A Systematic Process for Developing High Quality SaaS Cloud Services*

Hyun Jung La and Soo Dong Kim

Department of Computer Science
Soongsil University
1-1 Sangdo-Dong, Dongjak-Ku, Seoul, Korea 156-743
hjla@otlab.ssu.ac.kr, sdkim777@gmail.com

Abstract. Software-as-a-Service (SaaS) is a type of cloud service which provides software functionality through Internet. Its benefits are well received in academia and industry. To fully utilize the benefits, there should be effective methodologies to support the development of SaaS services which provide high reusability and applicability. Conventional approaches such as object-oriented methods do not effectively support SaaS-specific engineering activities such as modeling common features, variability, and designing quality services. In this paper, we present a systematic process for developing high quality SaaS and highlight the essentiality of commonality and variability (C&V) modeling to maximize the reusability. We first define criteria for designing the process model and provide a theoretical foundation for SaaS; its meta-model and C&V model. We clarify the notion of commonality and variability in SaaS, and propose a SaaS development process which is accompanied with engineering instructions. Using the proposed process, SaaS services with high quality can be effectively developed.

Keywords: Cloud Computing, Software-as-a-Service, SaaS, Commonality and Variability Analysis, Development Process.

1 Introduction

Cloud Computing (CC) is emerged as an effective reuse paradigm, where hardware and software resources are delivered as a *service* through Internet [1]. Software-as-a-Service (SaaS) is a type of cloud services, where the whole software functionality is run on provider' side and becomes available to consumers [2][3].

SaaS provides several benefits to consumers; no cost for purchasing, free of maintenance, accessibility through Internet, and high availability. To realize these benefits, there should be systematic and effective processes and methods to support the development of SaaS services. Conventional methods including object-oriented modeling would be limited in developing services, mainly due to the difference between their computing paradigms. That is, conventional development methods do not effectively

* This research was supported by the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development and Experts Education.

M.G. Jaatun, G. Zhao, and C. Rong (Eds.): CloudCom 2009, LNCS 5931, pp. 278–289, 2009.

support CC-specific engineering activities such as modeling common features and designing services. Hence, there is a great demand for effective processes for developing SaaS cloud services.

In this paper, we present a process for developing high quality SaaS. We first define criteria for designing the process model in section 3. And, we present a theoretical foundation of our work in section 4; a meta-model of SaaS and a tailored view of commonality and variability. In section 5, based on the criteria and foundation, we propose a whole life-cycle process and its instructions to develop SaaS. Using the proposed process, cloud services with high quality can be more effectively developed.

2 Related Works

A little work is known in the area of developing SaaS. We survey two related works. Javier and his colleagues present overall SaaS development process by tailoring traditional software development methodologies [4]. They first analyze how SaaS impacts each phase in the software development methodologies. Based on the analysis results, they redefine five-stage SaaS development process and a list of development artifacts. *Requirements* stage focuses on deriving business opportunities from market requirements, *Analysis* stage is also performed from a business perspective, *Design* phase is performed by using a set of technologies such as service-oriented architecture and business process modeling, *Implementation* stage is performed by considering the SaaS platform environment, and *Testing* stage focuses on validation of interaction between application and the SaaS platform, performance, and usage-metering. This work mentions a key development artifacts and essential techniques required in the development process. However, they do not cover a key characteristic of cloud services, *reusability*, in their process, and stepwise process and detailed instructions are required.

Mietzner and his colleagues present a package format for composite configurable SaaS application by considering requirements of different consumers [5]. They distinguish three key roles in SaaS environment; *SaaS consumer* using the SaaS software, *SaaS provider* selling the software as a service, and *SaaS application vendor* developing applications tha are offered as a service by SaaS provider. To be easily used by SaaS consumer, the package format contains a set of artifacts needed to provision the SaaS and customized application by using variability descriptors. This paper utilizes service component architecture (SCA) in customizing application template. This work focuses on the reusability of SaaS application by using SCA. However, they need to cover all the development process since the unique characteristics beyond the reusability can affect other stages in the process.

3 Design Criteria

Software/Service engineering processes largely depend on computing paradigms. In this section, we define two main design criteria for defining the development process for SaaS cloud services.

One criterion is to reflect the intrinsic *characteristics* of SaaS in the process. Since every well-defined development process should reflect key characteristics of its computing paradigm, this is considered as the main criterion. The other criterion is to promote developing SaaS with the *desired properties*, which are defined as the requirements that any SaaS should embed in order to reach a high level of QoS. Through our rigorous survey of literatures [4][6], we define key characteristics and desired properties of SaaS in Table 1.

Table 1. Characteristics and Desired Properties of SaaS

Characteristics	Desired Properties
♦ Supporting Commonality ♦ Accessible via Internet ♦ Providing Complete Functionality ♦ Supporting Multi-Tenants' Access ♦ Thin Client Model	♦ High Reusability ♦ High Availability ♦ High Scalability

We give explanation and brief justification for each characteristic given in the table. *Supporting Commonality*: As an extreme form of reuse approaches, an SaaS provides software functionality and feature which are common among and so reused by potentially a number of service consumers. Services with high commonality would yield high profits/return on the investment (ROI).

Accessible via Internet: All the current reference models of CC assume that cloud services deployed are accessed by consumer through Internet.

Providing Complete Functionality: SaaS provides the whole functionality of certain software in the form of service. This is in contrast to a *mash-up service* which provides only some portion of the whole software functionality.

Supporting Multi-Tenants' Access: SaaS deployed on providers' side is available to the public. And, a number of service consumers may access the services at the given time without advanced notices. Hence, SaaS should be designed in the way to support concurrent accesses by multiple tenants and handle their sessions in isolation.

Thin Client Model: SaaS services run on providers' side, while service consumers use browsers to access the computed results. Moreover, consumer-specific datasets which are produced by running SaaS are stored and maintained on providers' side. Hence, there will be nothing the browser-like user interaction tool installed and run on client/consumer side.

We now give explanation and brief justification for each desired property given in the table. *High Reusability*: Service providers develop and deploy cloud services and expect that the services would be reused by a large number of consumers. Services which are not much reusable by consumers would lose the justification for investment, while services that can be reused by many consumers would return high enough on the investment. Therefore, it is highly desirable for cloud services to embed a high level of reusability.

High Availability: Cloud services are not just for specific users; rather they are for any potential unknown consumers who may wish to use the services anytime and anywhere. Therefore, it is highly desirable for the service to be highly available if not always. Services with low availability would cause inconvenience and negative business impacts to consumers, and, as the result, they will suffer on reliability and reputations.

High Scalability: In CC, the amount of service requests from consumers, i.e. service load, is dynamic and hard to predict. Therefore, cloud services should be highly scalable even in the situation that an extremely high number of service invocations and so their associated resource requests are requested. Services with low scalability would suffer at the time of peak requests and so lose their reputations by consumers.

While typical web applications provide functionalities for sharing useful information and/or carrying relatively less business-intrinsic operations, SaaS is meant to substitute business-intrinsic or business logic-intensive functionality delivered by conventional software with a notion of internet-based service. Due to this observation, characteristics and properties presented above are more importantly considered in SaaS.

4 Theoretical Foundation for SaaS Services

4.1 Meta Model of SaaS

Before devising an effective SaaS development process, we need to define the key elements of SaaS. By considering general consensus on SaaS, we define its meta model in Fig. 1.

A SaaS-based system has two parties; *Client Node* is on the left, and *Provider Node* is on the right. As thin-client model, client node has only a user interface tool, typically a web browser. The provider node runs SaaS application which typically has components of three layers; *UI Component* as view, *Controller* as control, and *Entity component* as Model. Each component consists of common features and variable features which vary according to consumer.

Controller runs business logics and transactions, and hence it maintains a *Session Log* for multiple consumers. In CC, multiple consumers user the same SaaS and so their application datasets must be well maintained in isolated manner. Hence, in the figure, *Data Set* is maintained by *Entity Component*. Both *Session Log* and *Data Set* are specific to each consumer in CC.

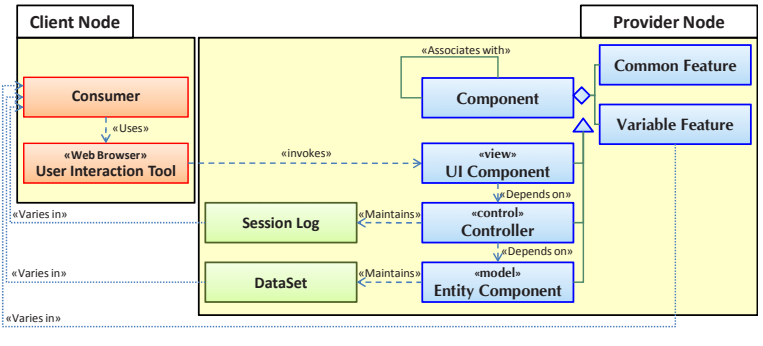

Fig. 1. Meta-Model of SaaS

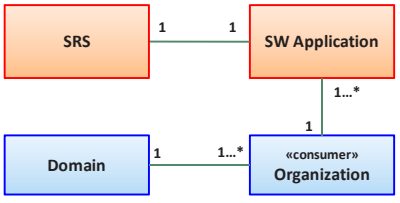

Fig. 2. Criteria on Commonality Determination

4.2 Commonality and Variability in SaaS

Reusability is a key criterion for cloud services, and commonalty is the main contributor to reusability. That is, cloud services with high commonalty will yield higher reusability. Variability is a minor difference among applications or consumers within a common feature. C&V analysis is an essential activity in component-based development (CBD) and product-line engineering (PLE). However, the some notions of C&V in SaaS are different from conventional C&V, mainly due to unique characteristics of SaaS. Hence, we provide its theoretical foundation in this section, so further sections would refer to.

Commonality in SaaS: Commonality in general denotes the amount of potential applications which need a specified feature such as a component or a service. To derive the common features which will be realized in SaaS, we define the relationships among requirement-related elements, as shown in Fig. 2.

A domain such as *finance* and *telecommunications* consists of several organizations, and an organization needs one or more software applications. Each application is associated with a SRS. Hence, a commonality of a feature can be computed as the followings;

$$\text{Commonality}(\text{FEA}_i) = \frac{\text{Number of Applications Needing FEA}_i}{\text{Total Number of Target Applications}}$$

If every application in the domain needs the given feature, the value of *Commonality* will be 1. It would be desirable to include features with high *Commonality* into the target SaaS. The range of the metric is between 0 and 1.

***Variability in SaaS*:** We present the notion of variability in SaaS using three aspects; *persistency, variability type,* and *variability scope*. In a target system or SaaS, there can be places where variability occurs, called *Variation Point*. Each variation point is associated with a set of values which can fill in, called *Variants*.

- Persistency on Variants Settings

Variability-related elements can have three different persistencies; *no persistency*, *permanent persistency*, and *near-permanent persistency*. A variable in a program can hold a value at a given time, and its value can be change as a new value is assigned. Hence, the value stored in a variable does not have an intended persistency, which is illustrated as *no persistency* in the figure.

A variation point in a component in CBD or core asset in PLE is a means for users to set a valid variant, and a variant set in the variation point is persistent, i.e. 'once set not changed.' We call this level of persistency *permanent persistency*.

A variation point in SaaS adds *near-permanent persistency* in addition to the *permanent persistency*, where the variant set in a variation point may be changed over time but in limited way. SaaS is for potentially many consumers, and it has to consider the consumer-specific context and environment in running SaaS application. Consumer may change its context such as *current location/time zone* and *various units used such as currency*. Once a variant is set, its value must be stored until a new variant is set within a session or across multiple sessions. Hence, the persistency of this variant is *near-permanent.* For example, SaaS consumers using mobile-internet device (MID) often travel, and their locations get changed and noticed by SaaS application. Then, SaaS may set new variants for the new locations, and provide services with the right mobile network protocol and its associated services and contents. In SaaS, we only consider the second and third type of variant persistency.

- Variability Types

The variability embedded in a variation point can be classified into several types; *attribute*, *logic*, *workflow*, *interface*, and *persistency* in CBD and PLE [7]. In addition to these, we define two additional types for SaaS; *context* and *QoS*. Variability can occur on the consumer's current context such as *location* and *time zone*, which is called *context* variability type. For a same SaaS, different consumers may require different levels of QoS attributes. Hence, variability can also occur on QoS required by each consumer, which is called *QoS* type.

- Variability Scope

Variability scope is the range of variants which can be set into a variation point. Typically, there are three scopes; *binary*, *selection*, and *open* [7]. When a service consumer wants to customize SaaS for their requirements, they can choose a variant in *binary* or *selection* scopes. However, there is a different implication of *open* scope in SaaS. Due to the thin-client characteristic, service consumers have a limitation on implementing and containing plug-in objects as variant on their client device. To overcome this limitation, we suggest two methods.

One is for service providers to implement and deploy capability to dynamically configure the required plug-in object, and add this object to the pool of variants. The other is for service consumer to implement his or her plug-in objects (may be on a fully powered computer) and to submit the object to the service provider so that it can be added to the pool.

In summary, we identify and distinguish SaaS variability from traditional variability in terms of *persistency, variability type* and *variability scope*. Clear definitions of SaaS variability will make the SaaS process and instructions more feasible, effective in designing reusable services.

5 The Process and Instructions

We define a process for developing SaaS, which has eleven phases as shown in Fig. 3. The five phases highlighted are SaaS development-specific, while other phases are

mostly generic and common among other development paradigms such as object-oriented development.

Phase, 'P1. Requirement Gathering' is to acquire a set of requirement specifications from multiple stakeholders or to define the requirements by considering marketability. Phase, 'P2. Domain Analysis' is to analyze commonality and variability in the target domain. Phase, 'P3. Functional Modeling' is to analyze the functionality in terms of use cases or business processes. Phase, 'P4. Structure Modeling' and Phase, 'P5. Dynamic Modeling' are to analyze structure and dynamic aspects of the SaaS. Phase, 'P6. Architectural Design' is to design architecture of SaaS system by considering SaaS-specific QoS attributes. Phase, 'P7. User Interface Design' is to design user interfaces of SaaS system so that the users can easily use developed SaaS through web browsers. Phase, 'P8. Database Design' is to design database schema of SaaS system including session information and data handled by the users. Phase, 'P9. Implementation' is to write source programs based on the all the design models. Phase, 'P10. Testing' is to test the developed SaaS system. And Phase, 'P11. Deployment' is to deploy all the developed SaaS onto server side.

From now, we provide detailed instructions only for SaaS-specific phases depicted with pink rectangle in subsequent sections.

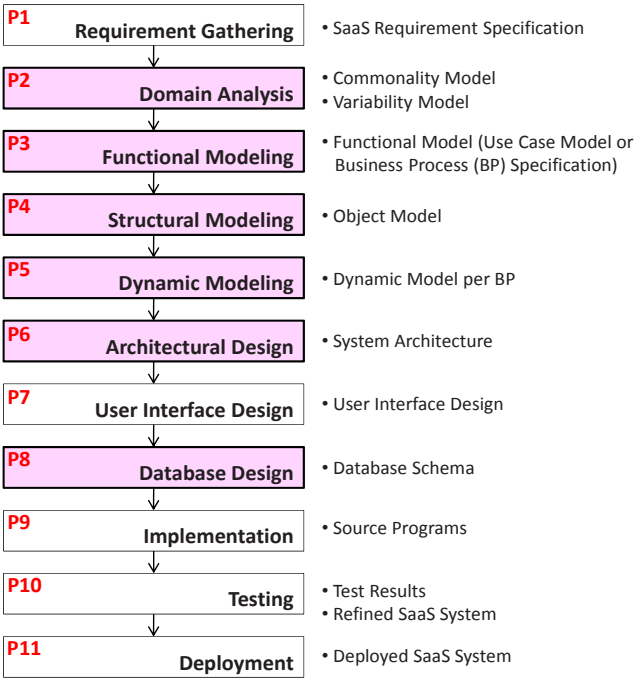

Fig. 3. Overall Development Process of SaaS

5.1 Domain Analysis

***Overview*:** Domain Analysis is an activity to identify common and variable features which will be used in scoping and modeling SaaS [8]. Domain analysis begins with a set of acquired SRSs.

***Instructions*:** This phase is carried out in three steps.

Step 1. Extract Commonality. A set of SRSs collected from different sources could have inconsistency in styles and terms used. Hence, we need to normalize the SRSs by defining standard terms in the domain, using techniques like [9] and [10]. Then, we compare functional and non-functional features of them by using *Commonality Analysis Table* shown in Table 2.

Table 2. Commonality Analysis Table

Feature ID & Feature Name	Feature Description	A set of SRSs				Degree of Commonality
		SRS_1	SRS_2	…	SRS_n	
F01. Generate ID	…	√	√		√	1
F02.CheckCredit	…		√		√	0.3

Step 2. Define Scope of SaaS development. This step is to choose the features which will be implemented as SaaS. We suggest the following criteria.

- A feature with a higher *Commonality*(FEA_i) tends to recruit a larger number of potential service consumers. The features with yellow color in Fig. 4 are the ones with common activities.
- A feature, FEA_i, on which other feature FEA_j depends at a relatively strong level, should be included in the SaaS scope if FEA_j is in the SaaS scope. If such FEA_i is not included, FEA_j would not fulfill its functionality. This is especially essential in SaaS which provides the whole application-level functionality.
- In addition to the two criteria, there exist other criteria for choosing the features such as ROI, marketability, degree of financial sponsorship in developing the SaaS, other essential criteria as defined in the target domain.

Based on the criteria, we now define guidelines for step 2.

- Case 1) *Commonality*(FEA_i) is zero or near-zero. If FEA_i has no other features which depend on $FEA_{i,,}$ exclude this feature. If there are other features which depend on FEA_i and those other features are in the SaaS scope, include this feature. These features are shown with only red color in Fig. 4.
- Case 2) *Commonality*(FEA_i) is one or near-one. Consider to include this feature unless this feature is evaluated as 'excluded' regarding other conditions. These features are shown with only yellow color in Fig. 4.
- Case 3) *Commonality*(FEA_i) is between zero and one, i.e. medium range. When this feature is evaluated as 'inclusion' regarding other conditions, include this feature. Otherwise, consider excluding this feature.

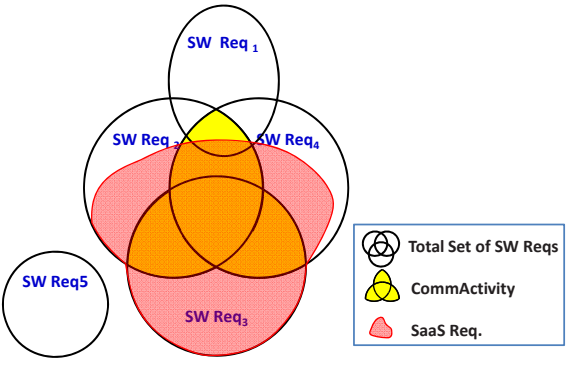

Fig. 4. Scoping the boundary of SaaS development

Table 3. Variability Analysis Table

Feature ID & Feature Name	Variation Point	Var. Type	Var. Scope	Variants for each SRS				Var. Persistency
				SRS_1	SRS_2	…	SRS_n	
F01. Generate ID	VP01	Attr.	Sel.	{a,b}	{a}	…	{b,c}	Permanent
	VP02	Logic	Bin.			…		Permanent
…	…	…	…	…	…	…		…

Step 3. Analyze Variability. For each feature in the scope of SaaS, identify its variation points and applicable variants using techniques like [11]. In addition to conventional variability types, we also identify SaaS-specific variability types as defined in section 4.2. *Variability Analysis Table* in Table 3 can be used in this step. The first column in the table lists features included in SaaS, the second column lists variation points found in the feature, and subsequent columns specify variability type, variability scope, valid variants for each SRS, and variability persistency in order.

For variability analysis for SaaS, we consider the persistency of variants in addition to conventional variability types. Two kinds of persistency could occur;

- *Permanent Persistency*;
  Variants for most variability types such as workflow, logic, interface, and persistency are permanent, meaning the setting lasts forever once it is set.
- *Near-permanent Persistency*;
  Some types of variability types have near-permanent settings such as variants about user sessions, dataset recovered from service faults. Also, attribute and logic variabilities can have near-permanent variant settings.

Once variation points and variants are identified, subsequent phases should be carried out by considering the variability analysis.

5.2 Functional Modeling

This phase is to model the functionality of SaaS by referring to C&V model. For each feature in the model, we analyze detailed functionality in two *use case modeling* and *business process modeling*. Use case modeling can be used when the target SaaS

embeds characteristics of conventional software applications. Business process modeling can be used when the functionality of the target SaaS largely consists of business processes as referred in service-oriented architecture and CC. This phase can be carried out by using well-established techniques such as [10]and [12].

5.3 Structural and Dynamic Modeling

These phases are to model structural and dynamic models of the SaaS based on the C&V model. Well-established techniques to design class and sequence diagrams and to define service WSDL interface can be utilized.

However, variability specified in C&V model should carefully be expressed on these diagrams and the interface. We suggest using stereotypes and/or element tags to express variability such as «variability», <*variability*>, and <*variation point*> [13].

Another consideration for this phase is to define public methods to set variants in the diagrams and interface. These methods should be defined by considering variability scopes.

5.4 Architecture Design

Architecture is an effective means for realizing non-functional requirements, i.e. quality requirements. SaaS applications also have strong requirements on QoS, representatively *scalability* and *availability* [14]. Hence, we suggest a typical architecture design practice as shown in Fig. 5.

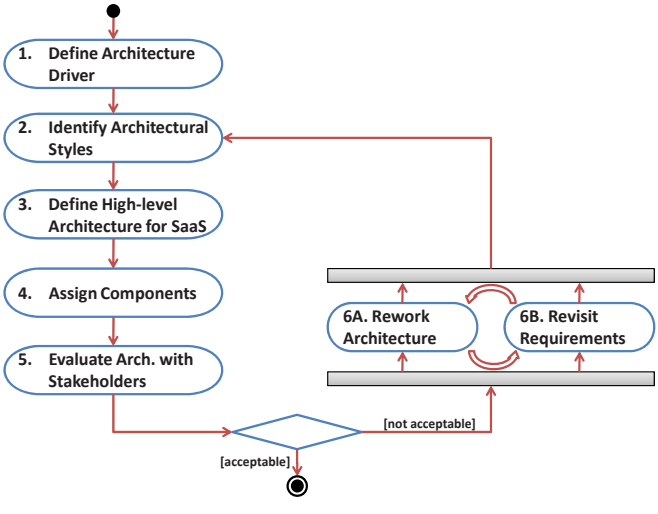

Fig. 5. Architecture Design Process

When considering *Scalability*, we can realize the following techniques in the architecture design;

- Resource Pooling with Multi-Clouds
- Dynamic Load Balancing and Migration
- On-demand Deployment

When considering *Availability*, we can realize the following techniques in the architecture design;

- Using proxy to access SaaS on Mirroring Nodes
- Downloading Lightweight Version of SaaS

5.5 Database Design

Since SaaS provides a complete application-level functionality, it typically maintains a database for strong consumer-specific data sets. Hence, in designing the database schema, two essential information should be maintained; *Session Log* and *Data Set*. Since a number of consumers may use the same SaaS, the service provider should maintain the records of all the sessions run by consumers. This is especially important for long-lasting transactions. *Data Set* is a collection of application-data specific to each consumer. Hence, SaaS should maintain multiple separate databases for the number of consumers.

6 Assessment and Conclusion

Our proposed process is evaluated by the design criterion defined in section 3, as shown in Table 4. The table shows that criteria defined in section 3 are all addressed in one or two phases. For example, *High Reusability* was addressed by *Domain Analysis* phase.

Table 4. Evaluating the Proposed Process

Design Criteria	Phase supporting Criteria	Remarks
Supporting Commonality	Domain Analysis	Proposed C&V methods.
Accessible via Internet	User Interface Design & Deployment	UI design for Browsers & Deployment for Internet Access
Providing Complete Functionality	Domain Analysis	Method to scope SaaS
Supporting Multi-Tenants' Access	Architecture Design Database Design	Architectural Consideration for this concern. DB schema design including Session and Data Sets.
Thin Client Model	Deployment	Deployed and Managed on Provider Side
High Reusability	Domain Analysis	Proposed C&V methods.
High Availability	Architecture Design	Architectural Consideration for these Quality concerns
High Scalability	Architecture Design	

SaaS provides several benefits to service consumers. To realize these benefits, it is essential to make a well-defined engineering process available. Conventional methodologies do not effectively support CC-specific engineering activities such as modeling common features, designing services and services components, and identifying variability among consumers.

In this paper, we presented a systematic process for developing high quality SaaS, and highlighted the essentiality of C&V modeling to maximize the reusability. We first defined criteria for designing the process model, and provides theoretical foundation for SaaS; its meta-model and C&V model. We clarified the notion of

commonality in SaaS, and extended the conventional variability with SaaS-specific consideration. Using the proposed SaaS development process, cloud services with high quality can be more effectively developed.

As the future work, we try to define effective quality assurance guidelines for the phases, and to define a traceability framework where all the artifacts can be cross-related and the consistency can be verified.

References

[1] Weiss, A.: Computing in the Cloud. Net Worker 11(4), 16–26 (2007)
[2] Gillett, F.E.: Future View: New Tech Ecosystems of Cloud, Cloud Services, and Cloud C omputing, Forrester Research Paper (2008)
[3] Turner, M., Budgen, D., Brereton, P.: Turning Software into a Service. IEEE Computer 3 6(10), 38–44 (2003)
[4] Javier, E., David, C., Arturo, M.: Application Development over Software-as-a-Service Pl atforms. In: Proceedings of the 3rd International Conference on Software Engineering Ad vances (ICESA 2008), pp. 97–104. IEEE Computer Society, Los Alamitos (2008)
[5] Mietzner, R., Leymann, F., Papazoglou, M.P.: Defining Composite Configurable SaaS A pplication Packages Using SCA, Variability Descriptors and Multi-Tenancy Patterns. In: Proceedings of the 3rd International Conference on Internet and Web Applications and Se rvices (ICIW 2008), pp. 156–161. IEEE Computer Society, Los Alamitos (2008)
[6] Manford, C.: The Impact of the SaaS model of software delivery. In: Proceedings of the 2 1st Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2008), pp. 283–286 (2008)
[7] Kim, S.D., Her, J.S., Chang, S.H.: A Theoretical Foundation of Variability in Componen t-Based Development. Information and Software Technology (IST) 47, 663–673 (2005)
[8] Mili, H., Mili, A., Yacoub, S., Addy, E.: Reuse-Based Software Engineering: Techniques, Organization, and Controls. Wiley Inter-Science, Chichester (2001)
[9] Choi, S.W., Chang, S.H., Kim, S.D.: A Systematic Methodology for Developing Compon ent Frameworks. In: Wermelinger, M., Margaria-Steffen, T. (eds.) FASE 2004. LNCS, vo l. 2984, pp. 359–373. Springer, Heidelberg (2004)
[10] Her, J.S., La, H.J., Kim, S.D.: A Formal Approach to Devising a Practical Method for Mo deling Reusable Services. In: Proceedings of 2008 IEEE International Conference on e-B usiness Engineering (ICEBE 2008), pp. 221–228 (2008)
[11] Kim, S.D.: Software Reusability. Wiley Encyclopedia of Computer Science and Engineer ing 4, 2679–2689 (2009)
[12] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manu al, 2nd edn. Addison Wesley, Reading (2005)
[13] Chang, S.H., Kim, S.D.: A SOAD Approach to Developing Adaptable Services. In: IEEE International Conference on Services Computing, SCC 2007, July 9-13, pp. 713–714 (2007)
[14] Rozanski, N., Woods, E.: Software Systems Architecture: Working With Stakeholders Us ing Viewpoints. Addison Wesley, Reading (2005)