Development of Software Engineering: Co-operative efforts from academia, government and industry

Fuqing Yang
School of Electronics Engineering and Computer Science, Peking University
Beijing, 100871, China
yang@sei.pku.edu.cn

Hong Mei
School of Electronics Engineering and Computer Science, Peking University
Beijing, 100871, China
meih@pku.edu.cn

ABSTRACT
In the past 40 years, software engineering has emerged as an important sub-field of computer science. The quality and productivity of software have been improved and the cost and risk of software development been decreased due to the contributions made in this sub-field. The software engineering community needs to invest much more efforts to cope with the drastically increasing demands on the information technology as well as the extremely open and dynamic nature of the Internet. The history of software engineering is reviewed with emphasis on the driving forces of software and the milestones of software engineering development. The history of software engineering in China is reviewed with emphasis on the relationship between software engineering and the software industry. Based on the above reviews, we argue that software engineering should become an independent discipline along with computer science and co-operative efforts from academia, governments and industries should be needed for the harmonious development of software engineering. Some results are presented based on China's experience of developing software engineering under this model.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]: General

General Terms
Design, Management, Human Factors, Standardization.

1. HISTORICAL REVIEW
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. Now it 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 to software crisis. And its goal doesn't change much: producing quality software using limited resources and time.

In a historical point of view, software engineering technology and software technology moves forward hand in hand. Technically, software technology has significant impact on software engineering development. So it is necessary to investigate the driven forces of software technology to understand the nature of software engineering and foresee its future trends.

1.1 Driving Forces of Software Technology
Software is essentially a computer program modeling the problem space of the physical world and its solution. Software can perform various tasks such as controlling hardware devices, computing, communicating, etc. It always pursues a computing model ease of construction and evolution.

Software technology has developed rapidly since its birth. Generally, there are four driving forces of software technology:

- 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. Now the booming of Internet and the ubiquitous computing devices pose great challenges to and require corresponding support from software and software engineering technology.
- Pursuing a computing model that is both expressive and natural. A basic software model comprises entity elements and the interactions among them. The 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 interoperation is always necessary. Operating system and program language makes heterogeneous hardware transparent to programmer and user, while middleware resolves the heterogeneity of OS and language, 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

ICSE'06, May 20-28, 2006, Shanghai, China.

programmer's abstraction level to work in. When commonalities are abstracted, reuse is possible to increase the productivity and quality of software development. Operating system relives 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.

The development of software engineering is closely related to that of software itself. When software technology evolves, it requires corresponding support from software engineering in theory, methodology and methods, or it won't satisfy the industry's requirements. The soundness of this rule is proved by the software engineering evolution road from structured analysis and design, object-oriented development, to the currently popular component-based software development.

1.2 Milestones of Software Engineering

In its three 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 repository emerged. The concept of software lifecycle boosted the development of programming methodologies and project management. Structured programming, ADT, and 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 and software process improvement, 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, 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. This technology is firmly supported by many leading companies such as Microsoft, SUN, IBM, etc, thus getting popular quickly. Component technology might be the key towards manufacturing software in an engineering approach.

2. SOFTWARE ENGINEERING IN CHINA

Software engineering in China was started in 1980. At that time, there was almost no software industry. Therefore, one of the most important goals of software engineering in China was to put forward Chinese software industry. As the development of software engineering in China was behind that in Europe and North America at that time, most research and development were conducted based on leading results produced by researchers from Europe and North America. As Chinese government also realized the importance of Chinese software industry, software engineering research gradually acquired much funding from the government.

An obvious effect of this way of software engineering development was that it can accelerate software technology maturation and thus boost the development of Chinese software industry. As a result, in less than 30 years, both the software industry and software engineering research in China achieved very rapid growth. Figure 1 depicts the increasing revenue of Chinese software industry in the past five years. Figure 2 depicts the changes of the software industry in the world. From these two figures, we can see clearly the rapid increase of Chinese software industry in recent years. Under this scenario of software engineering research, some researchers gradually targeted their research at some well-known challenges in software engineering. In the following, some representative research works are briefly introduced with special emphasis on the largest project for research on software technologies in China, known as the Jade Bird project.

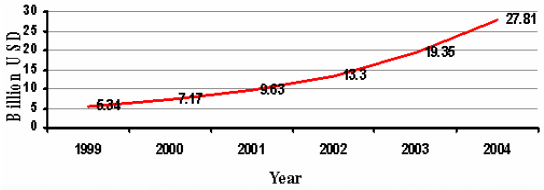

Figure 1. Revenue of Chinese software industry

Country or Area	China	U.S.	West Euro.	Japan	India	S. Korea	Others	Total
2000	7.17	240	186	57.2	8.85	8.32	88.46	596
Percentage	1.20%	40.20%	31.20%	9.60%	1.48%	1.39%	14.84%	100%
2004	27.81	311.5	238.2	83.2	20	20.7	81.19	782.6
Percentage	3.55%	39.80%	30.44%	10.63%	2.56%	2.65%	10.37%	100%

Figure 2. Changes of Revenues of 2000 and 2004 (billion$)

2.1 Representative Research Work

2.1.1 Software Automation

This research was conducted by a group from Nanjing University led by Prof. Jiafu 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. Please refer to [17] for more details.

2.1.2 Temporal Logic Based Software Engineering Environment

In 1980s, a research group from Institute of Software, Chinese Academy of Sciences led by Prof. Zhisong 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). This work received a top award issued by the government.

2.1.3 Acquisition and Reuse of Formal Specification

In 1990s, another research group from Software Institute, Chinese Academy of Sciences led by Prof. Yunmei Dong started research on acquisition and reuse of formal specification, called MLIRF [1]. 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.

2.1.4 Jade Bird Software Engineering Environment Series

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.

2.2 Jade Bird

The Jade Bird (JB) project, started in 1983 in the period of the 6th State Five-year Plan (1981-1985), is a key/major/grand national science and technology project. It has last more than 15 years through the 7th, 8th, and 9th State Five-year Plan (1986-2000) and is the largest software project getting continual support from the government. More than 20 institutions, 300 researchers and developers are involved in the development of JB. The name of this project comes from a lucky and divine bird in ancient Chinese mythology, who serves as a message deliverer in the world of gods. With 3 feet, black eyes and red head, this beautiful bird is viewed as a lucky symbol in China. There are many tales about her in the folklore and literature.

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 Figure 3) is one of the important moves aimed at building the necessary infrastructure for rapid development of the software industry.

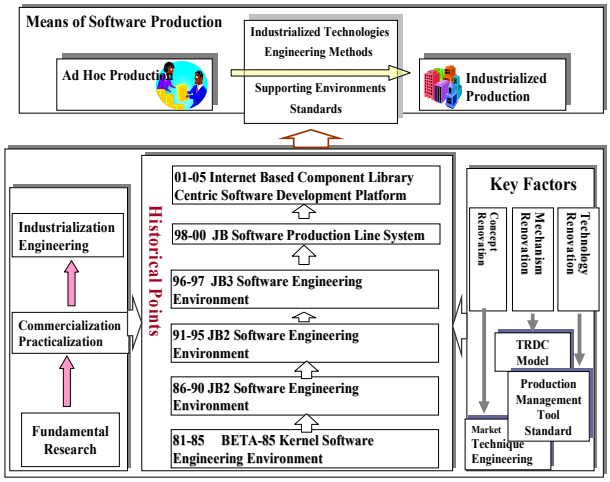

Figure 3. Overview of Jade Bird in the past 20 years

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:

- In 1985, BETA-85, a kernel software engineering environment, was released [18]. It was the foundation of the following development of JB project. The architecture of BETA-85 is depicted in Figure 4.
- 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 that the name Jade Bird was first introduced. This environment is depicted in Figure 5.
- 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. Figure 6 illustrates an overview of JB2. Please refer to [19] for details.

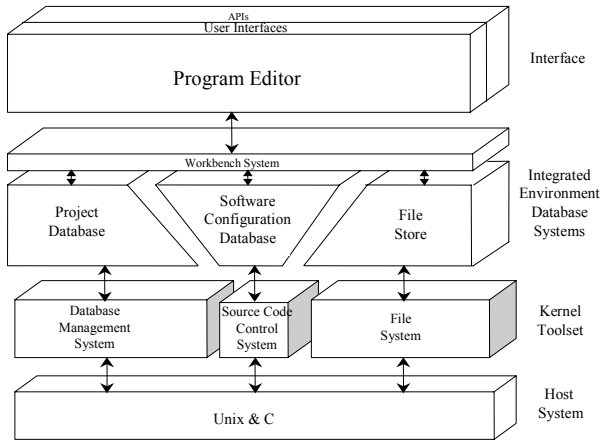

Figure 4. BETA-85: kernel software engineering environment

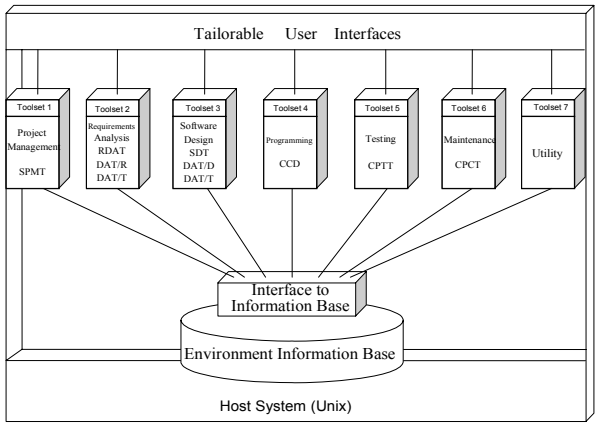

Figure 5. JB1: integrated software engineering environment

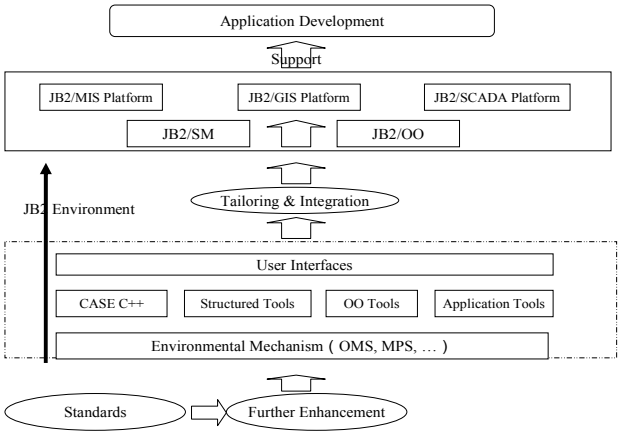

Figure 6. JB2: OO integrated software engineering environment

- In 1997, JB3, called JB Software Production Line System (depicted in Figure 7) that can support component-based reuse, was released [20]. 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 & specifications. In 2000, a new version of JB3 was released. In the past 3 years, JB3 system has been used in practice widely and got much feedback. By the end of this year, the task of JB Project in the 9th State Five-year Plan was finished.

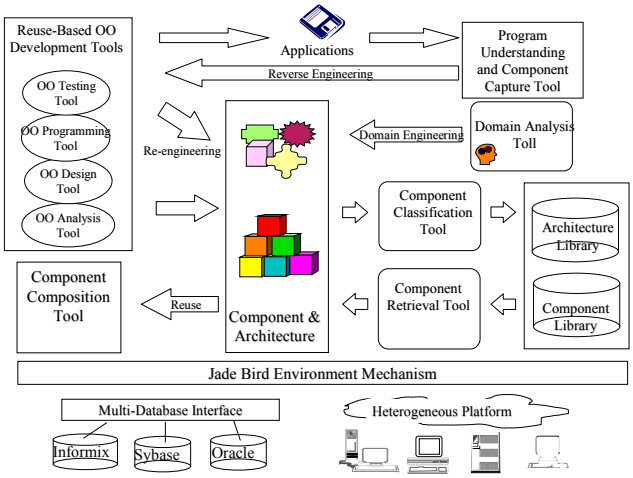

Figure 7. JB3: software production line system

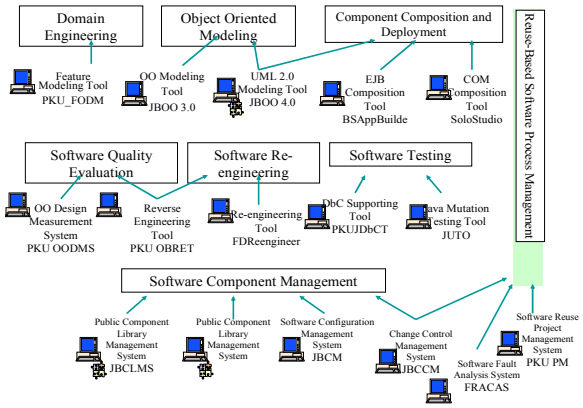

Figure 8. JB environment in the 10th State Five-Year Plan

- In the period of the 10th State Five-Year Plan (2001-2005), JB project was still keeping moving ahead under the funding of the State 863 High-Tech Program and the National 973 Key Basic Research and Development Program. It focused on the research on the architecture and component based software development technology and the promotion of the industrialization of software production in China. In 2005, a software development platform based on a nation-wide component library was set up and began to provide services for software enterprises. 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. Figure 8 depicts the most recent JB environment.

3. CO-OPERATIVE DEVELOPMENT

Software engineering is a typical cross-disciplinary. It applies technologies and practices from computer science, project management, traditional engineering, mathematics, management, psychology, economics, application domains, and other fields.

In many countries, software engineering affects economies and societies—especially the IT industry—in many ways. In China, software industry plays a very important role in the national economy and contributes a lot to the economic growth. Thus there is urgent demand of strong support from software engineering. To meet the requirements, a harmonious, co-operative software engineering development model suitable for the situation of China is practiced.

There are mainly three forces in this development model: the Chinese government, the industry and the academy. The government plays the role of making policies, sponsoring fundamental research projects and key technical-problem attacking projects, constructing public infrastructure for the whole industry. The academy plays the roles of education, training, taking projects funded by the government and industry, and making advanced research. The industry gets strong support from government and academy. Some of the technologies and tools that the industry need are either transferred from the research results from the academies or freely provided by the government as public infrastructure.

Practices in China show that this model is very effective and greatly promotes the ecosystem. The industry develops rapidly. Many companies grow up competitive in free market. The academy receives considerable funding to carry out research keeping the pace with the world.

3.1 Industry

The goals of China software industry development include: a) extending the scale of the industry and the share of the global market; 2) having kernel technologies and famous brand products; 3) having competitive multi-national companies. What is more, China is the world's largest developing economy, which requires that quantities of high-quality software are produced in low-cost within controlled time.

To meet the requirements, the software industry should change the production mode from handcraft to engineering. This transition requires strong supports from technology, infrastructure and human resources. In this era of no silver bullets, component based software development shows its potential and is selected as one of the cornerstones of the technology system. Figure 9 illustrates the desired industry structure and the underlying infrastructure.

In this industry structure, responsibilities are well divided: there are the component vendors providing reusable domain-specific components or general components, which are either reused by the system integration vendors or the service providers. These companies get support from the industry infrastructures. The most important infrastructure here is the various component libraries: enterprise component library, municipal component library, domain component library and the national component library. Among them the last three are public libraries. Software companies are encouraged to put their components that have been proved soundness into the public library. There are incentives and protections for sharing and fees for reuse.

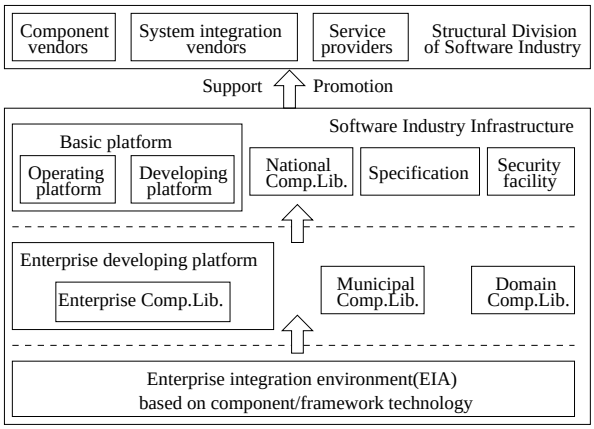

Figure 9. Structure of China software industry and the underlying infrastructures

Peking University has developed a component library management system with the following features: 1) Support component-centered software asset management: component description, classification, entity-relationship customization and management. 2) Support the description and management of on-line service components. 3) Support various access methods, e.g., the WWW method and the Web Services-based API method. Component library greatly facilitates reuse. Now this system has become one of the most important technical infrastructures in the national 863 software incubator centers. Up till present, there are more than eight active public software component libraries deployed in the cities of Beijing, Shanghai, Guangzhou, Shenyang, Changsha, Zhengzhou, and Xi'an, etc. These component library management systems are autonomous, each having different resources targeted for special domains and users. The interoperation among them is well supported.

3.2 Education

The immense growth of software industry requires corresponding increase of professional software engineers and researchers. Those making fundamental researches can be educated by the traditional computer science departments of universities. However, those making practical development (usually blue collar workers and white collar engineers) are badly short of. These practitioners are required to have not only the skills of software developing but also domain knowledge that is hard to obtain from the classical education systems. And another more severe problem is that the number of software engineers that the industry demands far exceeds that the education system could provide. In year 2001, the shortage was expected to be over 200,000. To meet the demands, since 2002, Chinese government has funded to set up more than 35 demonstrative schools of software, amongst of which the most representative one is the school of software, Peking University.

The school of software, PKU aims at training high-level talents with cross-disciplinary knowledge, domain expertise and global perspective. The graduates are expected to have strong professional background and balanced intelligent competence, capable of software development and project management, and have foreign language competence.

Currently there are over 2000 enrolled students, 33 full-time staffs (12 foreigners) and 50 part-time staffs. The curriculum is set up to meet the requirements of industry and market. Now the school offers 142 courses on software engineering, integrated circuit design and engineering, social sciences and humanities, advanced technologies, etc. Many courses are taught in English.

There are three national bases in the school of software, PKU: the National Talent Training Base in Software (Beijing), National Talent Training Base of Integrated Circuit, and the Beijing Engineering Base, National Engineering Research Center of Software Engineering.

The school has good relationship with industries. There are 12 joint-labs funded by well-known companies. And more than 200 companies provide about 1600 internships every year. Students have chances to intern in the leading corporations such as IBM, Microsoft, Intel, Motorola, Lenovo, Founder, Cadence, AMD, Huawei, etc. The fact that the students are well welcomed show that the educating and training mode of software school is successful.

These 35 schools of software contribute a lot to the whole software ecosystem. In 2004, the number of software engineers graduated was 170, 000, four times that in 2000. The human resource shortage problem was alleviated.

3.3 Research

3.3.1 Internetware: A New Software Paradigm

In recent years, many new concepts and research topics have been proposed. For example, Grid computing [3] 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. Service Oriented Computing [13] focuses on a new form of software with emphasis on collaboration and dynamism from the philosophy of software as a service. Almost all of the work, also including peer-to-peer computing, semantic web, autonomic computing, model driven architecture, etc., can be considered as attempts to review, rethink and evolve information technology from some new perspectives.

When looking 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 Figure 10. 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, such as the software structure becoming much more open and dynamic, the software entity becoming active and autonomous, the collaboration among software entities becoming more flexible and diverse, the software evolution continuing for a longer period and being performed at runtime and so on.

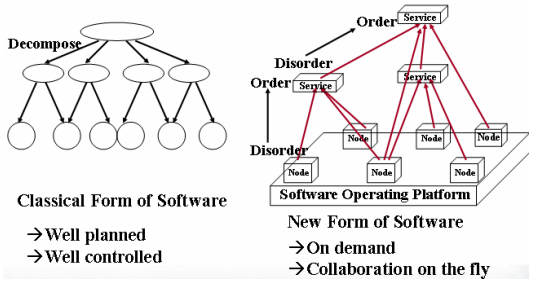

Figure 10. Challenges to development of Internetware

3.3.2 Chinese research on Internetware

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 Sciences, East China Normal University, Institute of Software of Chinese Academy Sciences and Tsinghua University, collaborate with each other under the support of the national basic research program (973), called "Research on Theory and Methodology of Agent-based Middleware on Internet Platform". The project aims to the formal model for capturing basic characteristics Internetware, the agent-based theory and method for incarnating the behavioral nature of Internetware, the component-based methodology for developing Internetware systems, the reference model of middleware for Internetware, the trust theory and measurement for Internetware, and the technical specifications and demonstration applications of Internetware.

Basically, this project takes two different ways, evolutionary and revolutionary, approaching Internetware. 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. 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.

3.3.3 *ABC as a methodology for Internetware*

Peking University focuses on the methodology of engineering Internetware. Belonging to the evolutionary approach, the methodology is called ABC (Architecture Based Component Composition) [10], which 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 Figure 11.

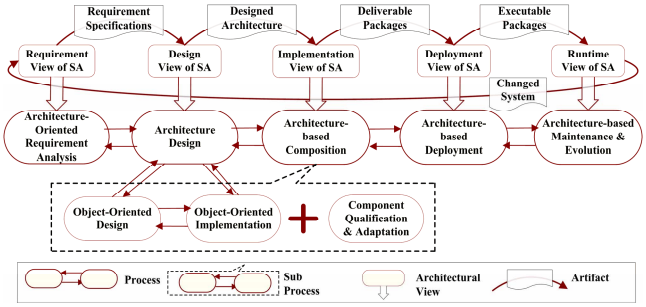

Figure 11. Process model and architectural views of ABC

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 requirements 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. Being implemented and tested, 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 some sense, the development of a system in ABC can be considered as a series of refinement and transformation of architecture models. The syntax and semantics of SA would become more accurate or complete after every refinement or transformation. SA in maintenance and evolution has the most accurate and complete details of SA describing the final system. Since it represents the actual situation of the runtime system, the runtime view of SA has a significant different perspective on components, connectors and constraints from the design view.

There are many challenges to the ABC approach as well as many novel ideas, including the feature modeling for reusability, the software architecting for changeability, the architecture based deployment for high performance and dependability, the architecture based reflection for autonomic management and so on.

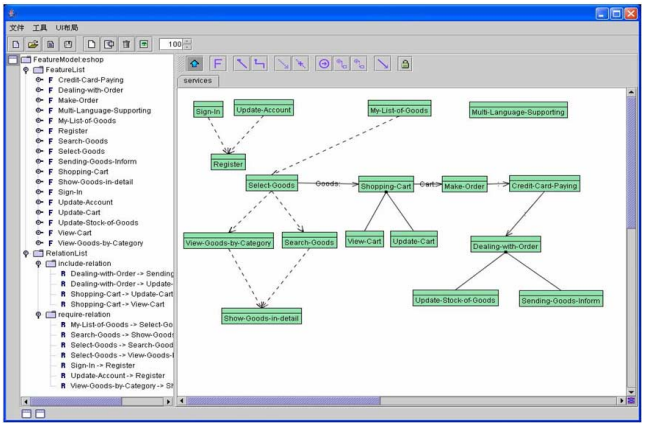

Figure 12. Feature modeling tool

As discussed above, new software systems can be implemented by reusing existing software in Internetware. Usually, reusability makes sense if and only if it is applied to an application domain or a product line. There are many domain oriented modeling methods, in which the feature oriented modeling is best-of-breed and selected by ABC [12] [21]. The feature-oriented approach to requirements modeling treats features as the basic entities in the problem space, and uses features and relations between features (namely, feature model) to specify the problem space, as shown in Figure 12. A feature describes a software characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements. Besides the nature of capturing commonality and variation, there are two important benefits to model requirements of component based systems in feature models. The first is to support software architecting at the requirements level. In the context of component-oriented programming, the design space is modularized by components and connectors. But there lacks a method to modularize the problem space. The second is to facilitate the reuse of SA. A carefully designed SA can be reused by a set of similar applications which share a set of common requirements. There needs an effective way to represent the requirements of the set of applications, rather than doing it repeatedly to each application.

Since feature model is very different to software architecture, there are two critical challenges, i.e. how to trace features to components and construct the software architecture based on the feature model. It should be noted that the software architecture here is abstract and acts as the architecture oriented requirement

specifications. The concept of responsibility is introduced to handle these two issues [22]. 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 is complex. One important reason is the complex n-to-n relations between features and components. By introducing responsibilities as the bridge, the n-to-n relations are decomposed into two sets of 1-to-n relations. One set contains the 1-to-n relations between features and responsibilities, indicating that a feature can be generally operationalized into several responsibilities. The other set contains the 1-to-n relations between components and responsibilities, showing that a component may be assigned several responsibilities. Based on the decomposition, tracing features to components can be done in a two-step way: first operationalizing features into responsibilities, then assigning responsibilities to components. As to the problem of the software architecture's construction, we decompose it into two sub-problems, namely, component construction and interaction identification. Based on the 1-to-n relations between features/components and responsibilities, components can be constructed by clustering responsibilities operationalized from features. The problem of interaction identification is resolved by analyzing interactions between responsibilities when operationalizing features, and using them as the source of interactions between components.

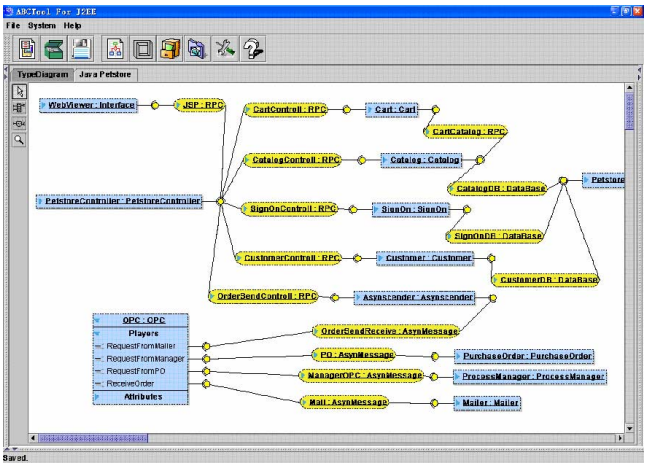

Figure 13. Software architecting tool

One of the most challenging issues of Internetware is how to survive in the rapid and continuous changes. Currently, there are plentiful research and practice on the adaptability. Particularly, adaptable or reflective middleware provides powerful and practical mechanisms for monitoring and changing runtime systems. However, these mechanisms only handle "how to do" other than "why, when and what to do" the adaptation. Then the adaptation for changes is still difficult, error-prone and time-consuming. On the other hand, developers can predict some changes and plan corresponding adaptations before the target system runs. Unfortunately, developers, especially designers, do not take the runtime adaptable mechanisms into account when developing the target system. For enhancing changeability of Internetware, ABC supports the modeling of self-adaptive software architecture (SASA), as shown in Figure 13. In details, we synthesize some sophisticated methods in architecture based software engineering, that is, 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 cases studies, such as adaptation of workflow [24] and exception handling in software architecture [2], demonstrate the effectiveness and feasibility of the idea.

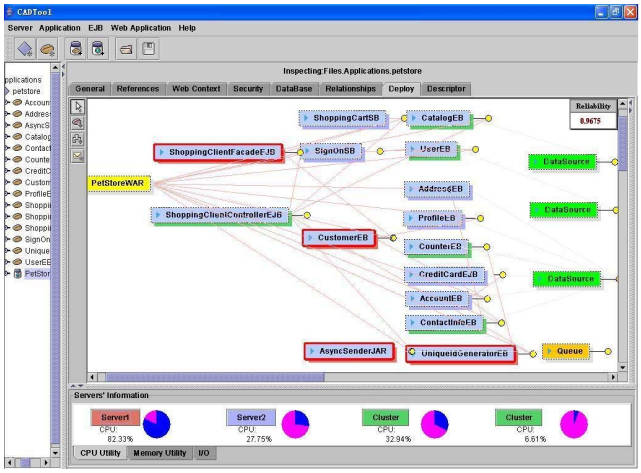

Figure 14. Architecture based deploy tool

When deploying a software system into a target environment, some challenging issues have to be handled, such as the configuration of the system to be deployed should match the configuration of the target environment, the new system should not put any bad impact on existing systems in the same environment and the resource requirements of the new system have to be satisfied. Otherwise, not only the new system but also existing systems cannot provide desired functions and qualities. These issues become much more challenging due to the extremely open and dynamic nature of Internetware. In ABC, software architecture is introduced into the deployment for facilitating and automating the analysis, reasoning and decision making [5]. As shown in Figure 14, the software architecture of the system to be deployed can be visualized, which help 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 all configuration information can be derived from the software architecture description, it is not necessary for deployers to write the configuration by hand any more.

Different to traditional software systems, Internetware has to be maintained and evolved frequently at runtime for keeping 7(days) x 24(hours) availability in the rapid and continuous changes of user requirements and operating environments. Since one of the most critical issues in software maintenance and evolution is the complexity of understanding the target system, software architecture is considered as a promising way. However, existing architecture based maintenance and evolution methods are usually based on source code and other documentation and then cannot capture the actual and up-to-date structure, states and behaviors of the target system and change the runtime system at once. In ABC, runtime software architecture is proposed to solve the problem of architecture based maintenance and evolution. Runtime software architecture is a model that represents a runtime system as a set of

architectural elements which are causally connected with the internal states and behaviors of the runtime system [4]. The causal connection is implemented as reflection so that changes to the runtime software architecture immediately cause corresponding changes to the runtime system, as shown in Figure 15. 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 [15]. 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 [5], the automatic coordinated recovery of correlated faults of middleware services [7], and so on.

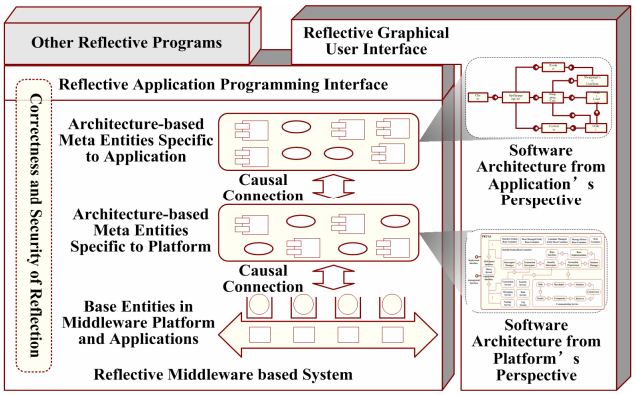

Figure 15. Architecture based reflective J2EE server

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 concluded, which then technically determined 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 for software engineering. This paper firstly reviews the history of software engineering in the world and in China especially. Based on these reviews, a new paradigm is proposed for the future development of software engineering. In this paradigm, software engineering is viewed as an independent discipline along with computer science and its development relies on co-operative efforts from academia, governments and industries.

The authors would like to thank our research group for their inspiration and support in the past two decades. Thanks Gang Huang, Lu Zhang and Donggang Cao in the preparation of the materials. Thanks to ShingChi Cheung and Robert Lai for their comments. This work was supported by several government-funded projects, including the National Key Projects in the State 6th to 10th Five-Year Plan, the State 863 High-Tech Program, the National Basic Research Program (973), the National Natural Science Foundation of China, the research program from Ministry of Education, etc.

5. REFERENCES

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.

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

[2] Feng, Y., Gang Huang, Yali Zhu, Hong Mei. Exception Handling in Component Composition with the Support of Middleware. Fifth International Workshop on Software Engineering and Middleware (SEM 2005), co-located with ESEC-FSE'05, Lisbon, Portugal, September 5-6, 2005, ACM Press, pp.90-97.

[3] Foster, I., C. Kesselman and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 2001, 15 (3), pp. 200-222.

[4] Huang, G., M. Hong, F.Q. Yang. Runtime Recovery and Manipulation of Software Architecture of Component-based Systems. International Journal of Automated Software Engineering, to appear in Vol. 13 No. 2, 2006, pp. 257-281.

[5] Huang, G., Tiancheng Liu, Hong Mei, Zizhan Zheng, Zhao Liu, Gang Fan. Towards Autonomic Computing Middleware via Reflection. In Proceedings of 28th Annual International Computer Software and Applications Conference (COMPSAC), Hongkong, China, September 28-30, 2004, pp.122-127.

[6] Lan, L., Gang HUANG, Liya MA, Meng WANG, Hong MEI, Long ZHANG, Ying CHEN. Architecture based Deployment of Large-Scale Component based Systems: the Tool and Principles. 8th International SIGSOFT Symposium on Component-based Software Engineering (CBSE), 2005, LNCS 3489, Springer, pp. 123-138.

[7] Liu, T., HUANG Gang, FAN Gang, MEI Hong, The Coordinated Recovery of Data Service and Transaction Service in J2EE, In Proceedings of 29th Annual International Computer Software and Applications Conference (COMPSAC05), Edinburgh, Scotland, July 2005, pp. 485-490.

[8] Mei, H. and G. Huang. PKUAS: An Architecture-based Reflective Component Operating Platform, 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2004, Suzhou, China. pp 163-169.

[9] Mei, H., A Complementary Approach to Requirements Engineering: Software Architecture Orientation, ACM SIGSOFT Software Engineering Notes, 2000, 25(2): 40-45.

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

[11] Mei, H., Feng Chen, Qianxiang Wang and Yaodong Feng. ABC/ADL: An ADL Supporting Component Composition. In proceedings of 4th International Conference on Formal Engineering Methods (ICFEM2002), 2002, pp. 38-47.

[12] Mei, H., W. Zhang, F. Gu. A feature oriented approach to modeling and reusing requirements of software product lines. In Proceedings of COMPSAC 2003, pp. 250-256.

[13] Papazoglou, M. P., D. Georgakopoulos. Editors. Special Issue on Service Oriented Computing. Communications of ACM. Oct. 2003, Vol. 46, No. 10, 24-60.

[14] Satyanarayanan, M. Pervasive computing: vision and challenges. IEEE Personal Communications, Aug 2001, Volume: 8, Issue: 4, 10-17.

[15] Shen, J., Xi Sun, Gang Huang, Wenpin Jiao, Yanchun Sun, and Hong Mei, 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.

[16] Tang, C.-S. Toward a Unified Logic Basis for Programming Languages. IFIP Congress 1983, pp.425-429.

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

[18] Yang, F., et al. Kernel Software Engineering Environment BETA-85, Science in China (A), Vol.19, No.7, 1989.

[19] Yang, F., Shao, W., Mei, H., The Design and Implementation of Object-Oriented CASE Environment Jade Bird 2(JB2), Science in China (A), Vol.25, No.5, 1995, 533-542. (in Chinese)

[20] Yang, F., Mei, H., Research on Technology for Industrialization Production of Software---Practice of JB (Jade Bird) Project, Symposium of Sino-American Engineering Technology, Oct. 1997, Beijing, 190-200.

[21] Zhang, W., Hong Mei, Haiyan Zhao, A Feature-Oriented Approach to Modeling Requirements Dependencies, in Proceedings of 13th IEEE International Requirements Engineering Conference (ICRE), pp,273-282, La Sorbonne,, France, August 29-September 2, 2005.

[22] Zhang, W., Hong Mei, Haiyan Zhao, Jie Yang, Transformation from CIM to PIM: Feature-Oriented Component-Based Approach, 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2005), Montego Bay, Jamaica, October 2-7, 2005, Proceedings. LNCS 3713: 248-263

[23] Zhu, Y., Gang HUANG, Hong MEI. Modeling Diverse and Complex Interactions Enabled by Middleware as Connectors in Software Architectures. 10th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS2005), 2005, pp. 37-46.

[24] Zhu, Y., Gang Huang, Hong Mei, Quality Attribute Scenario Based Architectural Modeling for Self-Adaptation Supported by Architecture-based Reflective Middleware, Asia Pacific Software Engineering Conference (APSEC 2004), Busan, Korea, November 30 C December 3, 2004, pp. 2-9.