Revisiting the Meaning of Requirements

Zhi Jin\textsuperscript{1,2} (金 致)

\textsuperscript{1} Academy of Mathematics and System Science, Chinese Academy of Sciences, Beijing 100080, P.R. China
\textsuperscript{2} Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, P.R. China

E-mail: zhijin@amss.ac.cn

Received May 12, 2005; revised October 11, 2005.

Abstract Understanding the meaning of requirements can help elicit the real world requirements and refine their specifications. But what do the requirements of a desired software mean is not a well-explained question yet though there are many software development methods available. This paper suggests that the meaning of requirements could be depicted by the will-to-be environments of the desired software, and the optative interactions of the software with its environments as well as the causal relationships among these interactions. This paper also emphasizes the necessity of distinguishing the external manifestation from the internal structure of each system component during the process of requirements decomposition and refinement. Several decomposition strategies have been given to support the continuous decomposition. The external manifestation and the internal structure of the system component have been defined. The roles of the knowledge about the environments have been explicitly described. A simple but meaningful example embedded in the paper illustrates the main ideas as well as how to conduct the requirements decomposition and refinement by using these proposed strategies.

Keywords problem decomposition, requirements analysis, requirements refinement

1 Introduction

“Requirements specification” is one of the frequently used terms in software engineering. But what are the requirements, what is the meaning of the requirements and how to make development of the specification to meet the requirements? These are long lasting questions since the birth of software engineering. In some of the well-known papers, e.g., [1, 2], in this area, the authors indicated that the requirements are located in the reality, i.e., in the environment, which is distinguished from the software system. It is argued that the relationship among the three frequently used terms, i.e., environment (\(\mathcal{E}\)), requirements (\(\mathcal{R}\)) and specification (\(\mathcal{S}\)), in requirements engineering can be revealed as an entailment:

\[ \mathcal{E}, \mathcal{S} \vdash \mathcal{R} \]

which means that if a software system whose behavior satisfies \(\mathcal{S}\) is installed in the environment described by properties \(\mathcal{E}\), then the environment will exhibit the properties described in \(\mathcal{R}\). Although the relationship is not an implication, it basically captures the essence that \(\mathcal{R}\) can be satisfied if a software system specified by \(\mathcal{S}\) is installed in the environment (\(\mathcal{E}\)).

Furthermore, the authors indicated that the requirements contain two parts of contents: the designation part and the definition part. Designation, which is always informal maps from the fuzzy phenomena to a formal language, singles out a phenomenon of interest in reality which tells us how to recognize it and gives it a name. Definition then gives a formal definition to a term that may be used in other descriptions. Since software is basically a symbolic system, making designation was recognized already and was called the symbol grounding\textsuperscript{[3]}. In other words, a designation is grounding a term represented by formal symbols to a phenomena of interest in the reality. This is the basis of modelling a real application. And giving a definition is for defining a higher (or a more abstract) level term by using grounding terms or any other lower level terms. That is straightforward to designate complex or abstract phenomena.

This is a powerful view in requirements engineering. However, as we know, the main tasks of requirements engineering are recognizing and expressing the requirements and developing a specification (which may be called as software requirements specification, or SRS for short) as the baseline for the subsequent software development. Now, the environment is there as the reality, and the requirements, originating from the environments, are desired or optative conditions (or assertions) over the phenomena in the environment, and can be expressed by using designations and definitions. Then, the question arises: how can the SRS be developed, or be derived from the requirements?

Obviously, there does not exist direct correspondence between them in the above mentioned entailment. There is a gap here. What can be used to bridge the gap? It is claimed that domain knowledge can be used for recognizing the requirements of environment constraints, of unshared phenomena and of future reference\textsuperscript{[2]}. But no argument has been made to specify the necessary domain knowledge and this important issue still remains un-addressed at present.

We suggest that the process from identifying the original requirements to obtaining a software specifica-
tion can be fractionized into the following continuous refinement:

\[ E \iff R = R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_n = S, \]

where \( R \) is the preliminary expectation of the effect on the environment the desired software system will be installed about. And for \( 1 \leq i < n \), we have, \( R_{i+1} \) is a refinement of \( R_i \), and finally \( R_n \) could be the SRS (S) of the desired software.

This paper is trying to reveal the meaning of the requirements by focusing on a promising continuous process for the requirements refinement. In the following, we first point out that we should take a system thinking to software systems in Section 2. Then we formulate two aspects of the requirements in Section 3 and Section 4. This formulation emphasizes both of the external manifestation and the internal structure of a software system. The other important point is that the two aspects depend on each other during the process of the requirements refinement. Based on this formulation, a requirements refinement model is given in Section 5. Some properties of the requirements refinement model also are given to help clarify the meaning of the requirements and the tasks of the requirements refinement. In Section 6, we compare our refinement model with some other representative requirements refinement processes and conclude the whole paper.

2 System Thinking on Software Systems

The outcomes of software engineering are software systems. As the first step of software engineering, requirements engineering produces the models and the specifications of software systems. But what is a system? Which properties ought it have? From the literature, we can obtain its properties as follows.

1) Wholeness and Openness. A system is some part of a reality. It is separated from its environments by a boundary and can be observed to interact with its environments.

2) Structure and Emergency. A system usually contains some subsystems. Such systems have interesting emergent properties.

These two properties are also suitable to software systems. Then, we can derive some properties of a software system. From the viewpoint of the wholeness and the openness, some concepts, such as the system environment, the boundary, the interaction and so on, are important in representing a system, so the same in a software system. In fact, to recognize the requirements of a desired software system, the environment does play an important role. Experience reveals that when a software system is desired to be built, it is the effects the software system can bring about to the environment that are wanted. That is why the requirements normally are not directly concerned with the software system, but with the environment with which the software system will interact, and in which the effects of the software system will be observed and evaluated. The environment, the interactions and the effects are the most important aspects of the requirements. That is the external manifestation of the desired software system.

However, from the viewpoint of the structure, there is another aspect, i.e., the internal structure. When we analyze the real world problem and try to get a solution, an important principle we may adapt is “decomposition”. The result of decomposition is to get a group of system components. This group with the relationships among its components is something like an internal structure of the system.

More precisely, the structure of life systems may inspire us with an idea that a system (which is also a component) could be a hierarchical cohesion of independent semi-autonomous components. In this cohesive hierarchy, a system and/or a component could be both a whole and a part. Maciaszek borrowed the word “holon” from Arthur Koestler’s exposition of the living systems for referring to this kind of components. By applying this point of view to the issues on the requirements of a desired software system, we could obtain the following points.

- When we talk about the requirements of a system, we just take the system as a whole component, a “holon” as a whole. This holon encapsulates the complexity of its internal structure and/or its subordinate hierarchy.
- From the top-down view, each holon could be decomposed into a set of smaller holons. From the bottom-up view, a group of holons may constitute an up-level holon. The relationships among these part holons decide the structure of the whole holon.
- For any two adjacent levels in this holon hierarchy, the group of holons on the lower level should be derived from their super-holon, and on the other hand, the holon on the upper level could be realized by the group of its sub-holons on the lower level. For each holon, its external manifestation and its internal structure are restricted by each other.
- In a broader sense, this hierarchy is endless, considering that the environment the system stands in is also a hierarchy of holons. That is, in this hierarchy of holons, we cannot say which holons are the roots, or which are the leaves. In other words, with regard to each holon, there is at least one “larger” holon outside it, and more than one “smaller” inside it. The requirements of each system may be a part of the universal “holon” hierarchy.

The last point may be out of the scope of this paper. In the rest of this paper, we only consider the holon hierarchy of the desired software system. And also for the sake of terminological uniformity, we will use a unified name for both of the systems and the components, the soft holons or the s-holons for short.

Thus, it is reasonable to conclude that the external manifestation and the internal structure might be the two dimensions for us to depict the models of a software system. Moreover, they are not separated from