Automatic Schedule Control for Distributed Software Development in Cloud Computing Environments

Chung Yung, Shao-Zong Chen, and Jen-Tsung Hsieh
Department of Computer Science and Information Engineering
National Dong Hwa University, Hualien, TAIWAN
Email: {yung@mail, m9721505@ems, 610021068@ems}.ndhu.edu.tw

***Abstract*—This paper proposes an extension of automatic schedule control to the WebSD management model of distributed software development in cloud computing environments. Cloud computing environments provide more flexibility than conventional computing environments. In particular, platform as a service (PaaS) provides more flexibility in application design, development testing, deployment, hosting, team collaboration, web service and database integration, scalability, and versioning. The WebSD model is a new management model of distributed software development for cloud computing environments. However, WebSD does not include functionality for schedule control. We design new operations and add into the WebSD model such that the distributed software development may be managed with automatic assistance in schedule control. We call the extended model as SDot. Inherited from WebSD, SDot also offers a common platform for various roles involved in the distributed software development, and SDot is also appropriate for the management of distributed software development in cloud computing environments. We present the application of SDot to a practical software project as a case study to show the effectiveness of SDot in schedule control for the management of distributed software development.**

***Index Terms*—schedule control, distributed software development, software development management, cloud computing environment.**

I. Introduction

Overall speaking, the objective of software engineering is to guarantee the delivery of high-quality software on time and within budget [1]. In the past decade, developing software systems with globally distributed teams is popularly applied to a lot software projects [2]. With the development of software technology and the rapid extension of application areas, the cost and schedule of distributed software development may get out of control easily if the projects are not managed with intensive care [3].

Based on 54 works of distributed software development published from 1998 to 2009, da Silva et al. concluded that the strong evidence about the effect of using the best practices, models, and tools in distributed software development projects is still scarce in the literature [4]. The ultimate goal of distributed software development is fully using all the resources, including computing devices and human resources, to achieve flexibility, quality and cost down. On the other hand, there exist several challenges in globally distributed software development, such as formalization in communication, formal change management, planning for system integration, project monitoring across distributed teams, standard distributed development tools, and integrated management tools [5].

Cloud computing is not only a term that refers to data, processing, or experiences that reside somewhere in the cloud that we call as the internet. Nowadays, cloud computing reforms the way how companies operate with data and applications in the processes of inventing, developing, deploying, scaling, updating, maintaining and paying for resources that undergo the changes [6]-[8].

A cloud computing environment can be defined as a computing environment that provides everything as a service, including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). The software systems developed in cloud computing environments suffer from the same problems that plague the conventional distributed and parallel software systems; they are complex to design, develop, test, deploy, and manage [9]. While various cloud services are either available or under development, the industry calls for a new model of distributed software development management that is specialized for cloud computing environments [6], [10].

With such a background, Yung et al. propose WebSD, which is a new management model of distributed software development management for cloud computing environments [6]. WebSD contributes in the following aspects.

- WebSD simplifies the conventional hierarchical architecture of distributed software development,
- WebSD extends conventional models to allow outsourcing parts of the software development to fellow companies, and
- WebSD provides various views to the distributed software development for the roles involved,

Manuscript received June 10, 2013; revised August 27, 2013.

doi: 10.12720/jiii.2.1.39-44

including project managers, software developers, software testers, and software debuggers, to cooperate in a common and open model.

However, WebSD does not include any specialized functionality for schedule control, which motivates our work presented in this paper.

We design new operations and add into the WebSD model such that the distributed software development may be managed with automatic assistance in schedule control. We call the extended model as *SDot*. Inherited from WebSD, SDot also offers a common platform for various roles involved in the distributed software development, and SDot is also appropriate for the management of distributed software development in cloud computing environments. We present the application of SDot to a practical software project as a case study to show the effectiveness of SDot in schedule control for the management of distributed software development.

This paper is organized as follows. The next section briefly describes the WebSD model. Our new SDot model is presented in section 3. Application of SDot to a practical case is described in section 4. And at last is a brief conclusion.

II. The WebSD Model

In this section, we briefly describe the WebSD model, which is a web-based management model of distributed software development for cloud computing environments [6].

In the WebSD model, a software project of distributed development is described as a well-designed set of modules, which are units of encapsulation. The life-cycle for developing a module in a software system is defined by the state transition diagram shown in Fig. 1.

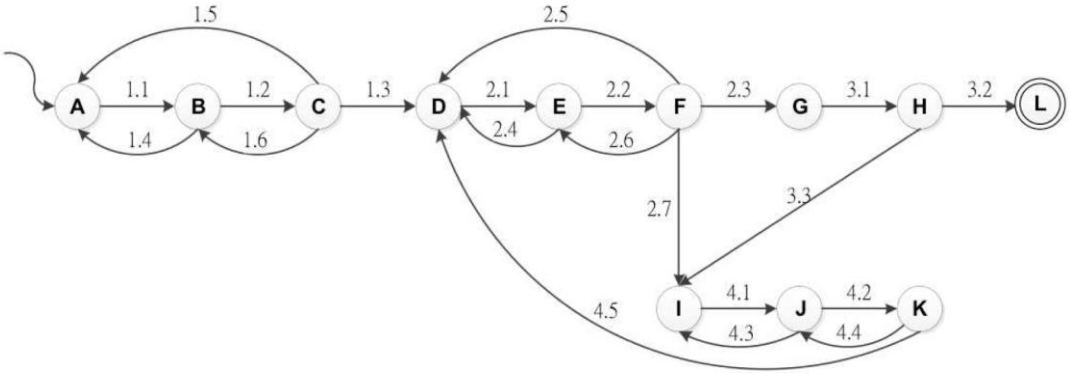


Figure 1. The state transition diagram of the WebSD model

As an example, the state of a module is initially *A*. After it is assigned to a group for programming, the state goes to *B*. Once the assigned group accepts the job of programming, the state goes to *C*. When the group reports the finish of programming, the state goes to *D*. Then, it is assigned for testing and the state goes to *E*. Once the assigned group accepts the job of testing, the state goes to *F*. When the group reports the finish of testing and no bug is found, the state goes to *G*. And then, the project manager performs the integration tests. If it passes the integration tests, the state goes to *L* and the development of the module is complete.

Extended from the definition of the life-cycle of developing a module, they define the status of a software project using distributed development as follows.

Definition (Status of a software project using distributed development, *P*)

Given a module set *M* of *k* modules and a software project consisting of the *k* modules, the status of the project *P* is defined as

$$P = \{p_i | 1 \le i \le k\},$$

where each p_i is a pair $\langle m_i, s_i \rangle$, $m_i \in M = \{m_1, ..., m_k\}$, and $s_i \in S = \{A, B, C, D, E, F, G, H, I, J, K, L\}$.

One of the advantages of the WebSD model is that with the definition of the status of a software project using distributed development, we may derive and keep record of the progress of distributed software development in an official manner.

III. Automatic Schedule Control

This section proposes an extension of automatic schedule control to the WebSD management model of distributed software development for cloud computing environments. We call the extended model as *SDot*.

The primary operations of distributed software development modeled in SDot are listed in Fig. 2. We briefly describe each operation as follows.

1.1. Assign_p: A project manager assigns the job of programming a module to a group.
1.2. Accept_p: A group accepts the job of programming a module.
1.3. Finish_p: A group finishes the job of programming a module.
1.4. Reject_p: A group rejects the job of programming a module.
1.5. $\mathsf{Withdraw}_p$: A group withdraws the acceptance of programming a module.
1.6. Expire_p: The job of programming a module gets expired.
1.7. Extend_p: A group applies for an extension in programming a module.
1.8. $\mathsf{Approve}_p$: A project manager approves the extension in programming a module.
2.1. Assign_u: A project manager assigns the job of testing a module to a group.

2.2. $Accept_u$: A group accepts the job of testing a module.
2.3. $Finish_u$: A group finishes the job of testing a module.
2.4. $Reject_u$: A group rejects the job of testing a module.
2.5. $Withdraw_u$: A group withdraws the acceptance of testing a module.
2.6. $Expire_u$: The job of testing a module gets expired.
2.7. $Extend_u$: A group applies for an extension in testing a module.
2.8. $Approve_u$: A project manager approves the extension in testing a module.
2.9. $Report_u$: A group reports bugs after testing a module.
3.1. $Assign_s$: A project manager assigns the job of integration testing for a module to a group.
3.2. $Accept_s$: A group accepts the job of integration testing for a module.
3.3. $Finish_s$: A group finishes the job of integration testing for a module.
3.4. $Reject_s$: A group rejects the job of integration testing for a module.
3.5. $Withdraw_s$: A group withdraws the acceptance of integration testing for a module.
3.6. $Expire_s$: The job of integration testing for a module gets expired.
3.7. $Extend_s$: A group applies for an extension in integration testing for a module.
3.8. $Approve_s$: A project manager approves the extension in integration testing for a module.
3.9. $Report_s$: A group reports bugs after integration testing for a module.
4.1. $Assign_d$: A project manager assigns the job of debugging a module to a group.
4.2. $Accept_d$: A group accepts the job of debugging a module.
4.3. $Finish_d$: A group finishes the job of debugging a module.
4.4. $Reject_d$: A group rejects the job of debugging a module.
4.5. $Withdraw_d$: A group withdraws the acceptance of debugging a module.
4.6. $Expire_d$: The job of debugging a module gets expired.
4.7. $Extend_d$: A group applies for an extension in debugging a module.
4.8. $Approve_d$: A project manager approves the extension in debugging a module.

PHASE		OPERATION	
P_1	Programming	1.1	$Assign_p$
		1.2	$Accept_p$
		1.3	$Finish_p$
		1.4	$Reject_p$
		1.5	$Withdraw_p$
		1.6	$Expire_p$
		1.7	$Extend_p$
		1.8	$Approve_p$
P_2	Unit Testing	2.1.	$Assign_u$
		2.2.	$Accept_u$
		2.3.	$Finish_u$
		2.4.	$Reject_u$
		2.5.	$Withdraw_u$
		2.6.	$Expire_u$
		2.7.	$Extend_u$
		2.8.	$Approve_u$
		2.9.	$Report_u$
P_3	Integration Testing	3.1.	$Assign_s$
		3.2.	$Accept_s$
		3.3.	$Finish_s$
		3.4.	$Reject_s$
		3.5.	$Withdraw_s$
		3.6.	$Expire_s$
		3.7.	$Extend_s$
		3.8.	$Approve_s$
		3.9.	$Report_s$
P_4	Debugging	4.1.	$Assign_d$
		4.2.	$Accept_d$
		4.3.	$Finish_d$
		4.4.	$Reject_d$
		4.5.	$Withdraw_d$
		4.6.	$Expire_d$
		4.7.	$Extend_d$
		4.8.	$Approve_d$

Figure 2. Primary operations in distributed software development

Extended from the WebSD model, the life-cycle of a module in the distributed software development is defined by a state transition diagram shown in Fig. 3.

In the design of SDot, we note the following:

- SDot is so flexible that it allows a module in the software be developed and tested by a group at different locations.
- In SDot, a group in the globally virtual team may get only the information involved with the group.
- In SDot, the project manager has the freedom of dynamic adjustment in distributing the job of programming, testing, or debugging a module to a group of his/her choice.
- SDot can be easily applied to the management of practical projects with distributed software development.

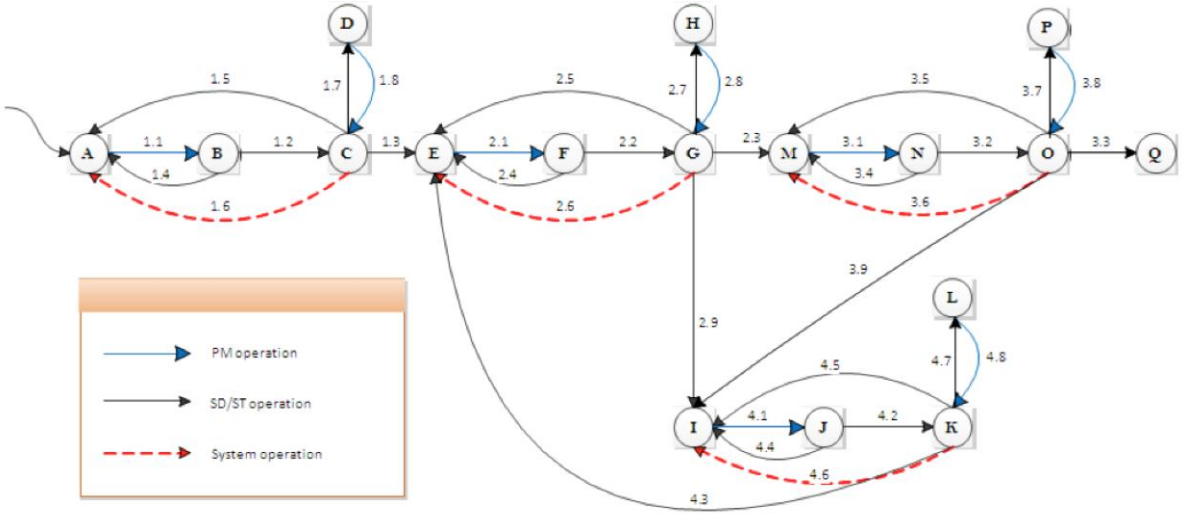


Figure 3. The state transition diagram of the SDot model

IV. Application of SDot to a Practical Project

For a validation, we apply the SDot model to a practical project called *ConsMan*. ConsMan is a project of distributed software development for building a web-based distributed information management system. The ConsMan project was executed between 2006 and 2007 for an energy and power company in Taiwan that our second author works with. Note that in this case, the SDot model is applied after the project is closed, based on the documentation kept during the execution of the project. While the whole details of ConsMan are available, simplification and adaption are applied for the purpose of clarity in presentation. In this section, we only show the top-level activities in executing ConsMan.

The ConsMan project with three top-level modules is developed by a virtual team consisting of five groups, of which one group is an in-house management group (g_1: TWPCMan), two groups are outsourced programming groups of agents and consultants (g_2: TWAC, and g_3: MLAC), and the other two groups are off-site testing groups (g_4: TWPCN, and g_5: TWPCS).

Here, we briefly describe the progress in developing the top-level modules as follows. ConsMan has a global virtual team consisted of 5 groups ($G = \{g_1, g_2, g_3, g_4, g_5\}$), of which g_1 is the project management group; g_2 and g_3 are the programming and debugging groups; g_4 and g_5 are the testing groups. The ConsMan project consists of 3 top-level modules ($M = \{m_1, m_2, m_3\}$), which are developed in a distributed way with the following steps:

1) On day d_1, g_1 performs $\text{Assign}_p(m_1, g_2, d_2)$, $\text{Assign}_p(m_2, g_3, d_2)$, and $\text{Assign}_p(m_3, g_3, d_2)$. g_2 performs $\text{Reject}_p(m_1)$. g_3 performs $\text{Accept}_p(m_2)$ and $\text{Accept}_p(m_3)$.
2) On day d_2, g_1 performs $\text{Assign}_p(m_1, g_2, d_3)$. g_2 performs $\text{Accept}_p(m_1)$. g_3 performs $\text{Withdraw}_p(m_3)$. The system automatically performs $\text{Expire}_p(m_2)$.
3) On day d_3, g_1 performs $\text{Assign}_p(m_2, g_2, d_4)$ and $\text{Assign}_p(m_3, g_3, d_4)$. g_2 performs $\text{Finish}_p(m_1)$ and $\text{Accept}_p(m_2)$. g_3 performs $\text{Accept}_p(m_3)$.
4) On day d_4, g_1 performs $\text{Assign}_u(m_1, g_5, d_5)$. g_3 performs $\text{Finish}_p(m_2)$ and $\text{Finish}_p(m_3)$. g_5 performs $\text{Accept}_u(m_1)$.
5) On day d_5, g_1 performs $\text{Assign}_u(m_2, g_5, d_6)$ and $\text{Assign}_u(m_3, g_5, d_6)$. g_5 performs $\text{Report}_u(m_1)$, $\text{Accept}_u(m_2)$, and $\text{Reject}_u(m_3)$.
6) On day d_6, g_1 performs $\text{Assign}_d(m_1, g_5, d_8)$ and $\text{Assign}_u(m_3, g_4, d_7)$. g_4 performs $\text{Reject}_d(m_1)$ and $\text{Accept}_u(m_3)$. g_5 performs $\text{Extend}_u(m_2, d_8)$.
7) On day d_7, g_1 performs $\text{Assign}_d(m_1, g_3, d_{10})$ and $\text{Approve}_u(m_2, d_8)$. g_3 performs $\text{Accept}_d(m_1)$. g_4 performs $\text{Withdraw}_u(m_2)$.
8) On day d_8, g_1 performs $\text{Assign}_u(m_3, g_5, d_9)$. g_5 performs $\text{Finish}_u(m_2)$ and $\text{Accept}_u(m_3)$.
9) On day d_9, g_5 performs $\text{Finish}_u(m_3)$.
10) On day d_{10}, g_3 performs $\text{Extend}_d(m_1, d_{12})$.
11) On day d_{11}, g_1 performs $\text{Approve}_d(m_1, d_{12})$.
12) On day d_{12}, g_3 performs $\text{Finish}_d(m_1)$.
13) On day d_{13}, g_1 performs $\text{Assign}_u(m_1, g_5, d_{14})$. g_5 performs $\text{Accept}_u(m_1)$.
14) On day d_{14}, g_5 performs $\text{Finish}_u(m_1)$.
15) On day d_{15}, g_1 performs $\text{Assign}_s(m_1, g_4, d_{22})$, $\text{Assign}_s(m_2, g_4, d_{22})$, and $\text{Assign}_s(m_3, g_4, d_{22})$. g_4 performs $\text{Accept}_s(m_1)$, $\text{Accept}_s(m_2)$, and $\text{Accept}_s(m_3)$.
16) On day d_{16}, g_4 performs $\text{Report}_s(m_1)$.
17) On day d_{17}, g_1 performs $\text{Assign}_d(m_1, g_3, d_{18})$. g_3 performs $\text{Accept}_d(m_1)$.
18) On day d_{18}, g_3 performs $\text{Finish}_d(m_1)$.
19) On day d_{19}, g_1 performs $\text{Assign}_u(m_1, g_5, d_{20})$. g_5 performs $\text{Accept}_u(m_1)$.
20) On day d_{20}, g_5 performs $\text{Finish}_u(m_1)$.
21) On day d_{21}, g_1 performs $\text{Assign}_s(m_1, g_4, d_{22})$. g_4 performs $\text{Accept}_s(m_1)$.
22) On day d_{22}, g_4 performs $\text{Finish}_s(m_1)$, $\text{Finish}_s(m_2)$, and $\text{Finish}_s(m_3)$.

As such, the ConsMan project is complete on day d_{22}.

The complete record of applying the SDot model to the top-level modules of ConsMan is shown in Fig. 4.

Module	m_1	m_2	m_3
States (d_1)	**A**	**A**	**A**
Operations	g_1: Assign$_p$(m_1, g_2, d_2) g_2: Reject$_p$(m_1)	g_1: Assign$_p$(m_2, g_2, d_2) g_3: Accept$_p$(m_2)	g_1: Assign$_p$(m_3, g_3, d_2) g_3: Accept$_p$(m_3)
States (d_2)	**A**	**C**	**C**
Operations	g_1: Assign$_p$(m_1, g_2, d_3) g_2: Accept$_p$(m_1)	S: Expire$_p$(m_2)	g_3: Withdraw$_p$(m_3)
States (d_3)	**C**	**A**	**A**
Operations	g_2: Finish$_p$(m_1)	g_1: Assign$_p$(m_2, g_2, d_4) g_2: Accept$_p$(m_2)	g_1: Assign$_p$(m_3, g_3, d_4) g_3: Accept$_p$(m_3)
States (d_4)	**E**	**C**	**C**
Operations	g_1: Assign$_u$(m_1, g_5, d_5) g_5: Accept$_u$(m_1)	g_3: Finish$_p$(m_2)	g_3: Finish$_p$(m_3)
States (d_5)	**G**	**E**	**E**
Operations	g_5: Report$_u$(m_1)	g_1: Assign$_u$(m_2, g_5, d_6) g_5: Accept$_u$(m_2)	g_1: Assign$_u$(m_3, g_5, d_6) g_5: Reject$_u$(m_3)
States (d_6)	**I**	**G**	**E**
Operations	g_1: Assign$_d$(m_1, g_2, d_7) g_4: Reject$_d$(m_1)	g_5: Extend$_u$(m_2, d_8)	g_1: Assign$_u$(m_3, g_4, d_7) g_4: Accept$_u$(m_3)
States (d_7)	**I**	**H**	**G**
Operations	g_1: Assign$_d$(m_1, g_3, d_{10}) g_3: Accept$_d$(m_1)	g_1: Approve$_u$(m_2, d_8)	g_4: Withdraw$_u$(m_3)
States (d_8)	**K**	**G**	**E**
Operations		g_5: Finish$_u$(m_2)	g_1: Assign$_u$(m_3, g_5, d_9) g_5: Accept$_u$(m_3)
States (d_9)	**K**	**M**	**G**
Operations			g_5: Finish$_u$(m_3)
States (d_{10})	**K**	**M**	**M**
Operations	g_3: Extend$_d$(m_1, d_{12})		
States (d_{11})	**L**	**M**	**M**
Operations	g_1: Approve$_d$(m_1, d_{12})		
States (d_{12})	**K**	**M**	**M**
Operations	g_3: Finish$_d$(m_1)		
States (d_{13})	**E**	**M**	**M**
Operations	g_1: Assign$_u$(m_1, g_5, d_{14}) g_5: Accept$_u$(m_1)		
States (d_{14})	**G**	**M**	**M**
Operations	g_5: Finish$_u$(m_1)		
States (d_{15})	**M**	**M**	**M**
Operations	g_1: Assign$_s$(m_1, g_4, d_{22}) g_4: Accept$_s$(m_1)	g_1: Assign$_s$(m_2, g_4, d_{22}) g_4: Accept$_s$(m_2)	g_1: Assign$_s$(m_3, g_4, d_{22}) g_4: Accept$_s$(m_3)
States (d_{16})	**O**	**O**	**O**
Operations	g_4: Report$_s$(m_1)		
States (d_{17})	**I**	**O**	**O**
Operations	g_1: Assign$_d$(m_1, g_3, d_{18}) g_3: Accept$_d$(m_1)		
States (d_{18})	**K**	**O**	**O**
Operations	g_3: Finish$_d$(m_1)		
States (d_{19})	**E**	**O**	**O**
Operations	g_1: Assign$_u$(m_1, g_5, d_{20}) g_5: Accept$_u$(m_1)		
States (d_{20})	**G**	**O**	**O**
Operations	g_5: Finish$_u$(m_1)		
States (d_{21})	**M**	**O**	**O**
Operations	g_1: Assign$_s$(m_1, g_4, d_{22}) g_4: Accept$_s$(m_1)		
States (d_{22})	**O**	**O**	**O**
Operations	g_4: Finish$_s$(m_1)	g_4: Finish$_s$(m_2)	g_4: Finish$_s$(m_3)
States (final)	**Q**	**Q**	**Q**

Figure 4. Applying the SDot model to keep record of the progress in the ConsMan project

V. Conclusion

In this paper, we present an extension of automatic schedule control to the WebSD management model of distributed software development for cloud computing environments. We call the extended model as SDot. We define the life-cycle of a module in the distributed software development using a state transition diagram. For validation, we include the application of the SDot model to the ConsMan project.

We are aware of that the SDot model only provides a simple and clear view to the software projects of distributed development for the project managers, with a special purpose of schedule control. For a sophisticated schedule control of software development, the managers are suggested to refer to various schedule control techniques for software project management [1], [3].

Acknowledgment

This work is supported in part by the National Science Council under grand number NSC 100-2218-E-259-002-MY3.

References

[1] X. Wang, C. Wu, and L. Ma, "Software project schedule variance prediction using bayesian network," in *Proc. IEEE International Conference on Advanced Management Science*, vol. 2, July 2010, pp. 26–30.

[2] A. Piri, "Challenges of globally distributed software development analysis of problems related to social processes and group relations," in *Proc. IEEE International Conference on Global Software Engineering*, August 2008, pp. 264–268.

[3] Y. Sun and R. Cui, "Workload point system based on project schedule optimization," in *Proc. International Conference on Management and Service Science*, September 2009, pp. 1–4.

[4] F. Q. B. da Silva, C. Costa, A. C. C. Franca, and R. Prikladinicki, "Challenges and solutions in distributed software development project management: a systematic literature review," in *Proc. 5th IEEE International Conference on Global Software Engineering*, August 2010, pp. 87–96.

[5] I. S. Wiese and E. H. M. Huzita, "IMART: An Interoperability model for artifacts of distributed software development environments," in *Proc. International Conference on Global Software Engineering*, October 2006, pp. 255–256.

[6] C. Yung, S.-Z. Chen, S.-C. Wu, J.-T. Hsieh, and K.-J. Peng, "A Web-based model of distributed software development management for cloud computing environments," *GSTF Journal of Computing*, vol. 2, no. 2, pp. 1–7, June 2012.

[7] V. Matveev, "Platform as a service–new opportunities for software development companies," Master's thesis, Department of Information Technology, Lappeenranta University of Technology, Finland, May 2010.

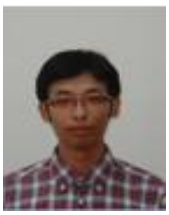
[8] Y. C. Zhou, X. P. Liu, X. N. Wang, L. Xue, X. X. Liang, and S. Liang, "Business process centric platform-as-a-service model and technologies for cloud enabled industry solutions," in *Proc. Third IEEE International Conference on Cloud Computing*, July 2010, pp. 534–537.

[9] J. S. Rellermeyer, M. Duller, and G. Alonso, "Engineering the cloud from software modules," in *Proc. ICSE Workshop on Software Engineering Challenges of Cloud Computing*, May 2009, pp. 32–37.

[10] K. A. Johnston and K. Rosin, "Global virtual teams: How to manage them," in *Proc. International Conference on Computer and Management*, May 2011, pp. 1–4.

Chung Yung received PhD degree in Computer Science from New York University (USA) in 1999 and BSc degree in Computer Science and Information Engineering from National Chiao Tung University (Taiwan) in 1988. He has been with the Department of Computer Science and Information Engineering of National Dong Hwa University (Taiwan) since 2000. He was a part-time senior consultant and project manager within the intelligent digital content industry between 2003 and 2007. He is currently leading Compiler Technology and Application Laboratory in National Dong Hwa University. His research interests include semantic methods of program analysis, optimizations for cloud software systems, compiler supported software engineering, and programming languages.

Shao-Zong Chen is currently with Taiwan Power Company as a senior software engineer since 2002. He is working part-time for his master degree in Computer Science and Information Engineering at Compiler and Technology Application Laboratory of National Dong Hwa University, Taiwan. He received BSc degree in Computer Science and Information Engineering from Tamkang University, Taiwan in 1992. His research interests include software project management, program analysis and distributed software development in cloud computing environments.



Jen-Tsung Hsieh is working for a master degree in Computer Science and Information Engineering in National Dong Hwa University (Taiwan). He received BSc degree in Computer Science and Information Engineering from University of Kang Ning (Taiwan) in 2007. His research interests include distributed software development and efficient implementation of programming languages.

