SArEM: A SPEM extension for software architecture extraction process

Mira Abboud
LaMA Laboratory
Lebanese University
Tripoli, Lebanon
And
LINA Laboratory
University of Nantes
Nantes, France
mira.abboud@univ-nantes.fr

Hala Naja
LaMA Laboratory
Lebanese University
Tripoli, Lebanon
hjazzar@ul.edu.lb

Mourad Oussalah
LINA Laboratory
University of Nantes
Nantes, France
Mourad.oussalah@univ-nantes.fr

Mohamad Dbouk
Lebanese University
Tripoli, Lebanon
mdbouk@ul.edu.lb

***Abstract*—In order to maintain a system, it's critical to understand its architecture. However even though every system has an architecture, not every system has a reliable representation of its architecture. To deal with this problem many researchers have engaged in software architecture extraction where the system's architecture is recovered from its source code. While there is a plethora of approaches aiming at extracting software architectures, there is no mean or tool measurement for these approaches; which makes the comparison between the different approaches a hard task. To tackle this lack, we developed a meta-model, based on SPEM meta-model, that specifies the software architecture extraction process. Such meta-model serves as a tool to compare, analyze and evaluate research field approaches. In this paper we detail our meta-model called SArEM (Software Architecture Extraction Meta-model) and clarify its concepts.**

Keywords-software architecture extraction; software architecture reconstruction; reverse engineering; SPEM extension.

I. INTRODUCTION

A software architecture is a bridge between the problem *(i.e. the requirements)* and the solution *(i.e. the implementation)*. It covers all the significant decisions that lead to the system construction. Formally, a software architecture is defined as the system structure(s), which comprises software elements, the externally visible properties of those elements, and the relationships among them [1]. As every system can be shown to be composed of elements and relations among them, every software system has an architecture.

In [2], Garlan reveals the importance of software architectures by projecting its role on six different aspects: understanding, reuse, construction, evolution, analysis and management. However, for many systems, the representation of their architectures is not reliable, due to two reasons: The first is the lack of documentations. Indeed, this is partly due to the fact that current-day development environments still focus too much on writing code and too little on designing and documenting tasks. The second reason is the architectural erosion. In fact, during the different evolution phases the system deviates from its original representation, causing a gap between the current system architecture and the documented architecture. So no reliable software architecture representation is available in the both cases.

In order to avoid the increasing of non-synchronization between current system architecture and its representation, several techniques and processes aiming at extracting software architectures have been proposed. Each approach has its own requirements, method, automation level, solution, limits and other properties. However, while there is an excess of approaches and technologies that support software architecture extraction, there is no reference or standard that allows their evaluation. This lack makes the selection of an extraction process a difficult activity for the architect.

To solve this problem, a comprehensive and effective comparison should be proposed. Such comparison must be based on several criteria that reflect the important elements of an extraction process. Thus, we propose a meta-model, called SArEM (Software Architecture Extraction Meta-model), that represents the basic concepts of software architecture extraction process in terms of entities and relationships. The rest of the paper is organized as follows: section II presents some related works on software architecture extraction. Section III provides the concepts on which SArEM is based, more specifically it describes the SPEM meta-model and its adaptation to the extraction processes. In section IV, our meta-model SArEM is detailed. And finally section V concludes with an overview of the major contributions of this paper.

II. RELATED WORKS

Several approaches have been proposed to extract system's architecture. Some propose a tool that supports the extraction and others propose a whole process that is human derived and tools supported.

A. Software architecture extraction tools

The most known tools that support software architecture extraction are the source code analyzers that parse the source code to represent the software structure at different levels of abstraction. We mention: CPP2XMI [3] which extracts the information from the C++ source code in UML diagrams like class, sequence and activity diagrams. Same for the reverse engineering framework Columbus [4] that analyzes large C++ projects and generates a schema that prescribes the form of the extracted data. Ptidej [5] builds class diagrams from Java source code. Rational Rose [6] supports reverse engineering of C++ and Java software systems. The output is the class diagram with the attributes, functions and variables. Add to this, there are many tools that extract source code information as graphs like the MDG graph. MDG is a model dependency graph such as the graph nodes represent the modules and the edges represent the dependencies between them. We could mention *Chava* [7] for the analysis of Java programs and ACASIA [8] for C++ code analysis.

There is many tools that support the software architecture extraction by providing a way to group source entities, for example: The *Bunch* tool proposed by Mancoridis et al. [9]–[14]. In fact, Mancoridis et al. introduce the extraction problem as a module clustering problem where search-based optimization algorithms can be applied. This idea led to the development of a tool called *Bunch* that supports a variety of search algorithms. The tool uses an MDG graph as an input, applies an algorithm and finds the best partition of the MDG according to a fitness function. Maqbool et al. propose a new algorithm for clustering finding inter-cluster instance during the clustering process [15].

Among the various way of software architecture extraction, matching tools are one of the ways: Sartipi [16], [17] considers the problem of extraction an optimal matching problem between two graphs: the pattern graph which represents the conceptual architecture proposed by the architect and the source graph which represents the system source code. The *reflexion model* tool [18], [19], which inspires many other approaches, is based on the elaboration of a model that computes the differences between the developers high-level model and the recovered model.

Since the design patterns reflect a design decision that has been made so they are a point of interest for software architecture extraction. Many tools support design pattern detection, one of them is done by Kraemer and Prechelt [20]. The tool works on design pattern detection from C++ code files. The approach uses PROLOG rules to detect the design pattern instance (the structural pattern) from a depository that contains the information. Also, MARPLE [21] supports design pattern detection and software architecture reconstruction.

B. Software architecture extraction processes

After studying a large number of research works, we could classify the extraction processes under two categories according to the techniques used.

The first category covers the processes that are based on the clustering technique. In these processes the architecture is extracted in terms of components such as each component is a group of related system entities. Claudio Riva has contributed in this field by proposing an approach [22] that extracts the software architecture of Nokia products, it is based on the principle of abstraction in order to pass from the source code level to a higher level. The process is composed of six phases: (1) the definition of architectural concepts; (2) the extraction of the source code model which contains entities that characterize the implementation; (3) the abstraction which's the grouping of the source model entities by using clustering techniques; and the remaining phases improve the architecture documents and identify architectural flaws. Many processes are based on *Bunch* tool which's a clustering tool: The IGA approach [23] integrates human interpretations in the extraction process. The genetic algorithm is executed iteratively and at each iteration new constraints given by the developers are added. Mahdavi et al. approach [24], which's also supported by *Bunch* tool, shows that combining the results from multiple hill climbs can improve the results for simple hill climbing and genetic algorithm.

The second category is based on the matching techniques. In such processes, the architecture is the result of a mapping between a high-level architecture and the source code. Among the processes that belong to this category, we mention the *reflexion model* process [18], [19], [25] which's based on the following steps: the definition of a high-level architecture, the extraction of a source model from the source code, a definition of a map between the two models, a computation of a reflexion model using the *reflexion model* tool and, the last step is the interpretation of the reflexion model by the reverse engineer. The output of the process is a reflexion model that highlights the convergences, divergences, and absences between the two models. Koschke [26] extends the original reflexion model to hierarchical architecture models. His extension allows the engineer to decompose his conceptual architecture at different level of details. Also, Bowman et al. approach [27] is based on the matching between a conceptual architecture and an extracted architecture. Bowman proposes the use of (1) an analysis tool, (2) a structuring tool to extract the relationships between elements based on the implicit relationships between them and (3) a visualization tool. Kazman et .al propose a tool called Dali [28] that integrates several techniques like the Rigi tool [29] and the POSTGREDSQL query tool. These tools allow the extraction of the source code, the registration of the results in a relational database, the execution of queries given by the reverse engineer which selects source entities according to the query and finally Rigi allows result visualization. In order to reconstruct the architecture in terms of patterns, Guo proposes a semi-automatic analysis method called ARM [30]. ARM is based on 4 phases that aim to (1) develop a recognition plan, (2) extract a model that represents the source code using *Dali* tool, (3) execute the plan on the source model also using *Dali* and (4) analyze the results using visualization tools.

Other processes could be classified under the two categories. Focus approach [31] takes into account the architectural concepts such as architectural style; it consists of six phases: (1) Components identification which's established by generating class diagrams, grouping related classes and packaging the classes using clustering tools; (2) Idealized architectural model proposition which's the creation of a conceptual architecture that respects an architectural style; (3) Mapping the components onto the idealized architecture; (4) Use case identification; (5) Components interactions analysis and (6) refined architecture generation. The approach proposed by Medvidovic [32] is based on Focus approach; it aims to combine techniques used to discover the architecture from functional requirements with techniques that extract an architecture model from the implementation.

III. SPEM

SPEM [33] (Software & Systems Process Engineering Metamodel Specification) is a meta-model that specifies the process engineering for system construction. It identifies the basic concepts of the software engineering processes in terms of elements and relationships. Three elements are defined in SPEM: The activities, the work product and the roles. An activity is a unit of work that must be realized in order to achieve the process goal. A work product is a concrete object that can be manipulated. And finally, a role represents a skill. Also, SPEM defines 4 relationships: The 'uses', 'produces', 'performs' and 'is responsible for' relationships. The 'uses'/'produces' relationships indicate that a product can be used/produced by an activity. The 'performs' relationship indicates that an activity is performed by a role. And, the 'is responsible for' indicates that a role is responsible for an artefact.

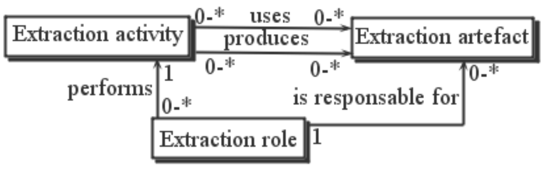

Figure 1: A meta-model based on SPEM.

While SPEM specifies software engineering processes, we believe that it can also specify software architecture extraction processes. Since an extraction process can be seen as a collaboration between activities, artefacts and roles, SPEM could be used as a meta-model for software architecture extraction modeling process. For this end, we propose the first meta-model presented in Figure 1 which reflects that a software architecture extraction process is a collaboration between extraction activities, extraction artefacts and extraction roles. We define an 'extraction activity' as a work task performed to meet the extraction process, the 'extraction artefact' as a concrete object that can be manipulated by an extraction activity during the extraction process, and an 'extraction role' as a skill able to execute an extraction activity. An extraction role performs an extraction activity and is responsible for the manipulated artefacts through the extraction activity.

IV. SArEM

Since the first meta-model, depicted in Figure 1, does not give specific details on software architecture extraction, several entities dedicated to the software architecture extraction must be added. For this, we expand the 3 elements of the first meta-model. An element expansion is realized by adding sub-elements that give more details.

A. Types of 'Extraction artefacts'

Figure 2 shows SArEM extraction artefacts metamodel. We distinguish between 4 artefact types: the initial artefact, the intermediate artefact, the pseudo-architecture and the final artefact. The initial artefact exists before the extraction process startups. The intermediate artefact are manipulated throughout the process. The pseudo-architecture is the first generated architecture and the final artefact is the final artefact constructed by the entire process.

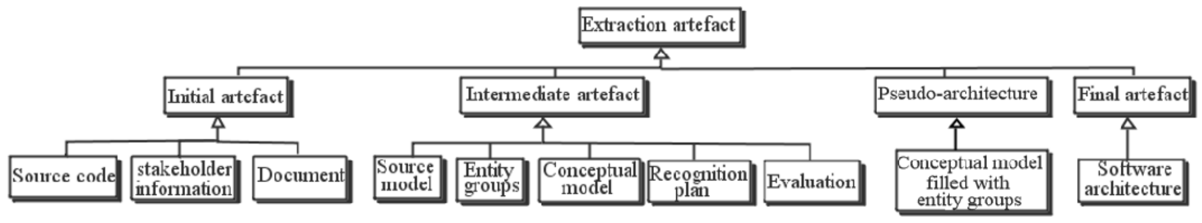

Figure 2: SArEM extraction artefacts.

1) Types of 'initial artefacts'

The initial artefact concept is extended by specifying several concrete artefacts that are itemized as follows:

- Source code: the system implementation.
- Stakeholder information: an idea acquired by the stakeholders through their knowledge about the system. The integration of this artefact is a good initiative to obtain a rewarding process that satisfies the stakeholders.
- Documents: software architecture description. Even if a gap exists between the documented and implemented architecture, there are still common traces that an extraction process can take advantages of.

2) Types of 'intermediate artefacts'

The intermediate artefact concept is extended by adding several concrete artefacts.

- Source model: a source code representation. It represents the source code in terms of entities and relationships. For example, a source model could be a class diagram or it could be a graph that represents the system files and their relationships.
- Entity groups: a partition of the source model such as each group of the partition is a set of related source model entities. This artefact can be considered as an abstraction of the source model.
- Conceptual model: a high-level architecture that represents the stakeholders view about the system functional aspect. It's symbolized in terms of blocks and relationships between them.
- Recognition plan: a map that shows how the functional aspects are realized in the entity groups. It is used as a plan to council the entity groups with a conceptual model.
- Evaluation: a representation of the stakeholders point of view. It expresses whether the pseudo-architecture satisfies them or not.

3) Types of 'pseudo-architectures'

The pseudo-architecture artefact is extended by adding the element 'Conceptual model filled with entity groups' which represents the entity groups assigned to the conceptual model according to the recognition plan.

4) Types of 'final artefacts'

The final artefact is extended by adding the element 'software architecture' which represents the system in terms of architectural elements and their interactions.

B. Types of 'extraction activities'

Extraction activities are work tasks performed throughout the process. As shown in Figure 3, these activities can be categorized under two types: Analysis activities and synthesis activities.

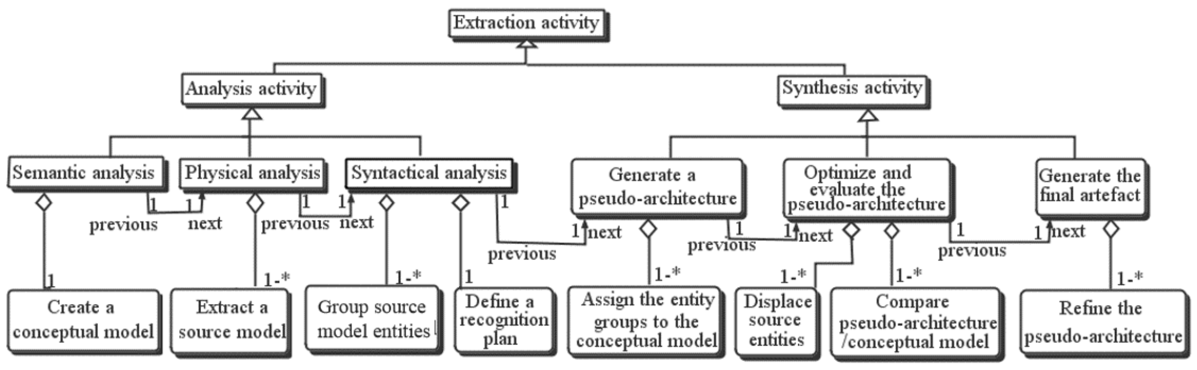
Extraction activity
Analysis activity
Synthesis activity
Semantic analysis
Physical analysis
Syntactical analysis
Generate a pseudo-architecture
Optimize and evaluate the pseudo-architecture
Generate the final artefact
previous
next
Create a conceptual model
Extract a source model
Group source model entities
Define a recognition plan
Assign the entity groups to the conceptual model
Displace source entities
Compare pseudo-architecture /conceptual model
Refine the pseudo-architecture
1
1-*

Figure 3: SArEM extraction activities.

1) Types of 'analysis activities'

An analysis activity studies an artefact, examines it in detail in order to conclude significant information useful in the pseudo-architecture generation. Usually, an analysis activity uses the initial artefacts, analyses them and produces intermediate artefacts. This activity type is extended by adding the following concepts: the semantic analysis which studies the meanings and concepts that occur in the initial artefacts in order to produce a system functional aspects representation; the physical analysis which focuses on the study of the source code to extract significant information from the code; and finally the syntactical analysis which specifies how the information at the source code level can realize the system functional aspects. Moreover, we detail the analysis activities concept by specifying more concretely the steps of each activity type:

- Create a conceptual model: an analysis of the documents and stakeholders information in order to provide a high-level system architecture in terms of blocks and relationships, called conceptual model.
- Extract a source model: a source code analysis that aims to extract a source model. It studies the source code to extract meaningful information at a higher level than the source code.
- Group source model entities: an analysis of the relations between the source model entities that aims to group the related entities. Thus, this analysis uses the source model and produces the entity groups.
- Define a recognition plan: a study of the relations that may exist between the entity groups and the functional aspects. In other words, it indicates how the conceptual model is realized in the source entities. It uses the entity groups and the conceptual model to generate the recognition plan that serves as a way of correspondence between them.

2) Types of 'synthesis activities'

A synthesis activity is the combination of several elements into a single element. It could be one of the following three activities: 'Generate a pseudo-architecture' which is a pseudo-architecture generation based on the analysis results; it uses the intermediate artefacts to produce a first architecture. 'Optimize and evaluate the pseudo-architecture' which is a pseudo-architecture evaluation and optimization. 'Generate the final artefact' which is a generation of the final desired artefact by using the pseudo-architecture and its evaluation. Moreover, we detail the synthesis activities concept by identifying more concretely the steps of each activity type:

- Assign the entity groups to the conceptual model: an integration of the conceptual model with the entity groups according to the recognition plan.
- Displace source entities: a displacement of a source entity from a conceptual model block to another.
- Compare the pseudo-architecture to the conceptual model: a comparison of the pseudo-architecture with the conceptual model to indicate whether the pseudo-architecture meets the conceptual model proposed by the stakeholders or not.
- Refine the pseudo-architecture: the refinement of the architecture according to the evaluation.

C. Types of 'Extraction roles'

The extraction roles represent the skills and knowledge that perform the extraction activities. They are specified by identifying two role types: the stakeholder and the technique. The stakeholders represent all those having an interest toward the extraction process. A technique is a tool or an algorithm able to execute an operation. Many techniques could be integrated in the extraction process; we classify them according to their role as follows:

- Extraction tool: a tool that performs the source model extraction activity. It uses the source code as an input and creates the source model as an output.
- Mapping tool: a role that takes the entity groups, the recognition plan and the conceptual model as input and assigns the groups to the conceptual model according to the plan.
- Search algorithm: a tool that performs the entities grouping activity according to many criteria. Usually the algorithm uses a fitness function that leads the algorithm according to the desired criteria.
- Visualization tool: a tool that transforms a pseudo-architecture in terms of entities, blocks and relationships to a graphical representation.

D. SArEM interactions

In a similar way to SPEM, the different entities of our meta-model interact with each other: The extraction activities use and produce the extraction artefacts and are performed by the extraction roles. Figures 4, 5 and 6 present some portions of SArEM meta-model.

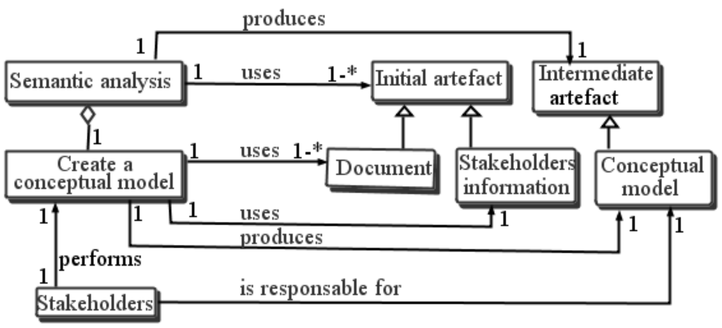

Figure 4: SArEM semantic analysis meta-model.

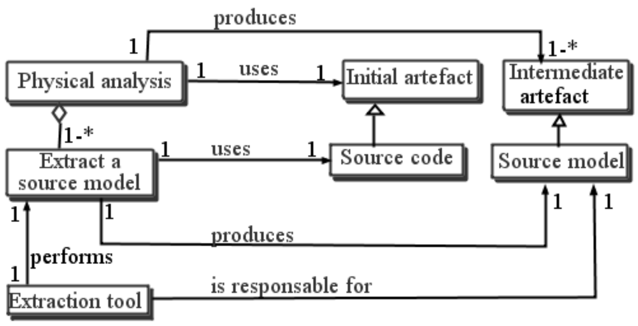

Figure 5: SArEM physical analysis meta-model.

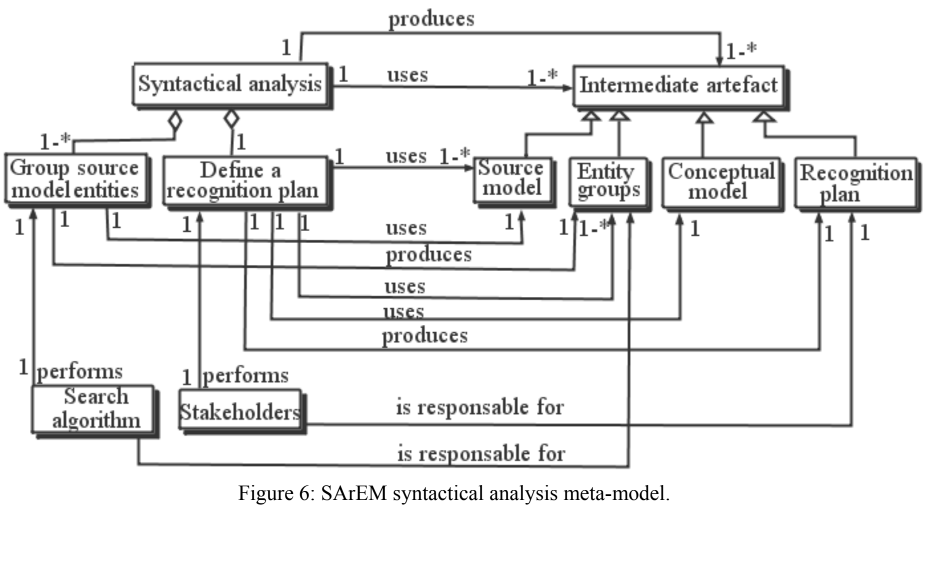

Figure 6: SArEM syntactical analysis meta-model.

V. CONCLUSION

This paper has presented a meta-model, called SArEM, for software architecture extraction. SArEM is centered on the following 2 concepts: First, it's based on SPEM meta-model by respecting its three fundamental principles: work products, activities and roles. Second, it extends each SPEM principle in order to identify a software extraction process concepts. SArEM does not aim to provide its own software architecture extraction process, rather SArEM define the ability for an architect to choose the extraction process that fits his needs. Furthermore, SArEM could be considered as a starting point to compare, analyze and evaluate existing approaches for software architecture extraction.

After the definition of the important concepts that an extraction process should be based on, we aim to define a practical tool that respects all SArEM concepts. Our future work is then to define a tool that meets all the elements of SArEM in terms of activities, artifacts and roles.

REFERENCES

[1] B. Len, C. Paul, and K. Rick, "Software architecture in practice," Boston Mass. Addison, 2003.

[2] D. Garlan, "Software architecture: a roadmap," in Proceedings of the Conference on the Future of Software Engineering, 2000, pp. 91–101.

[3] E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi, "CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code," in 13th Working Conference on Reverse Engineering, 2006. WCRE '06, 2006, pp. 297–298.

[4] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy, "Columbus-reverse engineering tool and schema for C++," in Software Maintenance, 2002. Proceedings. International Conference on, 2002, pp. 172–181.

[5] Y.-G. Guéhéneuc, "A reverse engineering tool for precise class diagrams," in Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, 2004, pp. 28–41.

[6] T. Quatrani, Visual modeling with rational rose 2002 and UML. Addison-Wesley Longman Publishing Co., Inc., 2002.

[7] J. Korn, Y.-F. Chen, and E. Koutsofios, "Chava: Reverse engineering and tracking of java applets," in Reverse Engineering, 1999. Proceedings. Sixth Working Conference on, 1999, pp. 314–325.

[8] Y.-F. Chen, E. R. Gansner, and E. Koutsofios, "A C++ data model supporting reachability analysis and dead code detection," Softw. Eng. IEEE Trans. On, vol. 24, no. 9, pp. 682–694, 1998.

[9] D. Doval, S. Mancoridis, and B. S. Mitchell, "Automatic clustering of software systems using a genetic algorithm," in Software Technology and Engineering Practice, 1999. STEP'99. Proceedings, 1999, pp. 73–81.

[10] S. Mancoridis, B. S. Mitchell, C. Rorres, Y.-F. Chen, and E. R. Gansner, "Using Automatic Clustering to Produce High-Level System Organizations of Source Code.," in IWPC, 1998, vol. 98, pp. 45–52.

[11] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures," in Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, 1999, pp. 50–59.

[12] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," Softw. Eng. IEEE Trans. On, vol. 32, no. 3, pp. 193–208, 2006.

[13] B. S. Mitchell and S. Mancoridis, "Using Heuristic Search Techniques To Extract Design Abstractions From Source Code.," in GECCO, 2002, vol. 2, pp. 1375–1382.

[14] B. S. Mitchell and S. Mancoridis, "On the evaluation of the Bunch search-based software modularization algorithm," Soft Comput., vol. 12, no. 1, pp. 77–93, 2008.

[15] O. Maqbool and H. A. Babri, "The weighted combined algorithm: A linkage algorithm for software clustering," in Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on, 2004, pp. 15–24.

[16] K. Sartipi, "Software architecture recovery based on pattern matching," in Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, 2003, pp. 293–296.

[17] K. Sartipi and K. Kontogiannis, "Pattern-based software architecture recovery," in Proceedings of the Second ASERC Workshop on Software Architecture, 2003.

[18] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: Bridging the gap between source and high-level models," ACM SIGSOFT Softw. Eng. Notes, vol. 20, no. 4, pp. 18–28, 1995.

[19] G. C. Murphy, D. Notkin, and K. J. Sullivan, "Software reflexion models: Bridging the gap between design and implementation," Softw. Eng. IEEE Trans. On, vol. 27, no. 4, pp. 364–380, 2001.

[20] R. K. Keller, R. Schauer, S. Robitaille, and P. Pagé, "Pattern-based Reverse-engineering of Design Components," in Proceedings of the 21st International Conference on Software Engineering, New York, NY, USA, 1999, pp. 226–235.

[21] F. Arcelli Fontana and M. Zanoni, "A tool for design pattern detection and software architecture reconstruction," Inf. Sci., vol. 181, no. 7, pp. 1306–1324, Apr. 2011.

[22] C. Riva, "Reverse architecting: an industrial experience report," in wcre, 2000, p. 42.

[23] Rajalakshmi, M, "Software System Re-Modularization using Interactive Genetic Algorithm," vol. 3, no. 4, pp. 105–107, Apr. 2014.

[24] K. Mahdavi, M. Harman, and R. Hierons, "Finding building blocks for software clustering," in Genetic and Evolutionary Computation—GECCO 2003, 2003, pp. 2513–2514.

[25] G. C. Murphy and D. Notkin, "Reengineering with reflexion models: A case study," Computer, vol. 30, no. 8, pp. 29–36, 1997.

[26] R. Koschke and D. Simon, "Hierarchical reflexion models," in null, 2003, p. 36.

[27] I. T. Bowman, R. C. Holt, and N. V. Brewster, "Linux As a Case Study: Its Extracted Software Architecture," in Proceedings of the 21st International Conference on Software Engineering, New York, NY, USA, 1999, pp. 555–563.

[28] R. Kazman and S. J. Carrière, "Playing detective: Reconstructing software architecture from available evidence," Autom. Softw. Eng., vol. 6, no. 2, pp. 107–138, 1999.

[29] S. R. Tilley, K. Wong, M.-A. D. Storey, and H. A. Müller, "Programmable reverse engineering," Int. J. Softw. Eng. Knowl. Eng., vol. 4, no. 04, pp. 501–520, 1994.

[30] G. Y. Guo, J. M. Atlee, and R. Kazman, A software architecture reconstruction method. Springer, 1999.

[31] L. Ding and N. Medvidovic, "Focus: A light-weight, incremental approach to software architecture recovery and evolution," in Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on, 2001, pp. 191–200.

[32] N. Medvidovic, A. Egyed, and P. Gruenbacher, "Stemming Architectural Erosion by Coupling Architectural Discovery and Recovery.," in STRAW, 2003, vol. 3, pp. 61–68.

[33] OMG Group and others, "Software & Systems Process Engineering Meta-Model Specification." 2008.