Development of Software Engineering: A Research Perspective

Hong Mei (梅　宏), Dong-Gang Cao (曹东刚), and Fu-Qing Yang (杨芙清)

School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P.R. China

E-mail: meih@pku.edu.cn; caodg@sei.pku.edu.cn; yang@sei.pku.edu.cn

Received April 17, 2006; revised June 27, 2006.

Abstract In the past 40 years, software engineering has emerged as an important sub-field of computer science and has made significant contribution to the software industry. Now it is gradually becoming a new independent discipline. This paper presents a survey of software engineering development from a research perspective. Firstly, the history of software engineering is reviewed with focus on the driving forces of software technology, the software engineering framework and the milestones of software engineering development. Secondly, after reviewing the past academic efforts, the current research activities are surveyed and new challenges brought by Internet are analyzed. Software engineering researches and activities in China are also reviewed. The work in Peking University is described as a representative.

Keywords software engineering, methodology, design, programming, software component

1 Introduction

The concept of software engineering was first discussed in the late 1950s and early 1960s to address the issue of the so-called software crisis. Many believe that the official start of the field of software engineering was the two conferences on software engineering that NATO sponsored in 1968 and 1969. Ever since then, software engineering has developed considerably and made substantial contribution to the computer industry. Currently software engineering has evolved into a separate profession standing beside computer science and traditional engineering regardless of the doubts on whether it is a true engineering discipline and whether there is the silver-bullet[1] to software crisis. And the goal of software engineering keeps unchanged: producing quality software with limited resources and time. However, compared to those classic engineering disciplines, this new profession is so young that there is still a long way towards reaching the goal. What is more, the booming of Internet has brought about more and more new challenges to software engineering.

Traditional software engineering theories and frameworks are developed in a relatively closed, static and stable environment. However, the Internet environment is highly open, dynamic, and in continuous evolution. Software entities in the Internet environment will be much different to those in the current environment. For example, new software entities may be more active and proactive. They can react to changing environments and evolve themselves accordingly. They can flexibly interact with each other in a rich set of means. For these new software entities, the traditional software engineering theories and technologies become insufficient. Therefore, new supports, both theoretically and technically, are strongly demanded from software engineering.

In order to better help address the old unresolved issues and rise to the new emerging challenges, it is necessary to look back the development traces of software engineering, to sum up the experiences and lessons, and to reflect on successes and failures. The well understanding to the past development of software engineering will of great importance to the whole software engineering community and will guide us for current and future development directions.

This paper will review the development of software engineering from a researcher's perspective. The review consists of mainly three parts: historical overview, status quo survey, and analysis of future directions.

The overview of the past practices can help identify the baseline of software engineering development evolution. It should be noted that the development of software engineering is closely related to that of software technology itself. So this paper tries to investigate the relationship between them to find the driving forces for them. Also overviewed and analyzed are the software engineering framework and milestones. Through the review, component-based reuse is believed to be a hopeful and practical approach towards the software engineering goals.

By surveying the state-of-the-art researches, we could have a better understanding of the progress we have made and the issues we are facing. So the most active research areas will be inspected, e.g., software process, model driven architecture, requirement engineering, aspect-oriented software development, service oriented architecture, and component-based software development, etc.

After reviewing the past and current researches, future directions of software engineering will be explored by analyzing the challenges and opportunities brought about by the extremely open and dynamic nature of the Internet.

In this paper, the history and achievements of soft-

Survey

This work was supported by several government-funded projects, including the National Key Projects in the State 6th to 10th Five-Year Plan, the National High Technology Development 863 Program of China, the National Grand Fundamental Research 973 Program of China, and the National Natural Science Foundation of China, etc.

ware engineering in China will be specifically introduced in detail. In China, software engineering was started in 1980. At that time, there was almost no software industry. With the great efforts of more than twenty years, both the Chinese software industry and the academia have made significant progress. And correspondingly there are also many precious lessons to learn. This paper mainly describes four representative research works in the past two decades and briefly introduces the current research activities with focuses on the representative work by Peking University.

The rest of this paper is organized as follows. Section 2 gives an overview of software engineering, surveying the history and state-of-the-art, outlining the future directions and introducing Chinese research activities. Section 3 presents the research and practices in Peking University. Section 4 concludes the whole paper.

2 Overview of Software Engineering

2.1 Historical Review

In a historical viewpoint, the development of software engineering always accompanies that of software technology. Technically speaking, software technology has significant impact on software engineering development. So it is necessary to investigate the development of software technology to understand the nature of software engineering and foresee its future trends.

2.1.1 Driving Forces for the Development of Software Technology

Software is essentially a computer program modeling the problem space of the physical world and the solution. Software can perform various tasks such as controlling hardware devices, computing, communicating, etc. In the past fifty years, software technology has developed rapidly since its birth. It has been recognized that every improvement of software technology has pushed the development of software engineering.

From the evolution history of software, we can find that the development of software can be attributed mainly to four driving forces as follows.

Better Utilizing Hardware Capabilities. Hardware devices are controlled by software, without which it is impossible for them to work efficiently and flexibly. However, computer hardware technology always develops much faster than that of software, as has been proved by Moore's law. Thus how to effectively and efficiently make use of the power of hardware with the help of software becomes an important goal of software technology. The development of operating system and other system software is a good example. Now the booming of Internet and the ubiquitous computing devices pose some new challenges to software technology. How to better explore the capabilities of networked devices and systems becomes a new task.

Pursuing a Computing Model That is Both Expressive and Natural. A basic software model comprises entity elements and the interactions among them. The software model has evolved from the initial machine language instruction and the sequence and jump relationship, to high-level language statement and the three control structures, procedures and the sub-procedures relationship, object and message passing, to the currently popular model of components and connectors. The evolution of computing model greatly facilitates software construction and maintenance.

Bridging Heterogeneity and Facilitating Interoperation. Heterogeneity is the natural result of free marketing. The need for open interoperability is also the demands of free marketing. Operating system and programming language makes heterogeneous hardware transparent to programmer and user, while middleware resolves the heterogeneity of OSs and languages, and so do the popular Web Service technology to different middleware.

Abstracting Commonalities to Promote Reuse. The road of software development is also the process of improving the programmer's abstraction level. When commonalities are abstracted, reuse is possible to increase the productivity and quality of software development. A system library provides system functions to user applications, which only need to link the library into the executable program without re-implementing the function. Operating system relieves programmers from managing low-level hardware devices. Middleware takes control of the hard network connection issues, so programmers can only focus on the high-level application business logic. In domain engineering, some application frameworks implement the domain commonalities to ease the development of specific applications.

The history of software has witnessed the four driving forces in every corner of software development. Since the development of software engineering is closely related to that of software itself, the four driving forces also have significant impact upon software engineering. When software technology evolves, it requires corresponding support from software engineering in theory, methodology and techniques, or it will not satisfy the industry's requirements. The soundness of this rule is proved by an evolution road of software engineering from structured analysis and design, object-oriented development, to the currently popular component-based software development.

2.1.2 Software Engineering Framework

To some extent, the essence of "software development" is the process of mapping "high level concepts and logics" to "low level concepts and logics". The mapping is usually rather complex especially for the development of large scale software system. There are many factors involved, e.g., humans, technologies, budgets and schedules, etc. Therefore, "software engineering" is defined as follows.

Software engineering is an engineering discipline aiming at producing software products that satisfy user's *needs by applying theories and technologies from computer science and project management. The products should be delivered on time and within budget*[2].

Like other engineering disciplines, software engineering also has its own objectives, activities and principles. The main contents of software engineering framework can be outlined in Fig.1.

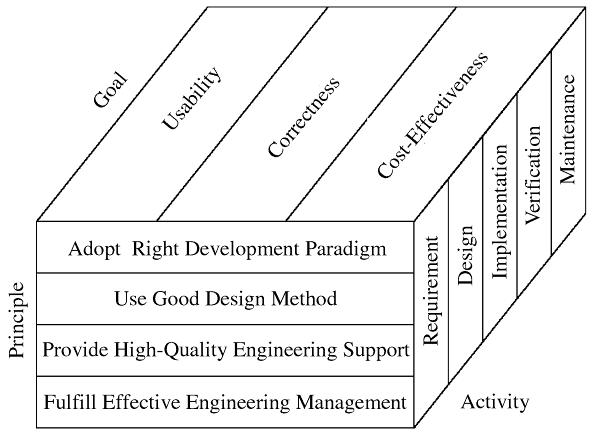

Fig.1. Software engineering framework[2].

The goal of software engineering is to produce correct, usable software with reasonable cost, i.e., correctness, usability and cost-effectiveness.

Software engineering activity is "the necessary processes or steps to produce a software product that both satisfy user's requirements and fulfill engineering objectives." Typical activities include requirement (analysis), design, implementation, verification and maintenance.

As an engineering discipline, software engineering has some software-specific fundamental principles: adopt the proper development paradigm, use sound design method, provide high quality engineering support, enforce effective engineering management, etc. Each of these principles generalizes one important aspect in software development and each has specific proven sub-principles. For example, "use good design method" abstracts the principles of separation of concern, information hiding, modulation, etc.

In the past forty years, a large number of research work has been carried out within this framework towards the goal of correctness, usability and cost-effectiveness. The software engineering community continues to make progress that can largely be attributed to the four driving forces. However, the progresses we made are still insufficient to address the "software crisis" issue. The whole community is still looking for the "silver-bullet" or silver-bullet-alike approaches.

In the long time researches and practices, after numerous successes and failures, people have recognized that *software reuse* may be the most promising approach towards reaching the software engineering goals.

2.1.3 Software Reuse

Traditional engineering industries have successfully developed solid model to promote productivity: first producing standard components, then composing (reusing) these standard components to form the target product. In this model, standard component is the key factor, whereas "reuse" is the necessary means. History and practices proved that this model should and must also be applied to the development of software industry.

Recently software reuse related researches are very active. From the technical perspective, there exist two major different technical approaches. One is the generative approach, which aims to reuse existing development process and then apply it to generating an application automatically from an abstract specification. The other is the compositional approach, which aims to reuse existing reusable assets and build an application by composing them together. Since the generative approach heavily depends on the development of software automation technology, it only succeeds in some specific application domains. At present, the compositional approach is practically more feasible. As a compositional approach, component-based reuse is currently a dominant approach to software reuse.

Fig.2 depicts the key technical and non-technical factors towards component-based software reuse, including: software component technology, domain engineering, software architecture, software reengineering, open system technology, software process, CASE technology, and various non-technical factors. These factors usually intermingle and affect each other.

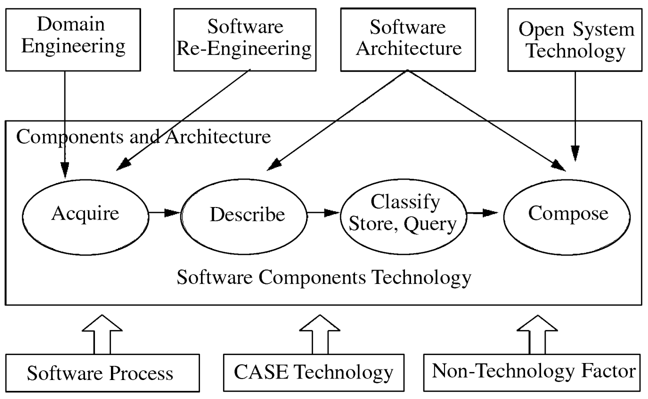

Fig.2. Key factors for component-based software reuse.

Component technology plays a key role in this approach. From a development process viewpoint, component-based software reuse can be divided into three relevant sub-processes[3]: development of reusable components, management of components, and application development by composing existing components. These three sub-processes treat components from different perspectives and at different detailed levels.

Currently the importance of component-based software reuse has been widely recognized by both academia and industry. Component-based software development is becoming the prevalent development paradigm.

2.1.4 Software Engineering Milestones

In its four decades of history, software engineering has made much progress. The following are some important events and achievements organized in not-so-strict chronological order:

1940s and 1950s: early preliminary tools, such as macro assemblers appeared.

1960s: the high-level programming languages such as FORTRAN, COBOL, and ALGOL were widely used. The notion of reuse flourished. Modular programming was being used in programming. Software engineering emerged as a new sub-area of computer science and engineering.

1970s: advanced development tools such as the utility of *make* and *code library* emerged. The concept of software lifecycle boosted the development of programming methodologies and project management. Structured programming, abstract data type (ADT), and the principle of information hiding were proposed. Some early CASE tools appeared.

1980s: the PC era began. The important results include: the prototyping technologies and formal methods, CASE tools and environment, software process, object technology, and the discussion about silver bullet.

1990s and the beginning of the 21st century: the attractive technologies in this network era include: matured object technology, middleware, process improvement, software architecture, open source software development, agile methods and web engineering. Currently software engineering is expected to be an independent discipline. In the late 1990s, component technology became prevalence. It is viewed as the key towards manufacturing software in an engineering approach.

2.2 State-of-the-Art Researches

Although the software engineering community has made significant progress, it has to be admitted that it is still far from the desired software engineering goals. And many issues are still open, such as how to make components reusable, how to reuse software assets in high level and early stages of development, how to better model non-functional properties of software systems, how to build software systems that are easy to configure and can adapt to changing environments, how to better support separation of concerns, how to utilize the advantages of domain engineering technologies, how to cope with the challenges brought about by the Internet, etc. For these issues, many new research directions are initiated. And some traditional software engineering areas also start new topics to address the new problems.

2.2.1 Component-Based Software Development

Component-based software development (CBSD), also referred to as component-based software engineering (CBSE), directly supports software reuse by shifting the development emphasis from programming to composition. CBSD embodies the "buy, do not build" philosophy[1]. It focuses on building large software systems by reusing pre-fabricated software components, especially those COTS (commercial-off-the-shelf) components. The reuse of existing high-quality components could considerably speedup the system construction while at the same time largely reduce development costs and maintenance costs.

Component based software development gets strong support from industry vendors. The component models and supporting middleware platforms of CCM/CORBA from OMG, Enterprise JavaBeans/J2EE from SUN, and DCOM/.Net from Microsoft dominate.

However, the mainstream component models and middleware platforms mainly focus on binary-level component reuse and interoperation among them, which is obviously insufficient. There also lacks of a systematic method to guide the whole developing process, to support component composition at higher abstraction levels. And the short of available COTS components are also a severe problem.

The prosperous of various Internet technology leads to the emergence of web service and service-oriented architecture (SOA). Web service is in essence a new kind of component model. This technology is expected to support reuse at a more abstract level and support new composition methods.

2.2.2 Service Oriented Architecture

In the traditional software engineering, the developer specifies requirements, makes architecture design, and then translates the designs into executables that meet the customer requirements. Different to the traditional approach, service-oriented architecture (SOA)[4] is a new paradigm that evolves from the object-oriented technology and component-based software development.

From the software engineering perspective, SOA provides a valuable approach of reusing software assets in standard way. The reuse granularity of SOA is in a high and abstract level. SOA also transfers the development focus from low level implementation to high level business process composition. Such a development model involves three collaborative parties: application builders, service developers, and service brokers. Services are platform-independent and loosely coupled so that services developed by different providers can be composed. Furthermore, SOA enables current development and future integration. The flexible and extensible features of SOA are effective for designing industry-level systems and large-scale distributed applications.

SOA is still far from mature. There are many challenges, e.g., how to compose services dynamically and on-demand, how to guarantee the trustworthiness and dependability of the selected services, how to validate and evaluate the composite service, etc.

2.2.3 Software Process

Software process, sometimes referred to as software development life cycle process, is primarily concerned with the production aspect of software development, as opposed to the technical aspect. Software process research mainly deals with the methods and technologies used to assess, support, and improve software development activities[5]. The field has grown up during the 1980s to address the increasing complexity and criticality of software development activities. Currently, three main interrelated themes can be identified in this area: process model, process modeling, and software process assessment and improvement.

The notion of software process incurs emergence of a series of process models to meet with the new requirements and trends in software development, such as waterfall model, iterative model, spiral model, etc. In order to support the creation and exploitation of software process models, a number of process modeling languages are proposed.

In development practices, people find that process need to be continuously assessed and improved to cope with changing requirements. Various software improvement models and standards have been developed, such as CMM and ISO9000, etc. Besides, cost and benefit analysis of process improvement, process reengineering, statistic process control in process measurement are also hot topics.

2.2.4 Requirement Engineering

Requirement engineering (RE) is in the early stage of software development. It has evolved as a branch of software engineering mainly concerned with the real-world goals for, functions of, and constraints on, the to-be-developed software systems[6]. One objective of RE is to discover the purpose the software system is intended for. It is very important in the software development lifecycle.

RE is a human-centered process demanding skills from multiple disciplines, e.g., computer science, logics, cognitive psychology, anthropology, sociology, linguistics, etc. Therefore, there are many difficulties in the requirement engineering process.

Technically, requirement engineering can be divided into four sub-processes: requirement elicitation, validation and verification, specification and documentation, variability and evolution management. The stakeholders and their requirements are identified at first and then documented in a form that is easy to understand and analyze in subsequent sub-processes.

There are several matured technologies in RE phase, e.g., entity relationship diagram, data flow diagram and state transition diagram, state machine, state chart and Petri-Net, etc. Object-oriented modeling techniques cover the structural, behavioral and process aspects of requirements. In recent years, feature oriented modeling and product line techniques become the hot spots.

Requirement engineering needs to better address several challenging issues[7], e.g., better modeling the problem domains, developing richer models for non-functional requirements, better understanding the impact of software architectural choices on requirements, reuse of requirement models, etc.

2.2.5 Model Driven Architecture

Model Driven Architecture (MDA)[8] is a new way of organizing and managing enterprise architectures based on a platform-independent model (PIM) of the application's business functionality and behavior. MDA is proposed by Object Management Group. It attempts to make application business logic specification independent of the various implementation level technologies and middleware platforms. In this way, it is not necessary to repeat the implementation functionality each time a new technology emerges.

MDA utilizes automated standard tools like UML for both defining the models and facilitating transformations between different models.

The MDA software development life cycle includes five steps: 1) capture business requirements in a computation independent model; 2) specify the business model in a specific technology context and create a platform independent model (PIM); 3) transform the PIM to one or more platform-specific model (PSM), and add platform-specific rules and code that the transformation did not provide; 4) transform the PSM to code; and 5) deploy the system in a specific environment.

Since MDA shifts the focus of software development from low-level implementation to high-level model design, computer-aided toolkits are necessary to help people work on the model. There are already some MDA environments and tools, like OptimalJ[9]. But there are still too few of them. Most of them have not been applied in industrial strength application development. For MDA to succeed, more and more automated tools are strongly needed.

2.2.6 Aspect-Oriented Software Development

In recent years, aspect oriented software development (AOSD) attracts much attention for its claim in supporting "separation of concern" in software development.

Separation of concern is a well-established principle in software engineering. Software developers tend to decompose problem domain into different concerns to separately tackle with. However, current programming methods usually provide only one way of decomposition, i.e., functional decomposition, resulting in one dominant kind of modules such as functions, procedures, etc. These modules effectively encapsulate business concerns. But there are always some concerns that cannot be easily encapsulated within the dominant modules but scatter across and tangle with other modules. These concerns are the so called crosscutting concerns. Crosscutting concerns are hard to implement and maintain in current programming framework.

Aspect oriented software development claims to be able to address the issue of "separation of concerns" by systematic identification, modularization and composition of crosscutting concerns throughout the software life cycle.

AOSD develops from aspect oriented programming[10] which works in programming level. The concepts of aspect-orientation are now progressing into the early stages of software development such as requirements engineering, architecture design. Some researchers are now exploring to extend the AOSD work into the area of component based software development. The idea of combining advantages from both CBSD and AOSD seams attractive. However, AOSD still needs to prove its value in large-scale industry level applications.

2.2.7 Agent Oriented Software Engineering

Agent-oriented software engineering is a new software paradigm. Its main purposes are to create methodologies and tools to enable development of agent-based software[11].

Agent-oriented software engineering suggests dividing software into a group of autonomous agents, which solve problems through flexible high-level interactions. This approach facilitates the natural description of complex system. Since agent is modeled as a high-level abstraction entity, it can reduce overall complexity and ensure loose coupling between system elements.

The adoption of autonomous agent makes agent-oriented software engineering have some unique features and advantage. It is considered as best suitable for future Internet environment. However, agent technology still lacks of support from industries and is far from practical application.

2.2.8 Open Source Software Development

Compared to the traditional development of proprietary software, open-source development[12] is a totally new paradigm. Open source relates to practices in the production of products which promote access to their sources. Many factors, technical and non-technical, economical and political, pushed the prosperous of open source movement.

The impact of open-source development is great. The number of open source projects available today is staggering: for example, there are 116,000 projects hosted on http://sourceforge.net. Many open-source applications are equal to, and even superior to, their proprietary competitors. Successful examples include: GNU/Linux and FreeBSD operating system, PostgreSQL and MySQL database, GNU Compiler Collection (GCC) compiler utilities, the integrated development environment of Eclipse, etc.

Open-source projects are usually developed and maintained by a team of programmers distributed all over the world. They communicate primarily through email, news, and Web documents. They heavily rely on automated tools such as *patch*, version-control systems like CVS, bug- and patch-tracking systems like Bugzilla.

From the development and technology viewpoint, open source affects two things: the systems we build (products) and how we build them (process). Open source enables reuse of high-quality components developed by third party. The open source software development has also evolved a process that works well where traditional processes fail. The traditional software engineering field can learn much from open source movement.

2.3 Challenges Brought by Internet

Internet is undoubtedly one of the greatest achievements human societies made in the last century. Internet has totally changed the traditional computing paradigms by extending the computing infrastructure into global-wide network environment. This brings about both great challenges and opportunities to the whole software engineering community.

In recent years, many new concepts and research topics have been proposed. For example, Grid computing[13] proposes a new model of networked applications from the perspective of resource sharing and management. Pervasive computing[14] discusses a new situation of networked applications from the perspective of human computer interaction. Almost all of the work, also including peer-to-peer computing, semantic web, autonomic computing, and so on, can be considered as attempts to review, rethink and evolve information technology from some new perspectives.

When looking at software in the above areas, it can be concluded that there is a new and significant trend that software becomes much more flexible, goal-directed and continuing reactive. Basically, the emerging evolution of software results from the four driving forces mentioned above in the era of Internet, whose characteristics include heterogeneity, openness, dynamism and variability. Technically, software will become autonomous, evolvable, cooperative, polymorphic and context aware for surviving and growing in the sea of rapid, continuous and unpredictable changes of users and environments. Such software is very different to current software. To distinguish them, we call such new paradigm of software "Internetware".

Obviously, existing software methodologies and technologies cannot support the development, deployment and management of Internetware in an efficient, cost-effective and low risk manner. For example, the development methods for open and dynamic systems are very different to those for closed and static systems, as shown in Fig.3.

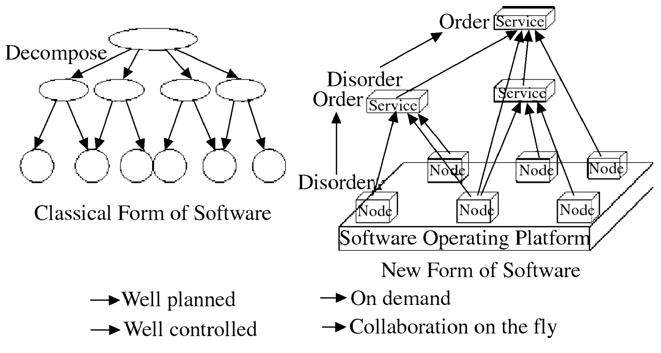

Fig.3. Challenges to development of Internetware.

In traditional software systems, the goals of the target systems are well-planned and deterministic and all resources involved in the development and deployment are available and well controlled. Therefore, the development is usually top-down, i.e., determining the scope of the target system and then decomposing the target system into small pieces of subsystems or modules which can be directly implemented.

On the contrary, the goals of Internetware systems are usually un-deterministic due to the on demand business and all required resources may only be available in a special period and usually controlled by different parties. As a result, the development of Internetware systems is usually bottom-up, i.e., determining the changes caused by on demand business, discovering available resources in Internet and composing them together in terms of desired functions and qualities.

There are many other critical changes in the shift from traditional software to Internetware. For example, the software structure will become much more open and dynamic. The software entity will be active and autonomous. The collaboration among software entities will be more flexible and diverse. The software evolution will continue for a longer period and will be performed at runtime, and so on.

Definitely, these above changes bring new challenges to almost all research and practice areas of software engineering. For coping with these challenges, several Chinese universities and institutes, including Peking University, Nanjing University, Academy of Mathematics and Systems Science of Chinese Academy of Sciences, East China Normal University, Institute of Software of Chinese Academy of Sciences and Tsinghua University, collaborate with each other under the project supported by the National Grand Fundamental Research 973 Program of China, called "Research on Theory and Methodology of Agent-Based Middleware on Internet Platform". The project focuses on four scientific issues:

- The open software model and its characteristics for Internetware, including the model and theory for autonomous entity in open environment, the theory and methods for self-adaptive software architecture, etc.
- Software architecture and methodology for Internetware, including the basic structure for Internetware applications, the methodology frameworks, etc.
- Trustworthy theory and measurement system, including the Internetware formal systems, the measurement and evaluation systems for trustworthiness, etc.
- The new middleware platform for Internetware, investigating the well-structured and adaptive middleware models.

Fig.4. outlines the main research contents of this project organized around the four scientific issues.

Basically, this project takes two different ways approaching Internetware: evolutionary and revolutionary. In the evolutionary approach, existing methods and techniques, e.g., the component model and software architecture, will be enhanced by innovative features required by Internetware. For example, Internetware component will be more active than the current component, and the software architecture will also be more dynamic. As a result, legacy systems or new systems developed by existing methods and techniques can be evolved to Internetware. In the revolutionary approach, agent-like active entities (existing software agents cannot sufficiently support the characteristics of Internetware) will act as the basic units of software systems and all Internetware features will be provided by the capabilities of individual entities and the collaboration between two or more entities.

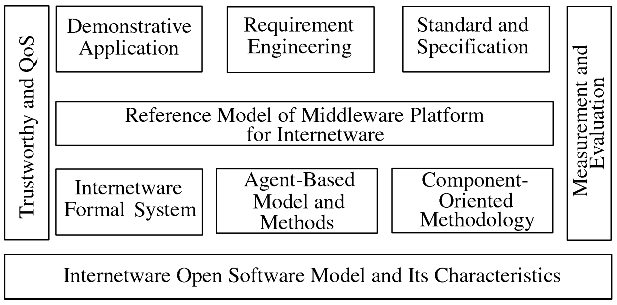
Trustworthy and QoS
Demonstrative Application
Requirement Engineering
Standard and Specification
Measurement and Evaluation
Reference Model of Middleware Platform for Internetware
Internetware Formal System
Agent-Based Model and Methods
Component-Oriented Methodology
Internetware Open Software Model and Its Characteristics

Fig.4. Chinese Internetware project.

2.4 Software Engineering in China

Software engineering in China was started in 1980s. At that time, some pioneering software researchers in China saw the importance of software engineering to the development of Chinese software industry and initiated software engineering research activities. At the same time, Chinese government also recognized the importance of promoting Chinese software industry and funded several projects to support software engineering technology and environment research. From then on, some researchers gradually targeted their research at rising to some well-known challenges in software engineering. In the following, some representative research works are briefly introduced.

2.4.1 Software Automation

This research was conducted by a group from Nanjing University led by Prof. Jia-Fu Xu from 1980s to 1990s, and its primary concern is to automate software production at different levels. Therefore, the results were several systems for automating different aspects of software production throughout the whole lifecycle of software. Typical research includes automated support for requirements analysis (for either structured or object-oriented programming), automated support for system design and programming (such as automation for algorithm design, program synthesis and design of self-adaptive software) and automated support for meta-level program transition etc. More details can be found in [15].

2.4.2 Temporal Logic Based Software Engineering Environment

In 1980s, a research group from Institute of Software, Chinese Academy of Sciences led by Prof. Zhi-Song Tang proposed XYZ/E, the first executable temporal logic language in the world[16]. This language is a practical programming language in which state transition of automata can be directly specified. In the following years, a software engineering environment was developed around this language. In this environment, software artifacts in different development phases can be represented using a unified framework based on XYZ/E. Furthermore, this environment can also support different paradigms of programming (such as object orientation) and/or domain specific programming (such as multimedia programming).

2.4.3 Acquisition and Reuse of Formal Specification

In 1990s, another research group from Software Institute, Chinese Academy of Sciences led by Prof. Yun-Mei Dong started research on acquisition and reuse of formal specification, called MLIRF[17]. It is a formal method for representation, acquisition, and reusing of formal specifications. The aim is to study how to assist human users by computer, through human-machine cooperation, to develop precise, complete and consistent formal specifications, which are approved by human users through verification and then used as the basis of software design and implementation, from human users' vague, incomplete and inconsistent informal statement of needs about target problems, together with, and making full use of, known specification knowledge. Key points in this work are a theory of recursive functions based context insensitive languages and a reuse based method for syntax deduction.

2.4.4 Jade Bird CASE Environment

During the past 20 years, a research group of researchers from Peking University and some other Chinese institutions were devoted in the development of large scale software engineering environments. The primary aim was to provide Chinese software industry with a series of effective software production technologies. As these environments were developed in the Jade Bird project, they were named as JB1, JB2 and JB3. Details of the Jade Bird project will be presented in the next section.

2.4.5 Current Researches and Practices

After more than twenty years of development, the Chinese software engineering research has improved significantly. The importance of software engineering has already been widely accepted by both industry and government. More and more researchers are working in this area. Some distinguished researches have had international impact. A large portion of research results are directly transferred into industry productivity. The software engineering community has made substantial contribution to the rapid development of Chinese information industry and Chinese economy.

The following outlines the representative software engineering research forces and their major researches.

Institute of Software, Chinese Academy of Sciences plays an important role in China software technology research and development. The major research fields and work include: semantic model for distributed object, theories and technologies of networked computing environment and several middleware platforms[18] like J2EE application server OnceAS and message middleware OnceASMQ; software process technology and software quality guarantee platform, CMM-based toolkits integration platform, CMM-based project management tool[19]; formal method for parallel programs[20], process algebras, model checking, modal logics and μ-Calculus, tools and algorithms for concurrent systems, etc.

Software Engineering Institute, Beihang University makes research in tools, methods and technology for testing networked software[21], typical work includes: component-based software analysis, maintenance and testing toolkits QESAT (Quality Easy-Software Analysis and Testing), which supports C/C++ and JAVA language; software testing process management tool QESuite; enterprise process modeling system EPMS based on VPML (Visual Process Modeling Language); visual modeling system UML_Designer. Other researches in Beihang University include: domain oriented application production platform; software system structure and computing environment in Internet, a Web service-based application supporting environment WebSASE[22], etc.

Academy of Mathematics and Systems Science, Chinese Academy of Sciences has long been investigating in the area of domain modeling and knowledge-based software engineering[23]. A domain-knowledge based rapid software production method was proposed, a knowledge based software development environment PROMIS was developed. Recently they focus on ontology based domain modeling method, including formal methods for software ontology, ontology-based requirement modeling and analysis technologies, etc.

Nanjing University mainly focuses on the areas of object-oriented technology and agent-oriented technology[24,25], software automation, formal methods for requirement specification and object specification[26], etc. Corresponding contributions include: UML-based requirement model verification technology, requirement specification language NDRDL and NDORDL, automation system NDRASS and NDORASS, an extension of formal specification Z named COOZ to support the concepts of object, formalization method DD-VDM for concurrent object system, agent-oriented software engineering and agent-based middleware system, mobile agent key technologies—structured mobilization mechanism, layered communication framework, mobile-agent based active component framework, etc.

National University of Defense Technology contributes in distributed object technology, agent technology, middleware technology, formal engineering, etc. Representative work includes: object-oriented CORBA middleware system StarBus[27], component-based application server StarAppserver, multi-agent information system growth environment MAISGE, ontology-based metadata processing system MetaData Pro[28], agent-oriented programming language SLABSp and related toolkits, agent-oriented software development methodology Gaia2S, etc.

East China Normal University specially studies formal methods of specification for computer systems, communications, application and standards, and the techniques for designing and implementing those specifications[29], etc.

Peking University (PKU) is one of those academic institutes that began the first software engineering researches in China. The researches of PKU include software engineering environment, object-oriented technology and component technology, etc.

3 Research Activities in PKU

Software engineering in Peking University started in 1980s. With the leadership of Prof. Fu-Qing Yang and the continuous support from the Chinese government, the software engineering research in Peking University has got many achievements. From 1983, as the leading participant, Peking University undertook the largest software engineering research project sponsored by the government, i.e., the Jade Bird (JB) project. Currently, the researches in Peking University mainly focus on component-based software reuse and Internetware.

3.1 Jade Bird Project

The Jade Bird (JB) project, started in 1983 in the period of the 6th State Five-Year Plan (1981–1985), is a National Science and Technology Major Project. It has lasted more than 20 years through the 7th, 8th, 9th and 10th State Five-Year Plan (1986–2005) and is the largest software project getting continual support from the government.

The objective of JB project is to boost the development of software industry in China by doing research and practice of industrialization of software production, providing advanced software engineering tools for software development enterprises, and helping them to improve their software processes. Software industry is selected as one of the key industries to be developed with top priority in China by the government. The conduction of this project (which is depicted in Fig.5) is one of the important moves aimed at building the necessary infrastructure for rapid development of the software industry.

Thanks to the great efforts and hard working of the project team, a lot of achievements have been obtained. Of these achievements, some have turned into commercial products used in software enterprises, some have been published as research results on national or international journals and conferences, and some have been applied by the project team and other enterprises for improving the productivity and quality of the software development. In the past ten years, this project has received several awards issued by the government and some industry societies, due to its outstanding achievements in research and practice of software engineering and great contribution to industry. It is worth mentioning that in the progress of the project, many young researchers and engineers became mature and have become the backbone of the project team or other software enterprises. The following are some milestones in the progress of the project:

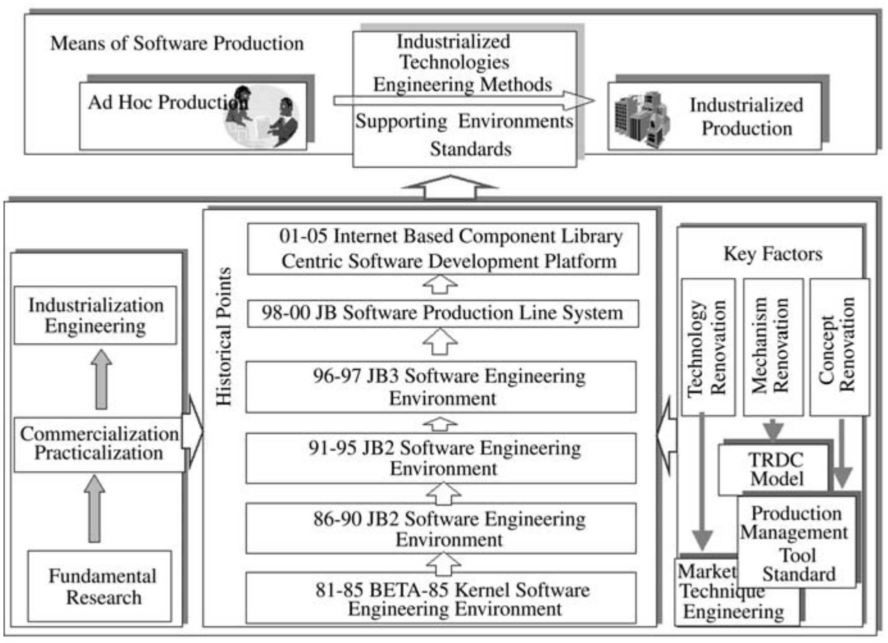
Means of Software Production
Ad Hoc Production
Industrialized Technologies
Engineering Methods
Supporting Environments
Standards
Industrialized Production
Industrialization Engineering
Commercialization Practicalization
Fundamental Research
Historical Points
01-05 Internet Based Component Library Centric Software Development Platform
98-00 JB Software Production Line System
96-97 JB3 Software Engineering Environment
91-95 JB2 Software Engineering Environment
86-90 JB2 Software Engineering Environment
81-85 BETA-85 Kernel Software Engineering Environment
Key Factors
Technology Renovation
Mechanism Renovation
Concept Renovation
TRDC Model
Production Management Tool Standard
Market Technique Engineering

Fig.5. Overview of Jade Bird in the past 20 years.

In 1985, BETA-85, a kernel software engineering environment, was released[30]. It was the foundation of the following development of JB project. The architecture of BETA-85 is depicted in Fig.6.

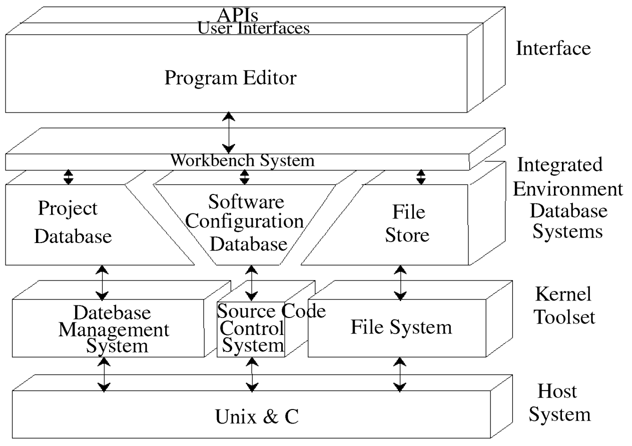

Fig.6. BETA-85: Kernel software engineering environment.

In 1990, JB1, an integrated software engineering environment supporting structured software methodology, was released. It was the first one in the JB environment series, and also the first large-scale commercial software engineering environment in China. It was then the name Jade Bird was first introduced. This environment is depicted in Fig.7.

In 1995, JB2, an object-oriented integrated software engineering environment supporting both OO and structured methodology, was released. It is an enhancement of JB1. At the time, a series of JB software engineering standards and specifications were published. Fig.8 illustrates an overview of JB2. Please refer to [31] for details.

In 1997, JB3, called JB Software Production Line System (depicted in Fig.9) that can support component-based reuse, was released[32]. It is an enhancement of JB2, consisting of a central component library and a set of tools. In addition, a set of technical specifications on software component technology were added into the list of JB standards and specifications. In 2000, a new version of JB3 was released. In the past 5 years, JB3 system has been used in practice widely and got much feedback. By the end of that year, the task of JB Project in the 9th State Five-Year Plan was finished.

In the period of the 10th State Five-Year Plan (2001–2005), JB project was still keeping moving along two directions: the one under the funding of the State High-Tech 863 Program mainly focuses on technology aspects and application promotion for component-based software development, and the other under the funding of the National Grand Fundamental Research 973 Program of China mainly focuses on fundamental theoretical researches for Internetware. In 2005, a software development platform based on a nation-wide component library was set up and began to provide services for software enterprises. Fig.10 depicts the most recent JB environment.

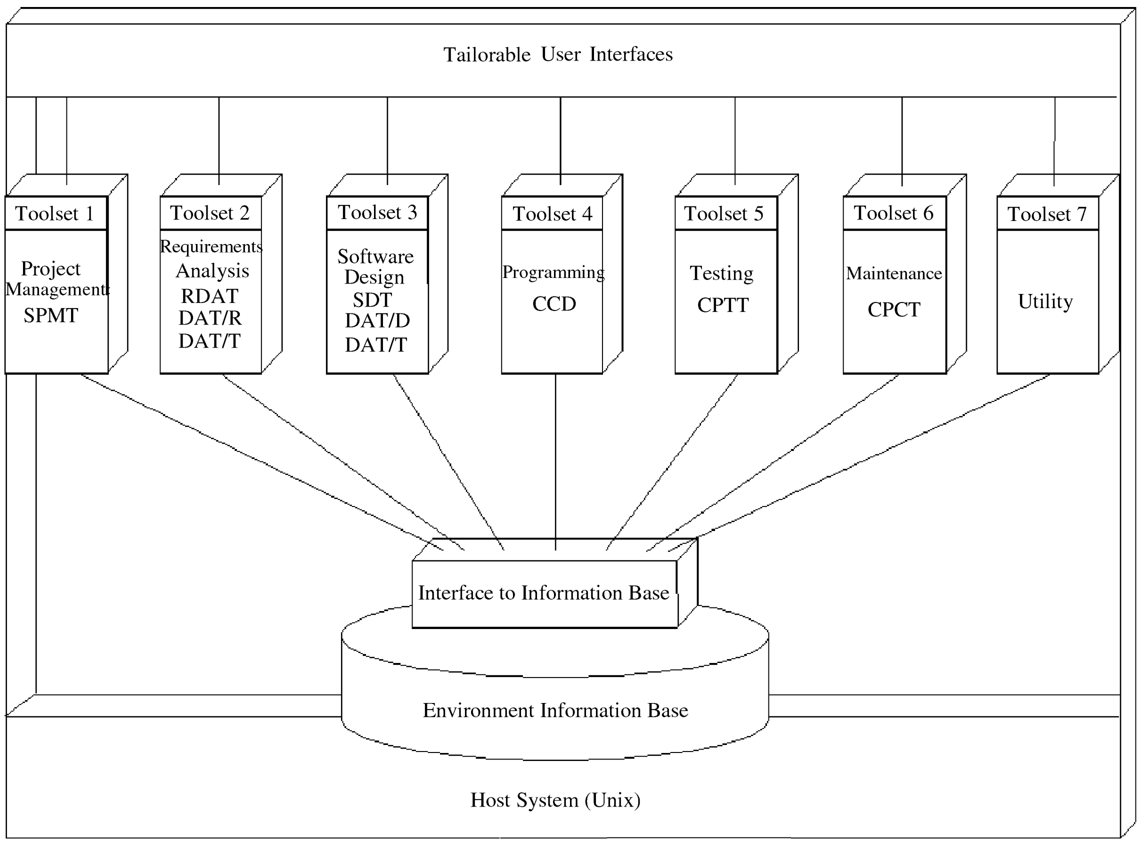

Fig.7. JB1: Integrated software engineering environment.

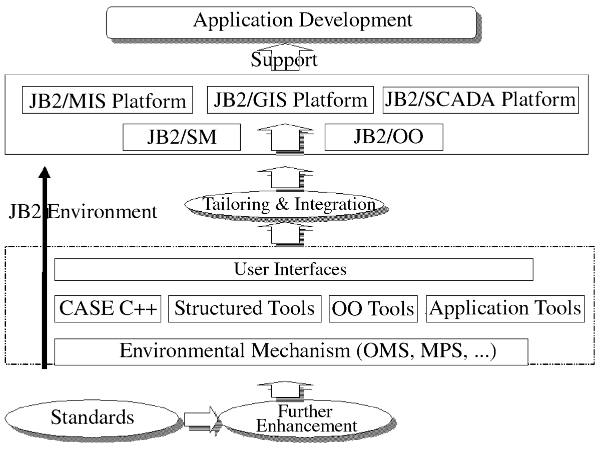

Fig.8. JB2: OO integrated software engineering environment.

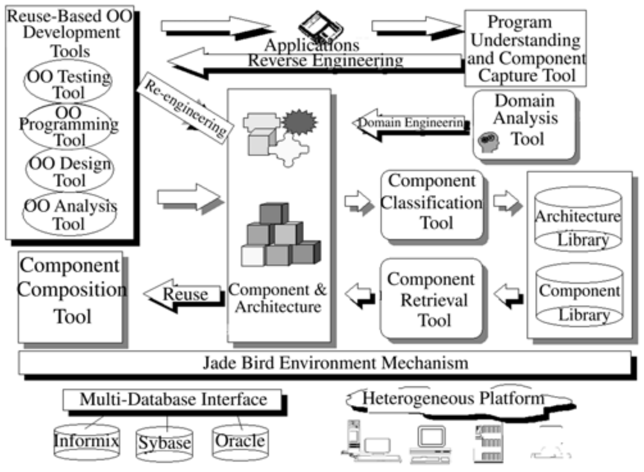

Fig.9. JB3: Software production line system.

3.2 Component Technology Promotion

With the continuous efforts of the JB project, most software enterprises in China have realized that architecture and component based software development is the key to their success. Now, the software industry in China begins to restructure itself spontaneously to meet the requirements of this new paradigm of software development.

In China, more and more software companies have recognized the promising future of component technology and realized the necessary of component library as a basic infrastructure. Chinese government has also been actively pushing software companies to adopt component technology. Some funding projects are also specifically setup for addressing the issues in component promotion.

There are several critical factors for the software companies to successfully adopt the approach of component-based software development: mastering the correct component development methodology, having a rich set of computer-aided toolkits, and the most important, having a large number of reusable components.

The Peking University research team has contributed the following in the promotion of component technology under the fund of the National High-Tech 863 Program of China in the 10th State Five-Year Plan (2001–2005).

• Developed the first software component repository management system in China—the Internet-based component library management system JBCLMS. JBCLMS is the prototype of all the commercial systems currently deployed in the national 863 software incubators.

• Introduced component-based software development method and technology to some selected superior software companies. A series of component-related curriculums are setup to help train software engineers.

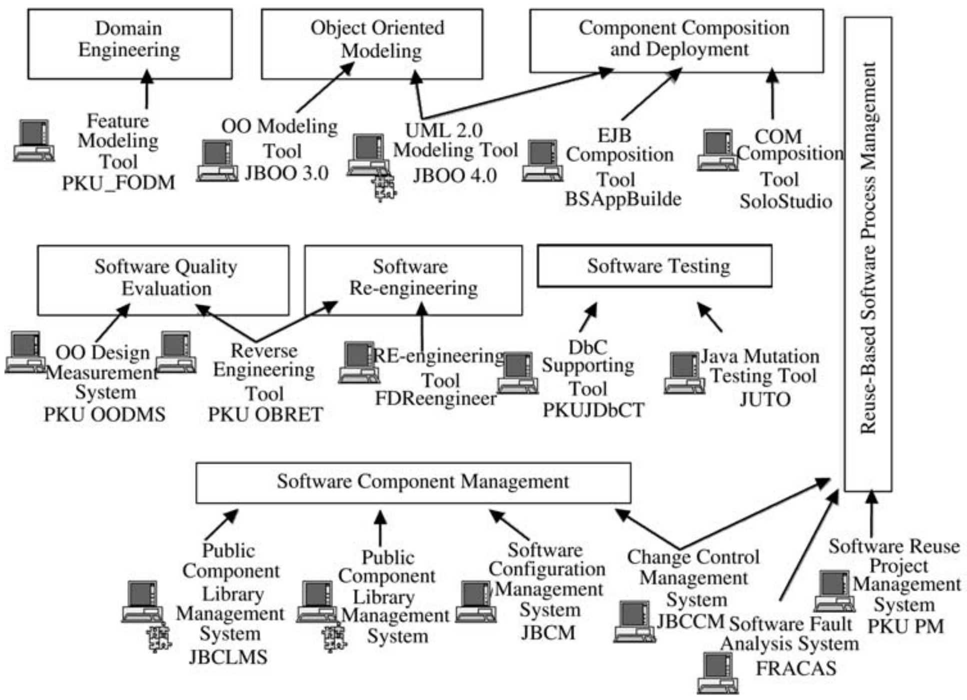
Domain Engineering
Object Oriented Modeling
Component Composition and Deployment
Reuse-Based Software Process Management
Feature Modeling Tool PKU_FODM
OO Modeling Tool JBOO 3.0
UML 2.0 Modeling Tool JBOO 4.0
EJB Composition Tool BSAppBuilde
COM Composition Tool SoloStudio
Software Quality Evaluation
Software Re-engineering
Software Testing
OO Design Measurement System PKU OODMS
Reverse Engineering Tool PKU OBRET
RE-engineering Tool FDReengineer
DbC Supporting Tool PKUJDbCT
Java Mutation Testing Tool JUTO
Software Component Management
Public Component Library Management System JBCLMS
Public Component Library Management System
Software Configuration Management System JBCM
Change Control Management System JBCCM
Software Fault Analysis System FRACAS
Software Reuse Project Management System PKU PM

Fig.10. Internet based, component library centric software development platform.

• Helped deploy several public national-wide component repository management systems in China to promote reuse of assets among different companies.

Since most of the small or medium sized software companies cannot afford a reuse repository of their own, Chinese government has funded to construct several public component libraries that are hosted in software parks and the national 863 software incubators. Software companies are encouraged to put into, and retrieve from the library high-quality domain components. Now the software component library management system has become one of the most important technical infrastructures in the national 863 software incubators. Up till now, there are more than eight active public software component libraries based on JBCLMS distributed in the cities of Beijing, Shanghai, Guangzhou, Shenyang, Changsha, Zhengzhou, and Xi'an, etc.

There are several technical requirements on the component library management system: distribution, autonomy, domains-orientation, etc. One of the most critical issues is that these component systems may use different component classification schemas, which makes them semantically different with each other.

The JBCLMS system developed by Peking University can effectively manage geographically distributed, autonomous component library systems. To connect autonomous libraries together and make the global resources accessible to local users transparently, JBCLMS is designed in layered architecture, as depicted in Fig.11. In this architecture, there are three kinds of nodes: the leaf node, the composite node, and the root node.

• Leaf node is actually a physical component repository. A leaf node is registered to the directory server in the root node. The leaf node provides two searching methods: 1) local searching: searching the reusable resources in local repository; 2) global searching: while doing local searching, also sending queries to parent node.

• The composite node comprises two parts: the physical component repository, and the similarity matrix of terms from different repositories. The composite node has the role of the leaf node plus: responding to the query requests from both parent and child nodes by forwarding the requests to appropriate adjacent nodes according to the terms similarity matrix. The composite node should also maintain the terms similarity matrix periodically.

• The root node is the top node in the interconnection architecture. It comprises two mandatory parts: the directory server, which records the architecture topology of all registered nodes, and the terms similarity matrix. The root node can optionally have a physical repository. However, it is recommended not to include a repository in root node to improve performance. A root node without a physical repository will have all the roles of the composite node plus managing the directory server and maintaining the architecture topology, including node adding, deleting, updating, etc.

Based on the infrastructures of software component library and the agreements between different software parks for sharing software assets, software enterprises all over China can transparently benefit from all the component libraries deployed in the eight software parks. Currently, there are more than 29,000 software components stored in the virtual component library. More than 670 software enterprises have registered to use the service provided by the virtual component library. With the mutual efforts from the government, the academic research institutions and the software companies, the component technology in China has advanced significantly. Those software enterprises that have transferred to component based software development have largely improved their productivity. More and more software companies are going to transfer to component-based approach.

3.3 ABC Methodology for Internetware

One important research contents of the Chinese 973 Internetware project is the Internetware methodology. ABC (Architecture Based Component Composition)$^{[33,34]}$ is such a methodology for Internetware. It is proposed by Peking University and is funded under the National 973 Program and the National Natural Science Foundation of China.

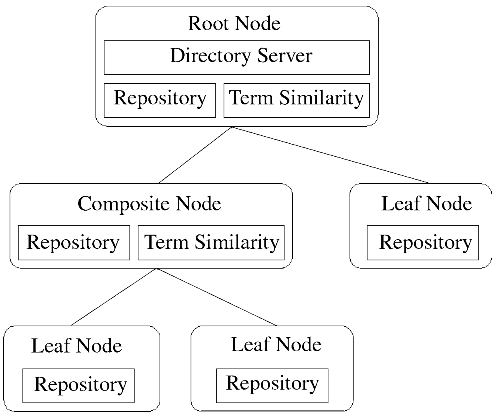

Fig.11. JBCLMS interconnection architecture.

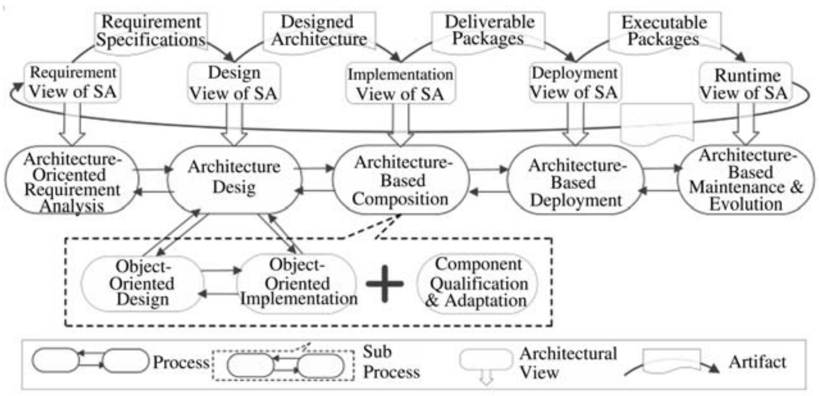
Requirement Specifications
Designed Architecture
Deliverable Packages
Executable Packages
Requirement View of SA
Design View of SA
Implementation View of SA
Deployment View of SA
Runtime View of SA
Architecture-Oricented Requirement Analysis
Architecture Desig
Architecture-Based Composition
Architecture-Based Deployment
Architecture-Based Maintenance & Evolution
Object-Oriented Design
Object-Oriented Implementation
Component Qualification & Adaptation
Process
Sub Process
Architectural View
Artifact

Fig.12. Process model and architectural views of ABC.

ABC employs software architecture (SA) as the blueprint and middleware technology as the runtime infrastructure for the development, deployment, maintenance and evolution of component based systems. The most distinguished feature of ABC is the introduction of SA into each phase of software life cycle, as shown in Fig.12.

To achieve the traceability and consistency between requirement specifications and system design, ABC introduces concepts and principles of software architecture into requirements analysis and specifications. In this phase, there is no actual SA but only the requirement specifications of the system to be developed, which are structured in the way similar to SA. It consists of a set of component specifications, connector specifications and constraint specifications and will be used as the basis for software architecting.

In the phase of software architecting, the requirement specifications are refined, and some overall design decisions are made. To produce SA meeting functional and non-functional requirements of the target system, the architects may study the requirement specifications, refine components and connectors in the problem space, create necessary artificial components and connectors, produce dynamic and static SA models, build mapping relationships between requirement specifications and SA, check SA and so on.

In the phase of component composition, the components, connectors and constraints in the reusable assets repository will be selected, qualified and adapted to implement the logic entities produced in architecting. However, there are still some elements unable to be implemented by reusable assets. These elements have to be implemented by hand in object-oriented languages or other ones. After implementation and testing, the elements will be stored into the repository and then composed into the target system as reusable assets. In that sense, the design view of SA can be fully implemented by the reusable assets.

Component-based systems are usually implemented and executed with the help of some common middleware, such as CORBA/CCM, J2EE/EJB and COM. Before the implementation of the system being executed, it must be deployed into the middleware platform. In this phase, SA should be complemented with some information so that middleware can install and execute the system correctly. Typically, the information includes declaration of its required resources, security realm and roles, component names for runtime binding, and so on.

In the phase of maintenance and evolution, software architecture also has an important role. ABC proposes a new concept of RSA (Runtime Software Architecture). RSA is based on reflection technologies and represents the actual situation of the runtime system. The runtime view of SA could help software maintenance and evolution.

There are many new ideas together with research results in the ABC approach, including the feature modeling for reusability[35,36], the software architecting for changeability[37,38], the architecture based deployment for high performance and dependability[39], the architecture based reflection for autonomic management[39], and so on.

3.3.1 Feature-Oriented Domain Modeling

ABC uses feature oriented modeling technology in requirement modeling[35,36]. This approach treats features as the basic entities in the problem space, and uses features and relations among them (namely, feature model) to specify the problem space.

A feature describes a software characteristic from user or customer viewpoint, which essentially consists of a cohesive set of individual requirements. Because feature model is structured, it is possible to map requirement specification to SA and facilitate the reuse of SA. But feature model is very different to software architecture, therefore there are two critical issues to address, i.e., how to trace features to components and how to construct the software architecture based on the feature model. It should be noted that the software architecture here is an abstract and acts as the architecture oriented requirement specifications.

The concept of responsibility is introduced by ABC to handle these two issues[41]. A responsibility is a cohesive set of program specifications from programmers' viewpoint, and can be used as a unit for work assignment. Tracing features to components can be done in a two-step way: first operationalizing features into responsibilities, then assigning responsibilities to components. The problem of the software architecture's construction can be decomposed into two sub-problems to separately tackle with: component construction and interaction identification.

3.3.2 Software Architecting

To make Internetware software able to adapt to changes, ABC proposes the modeling of self-adaptive software architecture. The idea behind this approach is simple. 1) Software architects can predict some changes and plan corresponding adaptations before the target system runs. But software architects usually do not take the runtime adaptable mechanisms into account when designing the target system. 2) The adaptable or reflective middleware provides powerful and practical mechanisms for monitoring and changing runtime systems. However, these mechanisms only handle "how to do" issue other than "why, when and what to do" issue.

The ABC approach utilizes both the design time decisions and runtime mechanisms. It synthesizes some sophisticated methods in architecture based software engineering, i.e., the quality analysis in software architecture for WHEN to change, the design and description of dynamisms in software architecture for WHAT to change, and the runtime software architecture for HOW to change. Some case studies, such as adaptation of workflow[37] and exception handling in software architecture[38], demonstrate the effectiveness and feasibility of the idea.

3.3.3 Deployment and Evolution

In ABC, software architecture is introduced into the software deployment phase for facilitating and automating the analysis, reasoning and decision making[39]. The software architecture of the system to be deployed can be visualized, which helps the deployers to understand the structures, behaviors, desired functions and qualities of the system. All machines in a local area network and their resource consumption can be also visualized so that the deployers can allocate enough resources to the new system by dragging-and-dropping the components to a machine. Moreover, since almost configuration information can be derived from the software architecture description, it is not necessary for deployers to write the configuration by hand any more.

ABC also proposes the concept of runtime software architecture (RSA) to help software maintenance and evolution. In ABC, runtime software architecture represents a runtime system as a set of architectural elements that are causally connected with the internal states and behaviors of the runtime system[39]. The causal connection is implemented as reflection so that changes to the runtime software architecture immediately cause corresponding changes to the runtime system.

The correctness of the reflection, that is, the changes taking place in the runtime software architecture and the runtime system are exactly equal and do not cause deadlocks, can be proved by some formalized methods[42]. Based on the architecture based reflection, a set of autonomic management programs can be implemented, such as the automatic configuration of the thread pool for achieving the best response time and throughput[39], the automatic coordinated recovery of correlated faults of middleware services[43], and so on.

4 Conclusion

The development of software engineering has been accompanied by the rapid growth of software technology and software industry in the past four decades. When looking back to the history, four main driving forces for software technologies can be identified. The driving forces technically determine the development of software engineering. Actually, milestones of software engineering in the history can also be viewed as reflection of the changes in software technologies.

In the new millennium, the drastically increasing demands on software technologies and the extremely open and dynamic nature of the Internet have presented new challenges to software engineering.

This paper reviews the history of software engineering in the world and especially in China. Based on these reviews, the possible future software engineering research directions under Internet environment are explored. As an example, the research activities in Peking University are introduced in detail.

Acknowledgements The authors would like to thank our research group for their inspiration and support in the past two decades. Due to the space limit and the language, so many papers and reports, especially in Chinese, referred by this paper are omitted here. We would like to thank the authors of these materials.

References

[1] Brooks F P Jr. No silver bullet: Essence and accidents of software engineering. *Computer*, April 1987, 20(4): 10-19.

[2] Zhang Xiaoxiang. Encyclopedia of Computer Science and Technology. Second edition, Beijing: Tsinghua University Press, 2005. (in Chinese)

[3] Hong Mei. A component model for perspective management of enterprise software reuse. *Annals of Software Engineering* USA, 11(1): 219–236.

[4] Papazoglou M P, Georgakopoulos D (eds.). Special issue on service oriented computing. *Communications of ACM,* Oct. 2003, 46(10): 24–60.

[5] Fuggetta A. Software process: A roadmap. In *Proc. the Conf. Future of Software Engineering*, Limerick, Ireland: ACM Press, 2000, pp.25–34.

[6] Zave P. Classification of research efforts in requirements engineering. *ACM Computing Surveys,* 1997, 29(4): 315–321.

[7] Bashar Nuseibeh, Steve Easterbrook. Requirement engineering: A roadmap. In *Proc. the Conf. Future of Software Engineering*, Limerick, Ireland: ACM Press, 2000, pp.35–46.

[8] Frankel D S. Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley, 2003.

[9] Meservy T O, Fenstermacher K D. Transforming software development: An MDA roadmap. *Computer,* 2005, 38(9): 52–58.

[10] Kiczales G *et al.* Aspect-oriented programming. In *Proc. 11th European Conf. Object-Oriented Programming* (*ECOOP*), Springer-Verlag, Finland, 1997, pp.220–243.

[11] Wooldridge M. Agent-oriented software engineering. In *IEE Proc. Software Engineering,* 1997, 144(1): 26–37.

[12] Diomidis Spinellis, Clemens Szyperski. How is open source affecting software development? *IEEE Software*, Jan/Feb 2004, 21(1): 28–33.

[13] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. *Int. Journal of High Performance Computing Applications,* 2001, 15(3): 200–222.

[14] Satyanarayanan M. Pervasive computing: Vision and challenges. *IEEE Personal Communications,* 2001, 8(4): 10–17.

[15] Xu J, Lu J. Software Languages and Their Implementation. Scientific Publishing House, 2000. (in Chinese)

[16] Tang C-S. Toward a unified logic basis for programming languages. *IFIP Congress,* North Holland, Amsterdam, 1983, pp.425–429.

[17] Dong Y, Li K, Chen H *et al.* Design and implementation of the formal specification acquisition system SAQ. In *Conf. Software: Theory and Practice, IFIP 16th World Computer Congress 2000,* Beijing, 2000, pp.201–211.

[18] Jinling Wang, Beihong Jin, Jing Li. An ontology-based publish/subscribe system. In *5th ACM/IFIP/USENIX Int. Middleware Conference* (*Middleware 2004*), Toronto, Canada, Oct. 2004, Springer, LNCS 3231, pp.232–253.

[19] Li Juan, Li Ming-Shu, Wu Zhan-Chun, Wang Qing. A SPEM-based software process metamodel for CMM. *Journal of Software,* 2005, 16(8): 1366–1377.

[20] Lin H. Complete inference systems for weak bisimulation equivalences in the pi-calculus. *Information and Computation,* 2003, 180(1): 1–29.

[21] Liu Chao, Zhang Mao-lin, Yan Hai-hua *et al.* Teamwork collaboration model of software integration testing and its characteristics. *Journal of Software,* 2000, 11(6): 841–847.

[22] Ge Sheng, Hu Chun-ming, Du Zong-xia *et al.* WebSASE: A web service-based application supporting environment. In *Proc. the 5th Northeast Asia Symposium,* Seal, Korea, Aug 6, 2002, pp.67–76.

[23] Ruqian Lu, Zhi Jin. Formal ontology: Foundation of domain knowledge sharing and reusing. *Journal of Computer Science and Technology,* September 2002, 17(5): pp.535–548.

[24] Lu Jian. Developing parallel object-oriented programs in the framework of VDM. *Annals of Software Engineering,* 1996, 2(9): 199–211.

[25] Lu Jian, Li Yingjun, Ma Xiaoxing *et al.* A hierarchical framework for parallel seismic applications. *Communications of ACM,* 2000, 43(10): 55–59.

[26] Li Xuandong, Zhao Jianhua, Pei Yu *et al.* Positive loop-closed automata: A decidable class of hybrid systems. *Journal of Logic and Algebraic Programming,* 2002, 52-53: 79–108.

[27] Wang H M, Wang Y F, Tang Y B. StarBus+: Distributed object middleware practice for Internet computing. *Journal of Computer Science and Technology,* 2005, 20(4): 542–551.

[28] Ji Wang, Wei Dong, Zhi-Chang Qi. Slicing hierarchical automata for model checking UML statecharts. *Formal Methods and Software Engineering. Lecture Notes in Computer Science 2495,* Springer, 2002.

[29] He Jifeng, Tony Hoare. Unifying Theories for Parallel Programming. *Lecture Notes in Computer Sciences 1300*, Springer, 1997, pp.15–30.

[30] Yang F *et al.* Kernel Software Engineering Environment BETA-85. *Science in China* (A), 1988, 18(5): 530–538.

[31] Yang F, Shao W, Mei H. The design and implementation of object-oriented CASE environment Jade Bird 2(JB2). *Science in China* (A), 1995, 25(5): 533–542. (in Chinese)

[32] Yang F, Mei H. Research on technology for industrialization production of software—Practice of JB (Jade Bird) project. In *Symp. Sino-American Engineering Technology,* Beijing, Oct. 1997, pp.190–200.

[33] Mei H, Chang J C, Yang F Q. Software component composition based on ADL and middleware. *Science in China* (Series F), 2001, 44(2): 136–151.

[34] Mei H, Chen Feng, Wang Qianxiang, Feng Yaodong. ABC/ADL: An ADL supporting component composition. In *Proc. 4th Int. Conf. Formal Engineering Methods* (*ICFEM2002*), Shanghai, China, 2002, pp.38–47.

[35] Mei H, Zhang W, Gu F. A feature oriented approach to modeling and reusing requirements of software product lines. In *Proc. COMPSAC,* Dallas, USA, 2003, pp.250–256.

[36] Zhang W, Mei H, Zhao H Y. A feature-oriented approach to modeling requirements dependencies. In *Proc. 13th IEEE Int. Conf. Requirements Engineering* (*ICRE*), La Sorbonne, France, August 29–September 2, 2005, pp.273–282.

[37] Zhu Y, Huang G, Mei M. Quality attribute scenario based architectural modeling for self-adaptation supported by architecture-based reflective middleware. In *Asia Pacific Software Engineering Conference* (*APSEC 2004*), Busan, Korea, Nov. 30–Dec. 3, 2004, pp.2–9.

[38] Feng Y, Huang G, Zhu Y L, Mei H. Exception handling in component composition with the support of middleware. In *Fifth Int. Workshop on Software Engineering and Middleware* (*SEM 2005*), co-located with ESEC-FSE'05, Lisbon, Portugal, ACM Press, September 5–6, 2005, pp.90–97.

[39] Huang G, Liu T C, Mei H *et al.* Towards autonomic computing middleware via reflection. In *Proc. 28th Annual Int. Computer Software and Applications Conference* (*COMPSAC*), Hong Kong, China, Sept. 28–30, 2004, pp.122–127.

[40] Huang G, Mei H, Yang F Q. Runtime recovery and manipulation of software architecture of component-based systems. *International Journal of Automated Software Engineering,* 2006, 13(2): 257–281.

[41] Zhang W, Mei H, Zhao H Y, Yang J. Transformation from CIM to PIM: Feature-oriented component-based approach. In *Proc. 8th Int. Conf. Model Driven Engineering Languages and Systems* (*MoDELS 2005*), Montego Bay, Jamaica, Oct. 2–7, 2005, LNCS 3713, 2005, pp.248–263.

[42] Shen J, Sun X, Huang G *et al.* Towards a unified formal model for supporting mechanisms of dynamic component update. *The Fifth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering* (*ESEC-FSE'05*), Lisbon, Portugal, September 5–9, 2005, pp.80–89.

[43] Liu T, Huang G, Fan G, Mei H. The coordinated recovery of data service and transaction service in J2EE. In *Proc. 29th Annual Int. Computer Software and Applications Conference* (*COMPSAC05*), Edinburgh, Scotland, July 2005, pp.485–490.

Hong Mei is a professor in computer science at the Peking University, China. He got the B.Eng. and M.S. degrees in computer science from Nanjing University of Aeronautics & Astronautics (NUAA) in 1984 and 1987 respectively, and the Ph.D. degree in computer science from Shanghai Jiao Tong University in 1992. His current research interests include: software engineering and software engineering environment, software reuse and software component technology, distributed object technology, etc.

Dong-Gang Cao is an assistant professor in the School of Electronics Engineering and Computer Science at the Peking University, China. He received the B.Eng. and M.Eng. degrees in computer science & technology from the Beijing University of Posts and Telecommunications in 1998 and 2001 respectively, and the Ph.D. degree in computer science from Peking University in 2005. His research interests include software engineering, Internet technologies and component-based software development.

Fu-Qing Yang is a professor in computer science in the School of Electronics Engineering and Computer Science, the Dean of the Faculty of Information and Engineering Sciences, Peking University. She is an academician of the Chinese Academy of Sciences and an IEEE fellow.