ISSN1000-9825 Journal of Software 软件学报 2000,11(7):863~870

A New Methodology for User-Driven Domain-Specific Application Software Development*

LI Ming-shu

(Lab. of Computer Science Institute of Software The Chinese Academy of Sciences Beijing 100080)
E-mail: mingshu@ox.ios.ac.cn

Abstract This paper presents a new methodology for application software development, named as "user engineering". It is a user-driven domain-specific application software development methodology based on component-based software architecture, strengthening driving effect of users to make software development as a detailed definition process rather than a coding process only. It indicates an effective way to meet increasing application software requirements.

Key words Software engineering (SE), requirement engineering (RE), user participation, component, software architecture, reuse, user engineering (UE).

The term of "SE (software engineering)" was named at a 1968 NATO (North Atlantic Treaty Organization) Conference in Garmisch, Germany[1], and a set of techniques for software development was introduced to combat the "software crisis".

However, the growing size and complexity of systems have revealed many shortcomings of existing software engineering practices. This in turn raised interest in component-based and architecture-driven software development[2]. Components are large-grain functional units of system and architectures are blueprints describing system composition. Some related researches include module interconnection languages[3], module interface specification and analysis[4], megaprogramming[5,6], software generators[7], architecture description and configuration[8,9], Polylith software bus[10], commercial off-the shelf[11], and domain-specific software architectures[12,13]. The shift toward developing systems from components has been more evolutionary than revolutionary. It has its roots in accepted architectural principles such as layering, modularization, and information hiding. But it also introduces its own principles and concepts and presents new challenges.

1 Software Engineering, Requirements Engineering and User Participation

SE is the application of scientific principles to[14]: (1) the orderly transformation of a problem into a working software solution, and (2) the subsequent maintenance of that software through the end of its useful life. People and projects that follow an engineered approach to software development generally pass through a series

* This research is supported by the National Natural Science Foundation of China (国家自然科学基金, Nos. 69773023, 69896250-3). **LI Ming-shu** was born in 1966. He is a research professor of the Institute of Software, the Chinese Academy of Sciences. He received a Ph.D. degree from the Department of Computer Science and Engineering at Harbin Institute of Technology in 1993. His current research interests are software engineering in particular requirements engineering and component-based software technologies, real-time systems and embedded operating systems, Internet/Web technology and applications, software quality and SE standards, intelligent agents and multi-agent systems, distributed artificial intelligence and cooperative problem solving.

Manuscript received 1999 05 12, accepted 1999 10 29;

of phases: software requirements, preliminary design, detailed design, coding, unit testing, integration testing, system testing, delivery, production, deployment, maintenance and enhancement. They can be mainly divided into five steps: requirements, design, coding, testing and maintenance.

The most important step in software development is to analyze, understand, and record the problems that the users are trying to solve, i.e., how to acquire the requirements from the application domain, especially from the end-users, and how to describe and analyze them completely and exactly. Requirements describe what the users want from the software. They should be helpful and understandable to end users, serve as a basis for design and testing, be suitable to the application, and encourage thinking in terms of external and not internal system behavior[15]. Most faults found during testing and operation result from poor understanding or misinterpretation of requirements. Analysis faults are expensive because it costs 100 times more to correct faults found after delivery than during analysis[16]. In spite of the progress in analysis techniques, CASE tool support, prototyping, and early verification and validation techniques, software development still suffers from poor requirements acquisition and analysis[17].

In response, the field of RE (requirements engineering) has experienced a sudden growth spurt in 1990s. RE is the science and discipline concerned with analyzing and documenting software requirements. It involves partitioning system requirements into major subsystems and tasks, allocating those subsystems or tasks to software, and transforming these allocated system requirements into a description of software requirements and performance parameters through the use of an iterative process of analysis, design, trade-off studies, and prototyping[18].

User participation in the RE process is hypothesized to be necessary for RE success[19]. A considerable amount of empirical research on the relationship between user participation and software development success has been conducted. Furthermore, user participation and influence are expected to increase the likelihood of user acceptance of the solution and of improved system quality[20~22]. Especially due to the increasing complexity of organizational life, it is difficult for analysts alone to design a system that will meet user requirements, and therefore user participation in system development is critical. In addition, the lack of user participation can lead to "many faults and economic disadvantages"[23] and is considered to be at least partially responsible for considerable increase in the cost of some systems development[24]. That means we have to move our attentions to the end-users themselves in application software development.

Recently, the competition among those famous software companies in the world has shifted focus to application software markets. The essential goal for some of them to launch a grueling drive on midware is application software systems. All of them have to closely integrate with application domains. It's thus clear that both trends of research and software industry ask a definite requirement for users to participate in the development of application software systems.

An effective way we think to meet increasing application software requirements is strengthening the driving effect of users to make software development a detailed definition process rather than a coding process only. Consequently, this paper presents a novel idea of user engineering methodology for development of user-driven domain-specific application software based on component-based software architecture.

2 Why a New Methodology and What's in a New Name?

2.1 Why a new methodology?

A software development methodology can be defined as[18]: (1) an integrated set of software engineering methods, policies, procedures, rules, standards, techniques, tools, languages, and other methodologies for

analyzing, designing, implementing, and testing software and (2) a set of rules for selecting the correct methodology, process, or tools for use. Methodologies to support software development appeared in great number and variety in the past, such as "stepwise refinement"[25], "structured programming"[26], "logical construction of programs"[27], "structured analysis and design"[28], "principles of program design"[29], "object-oriented methods"[17], "formal (mathematical) methods"[30], "plug-and-play"[2], and so on.

For many years, the acquisition of a new computer-based system was a significant one-time organizational investment that replaced entirely manual processes. As a result, we could talk about such things as "the system" (application software), and "the customer" (user), with the assumption that the application software was to be specified to meet the needs of the user and that the requirements could be "elicited" from the user and specified on paper. Today, things are far less simple. Now we develop "application","features", "services", and even "applets" that run on top of information infrastructures and enhance existing systems. The user often becomes a user (customer) only after a shrink-wrapped product has been shipped. Contractual development is no longer universal, and software development has therefore become partly services provision or integration, and partly user (customer) product design.

In addition, a basic cognition can be achieved in an application software development: (1) the (artificial) software system should be based on the current (human) system, and better than the current system; (2) each user clearly knows what things he (she) should do every day and can give a clear illustration; (3) each user clearly knows the relationships with other user(s); (4) almost all the users do not know exactly about what computers can do for them and what they should ask computers to do for them. That means the user(s) can do and have to do many things for the software development, but not everything.

These changes introduce exciting challenges to traditional SE practice and research. We can no longer assume the existence of a user with unquestioned legitimacy, but rather need to talk in terms of a variety of stakeholders; and can no longer assume groundless developments of systems and their "insertion" into an environment, but rather must deal with requirements for evolving systems. And as applications become more customizable by users or administrators, some of the development process becomes delegated to communities of users. A new methodology for application software development should be discussed to meet such a challenge. User Engineering (UE) in this paper is just an attempt.

2.2 What's in a new name

The name "User Engineering (UE)" means a new branch of traditional SE, even not everybody will agree that the epithet "engineering" is deserved. Can one "engineer" user(s), when user(s) is(are) only the customer(s) of an application software? Some people may object to the connotations that the term "engineering" suggests. Actually UE is not about the engineering of users themselves, but about the systematic production and manipulation of user requirements within a professionally responsible approach to application software development. By using the term "engineering", application software will be developed according to some user-driven engineered approach.

UE lies at the intersection of several previously existing fields without being a fully-fledged part of any of them, such as traditional SE and recent RE, HCI (human-computer interaction), CSCW (computer supported cooperative work) and AI (artificial intelligence).

One of the most important concepts in UE is user participation, which can be defined as referring to the behaviors and activities that the users perform during the software development. The other similar term is user involvement[31], which is defined as referring to the subjective psychological state of the users, and consists of "the importance and personal relevance that users attach to a particular system or information system in general". In this paper, the concern is with user participation.

In a sense, a user can be taken as an object or an agent. However, we only pay attention to the actions made by each user and ignore who is who. So we can apply object-oriented technologies and component-based software reusable technologies to reach a consistence between the problem description space and the problem solving space by some reasonable transfer. End-users (customers) have been the actual designers for application software systems. The dual identities of an end-user to be a designer and a user for an application system can be integrated, i.e., end-users have finished the development of an application software system as soon as they finished the problem definition process. As the action made by end-users in the above development is active, we call it "user-driven". Almost all the terms in the traditional SE life cycle model can be continued to use, requirement representations of end-users have, however, been used through all steps in the development of application software systems. The description and analysis for user's requirements have been entrusted to a new implication in the development of application software engineering. This kind of user-driven software engineering can be called user engineering. The key points of UE are emphasizing the effective participation of end-users in the development of application software systems, and using and integrating all current techniques, technologies and industries results.

Besides user participation, the other important aspects of UE are domain analysis (i.e., analyzing the common features and their variations across the current and future products in the application domain), software architectures (i.e., defining the high-level partitioning of the software into components) and component-based software reuse (i.e., organizing the design of each component using the commonalties and variations identified in the domain analysis).

2.3 Domain analysis

Domain analysis is seen by some in the reuse community to be a key process for achieving systematic and large-scale reuse. However, the success of a domain analysis is largely dependent upon how well the domain analysis process is carried out[32]. Domain analysis was first defined as "a process by which information used in developing software system is identified, captured, and organized with the purpose of making it reusable when creating new systems"[33]. It entails scoping the domain, gathering the relevant product information, identifying and classifying the features, and analyzing their commonalties.

Domain analysis is similar to system requirement analysis, but the result of this analysis, which is a domain model, is more generic and more oriented to real world processes than software system requirements. Domain models[34,35] describe a real world situation in terms of entities that produce and use information, types of the information, information flows among entities, responsibilities of entities and operational scenarios. For a well understood domain, the domain model can be created prior to implementing systems. But for unexplored domains, we may need to implement one or more software systems before we can build a domain model.

The typical domain analysis methods in literature are Feature Oriented Domain Analysis[36,37], Organizational Domain Modeling[38], Object Modeling Techniques[39], Shlaer Mellor Method[40], industrial methods[41,42] and so on.

2.4 Software architectures

The description of an architecture requires different views[43,44]. The architecture is described using four different views: (1) the object view provides the logical view of the architecture described objects and their relationships; (2) the layered view describes how components are organized in a hierarchy of layers, each one providing a well defined interface to the layers above it; (3) the task view analyzes and describes the concurrency and synchronization aspects of the architecture; and (4) the scenarios describe interactions between external actors (e.g., the end-user) and the system components. The steps for mapping domain models to architectural representations include identifying the application layer subsystems, extending the object model, defining the

layered view and the task view, and developing the scenarios.

2.5 Component-based software reuse

A component means a fragment of a software module—— a piece of a program, a subroutine, an object, ... one or more sentences expressed in any language[45,46]. Generalized components are not specific code fragments; each contains formal parameters and structures that allow it to be systematically modified to become any of a possibly infinite set of specific components. In this sense, generalized components transcend physical ones. What is impossible in the world of physical parts is natural and appropriate for software parts. Usability is essentially a run-time concept. A context of use is a special run-time architecture in which special software components are embedded. Object orientation exemplifies run-time architectures which enable high usability. To design highly usable systems means focusing on effective tradeoffs among usability's key properties: functionality, ease-of-use, and performance.

A component is a significant functional unit of a system. It can be anything from the high-level business function like "customer order processing" to a low-level technical task such as "roll back transaction". A component can be either atomic or composite. A composite component contains other components as its sub-components, which are themselves atomic or composite. Therefore, the architecture of a complete system can be described as a hierarchical organization of atomic and composite components. Moreover, a component is a reusable and sharable unit, i.e., it can be handed from one project to another.

The functionality of a component is defined by its interfaces. The ideal reuse technique is a component that exactly fits the application needs and can be used without being customized or forcing the user to learn how to use it. However, a component that fits today's needs perfectly might not fit tomorrow's. The more customizable a component, the more likely it is to work in a particular situation, but the more work it takes to use it and to learn to use it. Frameworks are a component in the sense that venders sell them as products, and an application might use several frameworks bought from various venders. Frameworks are much more customizable than most components, and becoming more important[47]. Systems like OLE, Open Doc, Java Beans, and DCOM are frameworks.

3 The Model and Tool Supporting User-Driven Domain-Specific Application Software Development

A model supporting the above idea has been presented, based on a general AI problem solving model, Function Module System[48]. It is called UserMODEL and can be written as,

$$\text{UserMODEL} = (\text{Users}, \text{Relations}, \text{Messages}, \text{Characteristics}, \text{Domain}).$$

In the model above, "Users" is a set of users, and an user is an entity of an information system end-user in the real world which possesses the ability of either carrying out some activity or accepting some activity, also possesses some specific characteristics, activity style and knowledge. All the users know how to finish their specific activities and how to get those necessary conditions and resources if needed. "Relations" is a set of relations among the users. It describes the model of static interaction and the structures of the information system. "Messages" is a set of messages transmitted among the users. All the users will communicate each other through messages passing. "Characteristics" is a set of the characteristics of each user, including intrinsic and variable ones. "Domain" is a set of background knowledge or other messages of the specific domain which this model will be used to.

Based on UserMODEL, a users-driven domain-specific application software development tool (called UserTOOL) is also being developed.

There is an increasing recognition that software development is not merely a mathematical or technological

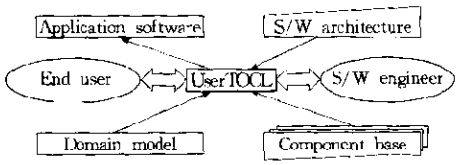

Fig. 1 Interacting with the UserTOOL

challenge, but a complex social process, in which the kind of communication and cooperative interaction between the participants determines the quality of the collaboratively developed product[49]. Figure 1 shows how the UserTOOL interacts with other agents involved in the software development process. Note that the UserTOOL interacts with an end user and a software engineer together. The focus of the UserTOOL is on bridging the gap between informal and formal specifications, i.e., between the two understanding interfaces of the end users (the application software definers) and the software engineers (the application software developers). It also serves as a bridge between the domain model and the final application software. It will adopt user-driven domain-specific application software development methodology based on component-based software architecture to develop application software systems.

According to the five steps of user-driven requirements analysis, the requirements analysis part of UserTOOL can be divided into five component sections[50]: requirement acquisition section, requirement interpretation section, requirement negotiation section, system definition section, system prototyping section.

4 Conclusions

There is a very interesting statement, i.e., the third one among the four rules presented by Rene Descartes (1637) to solve complex problems, "to conduct my thoughts in such order that, by commencing with objects the simplest and easiest to know, I might ascend by little and little, and as it were, step by step, to the knowledge of the more complex; assigning in thought a certain order even to those objects which in their own nature do not stand in a relation of antecedence and sequence ...". The work reported in this paper is only the first step, but may be a meaningful one.

References

1 Naur P, Randell B *et al*. Software Engineering: Report on a Conference Sponsored by the NATO Science Commission. Scientific Affairs Division, NATO, Brussels, Belgium, Jan. 1969

2 Bronsard F *et al*. Toward software plug-and-play. In: Proceedings of the 1997 Symposium on Software Reusability (SSR'97). Boston, MA, 1997. 19～29

3 Prieto-Diaz R, Neighbors J M. Module interconnection languages. Journal of Systems and Software, 1986,6(4):307～334

4 Perry D E. The inscape environment. In: Proceedings of the 11th International Conference on Software Engineering. 1989

5 Boehm B, Scherlis B. Megaprogramming. In: Proceedings of the DARPA Software Technology Conference. Arlington: Meridien Corp., 1992

6 Wiederhold G *et al*. Toward megaprogramming. Communications of ACM, 1992,35(11):89～99

7 Batory D, Geraci B J. Validating component composition in software system generators. In: Proceedings of the 4th International Conference on Software Reuse. 1996

8 Garlan D, Perry D E. Introduction to the special issue on software architecture. IEEE Transcations on Software Engineering, 1995,21(4):269～274

9 Magee J *et al*. A constructive development environment for parallel and distributed programs. In: Proceedings of the 2nd International Workshop on Configurable Distributed Systems. 1994

10 Purtilo J. The polylith software bus. ACM Transcations on Programming Languages and Systems, 1994,16(1):151～174

11 Mettala E, Graham M M *et al*. The domain-specific software architecture program. Technical Report, CMU/SEI-92-SR-9, June 1992

12 Fischer G. Domain-oriented design environments. Automated Software Engineering, 1994,1(2)

13 Davis M J, Williams R B. Software architecture characterization. In: Proceedings of the 1997 Symposium on Software Reusability (SSR'97). Boston, Massachusetts, 1997. 30~38
14 Davis A M. A comparison of techniques for the specification of external system behavior. In: Thayer R H, Dorfman M eds. System and Software Requirements Engineering. Washington, DC: IEEE Computer Society Press Tutorial, 1990
15 Rada R. Software reuse. Intellect, 1995
16 Boehm B W. Industrial software metrics TOP 10 list. IEEE Software, 1987,4(5):84~85
17 Coad P, Yourdon E. Object-Oriented Analysis. 2nd Edition, Prentice Hall, Englewood Cliffs, 1991
18 Thayer R H, Dorfman M *et al*. System and Software Requirements Engineering. Washington, DC: IEEE Computer Society Press Tutorial, 1990
19 Emam K E *et al*. User participation in the requirements engineering process: an empirical study. Requirements Engineering, 1996,1(1):4~26
20 Ives B, Olson M. User involvement and MIS success: a review of research. Manage Science, 1984,30(5):586~603
21 Berry D. Involving users in expert system development. Expert Systems, 1994,11(1):23~28
22 Torkzadeh G, Doll W. The test-retest reliability of user involvement instruments. Inform Manage, 1994,26:21~31
23 Sack K. User participation in software development: what is it, why, and how? In: Briefs U, Tagg E eds. Education for System Designer/User Cooperation, Elsevier, 1985
24 Mumford E. Defining system requirements to meet business needs: a case study example. The Computer Journal, 1985,28(2):97~104
25 Wirth N. Program development by stepwise refinement. Communications of ACM, 1971,14(4):221~227
26 McGrowan C, Kelly J. Top Down Structured Programming. New York: Petrocelli, 1975
27 Warnier J. Logical Construction of Programs. New York: Van Nostrand Reinhold, 1974
28 Yourdon E, Constantine L. Structured Design. Prentice-Hall, Englewood Cliffs, 1979
29 Jackson M. Principles of Program Design. London: Academic Press, 1975
30 Luqi, Goguen J. Formal methods: promises and problems. IEEE Software, 1997,14(1):73~85
31 Barki H, Hartwick J. Rethinking the Concept of User Involvement. MIS Quarterly, March 1989. 52~63
32 Lam W, McDermid J A. A summary of domain analysis experience by way of heuristics. In: Proceedings of the 1997 Symposium on Software Reusability (SSR'97). Boston, MA, 1997. 54~64
33 Prieto-Diaz R. Domain analysis: an introduction. ACM Software Engineering Notes, 1990,15(2):47~54
34 Jarzabek S. Modeling multiple domains in software reuse. In: Proceedings of the 1997 Symposium on Software Reusability (SSR'97). Boston, MA, 1997. 65~74
35 Tracz W. DSSA: pedagogical example. ACM Software Engineering Notes, 1995,20:47~54
36 Kang K *et al*. Feature-oriented domain analysis feasibility study. CMU/SEI-90-TR-21, 1990
37 Wartik S, Prieto-Diaz R. Criteria for comparing reuse-oriented domain analysis approaches. International Journal of Software Engineering and Knowledge Engineering, 1992,2(3):403~431
38 STARS. Organisation Domain Modelling Guidebook. STARS-VC-A023/011/00, March 1995
39 Rumbaugh J. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, 1991
40 Shlaer S, Mellor S. Object Lifecycles: Modeling the World in States. Prentice Hall, Englewood Cliffs, 1992
41 Lim W C. Effects of reuse on quality, productivity, and economics. IEEE Software, 1994,11(5):23~30
42 Joos R. Software reuse at Motorola. IEEE Software, 1994,11(5):42~47
43 Clements P. From Domain Models to Architecture. USC Center for Engineering Focused Workshop on Software Architectures, June 1994
44 Kruchten P. The 4+1 view model of architecture. IEEE Software, 1995,12(6):42~50
45 Bassett P G. The theory and practice of adaptive reuse. In: Proceedings of the 1997 Symposium on Software Reusability (SSR'97). Boston, MA, 1997. 2~9
46 Bassett P G. Framing Software Reuse: Lessons from the Real World. Prentice-Hall, Englewood Cliffs, 1997
47 Johnson R E. Components, frameworks, patterns. In: Proceedings of the 1997 Symposium on Software Reusability

(SSR'97). Boston, MA, 1997. 10～17

48 Li M. A cooperative solving model supporting users-oriented requirements analysis. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 1996

49 Macaulay L. Requirements capture as a cooperative activity. In: Proceedings of the IEEE International Symposium on Requirements Engineering. IEEE Computer Society Press, 1993. 174～181

50 Li M. Users-oriented requirements analysis in automated MIS production. In: Proceedings of the 12th International Conference on CAD/CAM Robotics and Factories of the Future (CARS & FOF'96). 1996

一种用户主导的面向领域应用软件开发新方法

李明树

(中国科学院软件研究所计算机科学开放研究实验室　北京　100080)

摘要　提出一种应用软件开发的新方法,称为"用户工程".这是一种基于构件化软件系统结构的用户主导的面向领域的应用软件开发方法,强调用户在应用软件开发中的主导作用,试图将应用软件的开发过程变成用户详细定义过程,而不仅仅是传统的编程过程.它为越来越多的应用软件开发需求提供了可能有效的一个途径.

关键词　软件工程,需求工程,用户参与,构件,软件体系结构,重用,用户工程.

中图法分类号　TP311