DESIGN GENERIC REENGINEERING TOOL ARCHITECTURE

* Rashmi Yadav, ** Ravindra Patel, and *** Abhay Kothari

* rashmi.yadaver@gmail.com, Acropolis Technical Campus, Indore

** ravindra@rgtu.net, UIT, Rajiv Gandhi Prouyodigiki Vishwavidalaya, Bhopal

*** kothari_abhay@hotmail.com, Indore Institute of Science and Technology, Indore

ABSTRACT

Software architectures capture the most significant properties and design constraints of software systems. In our research work, we intended to propose architecture for a reengineering tool. For that we will initially create a requirement set classified into functional and non-functional categories. Thereafter need to transform this requirement into architecture. It is found very difficult to transform requirement into architecture. So, this paper will present a survey of techniques and technologies that have been proposed over the years for transforming requirement into architecture to analyze their comparative strength, weakness and suitable environment. It will also highlight the method which is suitable for our reengineering tool requirement transformation and will try to give appropriate reasons for it.

Keywords: Use Case maps, Problem Domain, Requirement Set, User requirement notation.

1. INTRODUCTION

Requirements engineering aims to good understanding of the problem domain and users or customers needs through effective problem analysis techniques, and producing a correct, unambiguous, complete and consistent requirements specification which serves as a baseline for developers to implement the software system. Requirement engineering is concerned with discovering the purpose of a software system and the contexts in which it will be used [1]. 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.

Software architecture concerned with the shape of the solution space. It aims at making the architecture of a system explicit and provides a blueprint for the succeeding development activities. Software architecture is concerned with the study of the structure of software, including its topology, properties, constituent components and their relationships and patterns of combination [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 survey various methods available to transform requirement into architecture.

Research paper is organized as follows: Section 2 discusses the literature survey. Section 2 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

The software requirements specification describes the problem, not the solution. It rightly focuses on the behavior of the system. Bohem and colleague [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. They identify the ADR (architecture design requirement those affect the design decision of the architecture to identify and they treat as a critical inputs for the architecture.

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. REENGINEERING TOOL REQUIREMENT

Propose objective to design the architecture of reengineering tool. We study the available reengineering tool to analyzes the prons and corms and develop requirement set classify into functional and nonfunctional requirements. Prepare a functional requirement set for proposed reengineering tool. Analyze the existing tools find their merits and demerits. The functional
requirements are extractor, repository, analyzer, visualize. Nonfunctional Requirements are related with the quality Interoperability, Traceability, Compatibility, Scalability, reusability, Interconnectivity, Reliability, Modifiability, Efficiency, and Understandability.

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, or that it’s a good architecture for high availability.

R. Chitchat [9] in this work he 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 [FODA][1999]

In 1982, Davis [10] identified features as an important organization mechanism for requirements specification. In 1990 Kyo C. Kang [11] proposed feature-oriented domain analysis (FODA) method. In feature oriented Domain Analysis [FODA] the concept of using feature model for transform requirement into architecture. 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.

**Object Oriented Transition [1999]**

The idea of this methodology is to transform the object-oriented output of the Requirements engineering phase (analysis) into an object-oriented architecture design phase. Hermann kaindl [14] claim that the OO methodologist think object oriented development is that the developers can use objects in a uniform modeling approach throughout the process. In particular they can coherently apply the same notation for representing these objects and their relation in both analysis and design. Transition between the analysis and design are very easy and smooth. But in real life it is not the difficulty in moving from OOA to OOD is caused by the fact that OOA and OOD objects represent inherently different things. As a result, an OOA model cannot simply become an OOD model. Thus, 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 Map [1998]**

Use case maps 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. Use Case Maps provide a behavioral framework for evaluating and making architectural decisions at a high level of design. as 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. Use case maps can also be used to describe how organizational structures of complex systems and behavior are intertwined [15]. Use case maps 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. Use Case model dynamic (run time) refinement for variation of behavior and structure. Use case visually integrates behavior and structural components in a single view. Use Case Map diagrams focus more on a dynamic picture, not on source code organization. Use case maps provide a light-weight notation that scales up, while covering complexity factors. All artifacts of Use case maps (i.e. paths, components, responsibilities) can be developed in different orders,
depending on the starting point and objective. That makes Use case maps useful for requirements exploration and architectural design. Details can be added when needed by using notations like UML, ADL, and by providing additional system views and transformations. Use case maps can be combined with formal descriptions such as the User Requirements Notation (URN) and the Goal oriented Requirements Language (GRL). The lack of well-defined semantics and the large human input required when using Use Case Maps use. Use case maps always be used in most of the object oriented analysis and design representation of the commercial application is an additional feature this approach bridges the gap between Analysis and design problem by representing the scenario with special notation of uml etc.

**Weaving together requirements and Architecture [2003]**

Bashar Nuseibeh [16] mentioned in their paper that the Software-development organizations often choose between alternative starting points—requirements or architectures. This invariably results in a waterfall development process that produces artificially frozen requirements documents for use in the next step in the development life cycle. Alternatively, this process creates systems with constrained architectures that restrict users and handicap developers by resisting inevitable and desirable changes in requirements. This method uses the twin peak model it intertwines software requirements and architectures to achieve incremental development and speedily delivery. It gives flexibility to developers to do desirable change in requirements. This idea 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. Which requirements are more stable than others or what kind of changes can occur in a system are also open issues when applying the Twins Peak Model. 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 [2002]**

Problem frames allow 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 [17]. 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 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 [2003]
The 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 use the Preskriptko process which is a prescriptive architecture specification language that provides a high-level architecture [18].

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: [[2003]

W. Liu and S. Easterbrook [19] identified 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 [19]. 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 Requirements [2004]
Architecting requirement provide a systematic approach [20] 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 maintenance.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 [2006]
Lihua Xu and colleague [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
nonfunctional requirement it support early and explicit specification of nonfunctional 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 Nonfunctional requirements for which this is often difficult. It is used
where the nonfunctional requirement highly required.

5. SELECTION OF METHODS SUITABLE FOR PROPOSED REENGINEERING TOOL

Once requirement set for proposed reengineering tool is prepared we need to transform this
requirement into architecture. Researchers and Methodologies 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.

If we apply Feature oriented [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 fulfills all functional and nonfunctional requirement. 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 methods [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 proposed reengineering tool. Pattern methods[21]emphasize on the nonfunctional requirements, this method allows the modeling of dependability of nonfunctional 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.

Table 1 suitability of reengineering tool and method

<table>
<thead>
<tr>
<th>Method</th>
<th>References</th>
<th>Suitable Environments</th>
<th>Suitable for proposed Reengineering tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature oriented Domain Analysis</td>
<td>[10][11][12]</td>
<td>It is used in mature domain standard terminology domain expert and up-to-date documentation available</td>
<td>×</td>
</tr>
<tr>
<td>Object oriented transition</td>
<td>[14]</td>
<td>Commercial application</td>
<td>✓</td>
</tr>
<tr>
<td>Use Case Maps</td>
<td>[15]</td>
<td>Object oriented and Commercial application</td>
<td>✓</td>
</tr>
<tr>
<td>Weaving together requirement into architecture</td>
<td>[16]</td>
<td>This methods is suitable where we are not very confident for frozen the requirement or we development this type of software first time</td>
<td>×</td>
</tr>
<tr>
<td>Problem frame</td>
<td>[17] Need early delivery</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Goal based transition</td>
<td>[18] Nonfunctional requirement highly required</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Rule Based decision making</td>
<td>[19] Application domain</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Need flexibility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecting Requirement</td>
<td>[20] Requirement set not confidently design</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Develop first time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td>[21] Highly desirable Nonfunctional requirement</td>
<td>×</td>
<td></td>
</tr>
</tbody>
</table>

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 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. REFERENCES


6. Dan Calloway “Critique of Global Analysis: Moving from software requirements specification to structural views of the software architecture”.


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


