International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:1, No:11, 2007

Moving From Problem Space to Solution Space

Bilal Saeed Raja, M. Ali Iqbal, and Imran Ihsan

***Abstract*—**Extracting and elaborating software requirements and transforming them into viable software architecture are still an intricate task. This paper defines a solution architecture which is based on the blurred amalgamation of problem space and solution space. The dependencies between domain constraints, requirements and architecture and their importance are described that are to be considered collectively while evolving from problem space to solution space. This paper proposes a revised version of Twin Peaks Model named Win Peaks Model that reconciles software requirements and architecture in more consistent and adaptable manner. Further the conflict between stakeholders' win-requirements is resolved by proposed Voting methodology that is simple adaptation of win-win requirements negotiation model and QARCC.

***Keywords*—**Functional Requirements, Non Functional Requirements, Twin Peaks Model, QARCC.

I. Introduction

REQUIREMENT Engineering is one of the most important aspect of problem space. The deliverables of requirement engineering are eventually transformed into solution space. The outcome of requirement engineering is conventionally classified as both functional requirements (FRs) and non-functional requirements (NFRs), and they both constrain each other [2]. FRs is to be mapped as functional attributes & NFRs as quality attributes of a system. Before the emergence of software architecture as a new subject, only FRs was considered while devising the systems solution. But the devised architecture is getting its importunate place in the software development life cycle; it is arguing the importance of NFRs as well. This paper spots the impact of major drivers in problem space (FRs, NFRs & Domain Constraints.), over solution space (Architecture Decisions).

According to the survey conducted by Standish Group on software failures, it has been observed that only 61% of the originally specified requirements were available on the released software product. This led to 53% of the projects to be over budget and behind schedule, while 31% were deemed failure [1]. This shows the importance of mapping the requirements to the solution space.

Manuscript received May 20, 2005.

Bilal Saeed Raja is with the Hamdard Institute of Information Technology, Hamdard University, Islamabad, 44000, Pakistan (phone: 92 – 300 – 9540571; e-mail: bilal_saeed_raja@yahoo.com).

Muhammad Ali Iqbal is with the Hamdard Institute of Information Technology, Hamdard University, Islamabad, 44000, Pakistan (phone: 92 – 300 – 5235607; e-mail: allyiqbal@yahoo.com).

Imran Ihsan is with the Hamdard Institute of Information Technology, Hamdard University, Islamabad, 44000, Pakistan (phone: 92 – 333 – 5118897; e-mail: iimranihsan@gmail.com).

A. Problem Space and Solution Space

Stakeholders' are the key to both problem space and solution space, stakeholder identified by the problem space state the domain constraints (DCs), FRs and NFRs. Similarly stakeholders identified by solution space state architectural decisions (ADs) as well as FRs and NFRs as illustrated in the fig. 1. The relationship between set of requirements and an effective architecture for a desired system is not readily obvious. Requirements largely describe aspects of the problem to be solved and constraints on the solution [4][9].

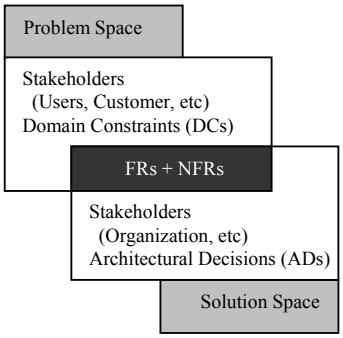

Fig. 1 Problem Space and Solution Space

In the literature, guidelines are available for modeling and understanding the impact of architectural decisions and requirements on each other. But there are some critical constraints, which need to be addressed when trying to reconcile requirements and architectures. These include:

- Requirements are frequently captured informally in a natural language. On the other hand, entities in a software architecture specification are usually specified in a formal manner.
- System properties described in non-functional requirements are commonly hard to specify in an architectural model.
- Iterative, concurrent evolution of requirements and architecture demands that the development of architecture be based on incomplete requirements. Also, certain requirements can only be understood after modeling and even partially implementing the system architecture.
- Mapping requirements into architectures and maintaining the consistency and traceability between the two is complicated since a single requirement may address multiple architectural concerns and a single architectural element may have numerous non-trivial relations to

various requirements [11].

- Real-world, large-scale systems have to satisfy hundreds, and even possibly thousands of requirements. It is difficult to identify and refine the architecturally relevant information contained in the requirements due to this scale.
- Requirements and the software architecture emerge in a process involving heterogeneous stakeholders with conflicting goals, expectations, and terminology.

Requirements are derived from the concepts and relationships in the problem space. Sometimes they reflect conflicting, interests of a given system's stakeholders (customers, users, managers, developers). The conflict resolution process for the right balance of requirements and architecture is complex and difficult due to the following obstacles [5][6]:

Difficulties in coordinating multiple stakeholders' interests and priorities. Users feel that full functionality, dependability, and ease of use are the most important attributes. The primary concerns of customers are cost and schedule and acceptable level of quality. Developers are mostly concerned about the low project risks and reusability of software components and assets. Maintainers are strongly concerned about the good diagnostics and easy maintenance means. Finding the middle ground among these requirements is quite difficult in reality.

Complicated dependencies and tradeoff analyses among quality attributes. Every decision to improve some quality or functionality may impact other quality attributes , particularly the cost and schedule. Some requirement and decisions may be not compatible with others.

Exponentially increasing resolution option space. In order to resolve a conflict, several items should be considered. For example, which functionality should be reduced and by how much to get the project back on track? Which constraints can be degraded in terms of solution architecture?

B. Dependencies among Factors

All Strong dependencies exist among DCs, FRs, NFRs and ADs (Fig. 2). They constrain or realize each other. So a natural and obvious architecture requires a balanced selection and alignment of these factors. Also it is ominous to overestimate any one and underestimate any other of these factors.

1. DCs and NFRs constrains each other
2. NFRs realizes DCs
3. FRs realizes DCs
4. DCs and FRs constrains each other
5. FRs realizes NFRs
6. FRs and NFRs constrains each other
7. ADs realizes NFRs
8. ADs and NFRs constrains each other
9. ADs realizes FRs
10. ADs and FRs constrains each other

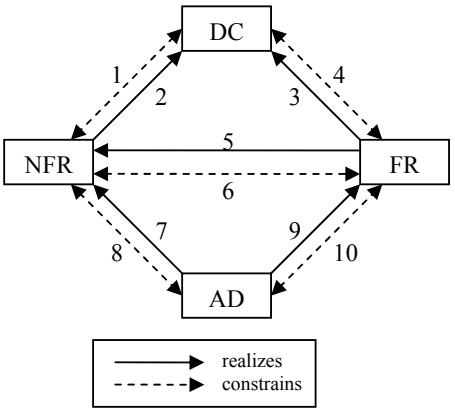

Fig. 2 Dependencies between FRs, NFRs, ADs and DCs

II. Existing Solutions

A. Classical Twin Peak Model

As illustrated in the fig 2, dependencies show that often we are restricted to specific architecture decision due to certain requirements. If the adopted life cycle model or the system lets us take specific architecture decision first then we have to compromise over FRs or NFRs some times. Specially talking about water fall model that yields frozen requirements leads towards constrained system architecture that not only restricts user but also handicaps the developer.

Except for well-defined problem domains and strict contractual procedures, most software-development projects address requirements specification and design issues simultaneously—and justifiably so. In reality, candidate architectures can constrain designers from meeting particular requirements, and the choice of requirements can influence the architecture that designers select or develop.

Twin Peaks Model emphasizes equally on requirements and architectures [3]. Although this model develops requirements and architectural specifications concurrently, it continues to separate problem structure and specification from solution structure and specification, in an iterative process that produces progressively more detailed requirements and design specifications.

The Twin Peaks model addresses the three management concerns IKIWISI, COTS, and Rapid Change.

- I'll Know It When I See It (IKIWISI). Requirements often emerge only after users have had an opportunity to view and provide feedback on models or prototypes. Twin Peaks explicitly allows the user to explore the solution space early, permitting incremental development and consequent risk management.
- Commercial off-the-shelf software (COTS). Increasingly, software development is actually a process of identifying and selecting desirable requirements from existing commercially available software packages. With Twin Peaks, developers can identify requirements and match architectures with commercially available products,

rapidly and incrementally. The developer benefits by quickly narrowing the selections or making key architectural decisions to accommodate existing COTS solutions.

- Rapid change. Managing change continues to be a fundamental problem in software development and project management. Focusing on finer-grain development, Twin Peaks is receptive to changes as they occur. Analyzing and identifying a software system's core requirements are requisite to developing stable software architecture amid changing requirements.

B. Win-Win Spiral Model

Win-Win is a groupware support system for determining software requirements as negotiated win conditions. It is based on the Win-Win Spiral Model which uses Theory W to generate the stakeholder win-win situation incrementally through the Spiral Model [5]. Win-Win assists the identified stakeholders in identifying and negotiating issues (i.e., conflicts among their win conditions), since the goal of Theory W, "Make everyone a winner," involves stakeholders identifying their win conditions, and reconciling conflicts among win conditions.

C. Win-Win Negotiation Model

The Win-Win negotiation model is based on four artifact types: Win Conditions, Issues, Options and Agreements [7]. Win conditions capture the stakeholder goals and concerns with respect to the new system. If a Win condition is non-controversial, it is adopted by an Agreement. Otherwise, an Issue artifact is created to record the resulting conflict among Win Conditions. Options allow stakeholders to suggest alternative solutions, which address Issues. Finally agreements may be used to adopt an Option, which resolves the Issue.

D. QARCC

QARCC (Quality Attribute Risk & Conflict Consultant) is an exploratory knowledge-based tool for identifying potential conflicts and risks among quality requirements early in the software life cycle. QARCC uses the "Attributes" portion of Win-Win's domain taxonomy to identify potential quality attributes conflicts [5]. As stakeholders enter Win Conditions, they identify which domain taxonomy elements are relevant.

III. Proposed Solution

A. Classical Twin Peaks Model Revisited

Although Twin Peaks Model is one of the most appreciated and adopted models that tackles the dilemma of requirements and architecture integration. While talking in the context of FRs, NFRs, DCs and ADs, it is observed to exhibit few lacking.

- It is deficient in narrating evolutionary aspects of the system with respect to system implementation.
- In twin peak model, requirements and architecture are placed independently, but as observed it's never so. Requirements and architecture strongly overlap and such demarcation is impossible [12].
- Model is quite abstract in terms of different types of requirements and architectural aspects and the relationship between requirements and architectures.
- It is hard to understand that where the model actually lies, whether it is in problem space or solution space domain.

B. Win Peaks Model

Win Peaks Model adapts Classical Twin Peaks Model and tries to remove the deficiencies observed in Classical Model. Name of the Model "Win Peaks" is of importance with respect to its adaptation from Classical Twin Peaks Model. It is why because it extends basic theme of Twin Peaks in more details and adopts Win-Win Spiral Model for incremental system evolution. Additionally "Win Peaks" is more justified by using Win-Win Negotiation Model to resolve conflicts between requirements. Distinctive factors of Win Peaks Model are:

- It maps requirements and architecture onto the demarcated problem space and solution space.
- It extends implementation details of the system in terms of FRs, NFRs, DCs and ADs.
- It ideally imposes the dependencies among FRs, NFRs, DCs and ADs, as described in Fig. 1.
- It relates these factors as they fall in contact with each other and affect each other. As FRs & NFRs related with DCs in problem space, FRs & NFRs related with ADs in solution space and then FRs & NFRs of both spaces related to each other that affects ADs.
- The model expresses the overlaps & intermixes of requirements & architecture.
- The model expresses that the requirements are more general and by the time system evolves they are transformed into more specific architecture.
- The spiral nature of this model tries to negotiate between requirements in each increment by using Requirements Voting Methodology as presented below.
- It also exhibits the benefits of Classical Twin Peaks Model.

C. Requirement Voting Methodology

Most of the stakeholders' Win Conditions are non-controversial [7]. Negotiating requirements is one of the first steps in any software system life cycle, but its results have probably the most significant impact on the system's value. However, the processes of requirements negotiation are not well understood.

The Win-Win Negotiation Model has been formally specified and analyzed for consistency but only little is known about the correctness and usefulness of assumptions made during this process. To challenge its acceptability certain questions have been raised which are given below [8]:

- How are the stakeholders' and negotiation results affected by using a negotiation tool such as Win-Win?
- How similar are the negotiation results if stakeholders for all groups have a similar win conditions to start with and

a pre-defined negotiation model to follow?

- How do people factors, like work experience or age, effect the process and the outcome of the negotiation?
- Do people use the tool as it was anticipated by the model?

There are also many other unanswered questions. Some of these questions are general; others are more related to the specific Win-Win methodology. However, knowing the answers to these questions is vital in providing more useful and powerful negotiation aids for stakeholders.

Requirement Voting Methodology is a solution for related problems as discussed above. It starts with QARCC to identify potential quality attributes (NFRs) and their related conflicts to stakeholders' win conditions. It helps in addressing the stakeholders and their respective NFRs in the problem space as well as solution space.

Then for each requirement, we cast votes of respective stakeholder and record their categorization for that requirement with respect to its impact on system success. Requirements' impact classification is made into two; high-medium and medium-low, for better applicability and ease in selection or rejection. Stakeholders can more easily categorize requirements into these two according to their assumption about requirements' effect on system success.

Requirement conflict will be resolved in the following fashion.

- If the number of votes is lying under high-medium, then particular requirement will be accepted.
- If the number of votes is lying under medium-low, then particular requirement will be rejected.
- If the equal number of votes is lying under high-medium and medium-low, that means this conflict requires more effort to be resolved. So it is put under Win-Win Negotiation Model. And as already discussed, stakeholders' Win Conditions, Issues and Options are to be prepared to get some agreement.

Requirement Voting Methodology can be practiced in any increment cycle of win-win spiral of Win Peaks Model. The proposed artifact that records requirement voting is sketched below.

NFR	High - Med		Med - Low
Stakeholder 1	-		-
:	-		-
Stakeholder n	-		-
Result	Accept	Reject	Conflict
Resolution			

Fig. 3 Requirement Voting Artifact

IV. Conclusion

Integrating requirements, domain constrains and architecture is still a striking issue in devising the win-solution. Proposed Win Peaks Model and Requirement Voting Methodology are placed in position to be practiced more for their acceptance. But certain inspirations tried to be imposed in proposed hypotheses are;

- Win Peaks Model helps avoiding Kitchen-Sink architectures, which tries to address all conceivable in problem space [10].
- It employs the theory "architecture should focus as much on what to leave out as on what to put in". [10]
- Win Peaks and Voting facilitates producing a natural and obvious solution architecture that is not seemed to be stuck with the problem [10].

References

[1] *Standish Research Paper*, Chaos Study, 1995, available on-line at http://www.standishgroup.com .
[2] B. Paech, A.H. Detroit, D. Kerkow and A. von Knethen, "*Functional requirements, non-functional requirements, and architecture should not be separated - A position paper*". REFSQ' 2002, Essen, Germany, September 9-10 2002.
[3] Bashar Nuselbeh, "*Weaving Together Requirements and Architectures*", in proceedings of *IEEE Computer, 34,* 3 (March 2001), pp. 115-119.
[4] Paul Grünbacher, Alexander Egyed, and Nenad Medvidovic. "*Reconciling Software Requirements and Architectures: The CBSP Approach.*" in Proceedings of the Fifth IEEE International Symposium on Requirement Engineering, *2001.*
[5] Boehm, Barry "*Conflict Analysis and Negotiation Aids for Cost-Quality Requirements*", Center for Software Engineering and Computer Science Department, University of Southern California, 1999.
[6] Hall, J.G.; Jackson, M.; Laney, R.C.; Nuseibeh, B.; Rapanotti, L. "*Relating software requirements and architectures using problem frames*". Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02), Essen, Germany, 9-13 September, 2002, Pages: 137- 144.
[7] Boehm, B. and Egyed, A., "*Software Requirements Negotiation: Some Lessons Learned*", in proceedings of the 20th International Conference on Software Engineering, April 1998.
[8] Alexander Egyed, Barry Boehm, "*Analysis of System Requirements Negotiation Behavior Patterns*". Proceedings of 7th Annual International Symposium on Systems Engineering, 1997.
[9] Robert Glass, "Software Runaways: *Lessons Learned from Massive Project Failure"* published in the magazine Computerworld, 1989.
[10] Steve McConnell, "*Software Project Survival Guide*" a book published by Microsoft Press, 1998.
[11] Awais Rashid , Ana Moreira , João Araújo, "*Modularisation and composition of aspectual requirements*", Proceedings of the 2nd international conference on Aspect-oriented software development, p.11-20, March 17-21, 2003, Boston, Massachusetts.
[12] Hall J.G., Rapanotti, L., "*A reference model for requirements engineering*", in *IEEE* Proceedings of International Requirements Engineering Conference (RE'03), USA, September 2003.