SCIENTIFIC PAPERS

Reuse-based software production technology

YANG Fuqing (杨芙清), WANG Qianxiang (王千祥), MEI Hong (梅 宏) & CHEN Zhaoliang (陈兆良)

Department of Computer Science and Technology, Peking University, Beijing 100871, China
Correspondence should be addressed to Yang Fuqing (email: yang@cs.pku.edu.cn)

Received April 9, 2000

Abstract Software reuse is viewed as a key technology to improve software product quality and productivity. This paper discusses a series of technologies related with software reuse and software component technology: component model, which describes component's essential characteristics; component acquisition technology, of which domain engineering is the main approach; component management technology, of which component library is the kernel; application integration and composition technology, of which application engineering is the main approach; software evolution technology, of which software reengineering is the main approach, etc. This paper introduces the software development environment: JadeBird Software Production Line System, which effectively integrates the above-mentioned technologies.

Keywords: **software reuse, software component, domain engineering, component library, application engineering, reengineering.**

For information technologies, microelectronics is the foundation, computer hardware and communication infrastructure are the carriers, and computer software is the kernel. Software is coded knowledge. More and more abstract experience and knowledge are accurately represented as software. Along with the appearing of cheaper and faster microprocessors, software will take up more and more functions. Software is becoming a new "physical infrastructure" of information age[1].

Although the demand for software is increasing rapidly during the information age, the ability of software development and production is relatively insufficient. This situation means that a lot of urgently needed software cannot be developed in time, and leads to the so-called software gap. Since the cognizance of software crisis and the introduction of software engineering in the late 1960's, many researchers and engineers have made great efforts in software development research and practice. It is realized recently that engineering development approaches and industrialized production technology should be adopted in order to improve software product quality and productivity[2,3]. This involves two aspects: in technology, reuse-based software production technology should be adopted; in management, multi-perspective engineering management pattern should be used.

This paper focuses on the technical issues in the industrialized software production. Sec. 1 introduces the definition of software component, and presents the Jade Bird (JB) component model (JBCOM). Sec. 2 discusses component acquisition, centering on domain engineering. Sec. 3 introduces component management: Component Library Management System. Sec. 4 discusses component reuse, centering on application engineering. Sec. 5 discusses software evolution centering, on reengineering. The last part of this paper introduces the JB project, which supports

this kind of industrialized production technology, especially the concepts, activities and architectures of the JB software production line system and its all-around support to software production.

1 Software reuse and software component

In recent years, it is realized that industrialized software production is the only feasible way to solve the software crisis problem. Research on the successful modes of traditional industry and computer hardware industry shows that these industrial development modes are all up to snuff parts (components) production and product manufacturing (composition) based on standard components, in which, components are the kernel and basis, and "reuse" is the necessary method. Practice has shown that this mode is the successful way for industrialization, and will be the only way that the software industry should go[2].

The idea of software reuse has been popular for a time and found broad application, such as subroutines reuse, generic classes reuse, and compilers reuse, etc. The introduction of the concept of software component lays the technical foundation for software reuse, and consequently makes software reuse attract more universal attention[4]. The component production and reuse is the key to form large scale software industry. This paper will give a brief introduction of a series of important concepts at first: component, component model, and component implementation, etc.

1.1 Component

In general, component is the system element that can be distinguished definitely; software component is a relatively independent constitutional element that has certain contents in software systems. Because the software component being talked about at present mainly concerns about its reuse aspect, software component mostly denotes reusable software component, viz. system constitutional element that has relatively independent functionality and can be reused by a number of software systems. In the following discussion, except specially indicated, the component we talk about all denotes reusable software component. In this paper, we define component as follows: component is a reusable software entity, and consists of two parts-component specification and component implementation. Component specification is mainly described by component model.

Architecture is a special kind of component. It depicts the components of the system, and relationships among these components.

1.2 Component model

Component model is the abstract description of component's essential characteristics. There are many component models at present. The objective and purpose of these models are different: some are reference models, e.g. 3C[5]; some are description models, e.g. RESOLVE[6] and REBOOT[7]; some are implementation models, of which the representative models are common object request broker architecture (CORBA)[8], distributed component object model (DCOM)[9], enterprise javabean (EJB)[10]. These implementation models effectively separate component implementation from its specification and provide the ability for component to interact, thus increasing the chances to reuse and accommodate to the needs of large-scale software systems in actual networking environment.

JB component model (JBCOM) fully assimilates the merits of the above-mentioned models, and is compatible with them. JBCOM consists of two parts——external interface and internal

structure——as illustrated in fig. 1.

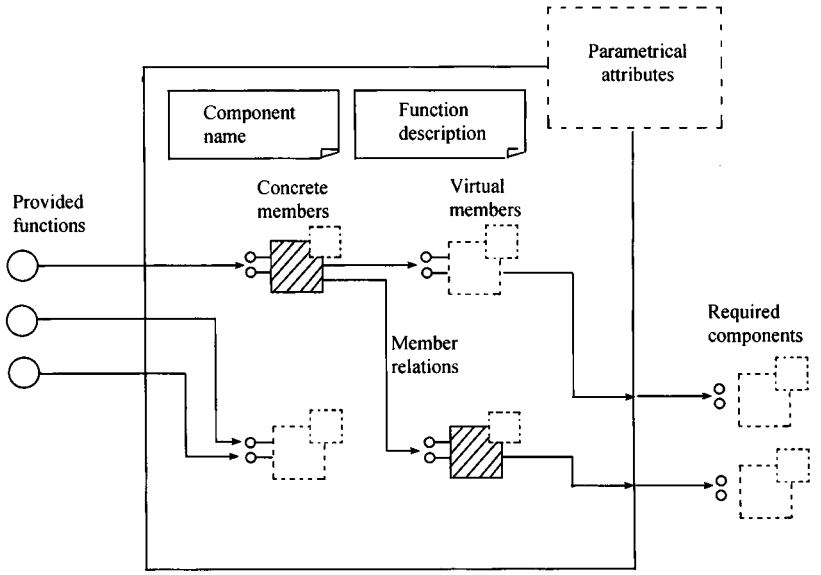

Fig. 1. JB component model.

1.2.1 External interface. Component's external interface is the essential information that a component provides for its reuser, including component name, function description, provided function, required components, and parametetrical attributes.

1.2.2 Internal structure. Component's internal structure comprises two aspects: internal members and their relations. Internal members include concrete members and virtual members, and member relations include the interconnections between internal members, and the interconnections between internal members and external interface.

1.3 Component implementation

Component implementation is the logical system (code) that implements the function of a component concretely, which is also called coded-level component generally. The component producer is responsible for component implementation, and the component reuser does not have to know the implementation details of a component. When reusing a component, the reuser can customize or specialize it.

2 Domain engineering: component acquisition

Having a large amount of reusable components is the precondition for using reuse technology successfully. By analyzing reusable information and domains, we find[11]:

(i) Reusable information is domain specific. Reusability is not an isolated property of information; it depends on a particular problem and problem-solving context. Therefore, domain-oriented strategy should be adopted when identifying, acquiring and representing reusable information.

(ii) Cohesiveness and stability of domains, viz. knowledge about solutions and in the domains are sufficiently cohesive and stable. So cohesiveness of specification and implementation knowledge in a domain makes it possible to solve a large number of problems through a finite, relatively small set of reusable information. Stability of domain makes it possible to reuse the captured information again and again over extended periods of time.

From the knowledge of these two aspects we can draw the conclusion: the components acquired in domain engineering are large numbers and have high reusability[12]. More than that, the domain-oriented software architecture acquired in domain engineering is more specific and easier to manipulate than the general software architecture. Therefore, domain engineering is the main approach to acquiring component and architecture.

Domain is the function area covered by a family of similar systems. Domain engineering is a process of creating basic capability and necessary foundation for application engineering of a family of similar systems. JB domain engineering method divides domain-engineering process into several activities: domain analysis, domain design, domain implementation, etc. The activities and products are illustrated in fig. 2.

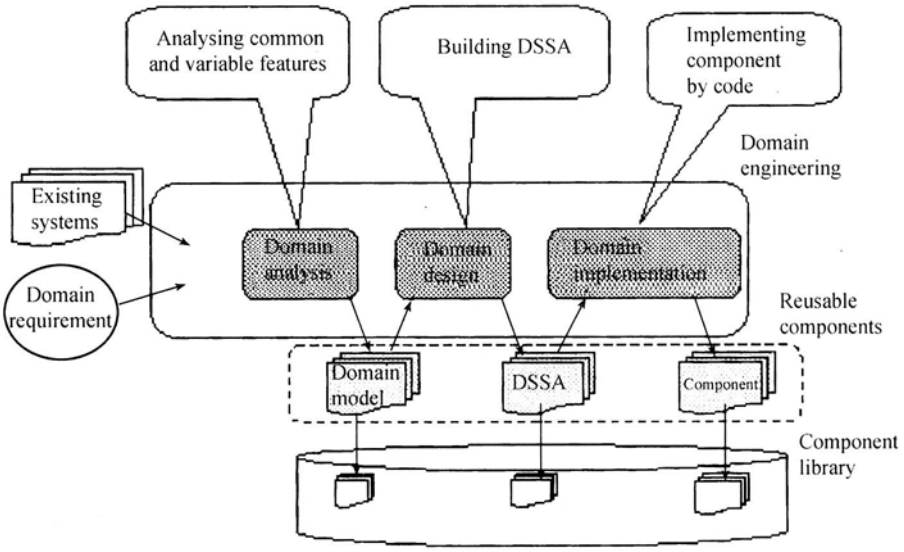

Fig. 2. Activities and products in domain engineering.

2.1 Domain analysis

The main activity in this stage is developing domain analysis model. It includes three subactivities: creating domain requirement definition, creating domain object-oriented analysis (OOA) model and creating domain terminology dictionary, of which, creating domain requirement definition can be further divided into three activities: defining domain business model, defining domain business process and defining domain requirements. Creating domain requirement definition and creating OOA model constitute the main contents of domain analysis; if necessary, we can define terminologies in domain terminology dictionary at any moment during these two activities.

During the domain analysis process, different requirements should be discovered and represented separately according to the domain characteristics. These requirements can be divided into

the following different classes:

Mandatory requirements: requirements that all systems in the domain must have. This kind of requirements is the essential requirement of systems in the domain.

Optional requirements: requirements that some systems in the domain may have. This kind of requirements reflects the differences between systems in the domain.

Alternative requirements: a set of requirements among which there exist special relations among them. This kind of requirements also reflects the variability of systems in the domain.

Domain model depicts the common requirements of systems in the domain, including domain terminology dictionary, domain requirement definition, and object oriented analysis model (OOA model), etc.

2.2 Domain design

The main activity in this stage is developing domain design model: domain specific software architecture (DSSA). Similar to the application design, domain design needs to consider implementation issues, e.g. operating system, programming environment, software deployment mode, and data-accessing mode, etc. And besides, it also needs to choose architecture style (e.g. two-tier C/S mode, B/S structure, three-tier structure, etc.) and component implementation model (e.g. DCOM and CORBA, etc.).

In JB domain engineering method, DSSA mainly considers the object-oriented design (OOD) model of which basic elements are attributes, services, classes, objects, and their relations. Classes, class attributes, methods, all kinds of relations are likely optional or alternative; there also exist dependency or mutually exclusion relations between these elements. There is traceability between DSSA and domain analysis model: mandatory requirements in domain analysis model must trace to the corresponding mandatory elements in DSSA; optional requirements in domain analysis model must trace to the corresponding optional elements in DSSA; alternative requirements in domain analysis model must trace to the corresponding alternative elements in DSSA.

2.3 Domain implementation

The main purpose of this stage is to develop domain specific components and architecture, which can be extracted from existing systems or developed from scratch according to domain analysis model and DSSA. Traceability should be established between reusable components and DSSA in order to connect reusable components to their specification. Black box reuse should be as convenient as possible in use.

Another important thing of domain implementation is component testing. It is especially important because the quality of components influences the quality of many future software systems.

Finally, we need to point out that there are iterations between the mentioned three stages, and inside each stage.

3 Component management

Managing a large amount of components efficiently to facilitate component storage, component searching, and component retrieval is the necessary assurance for successful component reuse[1,6]. Component management involves component description, component classification, component library organizing, users and privileges management, and user feedback, etc.

3.1 Component description

Component model is the abstract description of component's nature, and provides foundation for production and reuse of components. It also needs to describe component for management, including the following information: implementation way, implementation body, producer, data of production, size, price, version, related components, etc. which constitutes the full description of component together with component model.

3.2 Component classification

JB component library (JBCL) uses facet method to classify components, including the following facets:

(i) Application environment: the hardware and software platform that must be provided when the component is used.

(ii) Application domain: the names of domains (and its sub-domains) that the component is used or may be used.

(iii) Functionality: the set of software functions provided by the component in original or future systems.

(iv) Level: the abstraction level of component relative to the stages of software development process, e.g. analysis, design and coding, etc.

(v) Representation: the language or medium that describes the content of the component, e.g. programming language used in source code component, etc.

3.3 Component library organizing

In order to facilitate component searching and component reusing, component library builds various relationships between component entities. These relations include:

(i) Refinement: it denotes relationships between components that are in adjacent stages in software life cycle.

(ii) Version: it indicates relationships between components that are in the evolution series of a component. User may find all versions of a given component through version relations.

(iii) Cooperation: The component having cooperation relation with a component that cooperates with it to accomplish a certain task together.

(iv) Containment: the containment relationships between components in different forms, e.g. a class tree component may contain one or more class components, a framework component may contain one or more class components.

(v) Inheritance: An m:1 directed relation. In JB component library, there may be inheritance relation between components, but it is not encouraged to use complex inheritance hierarchy or multiple inheritance.

3.4 Users and privileges management

Component library system is an open public component sharing mechanism. Any users can access the component library through network. This is very convenient for the users, but also brings risks to system security. Therefore, it is necessary to properly restrict the users' accessing privileges to ensure the data security.

Component library system involves five categories of users: registered user, guest, component provider, system administrator and super system administrator. They have different responsibilities and privileges in component library system, and cooperate together to maintain the normal

operation of component library system. Moreover, system defines one privilege for each operation, including providing component, managing component, querying component, and retrieving component. To divide the work of the operation and provide flexible privileges assignment, each user can be endowed with one or several operational privileges, which combine together to form this user's privilege.

3.5 User feedback

The user feedback part of component library system provides assistant decision-making support for reuser to understand and select components, and for various component library administrators to manage and improve the component library system. Its main feature is: based on data warehouse technique, using multidimensional data model, and supporting subject-oriented and integrated data organizing mode, which has time attribute and contains history data. Subject table comprises multiple fact tables, dimension tables, and various levels of aggregation tables. It provides the basis of data storage and organizing for multidimensional analysis. To assist the users to identify and select components, the feedback library of component library system performs multidimensional analysis and measurement on component feedbacks, synthetically measures the reusability of components from four aspects separately: component documents, component composition, component testing, and component application. The architecture of JBCL system is shown in fig. 3.

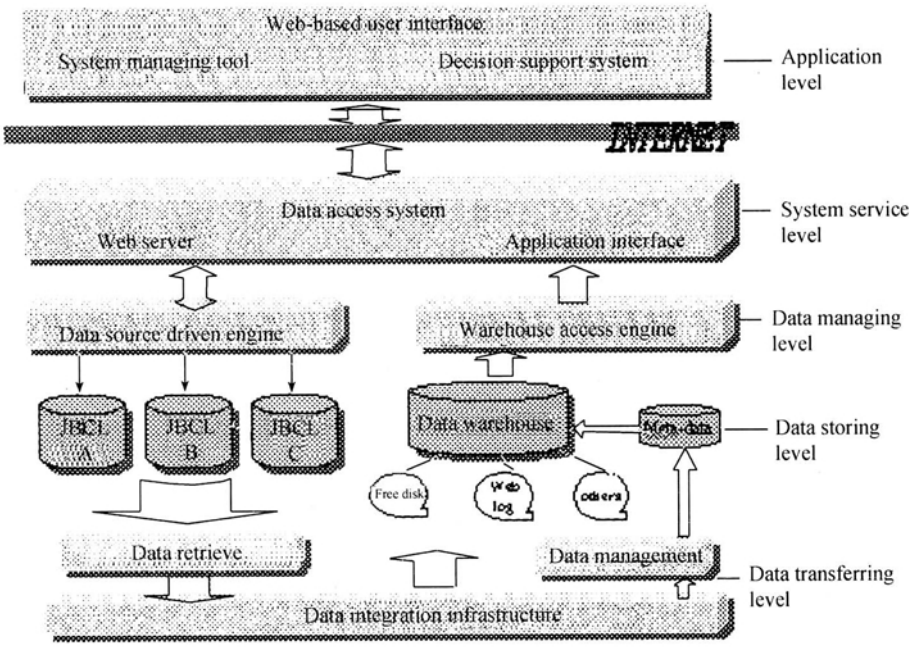

Fig. 3. Architecture of JB component library.

4 Application engineering: component reuse

The process that uses the results of domain engineering to develop software systems is usually called application engineering. Similar to the common software development process, application engineering can also be divided into several stages: analysis (requirement acquisition), de-

sign, and implementation, etc. The difference is, each stage of this process can obtain reusable domain engineering work products from component library and take it as the foundation of integration and development in this stage[15].

The activities and products in application engineering are shown in fig. 4.

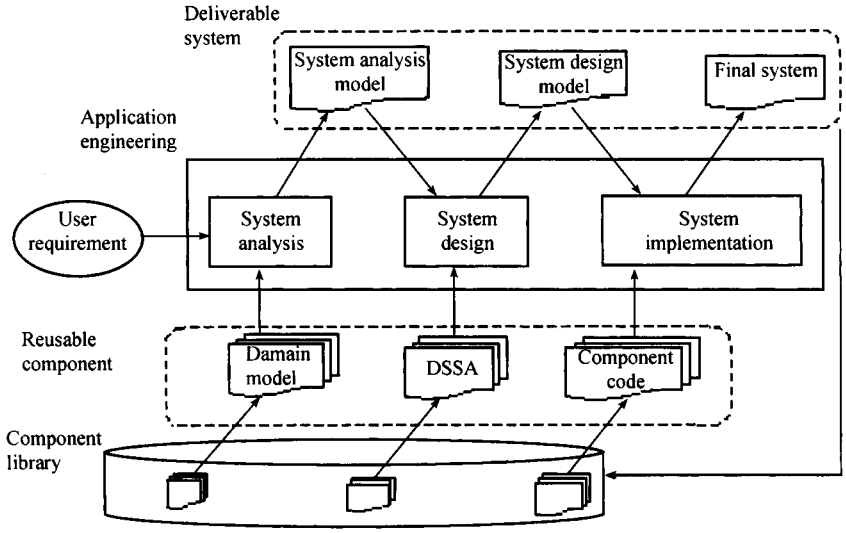

Fig. 4. Activities and products in application engineering.

4.1 System analysis (capturing requirements)

The objective of this stage is: based on the domain analysis model acquired by domain engineering, comparing with user requirements, confirming the variable requirements of domain analysis model or obtaining new requirements, to acquire the analysis model of the specific system. It involves the following activities: confirming the specific business model, fixing the variability of domain analysis model, adjusting domain requirement model, etc. The continuous participation of end-users is an important factor to acquire better model.

4.2 System design

The objective of system design is: based on DSSA acquired in domain engineering, to obtain the design model comparing with the analysis model of the specific system. Its kernel is to fix the corresponding variability of DSSA according to the system requirement model. This stage should design the corresponding model for the new requirements posed by users. In addition, it is necessary to make some relevant adjustment on DSSA if the domain knowledge is increased.

4.3 Implementation and testing

Based on domain architecture/components, this stage integrate and compose the components and architecture according to the design model of the special system, and do some necessary coding so as to implement and test the final application.

Application engineering fixes the variability of domain requirements in development phase. For a pretty matured domain, it is better to fix the variability of requirements in later stages. These later stages include setup, startup and running, etc. Variability can be implemented

through system clipping in setup stage, through parameter controlling in startup stage, and through dynamic configuration in running stage.

5 Reengineering: software evolution

With the development of software technology and the continuous change in application requirements, software should be evolotionary. It is one of the basic characteristics of software. However, the re-development method adopted currently is time-consuming and uneconomical. Moreover, the existing software is considered as a heavy burden. In fact, the procedure of reengineering occurs to almost everything. Component technology is effectively applicable to the evolution characteristics of the software, changing the legacy software into the valuable basis on which it develops. It not only expedites and regularizes the development procedure, but also accumulates the software assets.

Software reengineering is an engineering procedure. It combines reverse engineering, reconstructing, forward engineering, and reconstructs the existing system to a new form[17]. The foundation of the reengineering is system understanding, including the comprehensive understanding to running system, source code, design, analysis, documents, etc. However in most cases, because of the loss of various documents, the source code is the only source we can get, so program understanding will be the only way to understand the system.

The main activities of JB software reengineering are illustrated in fig. 5.

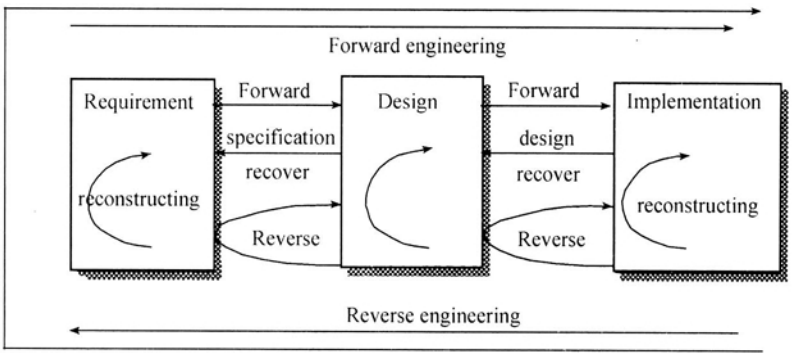

Fig. 5. Software reengineering.

6 Jade Bird Project

Jade Bird (JB) Project[3] is a national project that researches and resolves the problems of software development and production. It began in the late stage of the National Sixth "Five-Year Plan", and has gone through the Seventh, Eighth, and the first term of the National Ninth "Five-Year Plan". Now JB Project is in the process of the second term of the Ninth "Five-Year Plan". The goal of JB Project is: based on the practical techniques of software engineering, to popularize the software industrialized production technology and mode, provide necessary technology and tools supporting industrialized production of software to software enterprises, establish the fundament of Chinese software industry, and form the necessary accumulation of talents, techniques and products.

JB Project proposes the idea of software production line, dividing software production pro-

cess into three different sub-progresses: application architecture production, component production, and component-architecture-based application composition; application architecture library and component library are used to store the artifacts, and connect three sub-processes. The standards and quality assurance provide support for the whole production process. The conceptual diagram of JB software production line is shown in fig. 6.

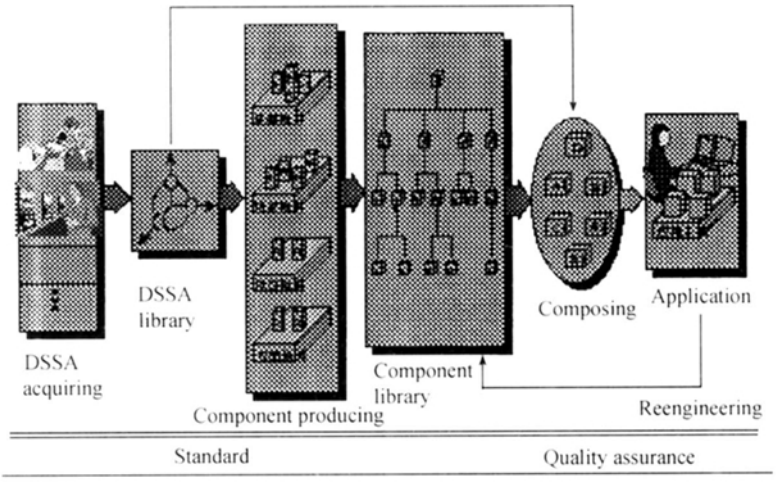

Fig. 6. Conceptual diagram of JB software production line.

The corresponding activities in JB software production line system for the mentioned conceptual diagram are shown in fig. 7.

We can make out from figs. 6 and 7 that software component and architecture technology is the kernel of JB software production line system, of which main activities are embodied in traditional domain engineering and application engineering but endowed with new contents and organically linked up through component management and re-engineering, etc. In addition, each activity of JB software production line system has corresponding methods and tools. Combining management issues: project management, organization management and quality management, etc. these activities form the fully integrated software production flow.

Because of the breakthrough and development of software component technology, software industry will be divided into three categories: software component industry, system integration and composition industry and component service industry, following traditional industry's pattern. This provides a reasonable mode for the large scale of software industry. JB software production line will facilitate the construction of the infrastructure of software industry, and provide direct support for the promotion of the capability of software enterprise.

7 Conclusion

This paper presents our research work on reuse-based software production technology and discusses the issues of component acquisition, component management, component reuse, and software evolution centering on the concept of component. The software engineering method based on software reuse and software component technology is the hotspot of current research and practice of software community. Refs. [4,15,20] have shown their research work of software reuse. Comparing with these methods, JB software production line system is more concerned with the

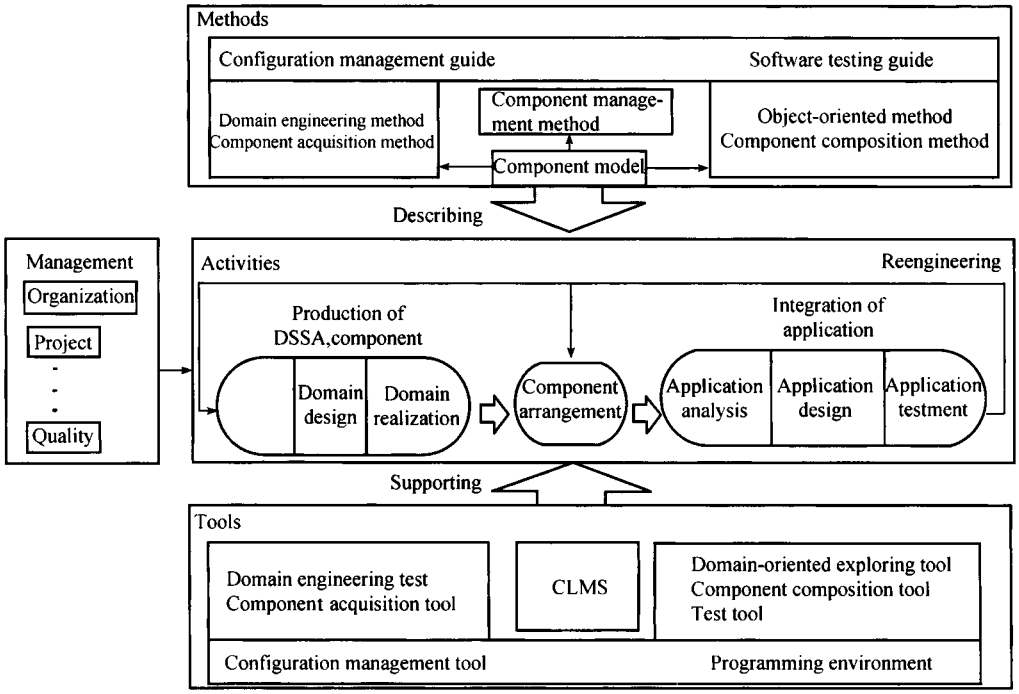

Fig. 7. Activities in JB software production line system.

production process and the maturity of production line system in order to improve the operability and practicability of software production line system.

Besides the technical contents introduced in this paper, it is also realized that management is an important issue of software engineering and a restraint factor of the exertion of advanced techniques. Therefore, besides deeper research in technology, we will do more work in management issues regarding the enterprise capability certification, and continue to cooperate with software enterprises. Considering the features of Chinese software enterprises, we wish to explore the software production methods to meet the needs of Chinese software enterprises.

Acknowledgements This research is supported by the Key Project of State Ninth "Five-Year Plan" the 863 High-Tech Program of CHina and the National Natural Science Foundatino of China.

References

1. President's Information Technology Advisory Committee Interim Report to the President, August, 1998.
2. Yang Fuqing, Shao Weizong, Mei Hong, The design and inplementation of object oriented CASE environment JBII system, Science in China, Ser. A, 1995, 38(5): 600.
3. Yang Fuqing, Mei Hong, Li Keqin, Software reuse and software component technology, Chinese Journal of Electronics, 1999, 27(2): 200.
4. Mili, H., Mili, F., Mili, A., Reusing software: Issues and research directions, IEEE Transactions on Software Engineering, 1995, 21(6): 528.
5. Tracz, W., Implementation working group summary, in Reuse in Practice Workshop (ed. Bald, J.), Pittsburgh, Pennsylvania, July 1989.

6. Paolo, B., Stephen, H. E., Special feature: Component-based software using RESOLVE, ACM SIGSOFT, Software Engineering Notes, 1994, 19(4): 21.
7. Sindre, G., The REBOOT approach to software reuse, System Software, 1995, 30: 201.
8. OMG 99, Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision 2.3, Oct. 1999.
9. Dale, R., Inside COM, New York: Microsoft Press, 1997.
10. Anne, T., Enterprise JavaBean, Dec. www.psgroup.com, 1997.
11. Arango, G., Prieto-Diaz, R., Domain analysis concepts and research directions, in Domain Analysis and Software System Modeling (eds. Prieto-Diaz, R., Arango, G.), Los Alamitos: IEEE Computer Society Press, 1991, 9—32.
12. Tracz, W., Confessions of a Used Program Salesman——Institutionalizing Software Reuse, New York: Addison-Wesley Publishing Co., 1995.
13. NATO, "NATO Standard for Management of a Reusable Software Component Library", Vol. 2, NATO contact number CO-5957-ADA, 1991.
14. STARS, Asset library open architecture framework version 1.2, Informal Technical Report STARS-TC-04041/001/02, 1992/8.
15. Ivar, J., Martin, G., Patrik, J., Software Reuse: Architecture, Process, and Organization for Business Success, New York: ACM Press, 1997.
16. Neighbors, J. M., The Draco approach to constructing software from reusable components, IEEE Transactions on Software Engineering, SE-10, September 1984, 564—573.
17. Feiler, P. H., Reengineering: An Engineering Problem (CMU/SEI-93-SR-5, ADA267117), Pittsburgh, Pa.: Software Engineering Center, CMU, July 1993.
18. Ralph, J., Foote, B., Designing reusable classes, Journal of Object-Oriented Programming, 1988, 1(2): 1.
19. Prieto-Diaz, R., Status report: Software reusability, IEEE Software, 1993, 10(3): 61.
20. Brownsword, L., Clements, P., A case study in successful product line development, Technical Report, CMU/SEI-96-TR-016.