social and Technical Issues*, Academic Press, London. pp. 17-40.

[64] Merlo, Ettore, Gagne, Pierre-Yves, Girard, Jean-Francois, Kontogiannis, Kostas, Hendren, Laurie, Panangaden, Prakash, and DeMori, Renato, "Reengineering User Interfaces", *IEEE Software,* January 1995.

[65] Moore, M., Rugaber, S., and Seaver, P., "Experience Report: Knowledge-Based User Interface Migration", in the *Proceedings of the 1994 International Conference on Software Maintenance*, Victoria, B.C., Sept 1994.

[66] Moore, M.. "A Technique for Reverse Engineering User Interfaces", in the *Proceedings of the 1994 Reverse Engineering Forum*, Victoria, B.C., Sept 1994.

[67] Moriconi, M. Qian, X. and Riemenschneider, R.. Correct architecture refinement. *IEEE Transactions on Software Engineering, Special Issue on Software Architecture*, 21(4):356–372, April 1995.

[68] Gail Murphy and David Notkin. Lightweight source model extraction. In Gail Kaiser, editor, *Proceedings of the Third ACM Symposium on the Foundations of Software Engineering*, Software Engineering Notes 20(4), pages 116–127. ACM Press, October 1995.

[69] Gail Murphy, David Notkin, and Kevin Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In Gail Kaiser, editor, *Proceedings of the Third ACM Symposium on the Foundations of Software Engineering*, Software Engineering Notes 20(4), pages 18–28. ACM Press, October 1995.

[70] Ning, J.Q., Engberts, A., and Kozaczynski, W. Automated support for legacy code understanding. Communications of the ACM, 37(5):50--57, May 1994.

[71] Nixon, B. (1993) Dealing with Performance Requirements during the Development of Information Systems, *Proceedings RE'93: International Symposium on Requirements Engineering*, San Diego, CA: January 4-6, IEEE Computer Society Press. pp. 42-49.

[72] Northop, L., Abowd, G. Bass, L., Clements, P. and Kazman, R. Software architectural evaluation: recommendations of best practice. SEI report on a joint academic/industrial workshop on architectural evaluation, Pittsburgh, November 1995. In preparation.

[73] Parnas, D.L.. On a 'buzzword': hierarchical structure. In *Proceedings of the IFIP Congress 74*, pages 336–339, 1974.

[74] Parnas, D.L.. On the design and development of program families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, 1976.

[75] Parnas, D.L., Paul C. Clements, and David M. Weiss. The modular structure of complex systems. *IEEE Transactions on Software Engineering*, SE-11(3):259–266, March 1985.

[76] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, October 1992.

[77] Potts, C. & Bruns, G. (1988) Recording the Reasons for Design Decisions. *Proc. 10th Int. Conf. Software Eng*., 418-427, Los Alamitos, CA: IEEE Comp. Soc. Press.

[78] Potts, C. (1989) A Generic Model for Representing Design Methods, *Proceedings. 11th International Conference on Software Engineering.*, May, Pittsburgh, PA. IEEE Computer Society Press. pp. 217-226.

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

[80] Potts, C. & Takahashi, K. An Active Hypertext Model for System Requirements. *Proceedings Eighth International Workshop on Software Specification and Design*, Redondo Beach, CA: December. IEEE Computer Science Press, 1993.

[81] Potts, C., Takahashi, K. & Anton, A. Inquiry-Based Requirements Analysis, *IEEE Software*, March 1994, pp. 21-32.

[82] 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.

[83] 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.

[84] Potts, C., Takahashi, K., Smith, J. & Ota, K. (1995) An Evaluation of Inquiry-Based Requirements Analysis for an Internet Service. *Proceedings RE'95: Second International Symposium on Requirements Engineering*,York, UK: March 27-29, IEEE Computer Society Press, pp. 172-180.

[85] Prabhakar, S. and Goel, A. Integrating Case-Based and Model-Based Reasoning for Creative Design: Constraint Discovery, Model Revision and Case Composition. In *Proc. Second International Conference on Computational Models of Creative Design*, Heron Island, Australia, December 1992, pp. 101-127.

[86] [Prabhaka, S. and Goel, A. Performance-Driven Innovation in Design. To appear in *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, Special Issue on Machine Learning.

[87] Pree, Wolfgang . Design Patterns for Object-Oriented Software Development. Addison-Wesley, 1995.

[88] Quilici, A. A memory-based approach to recognizing programming plans. Communications of the ACM, 37(5):84--93, May 1994.

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

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

[91] Rubin, K.S. & Goldberg, A. (1992) Object Behavior Analysis. *Communications ACM 35(9)*, pp. 48-62.

[92] Rugaber, S., Ornburn, S. and LeBlanc, R. Recognizing Design Decisions in Programs. *IEEE Software*, Vol. 7, No. 1, pp. 46-54, January 1990.

[93] Rugaber, S., Stirewalt, K, and Wills, L. "The Interleaving Problem in Program Understanding", *Second Working Conference on Reverse Engineering*, Toronto, Ontario, Canada, July 1995.

[94] Rugaber, S., Stirewalt, K., Wills, L. "Detecting Interleaving", *International Conference on Software Maintenance*, Nice, France, October 1995.

[95] Schwanke, R. An intelligent tool for re-engineering software modularity. In *Proceedings of the 13th International Conference on Software Engineering*, 1991.

[96] Schwanke, R., Altucher, R., and Platoff, M. Discovering, visualizing, and controlling software structure. In *Proc. 5th Int. Workshop on Software Specs. and Design*, pages 147--150, Pittsburgh, PA, 1989.

[97] Selby, R.W. and Reimer, R. M.. Interconnectivity analysis for large software systems. In *Proceedings of the California Software Symposium*, pages 3–17, March 1995.

[98] Shaw, M.. Elements of a design language for software architecture. Position paper for IEEE Design Automation Workshop, January 1990.

[99] Shaw, M.. Toward higher-level abstractions for software systems. In *Data & Knowledge Engineering*, volume 5, pages 119–128. Elsevier Science Publishers B.V., North Holland, 1990.

[100] Shaw, M.. Architectural issues in software reuse: It's not just the functionality, it's the packaging. In *Proceedings of the Symposium on Software Reuse*, April 1995.

[101] Shaw, M. and Garlan, D.. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1995.

[102] Shaw, M. Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young, and Gregory Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering, Special Issue on Software Architecture*, 21(4):314–335, April 1995.

[103] *Software Refinery Toolkit*. Reasoning Systems Incorporated. Palo Alto, California.

[104] Stroulia, E. *Failure-Driven Learning as Device Redesign: A Model-Based Approach*. Ph.D. Dissertation. Georgia Institute of Technology, Atlanta, December 1994.

[105] Stroulia, E. and Goel, A.. "A Model-Based Approach to Reflective Learning". In *Proc. 1994 European Conference on Machine Learning*, Catania, Italy, April 1994, pp. 287-306.

[106] Stroulia, E. and Goel, A.. "Reflective Self-Adaptive Problem Solvers". In *Proc. 1994 European Conference on Knowledge Acquisition*, Germany, September 1994; available in book form as "A Future for Knowledge Acquisition" Luc Steels, Guus Schreiber and Walter Van de Velde (editors), Berlin: Springer-Verlag, 1994.

[107] Stroulia, e. and Goel, A.. "Functional Representation and Reasoning in Reflective Systems". *Journal of Applied Intelligence*, Special Issue on Functional Reasoning, 9(1), 1995.

[108] Talyor, R. N. and Johnson, G. F. Separations of concerns in the Chiron-1 user interface development and management system. In Proceedings of the Conference on Human Factors in Computing Systems, pages 367-374, Amsterdam, April, 1993, Association for Computing Machinery.

[109] Ward, J., J. Krueger, and E. Engstrom. Arched: A visual graph-based approach to architecture specification. Technical Report CS-R93-008, Honeywell Technology Center, March 1993.

[110] Waters, R.C., A method for analyzing loop programs. IEEE Trans. on Software Engineering, 5(3):237--247, May 1979.

[111] Weintraub, M.. An Explanation-Based Approach to Assigning Credit. Ph.D. Dissertation. The Ohio State University, Columbus, Ohio, 1991.

[112] Weiser, M. Program slicing. IEEE Trans. on Software Engineering, 10:352--357, 1984.

[113] Wills, L. Automated Program Recognition, MIT Artificial Intelligence Laboratory Report 904, Master's Thesis, January 1987.

[114] Wills, L. "Automated Program Recognition", *Artificial Intelligenc*e, Vol 45, No 1-2, Sept 1990.

[115] Wills, L. Automated Program Recognition by Graph Parsing, MIT Artificial Intelligence Laboratory Technical Report 1358, PhD Thesis, July 1992.

[116] Wills, L. "Flexible Control for Program Recognition", First Working Conference on Reverse Engineering, Baltimore, MD, May 1993.

[117] Anderson, J.S. and Durney, B., "Using Scenarios in Deficiency-Driven Requirements Engineering," Proceedings of the IEEE International Symposium on Requirements Engineering: RE'93, San Diego, CA, January 4-6, 1993, pp. 134-141.

[118] Curtis, B., Krasner, H. and Iscoe, N., "A Field Study of the Software Design Process for Large Systems," Communications of the ACM, 31(11): 1268-1287, November, 1988.

[119] Carroll, J.M. and Rosson, M., "Getting around the Task-Artifact Cycle: How to Make Claims and Design by Scenario," ACM Transactions on Information Systems, 10(2): 181-212, April, 1992.

[120] Dardenne, A., Fickas, S. and Van Lamsweerde, "Goal-Directed Concept Acquisition in Requirements Elicitation, in Proceedings of the Sixth International Workshop on Software Specification and Design, Como, Italy, October 25-26, 1991.

[121] Feather, M.S. “Constructing Specifications by Combining Parallel Elaborations,” IEEE Transactions on Software Engineering 15(2): 198-208, February, 1989.

[122] Fickas, S., Collins, S. and Oliver, S. “Problem Acquisition in Software Analysis: A Preliminary Study,” University of Oregon, Department of Computer and Information Science, Technical Report CIS-TR-88-04, January 13, 1988.

[123] Green, S. Goal-Driven Approaches to Requirements Engineering, Imperial College of Science, Technology, and Medicine, Department of Computing Technical Report DoC TR-93-42, 1994.

[124] Van Lamsweerde, A., Darimont, A.R. and Massonet, P., “Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt,” *Proceedings RE’95: Second International Symposium on Requirements Engineering*,York, UK: March 27-29, IEEE Computer Society Press, pp.194-203.

[125] Yu, E.S.K. and Mylopoulos, J. “Using Goals, Rules and Methods to Support Reasoning in Business Process Reengineering,” Proceedings of the 27th Hawaii International Conference on System Sciences, Vol. IV: Information Systems: Collaboration Technology, Organizational Systems and Technology, Wailea, HI, January 4-7, 1994, pp. 234-243.

[126] Chung, L., Nixon, B.A. and Yu, E., “Using Non-Functional Requirements to Systematically Support Change,” *Proceedings RE’95: Second International Symposium on Requirements Engineering*,York, UK: March 27-29, IEEE Computer Society Press, pp.132-139.

[127] Maiden, N.A.M., Mistry, P. and Sutcliffe, A.G., “How People Categorize Requirements for Reuse: A Natural Approach,” *Proceedings RE’95: Second International Symposium on Requirements Engineering*,York, UK: March 27-29, IEEE Computer Society Press, pp.148-157.

[128] Lee, J. ‘Extending the Potts and Bruns model for recording design rationale,’ *Proc. 13th Int. Conf. Software Eng.*, 114-125, Los Alamitos, CA: IEEE Comp. Soc. Press, 1991.

[129] Conklin, E. J., & Yakemovic, K-C.B. A process-oriented approach to design rationale. Human-Computer Interaction, 6, 357-391, 1991.

[130] Hauser, J.R. and Clausing, D. The House of Quality ,Harvard Business Review, May/June, pp. 63-73, 1988.

[131] MacLean, A., Young, R.M., Bellotti, V.M. and Moran, T., Questions, Options and Criteria: Elements of Design Space Analysis, Human-Computer Interaction, vol. 6, pp. 201-250, 1991.

[132] Ramesh, B. and Luqi, Process Knowledge Based Rapid Prototyping for Requirements Engineering, Proc. RE’93: 1st Int. Symp. Requirements Eng., San Diego, CA, January, IEEE Comp. Soc. Press, pp. 248-255, 1993.

[133] Walz, D., Curtis, B. and Elam, J. “Inside a Software Design Team: Knowledge Acquisition, Sharing and Integration,” Communications of the ACM 36(10): 62-77, 1993.

[134] Catledge, L. & Potts, C. “Collaboration during Conceptual Design,” to appear in Proceedings of the Second International Conference on Requirements Engineering, Colorado Springs, CO, April 15-18, 1996.

[135] Cunningham, R.J., Finkelstein, A., ,Goldsack, S., Maibaum, T. and Potts, C., “Formal Requirements Specification: The FOREST Project,” Proceedings of the Third International Workshop on Software Specification and Design, London, UK, August 26-27, 1985.

[136] Finkelstein, A. and Potts, C. “Structured Common Sense: The Elicitation and Formalization of System Requirements,” in D. Barnes and P. Brown (Eds.) “Software Engineering 86,” Peter Peregrinus, 1986.

[137] Finkelstein, A. and Potts, C., “Building Formal Specifications Using ‘Structured Common Sense’,” Proceedings of the Fourth International Workshop on Software Specification and Design, Monterey, CA, April 3-4, 1987.

[138] Hsi, I. & Potts, C. “Toward Integrating Rationalistic and Ecological Design Methods for Interactive Systems,” Georgia Institute of Technology, Graphics, Visualization and Usability Center Technical Report 95-27, 1995.

[139] Kramer, J., Ng, K., Potts, C. and Whitehead, K. “Tool Support for Requirements Analysis,” Software Engineering Journal, May 1988, pp. 86-96.

[140] Lubars, M., Potts, C. & Richter, C., “Developing Initial OOA Models,” Proceedings of the 15th International Conference on Software Engineering, Baltimore, MD, May 17-21, 1993, pp. 221-230.

[141] Potts, C. Badre, A. and Bolter, J.D., “Collaborative Pre-Writing with a Video-Based Group Working Memory,” Georgia Institute of Technology, Graphics, Visualization and Usability Center Technical Report 93-35, 1993.

[142] Takahashi, K. & Potts, C., “Tuiqiao: A Hypertext Tool for Requirements Analysis,” submitted for publication, 1995.

[143] Takahashi, K., Potts, C., Kumar, V., Ota, K. and Smith J., “Hypermedia Support for Collaboration in Requirements Analysis,” to appear in Proceedings of the Second International Conference on Requirements Engineering, Colorado Springs, CO, April 15-18, 1996.