RC24464 (C0801-010) January 11, 2008
Computer Science

IBM Research Report

Case Study: CFI-enabled Application Development Leveraging Community Resource

Zhou Xin, Liu Ying, Su Hui, Zhang Xin
IBM Research Division
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, 100094
P.R.China

Case Study: CFI-enabled Application Development Leveraging Community Resource

Zhou Xin, Liu Ying, Su Hui, Zhang Xin

周欣, 刘英, 苏辉, 张欣

IBM China Research Lab,, Beijing 100094, China, {zhouxin, aliceliu, suhui, xinzh}@cn.ibm.com

Abstract: The abundant development resource in open community is promising to be a flexible and effective approach for relieving the enterprise's IT resource shortage. CFI (Call-For-Implementation) development method is proposed to facilitate leveraging open community resources in commercial application development by addressing the specific challenges: how to divide the whole implementation task into many smaller pieces so that each community developer can conveniently work on one piece and how to prevent the community developer from learning key knowledge embedded in the to-be implemented application. This paper reports our study on applying the CFI development method to a SOA application. The hypothesis to be validated in the study, the experiment approach, the data collected during the study and the result analysis are presented. Also problems to-be further explored are discussed.

Key words: Open community, Call for implementation, SOA, Knowledge protection, Work dispatch

CLC number：TP391 **Document code: A** **Article ID :**

1 Introduction

IT applications are playing increasingly critical role in supporting an enterprise's daily business. With the rapid growth of business, more and more new IT applications are required to be ready in very tight time schedule, which demands increasingly larger IT development team. However, the development resource in an enterprise cannot always grow accordingly due to budget and/or management limitation. Such status entails an approach to help smoothing out the development resource issue.

Numerous open-source software development practices reveal that there are abundant development resources in open community and they are able to delivery software varying from small utilities (like log4j[1], dom4j[2]) to large scale products (like Apache[3], Eclipse[4]). Leveraging open community resources, including campus students, programming fans, software amateurs, etc., seems to be promising for relieving current enterprises' resource pressure. Actually, some commercial companies have experienced this by involving in and incubating some open-source software projects and sharing the copyrights [4][5]. However, little case can be found that open community resource is leveraged to develop commercial applications. One major reason is that companies worry that their critical knowledge is prone to be exposed to open community resource. Also, the lack of effective work dispatching approach makes it very hard to use a large number of open community resources in parallel for tight development schedule.

CFI (Call-For-Implementation) development method is proposed to facilitate leveraging open community resource in commercial application development by addressing the specific problems including work dispatch and knowledge protection. With this method, the work of developing an application can be partitioned into many smaller work tasks so that open community developers can work on the tasks independently and conveniently. Meanwhile, key knowledge embedded in the application is identified and analyzed to apply appropriate protection mechanisms (like reserving, pretending and/or separation). As a result, the developer can hardly learn the key knowledge from the assigned work task. The benefit of CFI is supposed to be two-fold. First, the development duration can be dramatically reduced by leveraging a large number of developers in parallel. Secondly, the key knowledge is at a lower risk of exposure. To validate the hypothesis, we conduct a case study on applying the CFI method to the development of a SOA application – HR (Human Resource) with community resource. The paper reports our case study process and result analysis.

The rest of the paper is organized as follows. Section 2 introduces what major problems CFI method addressed and how. Section 3 gives the background of the application developed in the case study. In section 4, the experiment process and the data collection approach are introduced. The results are presented and analyzed in section 5. Section 6 summarizes some related works, and section 7 concludes the paper and presents the future works.

2 CFI Method Overview

Figure 1 [6] outlines the major participants and activities involved in a CFI-guided application development project (we call it CFI project for brief in the remaining part of the paper).

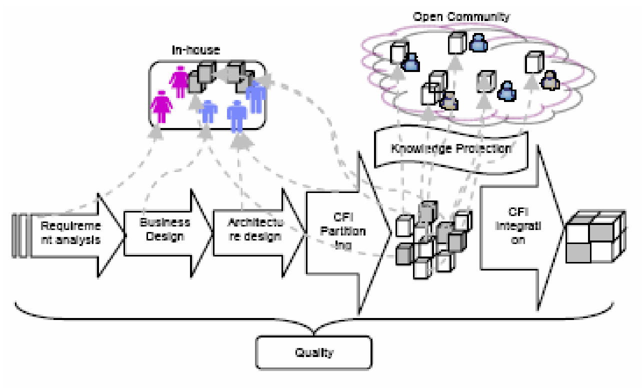


Figure 1: CFI Method Overview

The participants of a CFI project fall into two groups: one is in-house team and the other is community team. The in-house team includes project manager, requirement analyst, application designer, application integrator and so on. The in-house team members are usually the regular resource of an enterprise. They own the project's key knowledge and control the project. Besides the traditional development activities (requirement analysis, business design, architecture design, quality assurance, etc.), they should also take CFI specific activities like CFI partition, CFI integration.

The community team is a virtual team, including all the open community developers who work for this CFI project. They are not the owner of the project but are only hired to conduct the implementation work for lower cost and shorter development cycle. Keeping the community team away form the project's key knowledge during implementation is one of CFI's major concerns.

CFI partitioning technology is the first key to enable the CFI method. It partitions the design of an application into multiple smaller work tasks. Each work task will be specified and then distributed to an individual community developer for independently implementation without learning the whole picture of the project. The granularity of CFI work task can be very flexible according to community resources available, application characteristics and the knowledge protection requirement.

Knowledge protection technology is another key to enable the CFI method. It analyzes the characteristics of the to-be protected knowledge and how the knowledge is embedded in application design. Based on the analysis, individual knowledge protection mechanisms, like reserving, pretending, and separation, are selected, combined and applied to appropriate to-be distributed work tasks. With the proactive protection, key knowledge is at a low risk of exposure to open community developers.

As stated above, CFI method focuses on dealing with the challenges special for community-based commercial application development. It also leverages best practices from existing software development method, like RUP, XP, to form a holistic method. For instance, in order to reduce the quality risk due to the implementation work partitioning and the distribution to the community, CFI method adopts and enhances the quality assurance process of RUP [7].

3 HR Application Overview

The application developed in our experiment is a SOA system that realizes simplified payroll management and recruiting function. We call it HR (Human Resource) application for convenience in the paper. HR application includes 11 major functions, including user management, organization management, payroll template management, payroll structure management, payroll process, payroll reporting, vacancy management, application management, interview management, notification template management, and recruiting reporting.

The representation layer of this application consists of 109 JSP[8] pages and 181 Struts[9] actions. The business logic layer consists of 11 SCA (Service Component Architecture)[10] components. The data access layer consists of 26 SCA components. The data persistence layer consists of 26 tables. The representation, business logic and data access layers are all implemented with CFI approach, and the size of outcome source code is 28271 SLOC.

During the CFI development, Websphere Integration Developer 6.0.2 is used as the development tool and the application is finally deployed on Websphere Application Server 6.0.

4 Experiment Approach

We take the empirical study approach to investigate how CFI method can help community-based commercial application development. First of all, we define a set of metrics that link to our hypothesis about the CFI method. The metrics will guide data collection and analysis in the study. We define the detailed process for the study based on CFI method, and form an in-house team that includes 7 IBM employees and a community team with 43 college students. The students have average Java application development skill and have been trained to use technologies required by the HR application implementation. Also, the data collection approach is designed and data collection supporting tools are built.

4.1 Hypotheses and Metrics

The CFI method is intended to guide the community-based commercial application development. Applying this method, we expect the implementation duration will be greatly shortened as many community developers can implement in parallel and the key knowledge can be proactively protected from being learnt by community developers. Meanwhile, we don't expect that the overall application development effort is increased as extra effort is brought by CFI method (like CFI partition, CFI specifying, etc). For above hypotheses validation, we will observe the whole study process to record some facts and collect necessary data, and comparing the collected data with the reference data in industry or standard.

In order to effectively collect data from the study for the hypotheses validation purpose, we follow a widely adopted goal oriented measurement method - GQM (Goal-Question-Metric) [11] to design metrics, which are presented in table 1, 2, 3 and 4 by category.

Table 1: Product Size Metrics

Object	Metric Definition
Requirement	Number of use case
Design	Number of UI page
	Number of Struts Action
	Number of SCA component
	Number of SCA interface
	Number of DB table
Implementation	Line of JSP code
	Line of Struts Code
	Line of SCA code

Table 2: Process Metrics

Object	Metric Definition
Requirement Analysis	Requirement Analyst Number
	Requirement Analysis Duration
	Requirement Analysis Effort
Design	Designer Number
	Design Duration
	Design Effort
CFI Preparation (knowledge protection, partition, specification)	CFI Preparation Person Number
	CFI Preparation Duration
	CFI Preparation Effort
CFI Implementation	CFI Implementer Number
	CFI Implementation Duration
	CFI Implementation Effort
CFI Integration	CFI Integrator Number
	CFI Integration Duration
	CFI Integration Effort
Project Management	Project Manager Number
	Project Management Duration
	Project Management Effort

Table 3: Knowledge Protection Metrics

Object	Metric Definition
Community Developer	Exposed Knowledge Percentage Without Protection
	Exposed Knowledge Percentage With Protection
	Actually Learned Knowledge Percentage

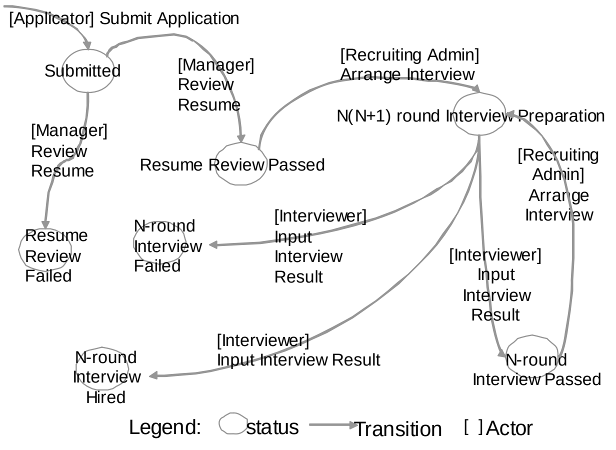
4.2 Experiment Process

** The CFI method Application Design*

In the experiment, only in-house team members involve in the application design activity. Firstly, the implementation technologies are determined: JSP and struts are selected to implement the application's representation layer, SCA is selected to implement the business logic layer and data access layer, DB2 is selected to implement the data persistence layer. Based on the technology decision, the design is performed in following sequence: UI page design first, then database table design first, then data access component/interface design, then business logic component/interface design, and finally action design. The application design is actually the same as that in traditional non-community-based development.

** Knowledge Protection*

In the HR application, we regard the position application process (as shown in Figure 2) in recruiting function as the key knowledge to-be protected.



[Applicator] Submit Application
Submitted
[Manager] Review Resume
[Recruiting Admin] Arrange Interview
N(N+1) round Interview Preparation
[Manager] Review Resume
Resume Review Passed
[Recruiting Admin] Arrange Interview
Resume Review Failed
N-round Interview Failed
[Interviewer] Input Interview Result
[Interviewer] Input Interview Result
N-round Interview Hired
[Interviewer] Input Interview Result
N-round Interview Passed
Legend:
status
Transition
[ ] Actor

Figure 2: Position Application Process

To quantitate the knowledge of this process for convenient knowledge protection and evaluation, we set the weight of each status as 1 and set the weight of each transition as 2. As there are 7 statuses and 8 transitions in the position application process totally, the quantitative knowledge of the process is 23 (i.e. 7*1 + 8*2 = 23).

Then we analyze each design element for the key knowledge implied by it. For instance, JSP page "NewInterviewArrangement" implies two position application statuses: "N-round Interview Passed", "Resume

Review Passed", and two transitions: "N-round Interview Passed -> N(N+1)-round Interview Preparation" and "Resume Review Passed -> N(N+1)-round Interview Preparation". So the quantitative knowledge for this design element is 6 (i.e. 2*1 + 2*2 = 6). Then the knowledge percentage implied by the this JSP page is 6/23 = 26%

After all design elements are assigned a quantitative knowledge, we set the knowledge value as a property of each design element and it will be used in later CFI partition phase. In that phase, we will make the design elements embedding key knowledge distributed to as many work task as much so that each work task will not reveal too much key knowledge.

** CFI partition and specification*

In CFI partition phase, in-house team divides the task for implementing all the pages, actions, business logic components and data access components into multiple smaller work tasks, so that each community member can take one work task and complete it. Our partition criteria include balanced key knowledge exposure of each task, balanced work load of each task and loose coupling between any work tasks. We have developed a tool to facilitate the CFI partition, which is implemented a RSA plugin. To perform the partition, in-house team firstly describes all the design elements and their inter-relationship with a UML class diagram. In the diagram, each design element is denoted as a class with according stereotype from CFI profile. For instance, a jsp page is denoted as a class with stereotype <<JSP Page>>, and a business component "InterviewManagement" is denoted as a class with stereotype <<SCA Component>>. Then, in-house team set the partition granularity and the partition tool automatically partitions the whole work into 43 work tasks. The CFI work tasks are distributed to the community group for implementation. In average, each task includes 2.3 JSP pages, 4 actions and 1.7 SCA interfaces. The average task effort is 36.6 person hours.

The partition tool also generates a preliminary specification for each task based on the design information. For instance, the implementation environment required for performing the task, the name and parameters of the operations to be implemented, are generated by the tool by default. Besides, in-house team supplements more contents to the specification, including UI style, code convention, unit test demand, etc.

** CFI Implementation*

After the CFI partition is complete and the specification for each work task is ready. Community group members prepare the required implementation environment, import required common libraries, and start their work strictly following the given specification. During the implementation, community members can communicate with in-house team for further task clarification and necessary technical support. However, we prohibit the communication among community members for knowledge protection purpose.

Unlike traditional distributed implementation, none of the community group members can access the whole design and implementation. They are only given the necessary interfaces implemented by other instead of the code details, which for sure brings challenges to the individual's debugging and unit testing. Fortunately, we leverage a technology called "surrogate" [12] to smooth the issue.

** CFI Integration*

The integration work is performed by in-house team based on the HR application design. Then, the integration testing is performed. Once a defect is found during the testing, the in-house team will investigate the reason and fix it with necessary help from according community implementer.

4.3 Data Collection Approach

In order to collect all kinds of data during the case study for later analysis, the activities we perform include:

- Establishing rules for in-house team and community team to make sure that size of the non-code work products (requirement, design…) is recorded and reported by the generator.

- Establishing rules for in-house team and community team to make sure that everyone reports the actual efforts spent on each CFI activity.

- Developing a daily log tool to collect effort. With the tool, in-house team members and community team members can report their effort by disciplines (e.g. requirement, analysis and design, test, implementation, etc) and sub-categories (e.g. CFI partition, CFI specification, CFI implementation, etc).

- Designing a questionnaire to test community team member's understanding of protected key knowledge.

- Designing a questionnaire to get experienced project managers' estimation about HR application's development effort.

- Using Eclipse Metric plug-in to calculate the size of HR application source code.

5 Experiment Analysis

The measurement collected during the experiment according to our predefined metrics and the further analysis are presented as below:

5.1 Product Size

Product related size measurement is summarized in Table 4, which include requirement size, design size and implementation size.

Table 4: Product Size Measurement

Object	Metric Definition	Measurement
Requirement	Number of use case	62 (Unit)
Design	Number of UI page	109 (Unit)
	Number of Struts Action	181 (Unit)
	Number of SCA component	37 (Unit)
	Number of SCA interface	75 (Unit)
	Number of DB table	26 (Unit)
Implementation	Line of JSP code	8989
	Line of Struts code	6977
	Line of SCA code	12305

All the implementation codes are new written codes without reuse. We calculate the total size of implementation as sum of JSP code line plus, Struts code line and SCA code line, i.e. 28271 lines. To be mentioned here is that about 50% SCA code is automatically generated by Websphere Integration Developer 6.0.2, so only 6152 lines of SCA code is actually written manually by implementers.

5.2 Process Effort

Process related measurement collected in this experiment is summarized in Table 5, which includes the number of persons involved in each application development phase, the duration of each phase and effort spent on each phase.

Table 5: Process Measurement

Object	**Metric Definition**	**Measurement**
Requirement Analysis	Requirement Analyst Number	2 (Persons)
	Requirement Analysis Duration	2.5 (Weeks)
	Requirement Analysis Effort	244 (PHs *)
Design	Designer Number	5 (Persons)
	Design Duration	3 (Weeks)
	Design Effort	490 (PHs)
CFI Preparation	CFI Preparation Person Number	5 (Persons)
	CFI Preparation Duration	2 (Weeks)
	CFI Preparation Effort	390 (PHs)
CFI Implementation	CFI Implementer	43 (Persons)
	CFI Implementation Duration	1 (Week)
	CFI Implementation Effort	1575 (PHs)
CFI Integration	CFI Integrator Number	5 (Persons)
	CFI Integration Duration	3 (Weeks)
	CFI Integration Effort	417 (PHs)
Project Management	Project Manager	1 (Person)
	Project Management Duration	12 (Weeks)
	Project Management Effort	542 (PHs)

(* PH is PersonHour)

To analyze the experiment result, we perform a survey on 12 experienced project managers by sending them the HR application design documents and a questionnaire. The purpose is to get their estimation about the total development effort, to get their experience on the appropriate number of persons assigned to each development phase and the duration of each phase given the number of persons are involved.

** Total effort analysis*

Total effort spent on the whole HR application development equals to the sum of requirement analysis effort, design effort, CFI preparation effort, CFI implementation effort, CFI integration effort and Project Management effort, i.e. 3658 PHs.

Table 6: Experienced PM's Estimation about the Effort

Estimated Effort	**Number of PMs giving the estimation**
1680 (PHs)	1
2160 (PHs)	1
2400 (PHs)	4
3600 (PHs)	5
6000 (PHs)	1

The survey result on total HR application development effort estimation is shown in Table 6. It shows that the actual total effort 3658 PHs spent in the experiment is very close to 3600 PHs estimated by 5 of the 12 experienced PMs.

** Project management effort analysis*

The percentage of project management effort on total development effort is 542/3658 = 14.8%. Compared to the reference percentage 11% from RUP [7], this value is higher.

The survey result on the saturated implementers for HR application implementation is shown in Table 7.

Table 7: PM Estimated Saturated HR Application Implementers

Saturated Implementers	**Number of PMs giving the suggestion**
0-4	3
5	3
10	5
20	1

We presume coordinating and communicating with the community group much larger than a typical implementation group causes the higher project management effort.

** Productivity analysis*

ISBSG (International Software Benchmarking Standards Group) investigates 196 java projects developed from 2000 to 2005 and gets an average productivity data: 12.87 Function Points/100Hours. [13]

To compare with the benchmark, we calculate the productivity in this study as:

- The manually written source code size is 8989 + 6977 + 6152 = 22118 lines.

- According to the empirical data "1 FP (Function Point) equals to 46 lines of java code" [14], the function points of HR application is 22118 / 46 = 481 FPs.

- So the productivity of the HR application case is 481 / 3658 *100 = 13.1 Function Points/100Hours.

The result indicates that productivity of the HR application case is very close to the benchmark.

5.3 Knowledge Protection Measurement

Knowledge protection measurement collected in the experiment is presented in a column diagram as shown in Figure 3. The x axis represents 43 community developers and y axis represents the knowledge percentage. Black column represents the knowledge exposure percentage to each community developer after using proactive knowledge protection mechanism. Grey column represents the actual learned knowledge percentage by each community developer after the CFI implementation. White column represents the knowledge exposure percentage to each community developer supposing we don't use any proactive knowledge protection mechanism, which we assume should be 100%.

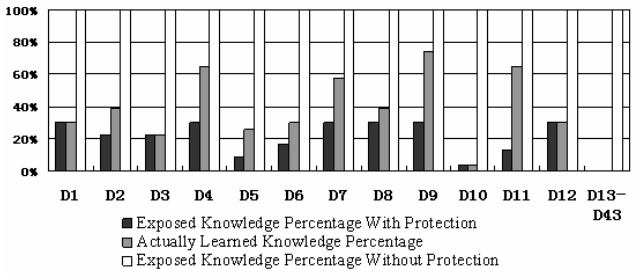


Figure 3: Knowledge Protection Measurement

In figure 3, we can find that 12 of 43 community developers are assigned work tasks containing the to-be protected key knowledge, and other 31 developers are not. Among the 12 developers, 4 of them finally learn exactly the same knowledge as we let them know, and 8 of them actually learn more than our expectation. But none of the developers learn the whole protected process, which is supposed to be 100% exposed to every community developers without protection. We also interview the 8 developers and find the major reason for their extra understanding of the protected knowledge - association of ideas. As the recruiting process to-be protected in our experiment is not very complex and complies with a person's common understanding about a general recruiting process, it's possible to make successful guess about more statuses and transitions based on the given ones.

5.4 Discussion

Based on the experience gained during the case study and the analysis on the experiment result, we come to following understandings:

- The HR application is developed in our experiment with a new development mode that a large number of community developers are leveraged for implementation.

- CFI development method and the supporting tool are successfully applied from end to end in this experiment to guide the very challenging activities: work dispatch, knowledge protection.

- In this case, CFI can reduce the implementation duration without sacrificing the overall development effort.

- In this case, CFI can guide protecting the key knowledge although it's still not as good as expected.

Through the experiment, we also acquire other interesting findings worth further investigating:

- As so many community developers contributes code to a single application, their diversity on coding skill, coding experience, coding custom makes the application harder to understand and maintain by others.

- It's not an easy task to effectively and efficiently communicate with such a large community group for prompt CFI development support.

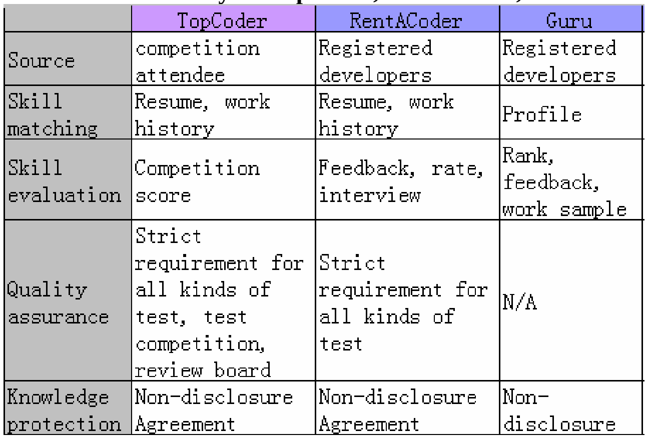
- The community developers are found and trained in advance. So it's relatively easy to guarantee their skill and quality. But a more real situation is that enterprises need to find matched resource from community in a short time. How to effectively select community resource and how to handle the mismatch are still questions.

6 Related Works

TopCoder[15], RentACoder[16] and Guru[17] are three mainstream platforms that support leveraging community resource for software development. We summarize their characteristics in table 8.

All of them provide a community resource pool, and their task-resource matching approaches, like work history, feedback, rank…, can be leveraged by CFI method. For work dispatch, we don't find any hints indicating that they can help partitioning a large work into smaller pieces for many developers to develop in parallel. As to the knowledge protection, they all require community developers to sign NDA (Non Disclosure Agreement). But this solution cannot absolutely stop violation since it's hard to forcing people to keep secret once they have learnt something. And when violation happens, enterprises need to spend significant time and cost on the lawsuit, which damages their normal business more or less. So the proactive protection technology presented by CFI provides a good supplement.

Table 8: Summary on TopCoder, RentACoder, Guru



	TopCoder	RentACoder	Guru
Source	competition attendee	Registered developers	Registered developers
Skill matching	Resume, work history	Resume, work history	Profile
Skill evaluation	Competition score	Feedback, rate, interview	Rank, feedback, work sample
Quality assurance	Strict requirement for all kinds of test, test competition, review board	Strict requirement for all kinds of test	N/A
Knowledge protection	Non-disclosure Agreement	Non-disclosure Agreement	Non-disclosure

7 Conclusion and Future Work

This paper presents an experimental study for CFI-enabled application development leveraging community resource. The experiment approach is expatiated and the experiment result is analyzed. The result validates our hypotheses:

- It's possible to make a large volume community developers implement a single application in parallel.

- The application implementation duration can be greatly shortened without sacrifice the overall development effort.

- The key knowledge can be proactively protected from being learnt by community developers.

- Presently, it's still imprudent to generalize the promising findings to other cases although it might be quite expectable.

In the future, we are going to experiment the new development mode and CFI method in more kinds of applications. Meanwhile, we will further refine the CFI method and supporting tool for community developer's diversity management, communication facilitation and task-resource matching.

Acknowledgements

We would like to thank Chen Hao, Niu Xuesong, Ye Hong, Xiong Yueda, Guo Anhui, Liu Hehui for their help on the experiment. We thank the project managers for participating in our survey. We also thank all the in-house team members and community team members.

References:

[1] http://logging.apache.org/log4j/1.2/index.html
[2] http://www.dom4j.org/
[3] http://www.apache.org/
[4] http://www.eclipse.org/
[5] http://www.mozilla.org/
[6] Liu Ying, Feng Chenhua, Zhao Wei, Su Hui, Liu Hehui, A Case Study on Community-enabled SOA Application Development, IEEE International Conference on Service-Oriented Computing and Application.
[7] Kruchten, P., Rational Unified Process – An Introduction, Addison-Wesley, 1999
[8] http://java.sun.com/products/jsp/
[9] http://struts.apache.org/
[10] http://www.ibm.com/developerworks/cn/webservices/ws-sca/
[11] Basili, G. Caldiera, and HD Rombach, The Goal Question Metric Approach, Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, 1994
[12] Zhong Jie Li, Jun Zhu, He Yuan Huang, Thomas Li, A simulation apparatus in service-oriented development, patent file number: 200710091775.4
[13] http://www.isbsg.org
[14] T. Capers Jones, Estimate Software Cost, McGraw-Hill Osborne Media, 2007
[15] http://www.topcoder.com
[16] http://www.rentacoder.com
[17] http://www.guru.com