Software Architecture Reconstruction: Practice Needs and Current Approaches

Liam O'Brien
Christoph Stoermer
Chris Verhoef

August 2002

TECHNICAL REPORT
CMU/SEI-2002-TR-024
ESC-TR-2002-024

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Software Architecture Reconstruction: Practice Needs and Current Approaches

CMU/SEI-2002-TR-024
ESC-TR-2002-024

Liam O'Brien
Christoph Stoermer
Chris Verhoef

August 2002

Product Line Systems

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Table of Contents

Executive Summary .... **v**

Abstract .... **vii**

1 Introduction **1**

2 Practice Scenarios **3**
- 2.1 The View-Set Scenario 3
- 2.2 The Enforced-Architecture Scenario 5
- 2.3 The Quality-Attribute-Changes Scenario 6
- 2.4 The Common and Variable Artifacts Scenario 7
- 2.5 The Binary Components Scenario 8
- 2.6 The Mixed-Language Scenario 9

3 Existing Approaches and Tools **11**
- 3.1 Manual Architecture Reconstruction 11
- 3.2 Manual Reconstruction with Tool Support 11
- 3.3 Query Languages for Reconstruction 13
- 3.4 Other Techniques 15

4 Evaluation **17**

5 Current and Future Work **19**

6 Conclusions **21**

References **23**

List of Tables

Table 1: Coverage of Practice Scenarios....17

Executive Summary

This report describes the needs, current approaches, methods, and tools for reconstructing software architectures that the Software Engineering Institute (SEI) has identified in its work with Department of Defense (DoD) and commercial organizations. These needs and approaches are presented through descriptions of several practice scenarios for architecture reconstruction. The following information is provided for each scenario: name, context, problem statement, example, and desired solution.

This report is intended for people who need to better understand existing systems or who are considering using architecture reconstruction but do not know about existing approaches, methods, and tools. The approaches covered in this document vary from manual reconstruction to tool-supported manual reconstruction and semi-automated reconstruction, and include data mining and the use of architecture description languages.

The existing approaches are evaluated in relation to the needs presented in the practice scenarios. The result is a list of deficiencies that could be overcome through improvements in the techniques used for architecture reconstruction. An example of such a deficiency is the lack of approaches, methods, and tools for reconstructing the architecture of a system in which several different languages have been used. Another example is the lack of approaches, methods, and tools for reconstructing an architecture that contains binary components but where the source code for these components is not available to the people doing the reconstruction.

This report concludes with a summary of the work that the SEI is doing to overcome some of these deficiencies. Further research should be conducted to identify better approaches and methods, and to develop tools to support them.

Abstract

Software architectures serve as the blueprints for systems, and they are central to the development of software product lines and the design of component-based systems. In existing systems, the architecture often must be reconstructed to reflect the as-built system accurately. This report presents the concept of practice scenarios for architecture reconstruction, which outline common problem/solution pairs that can be used in the strategic application of architecture reconstruction at Department of Defense (DoD) and commercial organizations. Based on an investigation of already developed and presented reconstruction approaches, the report describes deficiencies that have been uncovered in several practice scenarios and proposes improvements.

1 Introduction

Architecture reconstruction is the process by which the architecture of an implemented system is obtained from the existing system. Several approaches for architecture reconstruction have been developed and presented in the past. These approaches have been used for

- evaluating the conformance of the as-built architecture to the as-documented architecture
- reconstructing architecture descriptions for systems that are poorly documented or for which documentation is not available
- analyzing and understanding the architecture of existing systems to enable modification of the architecture to satisfy new requirements and to eliminate existing software deficiencies

There is continuing emphasis on software architectures in Department of Defense (DoD) and commercial organizations. Yet, research into software architectures is still maturing, and the role and potential of architecture reconstruction is not fully understood.

This report describes scenarios in which architecture reconstruction contributes to well-known challenges, such as reconstructing poorly documented software systems, as well as scenarios that expand the role for reconstruction in the overall development process. Reconstruction from available source code may no longer be a standard approach, because vendors of commercial components may purposely make source code unavailable. Organizations are more likely to support reconstruction if it provides a tangible benefit to the development effort.

We propose a set of practice scenarios, which are similar to patterns, that are based on our experiences of applying architecture reconstruction in industrial settings and on research in the architecture community. These scenarios include the identification of needs for methods and tools, and approaches to help satisfy those needs. Current methods, tools, and approaches have been investigated to determine if they cover the needs and to identify where gaps exist that could be filled by further research and development.

Patterns are problem/solution pairs that have, for example, been found to be very useful in architecture [Alexander 79]. They have been successfully applied in software settings (design patterns [Buschmann 96], product line practice patterns [Clements 02a], economics [Etzioni 64], and architecture [Alexander 79]). In this report, even though the practice scenarios that we describe are very similar to patterns, we use the term *practice scenario* rather than the term *pattern*, because so far we have described desired solutions rather than ready-to-use solutions. Our scenarios are relatively general; they can be applied to a large set of similar situations.

Practice scenarios for architecture reconstruction describe recurring situations in which certain problems can be solved by applying proposed solution strategies. Such scenarios are beneficial for development organizations as well as consulting companies that perform architecture reconstructions, because they allow them to identify how reconstruction can be used and possibly applied in their situation. These scenarios are useful for organizations and could guide the practice of architecture reconstruction at such organizations.

The practice scenarios provide approaches for achieving desired solutions. An investigation of current reconstruction approaches has resulted in the identification of some deficiencies and a need to improve the state of the art in architecture reconstruction.

The remainder of this report is organized as follows: Section 2 outlines a set of practice scenarios. Section 3 lists the current approaches in architecture reconstruction. Section 4 provides an evaluation of how well these approaches cover the solution approaches of the practice scenarios described in Section 2. Section 5 outlines our current research and future work. Section 6 summarizes our conclusions.

2 Practice Scenarios

This section captures practice scenarios that we have detected in applying architecture reconstruction in DoD and industrial settings. Our practice scenarios are not invented. Rather, they are discovered as useful solutions in recurring problem contexts.

The format used to describe the practice scenarios is derived from work by Buschmann [Buschmann 96] and consists of

- the scenario name with a short description
- the context in which the scenario applies
- the problem raised by the context
- an example experienced by the authors
- the desired solution the scenario should offer

The scenario format differs from Buschmann's format in two ways: First, an example is added to illustrate the industrial context. Second, the solution is a desired solution rather than the performed solution. The purpose is to offer evaluation criteria that can be used to measure how current approaches in architecture reconstruction are contributing to the desired solution space.

The described scenarios cover a set of architecture reconstruction uses that we have encountered. We do not believe that this is an exhaustive set, and we encourage readers to enhance the proposed scenarios, add solution alternatives, or add further scenarios from their experiences.

2.1 The View-Set Scenario

Name: The *view-set* scenario covers the identification of architectural views that sufficiently describe a software system.

Context: Architecture (re)documentation typically involves the use of a model from which a collection of architectural views and their interrelationships can be extracted. A view consists of a representation of a set of system elements and their interrelationships [Clements 02b]. Typical views include the module view, concurrency view, and deployment view, as well as the top-level context diagram, which presents a system overview. Several view sets are currently in common use. Examples include the 4+1 view by Kruchten [Kruchten 95], the four-view approach by Hofmeister, Nord, and Soni [Hofmeister 00], and the 2+2 view by Lassing [Lassing 02]. Views are categorized by view types (module, component-and-connector, and allocation) and view styles [Clements 02b]. An example is the component-and-connector view type containing a client/server or blackboard view style.

Problem: The problem is to determine which architecture views sufficiently describe the system and cover stakeholder needs.

Example: The process improvement group, within an organization that produces embedded software, would like to evaluate one of the organization's products in a specific market segment. The technical management team has experienced the recurring difficulty of how to decide how well customer requirements are covered by a software implementation. The product lacks an appropriate architecture description. With the exception of providing some interviews, the developers are not available because of other urgent commitments. One activity of the process improvement group is to contract with an analyst to reconstruct the architecture from existing source code to produce a set of architecture views that will reveal the required information.

Desired Solution: The desired solution consists of a method to determine the relevant architecture view set for a particular system. The selected view set will enable the organization commissioning the reconstruction to write a contract with the analyst performing the reconstruction.

The method should contain a catalog of architecture views, notations, and system approaches to enable view selection. The catalog has several dimensions. First, there are various stakeholders, such as developers, architects, project managers, maintainers, testers, or analysts (see our example of the process improvement group, above). The views must address the specific aspects that those stakeholders represent. A further dimension is the use of an appropriate notation. The purveyors of the unified modeling language (UML) claim that it is "the standard notation for software architectures" [Clements 02b]. However, other notations would also be appropriate. Finally, various types of systems are developed using different approaches. Examples include object-oriented systems or functional systems, and customized systems or product line systems.

A view catalog would benefit both the analyst and the organization. The analyst could select, adapt, and apply an appropriate view set for a specific system, and the organization would not be confronted with box-and-arrow figures, which are hard to comprehend and communicate. The view set, purpose, and notation would be streamlined in an overall architecture approach. Conclusively, the selected view set establishes the contract between the analyst and the organization.

2.2 The Enforced-Architecture Scenario

Name: The *enforced architecture* scenario covers the problem of consistency between the as-built architecture and the as-designed architecture.

Context: Architectural patterns and quality attributes are not “first-class citizens” during the implementation of a system. For example, a layer or blackboard has no corresponding language construct in many of the commonly used languages. As a result of this, the as-built and as-documented architectures are often not in conformance; in other words, “language invades design.” Another result can be an inappropriate or poor implementation of the architecture patterns inconsistent with quality attributes. Consistency and enforcement of design and implementation may be difficult to achieve in practice, especially in large development projects. An analysis of architecture conformance would show how well the as-built architecture complies with the as-documented architecture.

Problem: The problem is that consistent traceability information is missing, from architecture design through code implementation.

Example: An organization is developing a product consisting of both in-house-developed and outsourced components. The software architects in the organization previously developed the software architecture, and the participating development organizations (both in-house and external) came to a “final” agreement about the design. Some unexpected problems occurred during the integration tests. Although the product was delivered successfully, the organization contracted with a specialist to measure the conformance of the as-documented architecture with the as-built architecture to mitigate the risks for future product versions.

Desired Solution: The desired solution should consist of a method and supporting tools to enforce architecture conformance.

Ideally, the method should be an integral part of a general forward-engineering tool. A meta-model would capture traceability relationships, which would be analyzed, measured, and enforced by a tool.

A further application of architecture conformance could originate from an envisioned environment in which commercial off-the-shelf (COTS) software is used (see the binary components scenario later). Assume that component descriptions are available (of interfaces, for example) and that an organization has defined a software architecture for a system. The method and tool should measure how well components comply with the defined architecture.

The overall benefit of reconstruction in this scenario is consistency and compliance of architectures, decisions, and design guidelines throughout further design and implementation. Furthermore, contractual issues could be investigated, as in the example above.

2.3 The Quality-Attribute-Changes Scenario

Name: The *quality attribute changes* scenario covers the question of how architecture patterns are used to satisfy quality requirements and to what extent changes to quality attributes impact a system.

Context: Quality attributes are achieved primarily through attention to software architectures. Design decisions embodied by software architecture are strongly influenced by the need to achieve quality attribute goals. However, quality attributes are not orthogonal. That means that design decisions reflect tradeoffs between competing qualities. For example, a cyclic executive [Locke 92] contributes positively to memory performance but negatively to extensibility. The tradeoff decisions must be balanced by considering the business priorities on quality attributes. Typically those tradeoffs are done during very early design phases. Later changes to quality attributes may have deep impacts on the system.

Problem: The problem is how to determine the relationship among quality attributes and architecture elements.

Example: An organization wants to migrate one of its applications to a Web-based environment. One of the organization’s concerns is how a change of quality attributes (e.g., performance—the system must handle 100,000 transactions instead of 1,000 transactions per day—or security—security must be heightened in a Web environment) would impact the current system. To date, soft-real-time performance issues were not a critical factor in the product setting, because the transactions were settled in a batch environment. An appropriate architecture description for an assessment is not available. The organization orders an architecture reconstruction with the focus on determining how quality attributes are supported in its current architecture and which parts of the architecture would be affected by changes in the quality attributes.

Desired Solution: The solution should consist of a method and tool for recovering information about how a system contributes to particular quality attributes.

The solution addresses a couple of open research issues in the architecture community. For example, which architecture patterns support which quality attributes? If they support certain quality attributes, how do we measure their contribution?

This scenario provides two major benefits. First, it enables the identification of architecture and design patterns that contribute to certain quality attributes. Second, it can uncover design decisions that help developers balance competing quality attributes.

2.4 The Common and Variable Artifacts Scenario

Name: Commonality and variability are used in product line environments so that organizations can reduce costs by reusing common assets. The *common and variable artifacts* scenario provides models and techniques for analyzing the products in a domain with respect to their common and variable parts.

Context: Product lines embody a strategic reuse model of products sharing a market segment. As opposed to opportunistic reuse, in strategic reuse only those components that belong to the core assets of a product line are reused. The software architecture reflects common and variable parts of the system and offers appropriate design constructs. Product lines evolve out of the commonalities among existing products in a specific market segment. Typically, several products are delivered until a systematic migration to a product line takes place. To evaluate the potential for creating a product line from existing products, it is necessary to "mine" their architectures and analyze the commonality and variability across those architectures.

Problem: The problem is to identify the common and variable parts in several similar products.

Example: A business unit of a large organization has three development departments producing similar products worldwide. As part of a consolidation effort, a group of analysts investigates the potential of using a software product line approach to increase the business value of the organization's products. One task is to conduct a technical analysis of commonality and variability across products from the development departments. The group determines that the organization should conduct an in-depth architecture reconstruction for three representative products, one from each department, in order to reveal the parts of each system that are most amenable for consolidation into one overall system.

Desired Solution: The desired solution consists of methods and tools to identify and evaluate common and variable parts across products.

An analysis at the source level is difficult because different structures, naming conventions, or even implementation languages might have been used. Therefore architecture descriptions, including architecture patterns, quality attributes, component interfaces, and design rationales, provide an appropriate abstraction level for comparing existing products in a market segment.

This scenario provides two major benefits. First, the analysis can contribute rational arguments for product line migration in situations that may be politically difficult. Second, the insights gained provide useful information for applying an architecture-based design effort for the generation of a new product line.

2.5 The Binary Components Scenario

Name: The *binary components* scenario covers architecture reconstruction using binary component descriptions.

Context: The software industry is quickly moving toward systems based on commercial components. A component in this context has three characteristics: it is produced by a vendor, who sells the component or licenses its use; it is released by a vendor in binary form; and it offers an interface for third-party integration [Wallnau 02]. Existing architecture reconstruction methods abstract from the source code. Reconstructing software architecture in binary component settings is heavily dependent on the quality of the component interface descriptions. In addition, the detail in these descriptions may vary from vendor to vendor.

Problem: The problem is conducting architecture reconstruction in settings where COTS components are used.

Example: An organization is developing a Web-based application consisting of a Web server from one vendor and a database from another. In addition, two other commercial components must be integrated into the system. The organization's software architects want to define the entire software architecture and understand what "glue parts" the organization must develop between the components (such as forms and data transformations). The component interfaces and partial architectural descriptions (e.g., the Web server and database architectures) are available; the source code for the COTS components is not.

Desired Solution: The desired solution consists of methods and tools to support architecture reconstruction from binary components.

The solution addresses a couple of open research issues in the architecture field. For example, how do we sufficiently describe components to be able to extract a software architecture? How do we know that assembled components satisfy the desired software architecture? Furthermore, how trustworthy are component descriptions?

Assembling commercial components to achieve functional quality goals is a difficult task in the current component market. A commonly used approach is to build small toy examples to explore deficiencies so as to mitigate the risks for real products. Architecture reconstruction from commercial component descriptions could help architects detect the overall product structure and the dependencies among components.

2.6 The Mixed-Language Scenario

Name: Software systems implemented in several programming languages are commonplace today. Jones states that “about 30% of U.S. software applications contain at least 2 languages, based on our clients’ portfolios” [Jones 98]. The *mixed-language* scenario addresses the need for models and techniques that can be used to analyze products that are implemented in a variety of languages and language types (procedural and object oriented).

Context: Many existing systems are implemented in several programming languages, including C, C++, and Fortran. These systems may also include start-up files that configure the system at runtime based upon a set of script and data files. These systems may also have makefiles or build scripts that contain architecturally relevant information. How can all of the various components in several different languages and language types be modeled within a single reconstruction tool? What are the abstraction mechanisms for building architectural views from the source information from particular language types (procedural and object oriented), and how can these be combined to produce architectural views that incorporate the different types of information?

Problem: The problem is to reconstruct the architecture of a system that is implemented in more than one language.

Example: An organization wants to understand its existing system, because it must integrate parts of that system with one being built by another organization. No architecture documentation of the existing system exists, and there is no one in the organization who knows all of the existing system. Certain individuals know parts of the system. The system is implemented in several languages.

Desired Solution: Architecture reconstruction techniques and tools should be able to handle the reconstruction of a system that is implemented in more than one language. Techniques have already been developed to extract information from mixed-language applications [Brand 98], and the solution is to use that information to build architecture representations of systems.

Benefit: Architecture reconstruction mechanisms that apply to source information from systems implemented in particular language types are useful. Many new systems are being implemented in a variety of languages. Approaches to combining the abstractions for various language types will provide the ability to reconstruct the architecture of these systems.

3 Existing Approaches and Tools

Many approaches to architecture reconstruction and tools to support those approaches have been covered in the literature. Categories of approaches and tools include

- manual architecture reconstruction
- manual reconstruction with tool support
- query languages for writing patterns to build aggregations automatically
- use of other techniques, including clustering, data mining, and using architecture description languages

The following are some of the main approaches in each category. It is not an exhaustive list, but it enumerates a representative set of approaches and tools.

3.1 Manual Architecture Reconstruction

Laine presents work that he carried out in manually reconstructing the architecture of an object-oriented system to develop ideas that could be applied to the development of other object-oriented systems [Laine 01]. To reconstruct the architecture, a high-level overview of the system was generated, and code was assigned to various parts of the view. Examination of the code revealed architecture components. Clustering and abstraction were used to build component views. No tools were used to support the reconstruction effort. The only utilities used were the UNIX utilities Emacs and Grep. Any views that were generated were drawn using pen and paper.

3.2 Manual Reconstruction with Tool Support

The following set of approaches and tools support manual reconstruction.

Portable Bookshelf (PBS)

The Portable Bookshelf (PBS) is a toolkit used for generating a “software bookshelf” [PBS 02, Finnigan 97]. A software bookshelf for a large system can provide an easily accessible Web-based structure for storing information about a system. The information contained in the bookshelf includes source code, as well as other documentation about the system. Other information that can be accessed includes test cases, performance analysis, future plans, architectural diagrams, and information about a project's history. Bowman and others have presented a method for extracting architectural documentation from the code of an implemented system using parts of the PBS [Bowman 99]. In an example, they reconstructed the architecture of the Linux system. They analyzed source code using the cfx (c-code fact extractor) program to obtain symbol information from the code and generated a set of relationships among the symbols. They then manually created a tree-structured decomposition of the Linux system into subsystems and assigned the source files to those subsystems. Next, they used the grok fact manipulator tool to determine relationships among the identified subsystems, and they used the lsedit visualization tool to visualize the extracted system structure. Refinement of the resulting structure was carried out by moving source files among subsystems.

Rigi

Rigi is a tool for visualizing and manipulating software information [Rigi 02]. It is end-user extendable, contains an interpreter for applying operations to the visualized information, and allows for manual manipulation of the information that is presented to the user. For architecture reconstruction, one can apply groupings to the underlying elements by manually selecting nodes in the visualization and collapsing them, or by applying operations in the interpreter. The tool provides various capabilities for filtering node and arc types, and it also enables the application of various layouts to the presented views. In addition, Rigi provides parsers for extracting information in Rigi Standard Format (RSF) for various languages.

SHriMP

SHriMP is an information-visualization and navigation system [Shrimp 02, Storey 01]. It can be used to visualize information extracted from a system. When used for reconstruction, the tool can assist a user in generating high-level architectural views of a system by manually grouping and aggregating elements in a graph. The tool takes as input RSF files and, when used with the Rigi tool, can provide useful navigation and visualization of the architectural views generated using Rigi.

KLOCwork inSight Tool

KLOCwork inSight uses code-analysis algorithms to extract software architecture views, interactions, logic flow, and execution threads directly from the source code of both full and partial systems [Klocwork 02]. The product description for KLOCwork inSight states that it "allows for architectural comprehension, automatic control, and management through its graphic visualization of software architecture, architectural rules setting, and automatic tracking capabilities." The tool does not allow a user to build or apply patterns to abstract the architecture from the underlying information extracted from the source code. Rather, the tool allows the user to select source elements from the visualization and to create higher level groupings of those elements into architectural components, thus facilitating architecture reconstruction. It allows architectural control and management through its architectural rules-setting and automatic-tracking capabilities. This ensures that “no ‘risky’ code is submitted and keeps architectural integrity in check” [Klocwork 02].

3.3 Query Languages for Reconstruction

The following set of approaches and tools support the use of query languages for reconstruction.

Mitre

Harris and others have presented a framework for architecture reconstruction that uses a combined bottom-up and top-down approach [Harris 95a, Harris 95b]. The framework consists of three components: the architectural representation, the source code recognition engine and supporting library of recognition queries, and a “bird’s-eye” program-overview capability. The bottom-up analysis uses the bird’s-eye view to display the system’s file structure and components, and to reorganize information into more meaningful clusters. The top-down analysis uses particular architectural styles to define components that should be found in the software. Recognition queries are then run to determine whether the expected components exist. Harris’s approach is based upon a set of queries, which are independent of the implementation language and that are applied to an abstract syntax tree (AST). Parsing the source code of a system generates the AST, which in this case is specific to a particular programming language. The application mechanism of the queries is specific for each programming language. Thus, if a new language must be handled, a new AST must be developed, a parser must be written, and a new application mechanism must be derived. Lämmel and Verhoef report on efforts to solve this problem [Lämmel 01a, Lämmel 01b].

Dali

Dali is a collection of various tools in the form of a workbench [Kazman 99]. Included in the workbench are the Rigi tool and the PostgreSQL relational database. Rigi provides visualization and manipulation of the views that are generated, and the Dali extension to Rigi provides the capability of defining and applying query patterns to the underlying data to generate various architectural views of the system. Information is extracted from the source code of a system using software analysis tools and then loaded into Dali. Information can also be obtained from other sources (such as other forms of documentation) and loaded into Dali. This information is stored in the PostgreSQL database and is visualized in Rigi. Various queries can be written in a combination of Structured Query Language (SQL) and Perl and applied to generate abstractions of the information. The results of the queries are visualized in Rigi, and further queries can be written and applied, or the views can be manipulated manually to generate architectural views of the system.

Architecture Reconstruction Method (ARM)

Guo and others have presented a semi-automatic architecture recovery method called the software Architecture Reconstruction Method (ARM), which can be used to assist in architecture recovery for systems that are designed and developed using patterns [Guo 99]. The ARM consists of four major phases: (1) development of a concrete pattern-recognition plan, (2) extraction of a source model, (3) detection and evaluation of pattern instances, and (4) reconstruction and analysis of the architecture. Case studies have been presented showing the use of the ARM to reconstruct systems and check the conformance of these systems against their documented architectures. Pattern rules are transformed into pattern queries, which can be applied automatically to detect pattern instances from the source model. Refinement of the pattern queries can help to improve the precision of pattern recognition. Visualizations of the recovered patterns are presented to the tool user and aligned with the designed pattern instances.

Guo and others used the Dali workbench to perform architecture recovery work. An abstract pattern rule was mapped into a concrete pattern rule and was converted into a SQL query. This query was then applied to the database to extract instances of the pattern. This method is aimed particularly at systems that have been developed using design patterns. This limits the applicability of the method so that it may only apply to systems developed using design patterns or in cases where one can be sure that design-pattern implementations have not eroded over time.

Riva

Riva provides an approach to architecture reconstruction based on extracting information from source code, loading the information into a Prolog fact database, and using Prolog to build abstractions of that information and visualize those abstractions using Rigi [Riva 00]. The process that Riva describes consists of six phases: (1) develop a high-level architecture description; (2) extract source information; (3) abstract to generate an architecture model; (4) redocument the system; (5) analyze the system and come up with an improvement plan; and (6) reorganize the architecture.

3.4 Other Techniques

The following are examples of other techniques, approaches, and tools that have been used for architecture reconstruction.

Data Mining

Alborz is a user-assisted reverse engineering tool designed for use in analyzing and recovering software architecture in the form of cohesive modules and subsystems [Sartipi 01]. The tool's operation is based on techniques taken from the areas of data mining, pattern matching, and clustering. The tool user defines a graph-based architectural pattern of system modules (subsystems) and their interactions based on domain knowledge, system documents, and tool-provided clustering techniques. Through an iterative recovery process, the user constrains the architectural pattern, and the tool provides a decomposition of the system's entities into modules or subsystems that satisfy the constraints.

Software Architecture Reconstruction (SAR) Method

Krikhaar outlines the software architecture reconstruction (SAR) method based on a relation partition algebra [Krikhaar 99, Feijs 95, Feijs 98b, Feijs 99]. This method employs five levels of architecture reconstruction: initial, described, redefined, managed, and optimized. Krikhaar introduces the notions of InfoPacks and ArchiSpects. InfoPacks or information packages are packages of information extracted from a system. These packages can be extracted from the source code, design documents, and other sources. An InfoPack also contains a description of the extraction steps to be taken to retrieve this information from the software. ArchiSpect is a view of the system that makes explicit a certain architectural structure. A set of these ArchiSpects can be used to describe a system's architecture. InfoPacks are used to construct ArchiSpects. Philips uses the Teddy tool to visualize ArchiSpects [Feijs 98a].

X-RAY

Mendonça and Kramer have presented the X-RAY approach for recovering the architecture of distributed software systems [Mendonça 01]. X-RAY is implemented in a Prolog environment. Information extracted from the source is represented as Prolog facts. Clustering, search engines, and constructs for pattern description are implemented as Prolog predicates. Dot is used to convert the outputted views to Postscript drawings [Koutsofios 92].

Architecture Description Languages

Eixelsberger and others have presented a process for recovering the architecture of a program family [Eixelsberger 98]. This work was carried out as part of the European Commission ESPRIT project Architecture Reasoning for Embedded Systems (ARES). The process includes two tasks: identification and recovery of architectural properties, and construction of architectural descriptions for those properties. Eixelsberger and others developed a language for describing properties of software architectures called architecture structure description language (ASDL). A reference architecture representing the common architectural elements of a product family is recovered based upon the ASDL description of the members of the product family.

4 Evaluation

This section provides an evaluation of how well current architecture reconstruction approaches cover the practice scenarios presented in Section 2. The rating of each approach is performed according to the following scale:

ns	The approach does not seem to support the scenario.
u	It is unknown how the approach covers the scenario.
~	The approach must be adapted in order to be applicable.
+	The approach supports the scenario.
++	The approach supports the scenario with a specific method and tool.

The scenario coverage for each approach is illustrated in Table 1.

	View-Set	Enforced-Architecture	Quality-Attribute-Changes	Common & Variable Artifacts	Binary Components	Mixed-Language
Manual	~	~	~	~	~	~
PBS	~	~	~	ns	ns	~
Rigi	~	ns	ns	ns	ns	+
Shrimp	~	ns	ns	ns	ns	+
KLOCwork	~	++	ns	ns	ns	~
Mitre	~	ns	ns	ns	ns	+
Dali	~	~	~	ns	ns	+
ARM	~	~	~	ns	ns	u
Riva	~	~	u	ns	ns	+
Alborz	~	ns	ns	ns	ns	~
SAR	~	u	~	ns	ns	+
X_RAY	~	~	u	ns	ns	+
ASDL	~	ns	~	+	ns	u

Table 1: Coverage of Practice Scenarios

The ratings are sometimes fuzzy, because the approaches do not always address a specific scenario context. But overall, we can extract the following results for each practice scenario:

- *view-set*: No current approach or tool supports an explicit selection of architecture views that can be systematically reconstructed in order to describe a system sufficiently and address its stakeholders' needs. We assume that existing approaches and tools could be adapted to allow a view-set selection.
- *enforced-architecture*: This practice scenario is supported only by a commercial vendor with a specific method and tool. The tool is used for analyzing and refactoring a system. However, it is not embedded in a forward-engineering tool.

- *quality-attribute-changes*: No current approach explicitly supports this practice scenario. On one hand, further investigation is needed to determine if existing approaches and tools can be adapted to cover this practice scenario. On the other hand, further architecture research is necessary to uncover dependencies between quality attributes and architecture patterns.
- *common and variable artifacts*: Only the ASDL approach seems to cover this scenario. However, commonality and variability analysis, as reported, is not supported by a tool. Further research could lead to improved approaches and new tools to cover this scenario.
- *binary components*: No current reconstruction approach or tool supports this practice scenario. This deficiency should be addressed because of a fast-growing component market and demands for component certification.
- *mixed-language*: Several tools support mixed-language information. However, the approaches do not seem to describe how to build architectural views from information extracted from mixed-language systems. Further investigation is needed to determine how to incorporate into these methods the ability to carry out architecture reconstruction in mixed-language environments.

The manual approach neither supports nor fails to support a particular practice scenario. However, a manual architecture reconstruction approach may not be economically justifiable for an organization unless that organization can reap a lot of benefit from the approach.

5 Current and Future Work

Based on the evaluation results presented in Section 4, we are currently undertaking research to address the demands of DoD and commercial organizations as encapsulated by the practice scenarios presented in Section 2. We are

- enhancing the practice scenario catalog and adding solutions or similar business contexts to already existing scenarios. To do this, we are planning to offer a specific Web page for the architecture reconstruction community.
- creating a view catalog for architecture reconstruction with guidelines for stakeholders and particular system types, based on the SEI's architecture documentation work [Clements 02b]. This work offers substantial solutions to the view-set practice scenario.
- developing a methodology with a supporting tool set for commonality and variability analysis. The primary focus is to help organizations analyze and assess their products in a specific market for a potential migration to a product line. A second aspect of this research is the development of a successor to the Dali architecture reconstruction workbench [Kazman 99, Dali 02] with an enriched tool set and architecture reconstruction language capabilities. This work addresses the needs for enhancing the common and variable artifacts scenario.

6 Conclusions

Practice scenarios describe recurring problems of architecture reconstruction needs at organizations and present solutions to them. In their current state, the practice scenarios address both how organizations apply reconstruction and the need for architecture reconstruction research. Organizations can systematically apply architecture reconstruction techniques to achieve their specific business goals, which are normally broader than the results of an isolated reconstruction effort. The research community can contribute methods and tools to present economical and efficient solutions for each scenario's problem statement.

The evaluation has shown that current approaches do not cover the practice scenarios sufficiently. Furthermore, we assume that a single approach will probably not cover all practice scenarios. Therefore, the scenario solution space should describe approaches and strategies that best fit a specific problem.

We assume that the application of architecture reconstruction will be used in a much broader technical sense. The existing approach of abstracting from source code is difficult to apply in black-box component markets. On the other hand, companies are being forced to assemble "partial architectures" of commercial components to generate views of their overall system architecture.

Finally, we suggest that there be a closer relationship between the architecture reconstruction community and the architecture and component communities.

References

[Alexander 79] Alexander, C. *The Timeless Way of Building*. New York, NY: Oxford University Press, 1979.

[Bowman 99] Bowman, T.; Holt, R.C.; & Brewster, N.V. “Linux as a Case Study: Its Extracted Software Architecture,” 555-63. *Proceedings of the 21st International Conference on Software Engineering*. Los Angeles, CA, May 16-22, 1999. New York, NY: IEEE Computer Society Press, 1999.

[Brand 98] Van den Brand, M.G.J.; Sellink, M.P.A.; & Verhoef, C. “Current Parsing Techniques in Software Renovation Considered Harmful,” 108-17. *Proceedings of the Sixth International Workshop on Program Comprehension*. Ischia, Italy, June 24-26, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[Buschmann 96] Buschmann, F.; Meunier, R.; Rohmert, H.; Sommerlad, P.; & Stal, M. *Pattern-Oriented Software Architecture*. New York, NY: John Wiley & Sons, 1996.

[Clements 02a] Clements, P. & Northrop, L. *Software Product Lines*. Boston, MA: Addison Wesley, 2002.

[Clements 02b] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison Wesley, 2002.

[Dali 02] *The Dali Architecture Reconstruction Workbench.* <http://www.sei.cmu.edu/ata/products_services/dali.html> (2002).

[Eixelsberger 98] Eixelsberger, W.; Ogris, M.; Gall, H.; & Bellay, B. “Software Architecture Recovery of a Program Family,” 508-11. *Proceedings of the International Conference on Software Engineering*. Kyoto, Japan, April 19-25, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[Etzioni 64] Etzioni, A. *Modern Organizations*. Englewood Cliffs, NJ: Prentice-Hall, 1964.

[Feijs 95] Feijs, L.M.G. & van Ommering, R.C. *Theory of Relations and Its Applications to Software Structuring* (Phillips Research Internal Report, RWB-510-re-95011). Eindhoven, The Netherlands, 1995.

[Feijs 98a] Feijs, L.M.G. & de Jong, R.P. “3D Visualization of Software Architectures.” *Communications of the ACM 41*, 12 (December 1998): 73-78.

[Feijs 98b] Feijs, L.M.G. & Krikhaar, R.L. “Relation Algebra with Multi-Relations.” *International Journal of Computer Mathematics 70*, 1 (November 1998): 57-74.

[Feijs 99] Feijs, L.M.G. & van Ommering, R.C. “Relation Partition Algebra—Mathematical Aspects of Uses and Part-Of Relations.” *Science of Computer Programming 33*, 2 (February 1999): 163-212.

[Finnigan 97] Finnigan, P.J.; Holt, R.; Kalas, I.; Kerr, S.; Kontogiannis, K.; Mueller, H.; Mylopoulos, J.; Perelgut, S.; Stanley, M.; & Wong, K. “The Portable Bookshelf.” *IBM Systems Journal 36*, 4 (November 1997): 564-93.

[Guo 99] Guo, G.; Atlee, J.; & Kazman, R. “A Software Architecture Reconstruction Method,” 225-243. *Proceedings of the First Working International Federation for Information Processing (IFIP) Conference on Software Architecture*. San Antonio, TX, February 22-24, 1999. Norwell, MA: Kluwer Academic Publishers, 1999.

[Harris 95a] Harris, D.R.; Reubenstein, H.B.; & Yeh, A.S. “Recognizers for Extracting Architectural Features from Source Code,” 252-61. *Proceedings of the Second Working Conference on Reverse Engineering*. Toronto, Ontario, July 14-16, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Harris 95b] Harris, D.R.; Reubenstein, H.B.; & Yeh, A.S. “Reverse Engineering to the Architectural Level,” 186-95. *International Conference on Software Engineering*. Seattle, WA, April 23-30, 1995. New York, NY: IEEE Computer Society Press: 1995.

[Hofmeister 00] Hofmeister, C.; Nord, R.; & Soni, D. *Applied Software Architecture*. Boston, MA: Addison Wesley, 2000.

[Jones 98] Jones, C. *The Year 2000 Problem—Quantifying the Costs and Assessing the Consequences*. Boston, MA: Addison Wesley, 1998.

[Kazman 99] Kazman, R. & Carrière, S.J. "Playing Detective: Reconstructing Software Architecture from Available Evidence." *Journal of Automated Software Engineering 6*, 2 (April 1999): 107-138.

[Klocwork 02] *KLOCwork inSight.* <http://www.klocwork.com/products/inSight.html> (2002).

[Koutsofios 92] Koutsofios, E. & North, S. *Drawing Graphs With Dot*. Murray Hill, NJ: AT&T Bell Laboratories, 1992.

[Krikhaar 99] Krikhaar, R.L. "Software Architecture Reconstruction." PhD diss., University of Amsterdam, 1999.

[Kruchten 95] Kruchten, P. "The '4+1' View Model of Software Architecture." *IEEE Software 12*, 6 (November 1995): 42-50.

[Laine 01] Laine, P.K. "The Role of Software Architecture in Solving Fundamental Problems in Object-Oriented Development of Large Embedded Systems," 14-23. *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*. Amsterdam, Netherlands, August 28-31, 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[Lämmel 01a] Lämmel, R. & Verhoef, C. "Semi-Automatic Grammar Recovery." *Software Practice and Experience 31*, 15 (December 2001): 1395-1438.

[Lämmel 01b] Lämmel, R. & Verhoef, C. "Cracking the 500-Language Problem." *IEEE Software 18*, 6 (December 2001): 78-88.

[Lassing 02] Lassing, N. "Architecture-Level Modifiability Analysis." PhD diss., Vrije Universiteit Amsterdam, 2002.

[Locke 92] Locke, C.D. “Cyclic Executive vs. Fixed Priority Executives.” *Real-Time Systems 4*, 1 (March 1992): 37-53.

[Mendonça 01] Mendonça, N.C. & Kramer, J. “Architecture Recovery for Distributed Systems.” *SWARM Forum at the Eighth Working Conference on Reverse Engineering*. Stuttgart, Germany, October 2-5, 2001. Los Alamitos, CA: IEEE Computer Society, 2001. <http://www.program-transformation.org/twiki/bin/view/Transform/SwarmForum>.

[PBS 02] *The Portable Bookshelf.* <http://swag.uwaterloo.ca/pbs/> (2002).

[Rigi 02] *The Rigi Tool.* <http://www.rigi.csc.uvic.ca/> (2002).

[Riva 00] Riva, C. “Reverse Architecting: An Industrial Experience Report,” 42-50. *Proceedings of the Seventh Working Conference on Reverse Engineering*. Brisbane, Australia, November 23-25, 2000. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[Sartipi 01] Sartipi, K. & Kontogiannis, K. “A Graph Pattern Matching Approach to Software Architecture Recovery,” 408-419. *Proceedings of the IEEE International Conference on Software Maintenance*. Florence, Italy, November 7-9, 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[Shrimp 02] *SHriMP Views.* <http://shrimp.cs.uvic.ca/shrimp/shrimp_intro.shtml/> (2002).

[Storey 01] Storey, M.A.D.; Best, C.; & Michaud, J. “SHriMP Views: An Interactive Environment for Exploring Java Programs,” 111-12. *Proceedings of the Ninth International Workshop on Program Comprehension*. Toronto, Ontario, May 12-13, 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[Wallnau 02] Wallnau, K.; Hissam, S.A.; & Seacord, R.C. *Building Systems from Commercial Components*. Boston, MA: Addison Wesley, 2002.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **AGENCY USE ONLY**
(Leave Blank)

2. **REPORT DATE**
August 2002

3. **REPORT TYPE AND DATES COVERED**
Final

4. **TITLE AND SUBTITLE**
Software Architecture Reconstruction: Practice Needs and Current Approaches

5. **FUNDING NUMBERS**
F19628-00-C-0003

6. **AUTHOR(S)**
Liam O'Brien, Christoph Stoermer, Chris Verhoef

7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. **PERFORMING ORGANIZATION REPORT NUMBER**
CMU/SEI-2002-TR-024

9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**
ESC-TR-2002-024

11. **SUPPLEMENTARY NOTES**

12A **DISTRIBUTION/AVAILABILITY STATEMENT**
Unclassified/Unlimited, DTIC, NTIS

12B **DISTRIBUTION CODE**

13. **ABSTRACT (MAXIMUM 200 WORDS)**
Software architectures serve as the blueprints for systems, and they are central to the development of software product lines and the design of component-based systems. In existing systems, the architecture often must be reconstructed to reflect the as-built system accurately. This report presents the concept of practice scenarios for architecture reconstruction, which outline common problem/solution pairs that can be used in the strategic application of architecture reconstruction at Department of Defense (DoD) and commercial organizations. Based on an investigation of already developed and presented reconstruction approaches, the report describes deficiencies that have been uncovered in several practice scenarios and proposes improvements.

14. **SUBJECT TERMS**
architecture reconstruction, practice scenarios, software architecture, software product line

15. **NUMBER OF PAGES**
38

16. **PRICE CODE**

17. **SECURITY CLASSIFICATION OF REPORT**
Unclassified

18. **SECURITY CLASSIFICATION OF THIS PAGE**
Unclassified

19. **SECURITY CLASSIFICATION OF ABSTRACT**
Unclassified

20. **LIMITATION OF ABSTRACT**
UL