

Architectural Recovery of Large Legacy Systems

Samir BOUCETTA
S.Boucetta@ensi.rnu.tn

Henda HADJAMI BEN GHEZALA
Henda.BenGhezala@isd.rnu.tn

Farouk KAMOUN
Farouk.Kamoun@ensi.rnu.tn

PGL Laboratory - Ecole Nationale des Sciences de l'Informatique
Rue des entrepreneurs, 2035 Charguia II - TUNISIE
Tel : +216 1 704 911 , +216 1 704 267 - Fax : ,216 1 706 297

Abstract : Architectural recovery and evolution of legacy systems is a promising research field for the upcoming years. Most of current research works use a bottom-up approach by developing analysis and abstraction techniques of low level knowledge. Few recent works use a combined approach (top-down and bottom up) based on the modeling and the exploitation of domain models. This work proposes an architectural recovery method using a combined approach and based on the modeling and the exploitation of "real world" knowledge. It avoids the arduous and costly domain models building. The method establishes mapping links between "real world" knowledge and the recovered architectural components. These links may be used to better apprehend the system's evolution. This paper describes the method steps and discusses its contributions. Ongoing works and future perspectives are also reported.

Key words : Architectural recovery, systems evolution, reverse engineering, real world knowledge

I. Introduction

Nowadays many legacy systems suffer from maintenance costs and inadequacy to the users' needs. These systems have been maintained for many years by different maintainers and are being difficult to evolve. Their current state prevents them from meeting new users' requirements. Some of these requirements correspond to new functions dictated by business efficiency. Others are prompted by technology explosion. However, this requirements implementation faces many technological handicaps related to the hardware and the software in use. For many organisations, it is unthinkable to throw out the current implementation and redevelop from scratch. It is worth valuable to investigate new technologies to extend the system's time life and permit the "year 2000" (Y2K) transition [2]. It is preferable for this transition to be done according to an enhancement and a migration to a new architecture, thus making advantage from new technologies such as Object Orientation, Web, interoperability and distribution.

Architectural recovery and evolution of legacy systems is a recently research field that have gained much interest these last few years. This is due to the fact that only analysis on the architectural level permits a real evolution of a system [1]. Modifying isolated code fragments does not improve the system's quality attributes¹ [11]. These qualities are visible on the architectural level and can not be reached by individual components analysis.

Many legacy systems do not have a documentation (user's manuals, design documents, etc.). Their architectural recovery may use different information sources which have different reliability degrees. Although program sources are trusted to be most reliable [7], other sources such as the available documentation, the user's and the maintainer's knowledge are sometimes employed. Architectural recovery may be processed in a bottom-up or a combined (top-down+bottom-up) manner. However, a pure top-down process is used for designing new software architectures. Bottom-up approaches start with low level knowledge (program sources, documentation, etc.) and provide techniques to recover a system's architecture.

¹ examples of quality attributes : performance, availability, security, ...

Combined approaches, which start with high level knowledge ("real world" knowledge, domain knowledge, etc.), produce a model of this information and try to find model concepts instances in the system's implementation.

Bottom-up approaches are based on analysis and abstraction techniques of source code [1, 9, 10, 14]. These techniques, such as program slicing, run-time trace analysis, data-flows and control-flows analysis are closely related to human intervention. Automatic recovery of high level abstractions is difficult [3]. For example, automatic recognition of algorithms is complex due to the wide variety of coding techniques and the huge amount of code in which the algorithm may be embedded. In this case, human intervention is necessary to map code fragments to real world and domain concepts.

A combined approach is a two phase process. First, high level knowledge is analyzed and a model is produced. Second, program sources are explored to attempt to find instances of the previous model concepts. Code exploration is then driven by high level concepts recognition. Current research works [3, 5, 6, 12] focus on a particular application domain. A model of the domain is produced and model concepts are searched and located in the source code.

This paper presents a combined approach which makes use of the real world knowledge. This approach avoids arduous and costly domain models construction and can be applied to large legacy systems. It is supported by an information systems meta model which represents the main manipulated concepts.

The following section discusses the dilemma "real world" vs. "domain" model construction. The next two sections outline the meta model and the architectural recovery process. Finally, the main contributions and the perspectives are reported.

II. "Real world" vs. "domain" model construction :

Bottom-up approaches are mainly using source code and rarely make use of the system experts', the domain experts' and the users' knowledge. This information, which exist in human mind, is used informally; generally in the form of interactions with a reverse engineering tool. The user employs his knowledge to guide the reverse engineering process and construct a system's architecture.

Most combined approaches make use of human knowledge by building domain models. The target domain, to which the investigated system pertains, is studied and a model is produced. However, domain models building is difficult and costly [3]. It requires many system experts interviews and calls for "knowledge engineers" participation to organize raw knowledge into concepts and rules. The difficulty is also due to the models genericity and their ability to represent the application domain independently of a given system.

Moreover, a given system may belong to many application domains [3]. This yields that architectural recovery by a combined approach necessitates many domain models construction. For this reason, feasibility of combined approaches was demonstrated on small systems. Jean-Marc DeBaud [6] constructed a "Report Writing" domain model and showed that his method permits architectural recovery and evolution of Report Writing applications. Clements et al. [3] have developed a knowledge-based software assistant prototype called "Gadfly" to support development and comprehension of command, control and communication systems (C3) from a security perspective. They constructed C3 and information security domains, which were used in the reverse/forward engineering task.

Our approach, which aims at architectural recovery of large legacy systems, proposes a rigorous human knowledge exploitation and doesn't advocate domains modeling. This knowledge is unreliable but is in turn of high level of abstraction. In fact, a person may remember a system's function (even obsolete) but doesn't remember easily an implementation detail. This knowledge may correspond to information that appears anywhere or hasn't been updated in the system's documentation. The method

we are presenting consists in knowledge collection and real world concepts identification to make use of this knowledge in the architectural recovery.

The method is supported by an information systems meta model that represents the major manipulated concepts. The meta model is layered into two levels : a real world level and an architectural level. The following section describes the meta model and its concepts.

III. Meta-model presentation

The proposed meta model is described at figure 1 in an object oriented notation. It uses the concepts of relationship, inheritance and aggregation. The real world level summarizes the global system's context. The architectural level describes the main concepts necessary to architectural description.

III.1. Real world level

Real world level corresponds to the information system's view of the real world. In particular, it identifies the organization structure (internal and external **organizational units**²), the organization's **goals** and the **activities** that contribute to their achievement. An organizational unit references an activity when it uses its results. An activity is the responsibility of an internal organizational unit. Activities may be manual or automated. They use **resources** (forms, statements, ...) and generate **products** (processed or annotated documents, printed reports, ...). An activity has **constraints** : a **pre-condition** to trigger the activity and a **post-condition** to control its good execution.

III.2. Architectural level

Jerding [9] defines a Software Architecture as "*a high level program model that describes a system's major pieces (its Components) and how they interact (its Connectors)*". Intuitively, components correspond to boxes, and connectors accord to lines in box-and-line descriptions of software architectures. It is now admitted that architectural descriptions are based on components and connectors [3, 13, 14]. Architectural Description Languages (ADLs) are based, among other concepts, on the description of components and connectors [8].

The meta model reflects this comprehension by including the system's constituents : "**components**" and their interaction modes "**connectors**". It has been indicated that a component may be a **program sequence** or a **data store**. A connector may be uni/bi-directional and concerns a data or a control flow. An **automated activity** is implemented by ordered sequences of program chunks (executable procedures, batch files, etc.). A **product** is implemented by a component.

IV. The architectural recovery method

The method is composed of three main steps. The first step consists in gathering the real world knowledge of the information system. The second, uses the source code to automatically generate a preliminary system's architecture. The generated architecture corresponds to a macroscopic view of the system. The final third step scatters the initial architecture and makes links between the real world knowledge and the initial architecture components. This step allows verification of the information obtained during the first step and dictates the architecture granularity.

² Internal organisational units : departments, services, etc. External organisational units : customers, suppliers, etc.

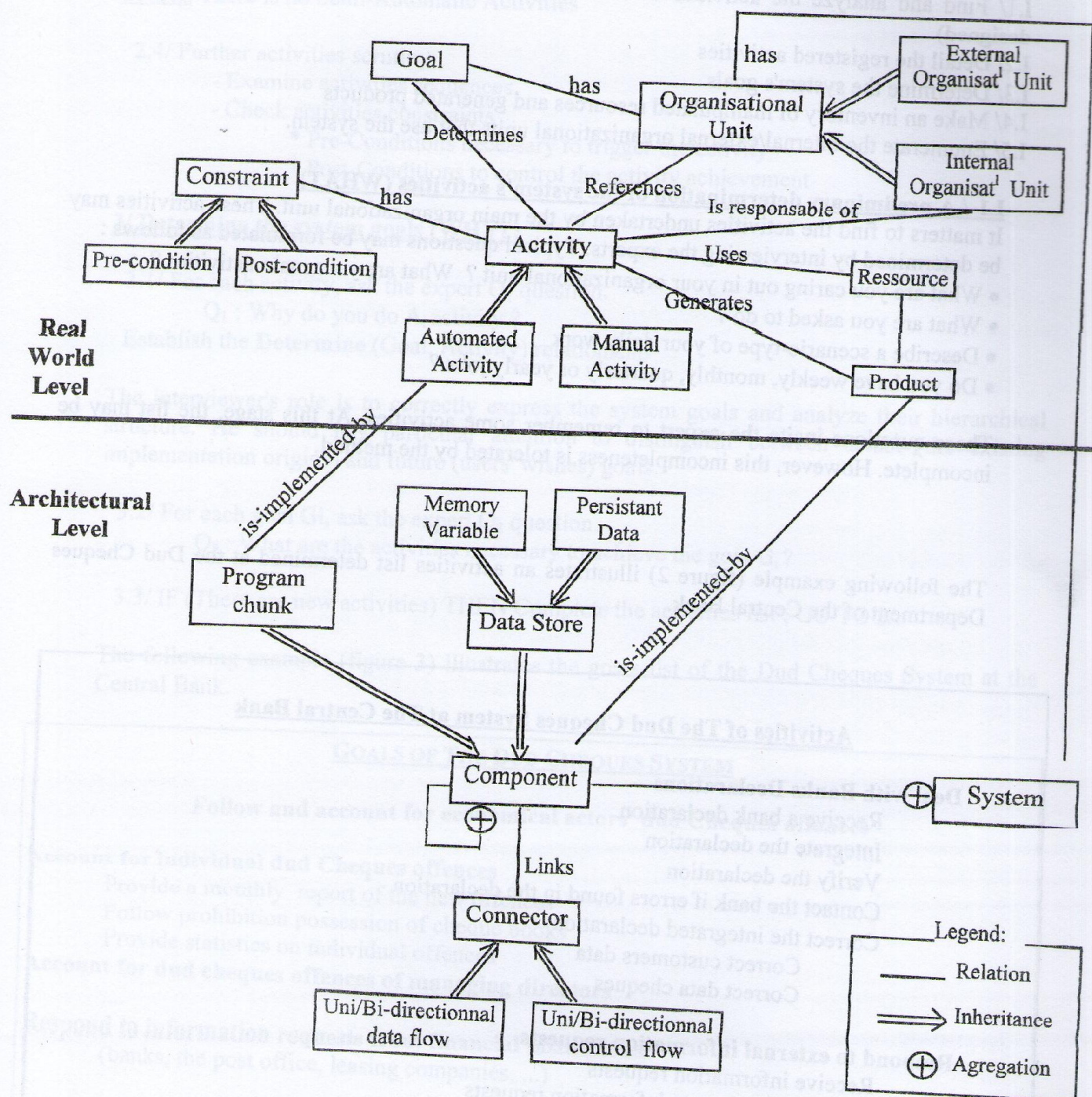


Figure 1 : The proposed Meta-Model

IV.1. First step : Real world model construction

This step is driven by interviews with the system experts, the users and the domain experts. A system expert corresponds to a person who has participated in the system design, development or maintenance. A user is a person that frequently makes use of the system to carry out his job. A domain expert refers to a person that masters the system's business rules. For paper's need, we refer to one of the persons above by the "expert" qualifier.

This first step is composed of five sub-steps as follows :

- I.1/ Find and analyze the activities of the main organizational unit (for whose the system is designed)
- I.2/ Detail the registered activities
- I.3/ Determine the system's goals
- I.4/ Make an inventory of manipulated resources and generated products
- I.5/ Enumerate the internal/external organizational units that use the system.

I.1-/ A preliminary determination of the system's activities (WHAT)

It matters to find the activities undertaken by the main organizational unit. These activities may be determined by interviewing the experts. Typical questions may be formulated as follows :

- What are you caring out in your organizational unit ? What are your main activities ?
- What are you asked to do ?
- Describe a scenario-type of your daily work
- Do you have weekly, monthly, quarterly or yearly jobs ?

These questions incite the expert to remember some activities. At this stage, the list may be incomplete. However, this incompleteness is tolerated by the method.

The following example (figure 2) illustrates an activities list determined at the Dud Cheques Department of the Central Bank.

Activities of The Dud Cheques System at The Central Bank

Deal with Banks Declarations

- Receive a bank declaration
- Integrate the declaration
- Verify the declaration
- Contact the bank if errors found in the declaration
- Correct the integrated declaration
 - Correct customers data
 - Correct data cheques

Respond to external information requests

- Receive information requests
- Prepare responses to information requests
- Prepare the mail

Figure 2 : Activities of the Dud Cheques System at the Central Bank

2-/ Detail the determined activities (HOW) :

REPEAT

{for each activity}

- 2.1/ Make an abstract³ description of the activity

³ The activity description should not reflect implementation details such as file, database, report or program/module names. Example: "Receive the floppy disk containing file FDECLAR.DAT" should be replaced by "Receive declarations"

2.2/ Determine the activity nature : Manual, Automatic, Semi-Automatic⁴

2.3/ IF (Activity IS Semi-Automatic) THEN Decompose the activity

UNTIL There is no Semi-Automatic Activities

2.4/ Further activities scrutiny

- Examine activities sequences
- Check activities constraints
 - * Pre-Conditions necessary to trigger the activity
 - * Post-Conditions to control the activity achievement

3/ Determine the system goals (WHY) :

3.1/ For each activity, ask the expert Q₁ question.

Q₁ : Why do you do A_i activity ?

Establish the **Determine** (Goal, Activity) relationship

The interviewer's role is to correctly express the system goals and analyze their hierarchical structure. He should pay particular attention to distinguish between actual (the existing implementation origins) and future (users' wishes) goals.

3.2/ For each goal G_i, ask the expert Q₂ question.

Q₂ : What are the activities necessary to achieve the goal G_i ?

3.3/ IF (There are new activities) THEN Complete the activities list ; GO TO 2/.

The following example (figure 3) illustrates the goals list of the Dud Cheques System at the Central Bank.

GOALS OF THE DUD CHEQUES SYSTEM

Follow and account for economical actors' dud Cheques offences

Account for individual dud Cheques offences

Provide a monthly report of the new offences

Follow prohibition possession of cheque books

Provide statistics on individual offences

Account for dud cheques offences of managing directors

....

Respond to information requests from financial institutions

(banks, the post office, leasing companies, ...)

Support new cheque books agreement

Figure 3 : Goals of the Dud Cheques System at the Central Bank

4/ Initial system manipulated resources determination

This step consists in gathering (in bulk) the resources manipulated by the main organizational unit. Each resource is linked to the activity that uses or generates it. At this point, some activities may be detected as they were previously forgotten.

⁴ A semi-automatic activity is composed of one or many manual activities and one or many automated activities

- 4.1/ Determine the manipulated resource list
- 4.2/ For each resource
 - 4.2.1/ find the activities that uses it (**Uses**(Activity, Resource) relationship)
 - 4.2.2/ find the activity that generates it (**Generate**(Activity, Resource) relationship)
- 4.3/ IF (There are new activities) THEN Complete the activities list ; GO TO 2/.

5/ System organizational units determination

Responses to the following questions allow the determination of organizational units that have a relationship with the prospected system. An example is presented at figure 4.

- 5.1/ For each activity, ask the expert Q_1 question.
 Q_1 : Who is driving/carrying A_i activity ?
- 5.2/ For each "product" resource, ask the expert Q_2 question.
 Q_2 : Who needs P_i product ?
- 5.3/ For each organizational unit, identify its nature : Internal or external.

Q_1 question establishes the **is-responsible-of** (organizational unit, activity) relationship
 Q_2 question establishes the **references** (organizational unit, activity) relationship

The knowledge collected during this step will be validated later by confrontation to source code and mapping links to the architectural components.

ORGANIZATIONAL UNITS REQUESTING THE DUD CHEQUES SYSTEM OF THE CENTRAL BANK	
Financial institutions (banks, the post office, leasing companies, ...)	External
Justice ministry (Law courts, ...)	External
The treasury	External
Dud cheques department	Internal
Central information department	Internal
Economical information department	Internal
Bank Inspections department	Internal
Banks department	Internal
Refinancing department	Internal

Figure 4 : Organizational Units requesting the Dud Cheques System of the Central Bank

IV.2. Second step : Preliminary architecture of the system

This step aims at generating automatically, an initial architecture of the system (components, connectors). Automatic decomposition is done according to the following rules :

Initial components are :

- Program sources : Separate program files

• **Data files** : Their list is determined by an automatic examination of program sources. Generally, this analysis concerns declarative sections. For example, in COBOL programs, examination concerns the INPUT-OUTPUT SECTION. For PASCAL language, the analysis deals with TYPE and VAR declarations in all programs and sub-programs. However, in some cases, analysis of all the source code is needed, especially with languages that do not require preliminary declarations or when external calls (intrinsic) are used.

Initial connectors are :

Connector	Linked Components
program call	Program-Program
Declaration	Program-File
Manipulate	Program-File

At the end of this step, the component list has a low level granularity. The final third step allow component refinement.

IV.3. Third step : Architecture refinement

This step constructs the matrix of the figure 5. Lines contain components that correspond to program sequences and columns list the automated activities. A checked cell means that the component C_i is needed to achieve the A_i activity.

Is implemented by Implement	Activity 1	Activity 2	Activity i	Activity n
Component 1	X	X		
Component 2		X		X
Component i				
Component n	X			

Figure 5 : Components/activities matrix

Matrix analysis allows determination of :

- Components that implement many real world activities
- Components that do not correspond to any real world activity
- Real world activities implemented by many components
- Real world activities that are not implemented by any component

The refinement stage consists in :

- Decomposing components that implement many activities
- Examining components that do not correspond to any of the real world activities
- Check if they do not implement a non listed activity(ies). If so, update the activities list
- Check if they do not correspond to obsolete functions or dead code. If so, eliminate the component.
- Inspecting real world activities that are implemented by zero components
- Check the activity nature (Manual, Automatic)

The refinement process is stopped when :

- A real world activity is implemented by one or many components
- A component does implement only one real world activity
- There doesn't exist any component corresponding to zero activities
- There doesn't exist any activity implemented by zero components

V. The method contributions

The main contribution of this architectural reconstruction is to allow reasoning about the system at a high level of abstraction. The resulting architecture permits change impact analysis and can be used to better apprehend the system's evolution.

The relationships ("determines", "generates", etc.) between real world concepts, and mapping links ("is-implemented-by" relationships) between real world concepts and system's components are the key features that permit to respond to questions such as :

- Which activities contribute to goal G_i satisfaction ?
- If goal G_i evolve to goal G_j , what should be updated in the system's implementation ? What are the involved automated activities ?
- If the content and/or the quality of the product P_i change, what are the required code source modifications ?

In these cases, mapping links provide the components and connectors lists associated with the implicated activities.

Moreover, it is possible to reason about the manual activities and plan their automation. Besides, the recovered architecture may be used to better apprehend the implementation of Business Process Reengineering (BPR). BPR solutions [4] influence considerably the activities natures and sequences, and affects the content and quality of the resources/products. This yields that some activities, resources or products may be created, deleted or redesigned. Relationships and mapping links, resulting from the application of our method, can support this reorganization effort. Architectural evolution is then done according to organisational needs.

VI. Conclusion :

In this paper, we have presented a reverse engineering method that enables architectural recovery of large legacy systems. This method makes a rigorous use of human knowledge by real world model construction and avoid the arduous and costly domain model building. The method is based on an information systems meta model, layered into two levels and which summarizes the main manipulated concepts. The method and its contributions were presented.

A first experimentation of our method was driven manually at the Central Bank of Tunisia, and had proven the method usefulness. The results of the experimentation were as follows :

- An architectural description of the prospected system
- a list of old programs which were no more used
- many versioning problems identification
- and many dead code elimination

We plan to develop a tool that proceeds to source code analysis, automatic generation of the initial architecture, and graphical visualisation. We are investigating techniques to aid in program slicing, to be used during the third step, when a component do implement many real world activities.

References :

- [1] G. Abowd, A. Goel, D. F. Jerding, M. M. McCracken, M. Moore, J. William Murdock, C. Potts, S. Rugaber, L. Wills, "MORALE Mission Oriented Architectural Legacy Evolution", Proceedings of the International Conference on Software Maintenance'97, Bari, Italy, September 29-October 3, 1997.

- [2] John K. Bergey, Linda M. Noethrop, Dennis B. Smith, "Entreprise Framework for the Disciplined Evolution of Legacy Systems", Technical Report (CMU/SEI-97-TR-007), Pittsburg, PA: Software Engineering Institute, Carnegie Mellon University, October 1997.
- [3] Paul Clements, Robert Krut, Ed Morris, Kurt Wallnau, "The Gadfly : An Approach to Architectural-Level System Comprehension", Proceeding of IEEE 4th International Conference on Program Comprehension, Berlin 1996.
- [4] T. H. Davenport, J. E. Short, "The New Industrial Engineering: Information Technology and Business Process Redesign", Sloan Management Review, Summer 1990, pp. 11-27.
- [5] Jean-Marc DeBaud and Spencer Rugaber, "A Software Re-Engineering Method using Domain Models", Proceedings of the International Conference on Software Maintenance, Nice, France, October 1995, pp. 204-213.
- [6] Jean-Marc DeBaud, "Lessons from a Domain-Based Reengineering Effort", Proceedings of the Working Conference on Reverse Engineering, WCRE'96, IEEE, Monterey, California, November 1996.
- [7] M. J. Freeman, P. J. Layzell, "A Meta-Model of Information Systems to Support Reverse Engineering", Information and Software Technology, Vol. 36, N°5, May 1994, pp. 283-294.
- [8] D. Garlan, R. T. Monroe, D. Wile, "ACME: An Architecture Description Interchange Language", Proceedings of CASCON'97, November 1997.
- [9] Dean Jerding and Spencer Rugaber, "Using Visualization for Architectural Localization and Extraction", Proceedings of the fourth Working Conference on Reverse Engineering, IEEE Computer Society, Amsterdam, the Netherlands, October 6-8, 1997, pp. 56-65.
- [10] Rick Kazman, Jeromy S. Carrière, "Playing Detective: Reconstructing Software Architecture from Available Evidence", Technical Report CMU/SEI-97-TR-010, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, October 1997.
- [11] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carrière, "The Architecture Tradeoff Analysis Method", Technical Report (CMU/SEI-98-TR-008, ESC-TR-98-008), Pittsburg, PA: Software Engineering Institute, Carnegie Mellon University, July 1998.
- [12] Melody M. Moore, Spencer Rugaber, "Domain Analysis for Transformational Reuse", Proceedings of the fourth Working Conference on Reverse Engineering, WCRE'97, IEEE, 1997.
- [13] Mary Shaw and David Garlan, "Software Architecture/Perspectives on an Emerging Discipline", Prentice Hall, 1996.
- [14] P. Tonella, R. Fiutem and G. Antoniol, "Augmenting Pattern-Based Architectural Recovery with Flow Analysis : Mosaic - A Case Study", Proceedings of the Working Conference on Reverse Engineering, WCRE'96, IEEE, 1996.