Requirement Engineering Errors: Errors and Ambiguities of Visualization

Shail K Dinkar
Assistant Professor, Department. Of Computer Science and Applications,
G B Pant Engineering College,
Pauri Garhwal-246194, UK , India

ABSTRACT

Requirement Engineering encompasses the tasks that go into identifying, determining and understanding the customer needs as well as the expectations from the proposed software system or an application, taking account of possibly conflicting requirements of various stakeholders. Requirement Engineering, also referred to by names like requirement gathering or requirement analysis or requirement specification, is a well-defined stage in the software development life cycle. This stage is not only well defined but also very crucial to the success of software development. Most of the project failures have been attributed to this stage only. It's a general and very often heard statement: 'Requirements were not clear or Requirements were ambiguous'.

So, what does this word 'Requirement' is all about, those requirement analysts try to establish and still, most of the projects fail due to lack of clarity in requirements. Requirements describe the behavior of the proposed software system but the roots of the same lie in the original real-time system for which software system has been proposed. Requirements engineering, in this manner, encompasses both the problem domain of the system as well as the solution domain of the system. The process of establishing the requirements is not an easy task as it is crossing over the boundaries of problem and the solution domain on a regular basis, and is bound to have errors. Errors in requirement engineering stage have many sources, ambiguous and unclear requirements are only a part of the problem. Having explored a number of common types of errors to find their source and strategies for dealing with them, Kurt Bittner [1] categorized requirement errors into three major groups: Errors of conceptualization (the ones that arise because of poorly conceived problem); Errors of specification (the ones that arise because of poor specification) and Error of implementation (the ones that arise because of poor coding and testing).

This paper is an attempt to put forward an idea of one more category of requirement errors: **Errors of visualization**. The difference between errors of conceptualization and error of visualization though apparently seems vague but is as clear as the boundaries between problem and the solution domain. The paper attempts to study the differences between these two types of errors and also studies the symptoms and possible solution strategies for errors of visualization.

Keywords

RE – Requirements Engineering. SDLC – Software Development Life Cycle, Aspect-oriented Requirement Engineering,

1. INTRODUCTION

The software development activities begin first with the identification and determination of the customer needs. The principles of software development and maintenance based on software engineering discipline suggest a systematic and disciplined approach towards the very cause. The result of this approach is a well-defined life cycle for software development, of which the customer requirements identification and analysis is the first step. Being the first step, requirement analysis is critical to the success of software project and the major problem with this phase is that it is the only phase of software development life cycle that speaks in both the problem domain and the solution domain. This problem has initiated a lot of research in this area and by early 1990's requirement analysis has emerged as a field of study in its own right by the name 'Requirement Engineering'. Requirements engineering is increasingly recognized as an important activity in any systems engineering process as many delivered systems fail to meet their customer's requirements due, at least partly, to ineffective RE. Requirements engineering is the process of gathering and structuring information both on the problem domain and on the expectations towards the new system. The aim of this phase is to establish the customer requirements, which have been defined in various ways by various software engineering practitioners. In nutshell, all the definitions point to one aspect: requirements are a description of how a system should behave or a description of the system properties or attributes. Alternatively, it's a statement of 'what' an application is expected to do and describes the mandatory as well as desirable aspects of solution. Various approaches including goal-oriented RE, aspect-oriented RE, model-driven RE are directed towards establishing the correct and complete customer requirements. Still, specified requirements suffer from the issues of ambiguities, lack of clarity, completeness and correctness. It's impossible to eliminate the ambiguities as a whole and doing so would be a waste of time, effort and money too. The following sections cover a study of various steps involved in requirements engineering and the errors associated with those steps. Section 4 proposes a new category of error: errors of visualization and details out on its symptoms and solution strategy.

2. STEPS OF REQUIREMENTS ENGINEERING

Like any other engineering discipline, Requirements engineering follows a sequence of systematic steps to establish customer requirements. These systematic steps are:

1. Requirement Elicitation or requirement gathering – This phase comprises activities that enable the understanding of goals, objectives and motives for building the proposed software system. It also involves identifying the

requirements that resulting system must satisfy to achieve these goals.

2. Requirement Analysis – This phase involves refining the requirements to ensure there are no conflicts and ambiguities in the requirements.
3. Requirement Definition – This phase establishes the analyzed requirements at a higher level.
4. Requirement Specification – This phase comprises activities to document the identified requirements as per the priority and in a manner that requirements are manageable.

As it can be judged from the RE steps description above, the first step begins with study of the problem domain and rest of the next steps span across both the problem domain and the solution domain. Whereas requirement elicitation phase starts identifying the entities, their relationships, domains and meta-classes, requirement analysis and definition steps attempt at resolving the conflicting views of the stakeholders, as gathered in first step. Additionally, these steps attempt to conceptualize the studied system so that it can be documented in terms of specification for the next steps in SDLC cycle to follow.

The conceptualization activity performed during analysis and definition, at times, tends to miss on one thing – **visualization of the proposed system**. This activity is inherently considered a part of conceptualization activity and requirement analysts keep themselves focused at clearly laying down the system as it is conceptualized in problem domain, missing on underlying requirements that may surface on visualization of the problem in solution domain. These requirements have been defined as 'Emergent Requirements', indicating the requirements that emerge as a consequence of adoption of computer-based system. A software system, when implemented in place of traditional system, changes the environment and operational behavior of the system. As the customer's understanding of the software system increases, he comes up with better-defined requirements. Such emergent requirements can be taken care of by visualizing the proposed system before actually developing the same. Of the various approaches to requirements engineering, aspect-oriented requirements engineering lays stress on not just the analysis of requirements but also on their subsequent mapping onto the solution space. This approach has resulted in uncovering the errors in requirements that remain dormant as errors of visualization and surface at the time of user acceptance testing.

3. ERRORS IN REQUIREMENT ENGINEERING

As seen in previous section, requirements engineering has 4 steps or activities to it and errors can be introduced at any of the stages. Saying that requirements are ambiguous or unclear is just one part of the problems encountered during requirements engineering phase. Having explored a number of common types of errors to find their source and strategies for dealing with them, Kurt Bittner categorized requirement errors into three major groups:

1. Errors of Conception
2. Errors of Specification
3. Errors of Implementation

3.1 Errors of Conception

Errors of Conception occur when the requirement is poorly conceived, i.e. when it solves the wrong problem or is based on flawed assumptions about the solution. These types of errors usually occur because the objectives of the solution are poorly understood, indicating that the concept of the system in problem domain was perceived wrongly. When these errors are present, the resulting requirements many be unambiguous and clear, but nevertheless wrong. As a result of these errors, even when the implementation is of high quality, the resulting system is of little or no value and resorting to the defense that "requirements were satisfied" is of little consolation since the concept of the problem itself was understood incorrect.

These errors generally arise when requirements were gathered from wrong source or someone actively communicated wrong information.

3.2 Errors of Specification

Errors of specification represent those errors that arise from the way a requirement is described. These are also very common types of requirement errors. These can be further divided into errors due to under-specification (resulting from incomplete and vague descriptions); incorrect specifications, inappropriate techniques for descriptions; and over-specification leading to confusion.

3.3 Errors of Implementation

Errors of implementation include those cases that arise where the concept was right, the specification was right, but the requirement was not implemented correctly, or possibly at all. It is easy to say that this is not a requirements problem but rather a project management problem or a testing problem, but such distinctions are artificial and ultimately not very useful. There is no point in writing requirements if you are not going to implement them, and the only way to know if they are implemented correctly is to test them. The functional areas of project management and testing are inextricably linked with requirements.

4. ERRORS OF VISUALIZATION

Further to the errors as discussed above, there can be one more category of error: Errors of visualization. These errors arise when problem-space domain was adequately conceptualized and represented through models but was not actually mapped onto solution-space domain. The software system, however best tried to emulate, remains different from real-time system; even if the attempt is to automate the processing of the real-time system while developing a software system, the problem domain and the solution domain still speak in entirely different languages. Its not just value-added software project but also an automated project that require visualization of the processing steps in solution space at the time of requirement analysis itself. The result of these errors, like errors of conception, is that the resulting solution is of little or no value to the customer and again saying that 'the requirements were satisfied' is of little solace. The customer expects the implicit requirements to be taken care of in the application.

The origin of these types of errors lies in the poor use of analytical skills at business analyst's or requirements engineer's end and insufficient observation of the problem domain. In such cases, the analyst fails to map the behavior of problem domain entities to the behavior of solution domain entities. Following examples explain in detail the errors of visualization:

1. Consider a data-intensive software system having features for a lot of data entry and report generation out of that data. The system appears to be simple to move from manual processing to a software solution in the sense that few screens need to be designed to enter some data and then few reports need to be generated out of that data. But, that's how the system is working in its problem domain. What kind of advantages is it expecting from software solution can be gauged by visualizing the system in solution domain. It's possible that few of the reports may not be required in the newer system as its data is already present in some GUI screens or possibly, one format of report may be used with various filter criteria and column visibility-hiding criteria, in turn reducing the effort and time to develop those.

2. Housekeeping requirements for software – Consider a hospital management system recording its employee and the patient's information along with the doctor's prescriptions to the patients. The patient's details constitute a huge amount of transactional data (day-to-day operations) in this case. Before deploying a computer-based system, the hospital is maintaining cards for patients where doctor's prescriptions are recorded and employee data is maintained in a register. Patient's registration to the hospital is maintained through card only. Any renewal / discharge is marked on this card.

   Looking at the problem domain and having interviews with the participants in the system, few requirements for the proposed software system can grossly be missed if the stress is simply on generating the concept of the system in problem domain. Examples of such requirements include auto-renewal of patient's registration, housekeeping of patient's details and prescriptions on frequent basis as these are going to consume a lot of space in database server and degrade the performance of software.

   A further step to generating system concept and going on further to visualize the system in solution domain will definitely bring out such requirements. Imagining the behavior of hospital system in solution domain where there is no patient's card. In that case, hospital will itself be maintaining the huge lot of data that was, otherwise, with patients. Similarly, in the absence of cards, auto-renewal of registration if treatment continues is an implicit requirement. Such requirements generally surface as a result of deployment of computer-based system and as the users of the software system start getting hold of the software system in the form of Emergent Requirements and often, result in prolonged software development life cycle.

3. Audit requirements – Consider the manual processing of purchase department of any manufacturing concern. The purchase cycle begins with inviting quotations from vendors for the requested items. One of the vendors is, then, selected and order is placed. On the delivery of requested items and payment processing, orders are then updated and the items are brought on charge. In this scenario, an important requirement may be missed if the requirement engineer goes by just drawing the functional model of the problem domain. If the development team develops the solution as per conceptual functional model, the solution is going to lose the history of an order status updation. A thorough visualization of the problem in solution domain (where there is just one record corresponding to the order) will bring out this fact in the very beginning that an order is processed by various people, each playing an important role and it's important to keep a chronological sequence of 'who' did 'what' to the order. Consequently, an audit trail of the order is required in the solution domain.

The above examples explain which type of dormant (dormant till conceptualization) /emergent (emergent with user's understanding of the solution system) requirements can be drawn upon in the requirement analysis phase itself using visualization of the system in the solution domain. It's a common error to think about the system the way it behaves in its problem domain. What visualization requires is a keen look at the aspect of the behavior of objects in problem domain as that can be mapped easily to its functioning in solution domain; looking for commonalities in behavior' looking for disjoints in behavioral/informational flow; looking for compositions in those flows.

4.1 Different from errors of conception

As we have seen before, visualization is inherently performed, as part of conceptualization but the focus in that case is more on conceptualizing the problem-domain. This is where the difference lies between the errors of conception and errors of visualization. When the problem is conceptualized, it is looked at from the perspective of the customer as he views the system processing in current environment or his problem space. On the contrary, when the system is visualized, it is looked from the perspective of the end-user interaction with system in solution domain. The user interactions in the solution domain are definitely going to be different the way user interacts with the system in problem domain.

Another source of problem in conceptualization is talking to the wrong person for information. Even while talking to the subject-matter expert, there are chances of poor understanding of the problem space. As a whole, error of conception fall under the domain of the problem – errors arising owing to poor understanding of problem space domain. On the contrary, errors of visualization arise due to absence of, or poor mapping of problem space to solution space domain. The errors of visualization fall under the domain of solution.

Errors of conception usually result in a system that is far away from the required system. But, if errors of conception have been covered and the errors of visualizations are present, then such errors result in a system that is not too far away from the user requirements but needs to be enhanced to improve usability or may be, performance of the system

4.2 Symptoms of errors of visualization

The presence of errors of visualization can be found through these symptoms:

1. Difficulty in using the system

   When the system is meeting the requirements and users have started using the system but the users are finding difficulty in using the solution system, then it indicates the presence of errors of visualization. Such levels of difficulty may include too many numbers of clicks to find a piece of information or too lengthy forms asking unnecessary or irrelevant details to fill out for saving certain details.

   This is usually a common symptom of visualization error and generally arises when usability of the system in solution space is not given attention while designing the solution, as it was not specified so in requirement specifications. The technical aspects or look-and-feel of the solution may be very good in terms of solution developers but it might fail to meet the user expectations if user doesn't find himself comfortable in using the system

2. Emergent requirements

When the system is under user acceptance testing or may be in production and there are numerous instances of enhancement requests, then it is an indicative of the fact that users are gaining better understandability of the system. And, they are now coming up with more and clearer requirements, as they are getting more comfortable with the software system. An early visualization of the user interaction with the software system can drastically reduce the number of enhancement requests.

3. Repeated occurrences of similar information in the system

In real-time environments, similar types of information lies at more than one places like departments of an organization keep employee data in the respective formats they need or the data that is collected at one place is then presented as report at another place. And, absence of visualization in solution domain results in simply automating the tasks and information displays as in the problem domain. There is hardly any value-add to the system in problem domain when solution is mapped as it is. Information still remains in the form of scattered data and it would be mistake to call such a scattered and raw data as information. A value-add to the problem-domain would be categorizing and classifying the data as information.

4. Performance issues after 2-3 years of usage of the system

The information stored in the software solution may be and will be different the way it is stored in its problem domain. Consequently, the sequence of steps to be followed for performing a task in problem domain will be different in solution domain. Imitating the same sequence will not make the task any simpler and no value-add and might also result in additional number of data fetches resulting in performance issues sooner due to more number of I/Os. Visualization in terms of criticality of operations and the amount of speed at which data would be processed and correspondingly time taken can help in containing the performance issues at an early phase of software development life cycle.

4.3 Solution strategy for errors of visualization

First, to avoid errors of visualization, perform this additional step of visualization of the system in solution domain in addition to system conceptualization that is generally being done in problem domain. Visualization should not be mixed with conceptualization of problem domain. During requirement analysis the system models, either in the form of data flow diagrams or object-oriented models, should be supported or supplemented by aspect-oriented models. Aspect–oriented models that reason about aspects in solution domain can help in preventing errors of visualization. The values that aspect-oriented models provide over data flow diagrams, process diagrams and object-oriented models include:

1. Abstraction: Abstract away from the details of how that crosscutting concerns, or aspect of the requirements of problem domain, might be scattered and tangled with the functionality of other modules in the solution domain of the system.

2. Modularisation: Keep crosscutting concerns, i.e aspects separated regardless of how they affect or influence various other modules in the system, so that we can reason about each module in isolation. The idea is more or less similar to what we know in object-oriented paradigm but in this, the unit of partitioning or modularity is an aspect and not merely an entity. Aspect is concerned about systematic areas of application or a characteristic of system for which components/services are required. Aspect covers a broader characteristic as compared to an entity and still, supports abstraction of these aspect (information-hiding) as well as modularity (problem-partitioning).

3. Composition: The various modules need to relate to each other in a systematic and coherent fashion so that one may reason about the global or emergent properties of the system. The aspect-oriented approach stresses on compositional reasoning – how modules relate and communicate with each other.

5. CONCLUSION

The paper is an attempt to segregate out system conceptualization in problem domain and the system visualization in the solution domain and to highlight the issue/errors that arise because of poor visualization of the system in solution domain at the time of requirement analysis and specification phase only. The kinds of problems grouped under errors of visualization have been known for long but generally have been attributed to client misunderstandings, new requirements or hardware issues. But, a careful study of these indicates that majority of client misunderstandings are because of poor visualization of how an end-user would be interacting with the system. Similarly, emergent requirements can be reduced to a large extent by exploring client's explicit and implicit expectations during requirement gathering and analysis time. The spectrum of such an exploration of requirements can be widened more by system visualization in its solution domain – how that would be working, what new requirements might emerge because of the proposed system. Aspect-oriented requirement engineering, that focuses on identifying and reasoning about system characteristics and managing conflicts arising due to tangled representations, cross-cutting requirements, can give a direction to contain errors of visualization in the requirement analysis itself before that these errors surface at later stages. And, reducing these errors can help in achieving higher client satisfaction and lower costs of software solutions – a dream to be realized!

6. REFERENCES

[1] Bittner, Kurt, 'When Requirements Go Bad', http://www.ivarjacobson.com/resources.cfm

[2] Chang, Betty H.C. and Atlee, Joanne M., "Research Directions in Requirements Engineering", International Conference on Software Engineering 2007 – Future of software engineering

[3] Dardenne, Anne and Lamsweerde A.L. and Fickas, Stephen, 'Goal Directed Requirement Acquisition', Proceedings of the 6th international workshop on Software specification and design

[4] Easterbrook, Steve and Nuseibeh Bashar . 'Requirements Engineering: A Roadmap' , Proceedings of the Conference on The Future of Software Engineering - 2000

[5] Hausmann, Jan Hendrick and Heckel, Reiko and Taentzer, Gabi. 'Detection of Conflicting Functional Requirements

in a Use Case Driven Approach.' , Proceedings of the 24th International Conference on Software Engineering - 2002

[6] Rashid, Awais and Chitchyan, Ruzanna 2008. 'Aspect-Oriented Requirements Engineering: A Roadmap', ICSE 2008

[7] http://www.resg.org.uk/EAslides/Awais%20Rashid%20Slides.pdf.