International Journal of Computer Science and Engineering (IJCSE)
ISSN(P): 2278-9960; ISSN(E): 2278-9979
Vol. 4, Issue 4, Jun - Jul 2015, 9-22

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

TRACEABILITY METRICS PREDESTINED FOR TRANSFORMATION REQUIREMENT TO ARCHITECTURE (DESIGN REENGINEERING TOOL)

RASHMI YADAV[1], RAVINDRA PATEL[2] & ABHAY KOTHARI[3]

[1,3]Acropolis Indore, Department of Computer Science Engineering and Application,
Rajiv Gandhi Technical University, Bhopal, India

[3]Department of Computer Science Engineering and Application, Rajiv Gandhi Technical University, Bhopal, India

ABSTRACT

Traceability Metrics are measures describing how well traceability is performed from original allocated requirements to software through design, coding and testing. In this paper propose Traceability Metrics of Reengineering Tool for transformation requirement to Architecture. For that we will initially frame a requirement Thereafter need to transform this requirement into architecture. It is found very difficult to transform requirement into architecture. When we map requirements into architecture most of the important information generated during mapping process is lost in the final architecture representation. In this paper proposed Traceability Metrics for measure quality of system and selection of traceability metrics at the instance when the requirement method used for transform requirement to architecture.

KEYWORDS: Traceability Metrics, Requirement Set, Software Architecture

1. INTRODUCTION

Traceability metrics ought to consider both directions from individual requirement to test results, and from test results to individual requirements. Requirements engineering only focuses on problem domain and system responsibilities, but not design and implementation details. Most important measure of success of a software system is the degree to which it meets the purpose for which it was intended **[1]** Software architecture concerned with the shape of the solution space **[2]**.

When we transform requirement into architecture it is difficult to transform requirement into architecture, because there are quite different perspectives in user requirements and software architecture. Requirement and architecture control different term and artifacts. Several researchers work on this and provide different view: a) requirement and architecture are different to each other b) they are related to each other c) There is gap between requirement and architecture.

In this paper proposed Traceability Metrics for measure quality of system and selection of traceability metrics at the instance when the requirement method used for transform requirement to architecture.

Research paper is organized as follows: Section 2 discusses the literature survey. Section 3 discusses Reengineering tool requirement. Section 4 presents methods for transforming requirement into architecture. Section 5 presents Selection of methods suitable for proposed reengineering tool. Section 6 conclusion and future work.

2. LITERATURE SURVEY

In reengineering, the authors argue for complete traceability from code through specifications to code. In the course of reverse engineering a legacy software system and its subsequent redesign and reimplementation, they found

several cases where traceability provided immediate benefits that appear to be specific to reengineering, Here we propose tracing metrics "all around"—from existing code to the legacy systems design to its requirements during reverse engineering, and from the modified requirements to the new design and the new code during development of the successor system. In effect, this means integrating traceability during reverse engineering with (more standard) traceability during forward engineering [32]. During the project, developers installed and used traceability in real time. Here, we describe the task, the project's basic reengineering approach, and some experiences, focusing exclusively on traceability and its benefits for developers of the new software.

For to find traceability metrics we studies several literature at real time conversion of requirement method used for transform requirement to architecture.

The software requirements specification describes the problem, not the solution. It rightly focuses on the behavior of the system. **Bohem [3]** identified the problem that requirement change dynamically even SRS (Software Requirement Specification) is unambiguous, complete, and consistent if requirements change architecture also change. It is difficult to transform dynamic change requirement into architecture. **Nicholas May [4]** give the survey of architecture viewpoint models and mentioned that there exist quite different perspectives in user (or customer) requirements and software architecture. **C. Hofmeister, Nord and D. Soni [5]** conclude that the concepts, languages, notations, and tools for architecture are much more closely related to detailed design and implementation rather than software requirements they use **Siemens** four views architecture design approach and try to reduce the gap between requirement and design. Again they also reexamine the global analysis it small the gap between requirement and architecture but not completely fill the gap**.** SEI model, Siemens model and Rational model for architecture documentation this documentation to use multiple concurrent diagrams to describe the entire software architecture of a system using Crowded diagrams, inconsistent notation, and mixing of architectural styles, he propose the need of separate subsystem those specify the separate requirements.. According to **Dan Calloway[6]** global analysis activities help to significant benefit in achieving their goal and, in some cases, the benefit went beyond what they had anticipated but, the use of global analysis activities was not applied as expected. **G. Hall and his colleague [7]** identify four differences and relationship between these two areas. Architecture developed formally and requirements are expressed in the language of customer. requirement are expressed in terms of characteristics of system for mission critical application give the equalities such as security, dependability safety, reliability, maintainability, and portability, which are often in conflict. For such systems, the trade-offs between conflicting requirements are often expressed through the choice of high-level architecture. Requirements are problem space and architecture is solution space it is very difficult to cooperate with these two dimensions. Managing the evolution of software successfully depends upon the stability of architecture the system contains the volatile requirements that is work again stability.

3. CHALLENGES TO MAPPING THE REQUIREMENT INTO ARCHITECTURE.

The functional requirements are extractor, repository, analyzer, visualize. Rick Kazmar [8] indentifies the challenge to mapping requirements he explains there are two types of Requirement Functional and Quality that is non functional requirement. Most customers and developers have focused on functional requirements what the system does and how it transforms its input into its output. But while functional requirements are necessary, quality requirements are critical to the software architecture and significantly influence the shape of the architecture. Choices among different quality requirements shape the architecture, Kazmar explains. Each requirement suggests certain architectural structures and rules

other ones out will choose one set of architectural structures over another because we know that it's a good architecture for being able to predict and control end-to-end latency or throughput, R. Chitchat et al. [9] identified the problem that when we map the requirements into architecture large amount of important information generated during the mapping of requirements to architecture is lost in the final representation of the architecture. He proposed the traceability schema that will provide support for recording such information generated during the mapping process. He focuses on the mapping from requirements to architecture though the schema could be used for relating other development stages. The problem with this schema is that mapping schemas is not automated and time-consuming activity it increase the cost and time. We discuss in details methods to transform from requirement to architecture identify the strength and weakness and which method is suitable in which particular scenario. Then we discuss which method is suitable for transform our requirement into architecture and why.

4. METHODS FOR TRANSFORM REQUIREMENTS TO ARCHITECTURE

Feature Oriented Domain Analysis Method

In 1982, Davis [10] gave Feature oriented Domain Analysis method which identified features as an important organization mechanism for requirements specification. In **1990 Kyo C. Kang [11]** proposed feature-oriented domain analysis (FODA) method. The merits of feature analysis are intended to capture the end-user's (and customer's) understanding of the general capabilities of applications in a domain which is the Limitation of Direct Mapping. FORM: A feature-oriented reuse method with domain-specific reference architectures it extends to FODA Method it emphasis software design and implementation phases and prescribes how the feature model is used to develop domain architectures and components for reuse. FORM method is quite fit for software development in mature domain where standard terminology, domain experts and up-to-date documents are available**. Reid Turner [12]** puts forward a conceptual framework for feature engineering in 1999. It prefers to look feature as an important organizing concept within the problem domain and proposes carrying a feature orientation from the problem domain into the solution domain. It shows that it is feasible and effective to make features explicit in software development and to take feature orientation as a paradigm during the software life cycle. Turner's framework comes from software development experience in telecommunication domain, and is still conceptual and incomplete. It does not provide particular solution for mapping requirements to Software architecture from software engineering perspective. **Dongyun Liu and colleague [13]** explore how to apply feature orientation as a solution for the mapping problem between requirements and Software architecture from general software engineering perspectives, focusing on the mapping and transformation process. is to organize requirements in problem domain into a feature model, and then base our architectural modeling on the feature model, with the goal maintaining direct and natural mapping between requirements model and architecture models address functional features and nonfunctional features separately in different architectural models. It is not replacement of traditional method it is an improvement on traditional methods. This approach can integrate closely with OO method. The modeling concepts and notation adopted in this paper are based on UML, but have appropriate extension.

The **feature-oriented** concept is based on the emphasis placed by the method on identifying those features a user commonly expects in applications in a domain. This method, which is based on a study of other domain analysis approaches, defines both the products and the process of domain analysis. Feature oriented domain analysis done by domain analysis based on features. Feature model is introduced & take a key role to identify commonalities and to generate architecture model. FOSD is not a single development method or technique, but a conglomeration of different ideas, methods, tools, languages, formalisms, and theories. What connects all these developments is the concept of a feature. Due

to the diversity of FOSD research, there are several definitions of a feature [22], e.g. (ordered from abstract to technical):

Kang et al. [29]: “a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems”. Kang et al. [30]: “a distinctively identifiable functional abstraction that must be implemented, tested, delivered, and maintained”. Czarnecki and Eisenecker [28]: “a distinguishable characteristic of a concept (e.g., system, component, and so on) that is relevant to some stakeholder of the concept”. Bosh [24]: “a logical unit of behavior specified by a set of functional and non-functional requirements”. Chen et al. [25]: “a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements”. Batory et al. [23]: “a product characteristic that is used in distinguishing programs within a family of related programs”. Classen et al. [22]: “a triplet, f = (R, W, S), where R represents the requirements the feature satisfies, W the assumptions the feature takes about its environment and S its specification”. Zave [31]: “an optional or incremental unit of functionality”. Batory [27]: “an increment of program functionality”. Apel et al. [26]: “a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder’s requirement, to implement and encapsulate a design decision, and to offer a configuration option”

Object Oriented Transition Method

Dongyun Liu and colleague [13] explore how to apply feature orientation as a solution for the mapping problem between requirements and Software architecture from general software engineering perspectives, focusing on the mapping and transformation process. is to organize requirements in problem domain into a feature model, and then base our architectural modeling on the feature model, with the goal maintaining direct and natural mapping between requirements model and architecture models address functional features and nonfunctional features separately in different architectural models. It is not replacement of traditional method it is an improvement on traditional methods. This approach can integrate closely with OO method. The modeling concepts and notation adopted in this paper are based on UML, but have appropriate extension.

Object Oriented Transition method is to transform the object-oriented output of the Requirements engineering phase (analysis) into an object-oriented architecture design phase.

Hermann kaindl [14] object oriented analysis and design are different. Developers have to perform two difficult OOD tasks concurrently: they have to specify architecture for the software and build a model of the domain to be used by that software. Merits of this method the Traceability seems to be possible in object-oriented models. This method not provides a complete solution for mapping requirements into architectures. This is due to the fact that internal structures of a system are described from developer’s point of view in the design; object-oriented analysis describes the user view. Both stages present different information or lack information that is of interest to the developer or the customer Moreover, a transition from analysis objects to design objects implies that real world objects become software objects and that the object-oriented analysis model deals with internal design. Following this, objects at different stages have different abstractions levels and different purposes. As a consequence object orientation does provide similar models but does not allow mapping between the requirement and architecture stage. It is very generic approach it can be used in most of the commercial application where object and classes can be identified.

Use Case Maps

Use **case maps** method are scenario based software engineering technique most useful at the early stage of software development. The notation is applicable to use case capturing and elicitation use case validation as well as high

level architecture design and test case generation. UCMs provide a behavioral framework for evaluating and making architectural decisions at a high level of design. A visual behavior structures, manipulated, reused, and understood as architectural entities. The primary objective is to ease understanding in all phases of development by neglecting details. UCMs can also be used to describe how organizational structures of complex systems and behavior are intertwined **[15]**. UCMs show causal paths directly between responsibilities in organizational structures of abstract components. They combine behavior and structure into one view and allocate responsibilities to architectural components. Related use cases are shown in map-like diagrams. The Use Case Map notation was developed to capture scenario descriptions as causal flows of responsibilities for object-oriented design of real-time systems

Use Case Maps (UCMs) represent scenarios executing across a system:

- show the emerging behavior of the system
- The basic UCM constructs are paths, components, and responsibilities
- use them to reason about design and performance

UCM (Use Case Map) R. J. A. Buhr [33] describe in his book "a high level scenario modeling technique defined for concurrent and real-time system design. It is based on a simple and expressive visual notation that allows describing scenarios at an abstract level in terms of sequences of responsibilities1 over a set of components. The primary objective of the UCM modeling technique is to capture and analyze system behavior at an abstract level; UCM describes scenarios at an abstraction level that is above both inter-component communication and detailed level component behavior. It allows focusing on individual scenario description, scenario interaction, and responsibility allocation, before introducing inter-component communication". As is, UCM models can be viewed as a specification for the modeling of inter-component communication.

UCM also provides important features:

- Superimposition of scenarios on system structure. This enables designers to visualize scenarios in the context of a system structure. It also provides a mechanism by which responsibilities can be allocated to system components.
- Combination of sets of scenarios in a single diagram. This enables designers to express scenarios and scenario interactions in a graphical manner. It also provides a mechanism that can be used by designers to analyze the overall system behavior that emerges from scenario combinations.
- Description of system dynamics both at the component level, where components may be created, destroyed, and moved from one location to another in the system, and at the scenario level, where the a scenario may be dynamically modified as the system evolve.

Weaving together requirements and Architecture method [16] weaving together requirement into architecture it gives flexibility to change requirement by using Twin speak model whenever require changing into architecture. It does not freeze the requirement at early stage but this method provides a high level-process framework and no detailed description on how to perform the transition. Additionally, it does not provide information on what software architectures are stable when dealing with changing requirements

Problem frames method allows the classification of software problems and the decomposition of a large problem

into sub-problems. The developer can focus on the problem domain instead of inventing solutions because the idea is to delay solution space decisions until a good understanding of the problem is gained. These sub-problems can then be solved and combined into a solution of the original problem**.** Problem frames can express the relationship between requirements, domain properties, and machine descriptions. When bridging requirements and architecture, problem frames can model the organization of requirements in the architecture. This allows us to deal with undesired effects, e.g. overlapping event reactions. **Jon G. Hall Michael Jackson and Colleague [17]** proposed twins peak model that illustrates the iterative nature of the development process. This is a process during which both problem structures and solutions structures are detailed and enriched. They also give the extension of problem frame they identified that most real problems are too complex to fit within a problem identification/solution description model. They require a third level of description. That of structuring the problem as a collection of interacting subs problems, each of which is smaller and simpler than the original, with clear and understandable interactions. Problem frame are give the notation for the third level. It is good in problem frame method is it use hierarchical solution structure ensures scalability of the system and traceability of architecture decisions. Problem frames describes architectural structures, services, and artifacts as part of the problem domain. A drawback rotationally the extension is slight, is that it is not clear that the notation covers all aspects for creating proper architectures. It is use where the development time require short because developers to describe the problem domain more abstractly, closer to business logic that operate in the domain.

Goal Based Transition approach performs a transition from requirements to architecture to meet functional and non-functional requirements. It can be regarded as a combination of qualitative and formal reasoning based on KAOS (it is goal oriented requirement specification Language), Perry's **[18]** use the Prescript to process which is a prescriptive architecture specification language that provides a high-level architecture**.** The process starts with analyzing the global impact of goals on architectures. The software specification is created based on underlying system goals by deriving requirements. Functional specifications are considered in the architectural draft that is built in a second step. This draft is then refined to fulfill the domain constraints. The final architecture which complies with all non-functional requirements is achieved using recursive refinement. This approach supports intertwinement of requirements and architecture creation and allows the extraction of different views (e.g. security view, fault tolerance view).goal based transition method is the qualitative reasoning in there refinement process that should be more formal to allow extended tool support. Also, when architectural features need to be propagated bottom-up this approach is limited as it focuses on refinement. A combination of bottom-up and top-down might help. If the relation between global architecture decisions at early stages of the process and meet all nonfunctional requirement at first stage then we use the Goal Based transition. After early stage when we do final refinements to meet all non-functional requirements should be more difficult. Recursive Refinement several time are time consuming.

Rule Based Decision Making method describe by W. Liu and S. Easterbrook [19] that the making architectural decision based on requirements, analyzing cost benefit analysis tradeoff and keeping design options open is a difficult task. Existing work on classification of architectural styles and features reusable components and derivation of relevant architectural styles provides useful heuristic to the task but it is highly labor intensive. It presents a framework is based on the assumption design options that architectural decisions are labor-intensive and difficult to make**.** The framework supports automated reasoning for eliciting architectural decisions based on requirements. In rule based framework consists of two main modules, a reasoning module and presentation module. The reasoning module contains a mapping process which allows the generation of decision trees. These trees provide guidance through the decision making

process. They are used to manually map each requirements specification into architectural properties. An addition to the actual transition process is the capturing of mappings and the process of mapping to study how decisions are made. This supports later architectural decisions. this method explores the applicability of a unified description language for requirement specifications architecturally significant properties. Rule Based decision making need significant human interaction is required to perform the transition from requirements to Architecture. Framework can be customized for any application domain. The rule base can be easily updated as new mapping are required.

Architecting requirement method [20], provide a systematic approach to produce design of more consistent quality reduce human labor and error train new designers effectively and relate the design more closely to the requirements. Managing changes effectively can reduce cost and effort during maintaince. The methodology proposed implicit analysis in a separate phase as part of the requirement realm thus architecting requirement that the end products of requirement analysis have a structure that represent the logical view of the system. This structure also enable the incremental analysis of change request at the requirement level before propagating to the design and implementation. Architecting requirements replaces the architectural design phase and fit in the development process between the requirement elicitation / modeling phase and the design implementation phase. This structure also enables incremental analysis it uses hierarchies to structure requirements and provides analysis techniques for designers to refractor requirements in order to identify the right form. This is not only the gives rise to the system decomposition but also provides a foundation for further design and change management. Compared to the approaches mentioned so far, this work uses new ideas to solve the problem of transition from requirements to architecture. This method models entities and relationships, it allows managing change request. This method reduces the manual work but tool support is mandatory. This methods is suitable where we are not very confident for frozen the requirement or we development this type of software first time.

Patterns method [21] give the answer of research question of transforming the dependability requirements into corresponding software architecture constructs by proposing first that dependability needs can be classified into three types of requirements and second an architectural pattern that allows requirements engineers and architects to map the three types of dependability requirements into three corresponding types of architectural components. The pattern proposed by Lihua Xu allows the modeling of dependability Non Functional Requirements as first class requirements elements during software development, followed by explicit mapping of such NFRs into software architectures, all while embracing traditional architectural design principles for meeting the stated Functional requirements. Previously it was said that NFRs are considered to be an integral part of the system and used them to drive the development process and according to further research it is considered that both FRs and NFRs to be parts of the requirements elements that will be mapped into architectural design elements to be implemented later.

In the previous approaches the design model does not require particular specific techniques to be used by the designer and presently it can be used together with any traditional design technique, including architectural styles, design patterns, UML and so forth. This method emphasis on non-functional requirement it support early and explicit specification of non-functional requirements during requirement gathering followed by design of corresponding software architectures. The problem with this method Tracing software requirements to architecture level including dependability and other non-functional requirements for which this is often difficult. It is used where the non-functional requirement highly required.

5. TRACEABILITY METRICS FOR SELECTION METHOD

Once requirement set for proposed reengineering tool is prepared we need to transform this requirement into architecture. Researchers and Methodology mentioned that there is no clear way to select one method for the transforming requirement into architecture. We read the almost all methods individually and try to find out which method give the appropriate architecture. We analyze the detailed impact of each and every method of proposed reengineering tool. Identify Traceability metrics is pedestal on following features of traceability concept according to our requirement: Functional Requirement, Non function Requirement, Support changes in Requirement, Design Technique Used, Support for Traceability, Principal underlying the method, Suitable Environment, Traceability in Context with Proposed tool.

The Feature oriented method [**11][12][13]**. It support the user view but it use in mature domain and where standard terminology, Domain expert and up-to-date documents are available we cannot use this. Object oriented transition **[14]** give the Traceability which is desirable in our tool but in object oriented method emphasis the object is uniform it can be use anywhere but object at different stages have different abstraction level and different purposes we can use this method for proposed tool. Use case Maps methods**[15]** focus on the dynamic picture ucm are useful for the requirements exploration and architectural design details can be further added when required by using UML, ADL. UCM combine with formal description such as user requirements Notations(URN)and the goal requirement oriented language(GRL) although this method have lack of well defined syntax and large numbers of human involvement required this makes the methods slower but the GUI interface provide the good support for map requirement into architecture. If we use method weaving together requirement into architecture **[16]** for our tool it gives flexibility to change requirement by using Twin speak model whenever require changing into architecture. It does not freeze the requirement at early stage but this method provides a high level-process framework and no detailed description on how to perform the transition. Additionally, it does not provide information on what software architectures are stable when dealing with changing requirements. We cannot use this method. If we use problem frame methods[**17]** this is very good method it give the scalability and traceability which is the requirement of our reengineering tools. Goal based transition method[18]. Generates the architecture by recursive refinement and fulfils all functional and non-functional requirements. This method is suitable for proposed reengineering tool although the recursive refinements are time consuming. rule based method**[19]** are applicable for our reengineering tool this method using two main modules reasoning module contains a mapping process which allow the generation of decision trees these tress provide guidance through the decision making process. But in this method need to manually map the requirements specification into architectural properties. This methods need large human involvement this makes method slow. Architecting requirement method**s [20]** are faster it using automated tool it also model the entity and relationship. It allows managing the change request and incorporating the change into the Architecture. This method more suitable for anticipated reengineering tool. Pattern methods**[21]**emphasize on the non-functional requirements, this method allows the modeling of dependability of non-functional requirements as first class requirements elements during software development while other method emphasize on functional requirements but traceability between the dependability non functional requirement into architecture it is difficult so this method not suitable for proposed reengineering tool. Using the recommended methods we have come out with traceability Metrics Table: 1 show “Requirement Set and function supported by Reverse Engineering tool”.

6. CONCLUSION AND FUTURE WORK

There is no straight forward way to select a single method to transform requirement into architecture to lead a better architecture of proposed reengineering tool. We conclude that according to requirement set, we are using some selected methods. We can use Object Oriented Transition as it supports the traceability, which is the requirement of our proposed tool. We can also use Use-Case method as it focuses on the dynamic picture and give GUI support which lead better architecture. We can also use the Twin Peak Model because it gives the flexibility to change our architecture if needed. Although it is little bit time consuming. We can use Architecting Requirement by using automatic tool as it is a faster method to lead to architecture. In my opinion, the selection of methods depends on the requirement set. Even though, we are giving some features and environment which give the aptness of the methods. We cannot say our feature and environment are sufficient for developing Reengineering tool architecture. But it is just one way to decide the suitable method to transform requirement into architecture. In future we will take some different requirement sets of reengineering tool and find out some more features which can be apply on the chosen methods in a limited manner to lead better architecture.

7. CONCLUSIONS

There is no straight forward way to select a single method to transform requirement into architecture to lead a better architecture of proposed reengineering tool. We conclude that according to requirement set, we are using some selected methods. We can use Object Oriented Transition as it supports the traceability, which is the requirement of our proposed tool. We can also use Use-Case method as it focuses on the dynamic picture and give GUI support which lead better architecture. We can also use the Twin Peak Model because it gives the flexibility to change our architecture if needed. Although it is little bit time consuming. We can use Architecting Requirement by using automatic tool as it is a faster method to lead to architecture. In my opinion, the selection of methods depends on the requirement set. Even though, we are giving some features and environment which give the aptness of the methods. We cannot say our feature and environment are sufficient for developing Reengineering tool architecture. But it is just one way to decide the suitable method to transform requirement into architecture. In future we will take some different requirement sets of reengineering tool and find out some more features which can be apply on the chosen methods in a limited manner to lead better architecture.

REFERENCES

1 Nuseideh, V. and Easter brook S., “Requirement Engineering: a Roadmap” Proceedings of the International conference on Software Engineering (ICSE2000), 2000.page 35-46.

2 Perry D. E. and Wolf A. L., “Foundations for the study of software architecture” SIGSOFT Soft. Eng. Notes, 1992, 17 (4), pp. 40–52.

3 Boehm B. W.,”Requirements that handle IKIWISI, COTS, and rapid change” Computer, 2000 33, (7), pp. 99–102.

4 Nicholas May, “A Survey of Software Architecture Viewpoint Models” 6th Australasian Workshop on Software and System Architectures (2004).

5 C. Hofmeister, R. L. Nord and D. Soni, “Global Analysis: moving from software requirements specification to

structural views of the software architecture" IEE Proc.-Soft., Vol. 152, No. 4, August 2005.

6 Dan Calloway, "Critique of Global Analysis: Moving from software requirements specification to structural views of the software architecture". THE CHRONICLER'S WEB LOG, 6 June 2010.

7 Jon G. HallIvan Mistrik Balshar Nuseibh Dr. Andres Silva, "Relating Software Requirements and Architectures" IEEE Proceeding Software Volume 152 no. 4 August 2005.

8 Rick Kazmar, "Meeting the challenges of Requirements Engineering" news at SEI on 1 March 1999.

9 Chitchyan, Pinto, Fuentes, Rashid, "Relating AO Requirements to AO Architecture, Early Aspects" 2005 at OOPSLA.

10 Davis, A. M., "The design of a family of application-oriented requirements languages." Computer 15 (5) (1982) 21-28.

11 Kang, Kyo C."Feature-Oriented Domain Analysis Feasibility Study "(CMU/SEI-90-TR-21, ADA235785) CMU-SEI 1990.

12 C. Reid Turner, "A conceptual basis for feature engineering", The Journal of Systems and Software 49 (1999) 3-15.

13 Dongyun Liu Hong Mei, "Mapping requirements to software architecture by feature-orientation" Volume: 25, Issue: 2, Pages: 69-76 Requirements Engineering (2003).

14 H. Kaindl, "Difficulties in the Transition from OO Analysis to Design," IEEE Software, vol.16, pp.94-102, 1999

15 J. A. Buhr,"Use Case Maps as Architectural Entities for Complex Systems," IEEE Transactions on Software Engineering, vol. 24, pp. 1131-1155, 1998.

16 Nuseibeh, "Weaving together requirements and architectures," IEEE Software, vol. 34, pp. 115-11, 2001.

17 J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti, "Relating Software Requirements and Architectures using Problem Frames," IEEE Joint International Requirements Engineering Conference (RE'02), Essen, Germany, 2002.

18 M. Bradozzi and D. E. Perry, "From Goal-Oriented Requirements to Architectural Prescriptions: The Preskriptor Process," The Second International Workshop on Software Requirements and architectures (STRAW '03) at ICSE'03, Portland, OR, 2003.

19 W. Liu and S. Easterbrook, "Eliciting Architectural Decisions from Requirements sing a Rule-based Framework," The Second International Workshop on Software requirements and Architectures (STRAW '03) at ICSE'03, Portland, OR, 2003.

20 W. Liu, "Architecting Requirements," Doctoral Consortium at RE'04, Kyoto, Japan, 2004.

21 Lihua Xu, Hadar Ziv, "An Architectural Pattern for Non Functional Requirements "Elsevier Science Direct 2006.

22 A. Classen, P. Heymans, and P. Schobbens, What's in a Feature: A Requirements Engineering Perspective. In

Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), volume 4961 of Lecture Notes in Computer Science, pages 16–30. Springer-Verlag, 2008.

23 D. Batory, J. Sarvela, and A. Rauschmayer, Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering (TSE), 30(6):355–371, 2004.

24 J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. ACM Press / Addison-Wesley, 2000.

25 K. Chen, W. Zhang, H. Zhao, and H. Mei, An Approach to Constructing Feature Models Based on Requirements Clustering. In Proceedings of the International Conference on Requirements Engineering (RE), pages 31–40. IEEE CS Press, 2005.

26 S. Apel, C. Lengauer, B. M¨oller, and C. K¨astner, An Algebra for Features and Feature Composition. In Proceedings of the International Conference on Algebraic Methodology and Software Technology (AMAST), volume 5140 of Lecture Notes in Computer Science, pages 36–50. Springer-Verlag, 2008.

27 D. Batory, Feature Models, Grammars, and Propositional Formulas. In Proceedings of the International Software Product Line Conference (SPLC), volume 3714 of Lecture Notes in Computer Science, pages 7–20. Springer-Verlag, 2005.

28 K. Czarnecki and U. Eisenecker, Generative Programming: Methods, Tools and Applications. Addison-Wesley, 2000.

29 K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

30 K. Kang, S. Kim, J. Lee, K. Kim, G. Kim, and E. Shin, FORM: A Feature-Oriented ReuseMethod with Domain-Specific Reference Architectures. Annals of Software Engineering, 5(1):143–168, 1998.

31 P. Zave, An Experiment in Feature Engineering. In Programming Methodology,pages 353–377. Springer-Verlag, 2003.

32 Ebner, G. ; Kaindl, H., Tracing all around in reengineering, Software, IEEE (Volume: 19, Issue: 3), 2002.

33 R. J. A. Buhr, R. S. Casselman, *Use Case Maps for Object-Oriented Systems.* Prentice Hall, 1996.

APPENDICES

Table 1: Traceability Metrics for Transformation Requirement to Architecture in Reengineering Domain

Methods	Functional Req.	Non Functional Req.	Support Changes in Requirement	Design Technique Used	Support for Traceability	Principal Underlying the Method	Suitable Environment	Traceability in Context with Proposed Tool
Feature oriented Domain Analysis [10][11][12]	Yes, Separate architecture model than nonfunctional	Yes. Separate architecture model	It emphasizes on users understanding of how the application will works on live domain so requirement are and support separation of concern then it easy to change	Architecture model is based on feature model Has some resemblance to object oriented techniques.	Feature model	Feature oriented	It is used in mature domain standard terminology domain expert and up-to-date documentation available	Reengineering tool have four basic components extractor, repository, analyzer and visualize out of which extractor & analyzer are relatively complex and All together there is need for numerous quality or nonfunctional requirements, so here methods are recommended keeping these facts in view
Object oriented transition [14]	Yes	Yes	Yes	Object & class related diagram	Yes	Convert Object oriented Analysis model to object oriented Design Model	Commercial Application	√
Use Case Maps [15]	Yes	Yes ,as behavioral frameworks are used to evaluate and make architectural decision at higher level of design	Yes	Related use cases are shown in map like diagrams, this notation is useful for capturing, elicitation and validations of use case this helps in architecture design and test case generations	Yes	Scenario based ,behavioral framework is used to evaluating and make architecture decision.	Object oriented and commercial application	√
Weaving together requirement into architecture [16]	Yes	yes	Yes, Very flexible		Implicitl-y yes	Twins Peak model is used which supports changing requirements	This methods is suitable where we are not very confident for frozen the requirement or we development this type of software first time	
Problem frame [17]	Yes	Yes	They work on frame formats and short delivery so it is unlikely as problem frames.	Real problems can be modeled as problem frames which describes architectura-l structures, services and artifacts as a	yes	A problem is collection of many simple sub-problem	Need Early delivery	√

				part of problem domain.		s.		
Goal based transition[18]	Yes	Same functional architecture is recursively refined to accommodate non functional requirements.	Yes	Goal based transition method uses architectural specification language.	yes	Requirement architectural from system goals.	Non functional requirement highly required	√
Rule Based decision making[19]	Yes	-	Rule base can be easily updated so, yes,	Reasoning and organization module	yes	Automated rule based reasoning	Application domain need flexibility	-
Architectring Requirement [20]	Yes	Yes, as refactoring of requirements is there	Yes	Requirements elicitation, architecturin-g requirement, design implementation phase.	Implicitly implemented, as architect-ring requirements phase replaces architecture design phase.	Implicit analysis	Requirement set not confidently design software design first time	√
Patterns[21]	yes	Yes on priority	yes	Design patterns	Poor	Non functional requirements then functional	Highly desirable for non functional requirement	