Requirements Analysis: A Review

Joseph T. Catanio
LaSalle University
Mathematics and Computer Science
Department
Philadelphia, PA 19141
1.215.951.1142
catanio@lasalle.edu

Abstract - **Many software organizations often bypass the requirements analysis phase of the software development life cycle process and skip directly to the implementation phase in an effort to save time and money. The results of such an approach often leads to projects not meeting the expected deadline, exceeding budget, and not meeting user needs or expectations. One of the primary benefits of requirements analysis is to catch problems early and minimize their impact with respect to time and money.**

This paper is a literature review of the requirements analysis phase and the multitude of techniques available to perform the analysis. It is hoped that by compiling the information into a single document, readers will be more in a position to understand the requirements engineering process and provide analysts a compelling argument as to why it should be employed in modern day software development.

I. INTRODUCTION

Requirements analysis, as it relates to software projects, is the process of studying, determining and documenting user needs and expectations of the software system to be designed that solves a particular problem. The process is referred to as requirements engineering and entails feasibility studies, elicitation, specification, and validation process steps. The process generates software requirement documents that capture what is to be implemented by fully describing the software system's functionality, performance, design constraints, and quality attributes. Precisely documenting what to build helps to reduce uncertainty and equivocality [1]. Determining and documenting the requirements of an information system, is arguably the key to developing successful information systems [2]. Not getting the correct final software system requirements at the project onset is largely responsible for the cost and schedule overruns plaguing the information system development process [3][4][2]. Table 1 shows that the earlier in the development process an error occurs and the later the error is detected, the more expensive it is to correct [5].

Table 1 Relative Cost to Repair a Software Error in Different Stages

Stage	Relative Repair Cost
Requirements	1-2
Design	5
Coding	10
Unit Test	20
System Test	50
Maintenance	200

Thus, performing software requirements analysis at the project onset helps to identify and correct problems early. Consequently, the relative repair cost is low and reduces the chances of project cost overruns.

Much of the literature describes the requirements analysis process as three sub-processes and compares it to an engineering methodology [6][7][8][2][9]:

- Elicitation
- Specification
- Validation & Verification

Each sub-process addresses the problem definition aspects from different angles during the requirements creation process to collectively describe the software system to be built. The software specification documents generated, as a result of the analysis, captures the user needs and describes the operation of the proposed software system. The software system description is generally comprised of three types of documents [6][9][10].

- Functional Requirements
- Non-functional Requirements
- Design Constraints

Functional requirements describe what the software system should provide and how it should behave. In contrast, non-functional requirements are not concerned with specific functionality delivered by the system. Instead non-functional requirements relate to system properties such as

T. Sobh and K. Elleithy (eds.), Advances in Systems, Computing Sciences and Software Engineering, 411–418.

reliability, response time, memory space, portability, maintainability, ease-of-use, robustness, security, and re-usability [10][9]. Design constraints describe the requirements that are specific to characteristics of the problem domain that do not easily categorize into the other two types of documents.

The hardest single part of building a software system is determining and documenting precisely what to build [12][10]. The difficulties of documenting and specifying software requirements are primarily due to human problem-solving limitations [13]. Davis also points out that these limitations are because human beings have a limited ability to process information. Much of the time humans leave or filter out information to prevent information overload. To help include all the available information, methodologies have been developed to provide a systematic repeatable approach to the description and development of software requirements and systems [14][15][16][17][10]. Methodologies help to structure the problem and solution domain into a collection of smaller sub-problems. These sub-problems are then individually described and eventually implemented. The aggregate of all the components, describe the entire problem domain. This approach helps to divide large complex problems into smaller, more manageable components. In addition, complexity is reduced since the amount of information that a sub-component must consider is also reduced. Therefore, the documentation of sub-problems help achieve a goal of requirements analysis, namely to understand and capture what is to be solved in a component-oriented manner using all available information.

II. ELICITING REQUIREMENTS

Requirements elicitation is the process of identifying the application domain by determining the desired software system functionality. This activity should involve many different kinds of people that have a stake in the system being built. These stakeholders work together to define and scope out the application domain. Each participant is a stakeholder and represents a different interest in the project. These interests are dependent upon the role the individual performs. Therefore, the elicitation process should include all people that are either directly involved with the project or indirectly affected. Once the stakeholders are identified, the process enters the problem domain description phase. The description is realized either in an independent or a team-oriented collaborative manner. Gause points out that many organizations reinforce a negative image of cooperative work, encouraging instead competition among employees by such devices as individual achievement awards [18]. However, software development projects should not be realized alone and need the diversity and ability that a collaborative team approach can provide. Much of the literature supports the concept that groups generate more and better solutions to identifying, describing, and solving a problem [19][20][21][22][10]. The general consensus is that problem definition is likely to be more complete when realized by participation in a collaborative team environment.

The literature identifies many different techniques that are possible to elicit requirements of a computer or information system in both a group team and single person team approach [23][24][25][9]. Some examples of a non-group approach are:

- Introspection
- Questionnaires
- Interviews
- Protocol Analysis
- Ethnography

The introspection technique attempts to elicit requirements of the desired computer or information system by having the development team members individually imagine the system they want. Thus, many perspectives and interpretations will result from introspection. Many viewpoints help to identify all aspects of the problem domain, but these do not necessarily reflect the needs of the end-users of the system. The literature does not consider this technique a practical way to elicit requirements due to its apparent lack of end-user involvement.

Similar to introspection, questionnaires and interviews attempt to elicit requirements by asking questions in a non-group oriented fashion. Questions are presented to individuals in either a written or verbal format, and the answers recorded. Although this is a systematic process Suchman argues that these approaches lead to multiple interpretations in both the questions and the answers [26]. To reduce misinterpretations, the interview technique can be extended to permit dialog between the interviewer and interviewee. However, the literature indicates that the interview process usually involves assumptions concerning the interaction among participants. Goguen strongly argues that assumptions and misinterpretations that can result from questionnaires and interviews make this technique impractical to elicit computer and information systems requirements [23].

Protocol analysis is a process in which a person performing a task does so aloud while his or her thought processes are observed and recorded. This represents direct verbalization of specific cognitive processes [27]. This technique helps to understand an individual's approach to problem solving. Therein lies the problem; it is not a team-oriented approach. The project team consists of many different kinds of people operating in different roles. Some of these individuals have knowledge about the business and organizational needs,