Chapter 25
Knowware-Based Software Engineering: An Overview of its Origin, Essence, Core Techniques, and Future Development

RuQian Lu
Chinese Academy of Sciences, China & Peking University, China

Zhi Jin
Peking University, China & Chinese Academy of Sciences, China

ABSTRACT

*The first part of this chapter reviews the origin of knowware-based software engineering. It originates from the authors' experiences in finding new techniques for knowledge-based software engineering while performing PROMIS, a continuing project series from the 1990s. The key point of PROMIS is to generate applications automatically by separating the development of domain knowledge from that of software architecture, with an important innovation of acquiring and summarizing domain knowledge automatically based on the pseudo-natural language understanding techniques. However, during PROMIS development, the authors did not find an appropriate form for the separated domain knowledge. The second part of the chapter briefly describes how the authors came to the concept of knowware. They stated that the essence of knowware is its capacity as a commercialized form of domain knowledge. It is also the third major component of IT after hardware and software. The third part of the chapter introduces the basic concepts of knowware and knowware engineering. Three life cycle models of knowware engineering and the design of corresponding knowware implementations are given. The fourth part of the chapter introduces object-oriented mixware engineering. In the fifth part of the chapter, two recent applications of knowware technique regarding smart room and Web search are reported. As a further development of PROMIS, the sixth part of the chapter discusses knowware-based redesign of its framework. In the seventh part of the chapter, the authors discuss automatic application generation and domain knowledge modeling on the J2EE platform, which combines techniques of PROMIS, knowware, and J2EE, and the development and deployment framework (i.e. PROMIS/KW**).*

DOI: 10.4018/978-1-4666-6026-7.ch025

1. EXPERIMENT OF SEPARATING APPLICATION KNOWLEDGE DEVELOPMENT FROM SOFTWARE DEVELOPMENT: THE ORIGIN

The practice of software engineering shows that most failures of software development are caused by failure of requirement analysis, and the reason for that falls upon lack of good cooperation between users and software engineers. Users, usually being unable to exactly and clearly state their requirements, often change their requirements freely during the process of software developing, which makes it difficult for software engineers to perform a proper requirement analysis and to guarantee the accomplishment of the developing job successfully.

At present, requirements analysis researchers and practitioners use either formal methods or semi-formal methods with different requirement specification languages. The advantage of formal methods is that they provide strict guidance to software engineers or programmers for writing requirement specification with the assumption that user requirement is complete and precise (Chakraborty et al., 2012; Vassiliou et al., 1990; Mulopoulos et al., 1999; Kundu, 2007; Yu, 1997; Wang et. al., 2001; Castro et al., 2002; Fuxman et al., 2004); otherwise they cannot give any help, no matter how perfect they are in theory. One of the solutions to this problem is involving users into the process of software development as much as possible, so that they can realize the differences between the software under development and that they really need, or between the drafted requirement specification and their real requirements. In this way, users can find the software design deficiencies at the earliest time. However, because of the big difference between the knowledge backgrounds of software engineers and users, formal methods often cause serious problems of bad communication between them. The changing nature of requirements during software design and development process makes the situation even worse.

We believe that it is not enough to only attract users to join the development process, but we should also give the key of developing software to users, whenever it is possible. That is to let users themselves define, design, develop, maintain and modify their software. This is possible for some kinds of software, for example, management information system (Mansour et al., 2009; Jarke et al., 1990; Engels et al., 1995; Engels et al., 1992; Vilkomir et al., 2004; Bhuiyan et al., 2007; Monroe et. al., 1996). To achieve this goal, we must remove from users the burden of learning and mastering the knowledge about software development and also the burden of requirement analysis with formal methods. One way for achieving this is using knowledge. As a result, we proposed a knowledge-based software engineering method, KISSME (Knowledge Intensive Software System Manufacture Engineering), and developed a tool for supporting this method that is named as PROMIS (PROtotyping MIS)(Lu et al., 1994, 1995, 1996 (journal), 1996, 1997 (Spain), 1997, 1998, 1998 (journal), 1999, 2000, 2000 (book), 2002, 2003, 2003 (journal); Jin et al., 2003). The essence of this method is that by using a large knowledge base to support software development, users do not need to master knowledge of software development or requirements analysis of related domain. This approach is also made possible by a requirement description language BIDL (Business Information Description Language). This language is in pseudo-natural style and contains only expressions and terminology of the application domain, without any jargon from the software engineering area. Users who are not software professionals can use this language to describe their business. This description will then be transformed into the final program under the support of a domain knowledge base throughout the whole lifecycle of application development.

The following process has been used to design a pseudo natural language like BIDL: