Evaluating the Release, Delivery, and Deployment Processes of Eight Large Product Software Vendors applying the Customer Configuration Update Model

Slinger Jansen
Institute for Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
slinger.jansen@cs.uu.nl

Sjaak Brinkkemper
Institute for Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
s.brinkkemper@cs.uu.nl

ABSTRACT

For software vendors the processes of release, delivery, and deployment to customers are inherently complex. However, software vendors can greatly improve their product quality and quality of service by applying a model that focuses on customer interaction if such a model were available. This paper presents a model for customer configuration updating (CCU) that can evaluate the capabilities of a software vendor in these processes. Eight extensive case studies of medium to large product software vendors are presented and evaluated using the model, thereby uncovering issues in their release, delivery, and deployment processes.

Categories and Subject Descriptors: D.2.7 SoftwareSoftware Engineering[Distribution, Maintenance, and Enhancement]

General Terms: Management, Reliability, Design

1. INTRODUCTION

With the advent of increased amounts of bandwidth the communication between software vendors and their customers can greatly be improved. Whereas in the past customers and vendors could only communicate by mail and phone, the web can now function as a lifeline between customers and software vendors, to allow for automatic license retrieval, deployment and error feedback, automatic updates, and automatic provision of commercial information to customers. Product software vendors, however, generally do not implement any of these key practices.

To date product software is a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market [1]. Product software vendors encounter many problems when attempting to improve customer configuration updating of their product software. Customer configuration updating is defined as the combination of the vendor side release process, the product or update delivery process, the customer side deployment process, and the activation and usage process. To begin with, these processes are themselves highly complex considering vendors have to deal with multiple revisions, variable features, different deployment environments and architectures, different distribution media, and dependencies on external products. Also, there are not many tools available that support the delivery and deployment of software product releases that are generic enough to accomplish these tasks for any product [2]. Finally, CCU is traditionally not seen as the core business of software vendors, and seemingly does not add any value to the product, making software vendors reluctant to improve CCU.

A number of sources show that CCU is often underestimated and requires more attention in the quickly changing software industry. First, the quality of deployment and upgrade processes can increase customer perceived quality of a software product significantly [3], making it important that these processes are managed explicitly. Also, field research has shown that by explicit management of CCU, software vendors are able to handle large amounts of customers [4]. Finally, Niessink et al. have shown that the development of software should be seen as product development, whereas maintenance should be seen as a customer service, thereby improving customer interaction, the latter being stressed again by the introduction of the Software Maintenance Maturity Model [5].

Even though the previous sources call for more attention to CCU, it is underemphasized in literature. The SWEBOK, for instance, gives a generic description in the software configuration management (SCM) chapter of the processes of release and delivery. The Capability Maturity Model (CMM) [6, 7] also does not provide adequate descriptions for CCU, which is explained by the fact that the CMM does not focus on product software specifically. Attempts have been made in the release candidate of the IT Service CMM [1], although the IT Service CMM also does not provide an elaborate description for the processes of release, delivery, and deployment. Clearly, even though there is a need for process definitions, there are no adequate process descriptions available for product software vendors. This paper attempts to satisfy that need by shedding light on the ugly duckling that is customer configuration updating. To do so the SPICE[2] framework for evaluation is used, a widely used standard for software process analysis and improvement. The SPICE framework fits CCU because SPICE does not prescribe, contrary to CMM-i and the IT Service Maturity Model, large parts of the processes that make up CCU.

The contribution of this paper is twofold. First, it attempts to answer the need for adequate process descriptions by presenting a model that describes and identifies CCU. Secondly, eight descriptive [8] case studies that have been performed at medium to large

[1]http://www.itservicecmm.org/
[2]http://www.sei.cmu.edu/iso-15504/

WISER'06, May 20, 2006, Shanghai, China.

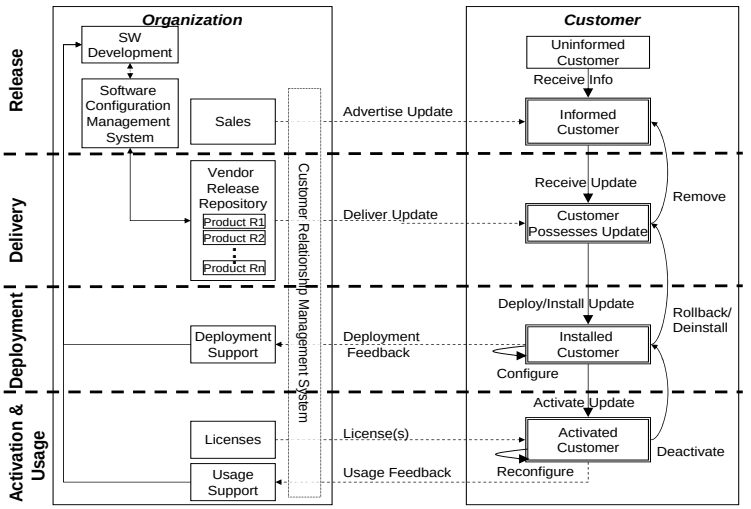

Figure 1: CCU Model

software vendors into their development and CCU processes are presented. The cases are evaluated using the model, which reveals that several key practices are left completely uncovered, due to the implementation effort involved, the lack of sufficient process descriptions, and the lack of sufficiently equipped CCU support tools. Of the eight case studies two are large open source software producers. These case studies resulted into eight case study reports[3] for which facts have been collected from several sources, being interviews, software study, document study, and direct observations while performing the case studies on site.

To produce these results six descriptive case studies [8] were performed at Dutch software vendors and two of open source organisations. These case studies resulted into six case study reports. During several months of doing the case studies, facts have been collected from the following sources:

- **Interviews** - To study the cases and confirm our hypotheses, interviews were held with the people responsible for the development and usage of the studied product.
- **Studying the software** - Academic licenses were granted to the products. These licenses helped to gather many facts by examining, using, and experimenting with the software and its updating capabilities.
- **Document study** - Document study was performed to evaluate the development process and cross check the answers provided by the other sources of information.
- **Direct observations** - Since our research took place at the development departments (of the non-open source cases), we were able to directly observe and document day to day operations.

The interviews consisted of two sessions, one to explore and elaborate, and one to cross-reference answers from other interviews. The second session was also used to cross-reference documentation and confirm the facts stated in these documents. Besides these reviews we also created a case study protocol and a case study database. To ensure reliability, the case study report was reviewed by key informants. Two open source organisations were included to evaluate their key CCU practices. For these two cases all on-line material was used, including the source code of the products and the products themselves were tested extensively. The open source cases' high numbers of employees can be explained by the fact that open source developers are not working on a product full-time. The open source cases can therefore not be compared to the commercial cases in terms of size. The open source cases have been added to show that the CCU model can be used for any type of software vendor or distribution organisation. Also, the open source cases contrast with the commercial cases in a number of interesting ways.

The validity threats to our case studies are construct, internal, external, and reliability [8] threats. With respect to construct validity, the same protocol was applied to each case study, which was guarded by closely peer reviewing the case study process and database. To create a complete and correct overview, both the development and CCU processes have been documented extensively. The internal validity was threatened by incorrect facts and results from the different sources of information. By crosschecking these results and observing the processes as they were going on a complete view could be created. With respect to external validity, the cases are representative for the Dutch software vendor market domain because each software vendor has a different number of customers and is active in a different problem domain. Also, the general information about these vendors has been compared to other vendors that are active in the Platform for Productsoftware[4], an organization that aims to share knowledge between research institutes and software vendors in The Netherlands, with over 100 members.

2. PROCESS AREAS FOR CCU

In this section the process areas of customer configuration updating are defined. The SPICE model, which encourages self analysis for vendors, defines a process as "a statement of purpose and an essential set of practices that address that purpose". These practices are software engineering or management activities that contribute

[3]http://www.cwi.nl/projects/deliver/

[4]http://www.productsoftware.nl/

Table 1: Release key practices

Release Key Practice	Software product vendors							
	ERPC	CMSC	FMSC	HISC	CADC	OCSC	Mozilla	Apache
The organisation has pilot customers to test early releases	●					●	●	●
Release planning is published internally	●	○	●	○	○	●		○
Restrictions on configurations due to internal components are managed	○	●	●	○	○	●	●	●
Restrictions on configurations due to external components are managed	○	○	●	○	○	●	●	●
The tools that support release, delivery, and deployment are managed	●	●	○	○	○	●	●	●
There is a formalized release procedure	●	●	●	●	●	●	●	●
Releases are stored in repositories (that mirrors the SCM)		●	●	●	●	●		●
The formalized release planning is adjusted to customer requirements	●	●	●					
Sending information regularly to customers	●	○	○	○	○	●	○	○
Vendor uses all possible channels for informing customers	○					●	○	●
Legend: ●: Currently implemented; -: Not applicable; empty: Not implemented ○ Requires some manual steps, yet would be easy to automate;								

Table 2: Delivery key practices

Delivery Key Practice	Software product vendors							
	ERPC	CMSC	FMSC	HISC	CADC	OCSC	Mozilla	Apache
Automatic pull is available	●					-	●	
Delivery through any medium (Internet, DVD, Floppy)	●	○	○	○		-	●	●
Download site abstraction	●		○	●	○	●		●
Sending information regularly to customers	●	○	○	○	○	●	○	○
Vendor uses all possible channels for informing customers	○					●	○	●
Legend: ●: Currently implemented; -: Not applicable; empty: Not implemented ○ Requires some manual steps, yet would be easy to automate;								

to the creation of the output (work products) of a process or enhances the capability of a process.

The CCU model, as seen in Figure 1, displays the states a customer can move through after a product or update release on the right side. On the left side, the organisational structures that facilitate interaction are displayed. Within the CCU model four processes are separated by dotted lines in figure 1, being release, delivery, deployment, and the activation and usage process. Both the process models of the Software Dock [9] and SOFA [10] are contained in the presented model.

The processes in the model are triggered by customer and vendor actions. When a vendor decides to notify the customer or receives a customer request, the customer relationship management (CRM) system is used to identify the customer. The vendor then handles the request and interacts with the customer. The customer moves through a number of states when about to update its configuration. At first the customer is unaware of the update, until the customer requests information about a product. Once received, the customer downloads, deploys and activates it for use, in the mean time communicating with the vendor in the form of software, licenses, feedback, and product knowledge.

Release management encompasses the planning of releases, configuration management of released artifacts and external components, and proper distribution of releases and release planning to development, quality assurance, pilot customers, and customers. The delivery process concerns the delivery of software, feedback, licenses, and commercial information between the customer and vendor. Maturity in this process is indicated by a vendor that uses all possible channels for software and knowledge delivery, can apply any type of policy, such as push and pull methods, and enables internal delivery and distribution within the customer organisation (for instance by enabling ITIL [11] practices). Thirdly, the deployment process concerns correct deployment, automatic deployment error resolution, local configuration management (at the customer site), and software rollbacks. A software vendor is considered to have a mature deployment process once the deployments of updates and software are done consistently and correctly, support automatic error resolution, send post deployment feedback, support customisations, and uninstall correctly. One important aspect of deployment is the externalisation of customer data [12], such that the customer data can be backed up sepa Slinger rately from the product. Finally, the usage and feedback process. A software vendor is considered mature in this process once it is able to process feedback automatically, it uses deployment and usage feedback for requirements engineering and development, and it can generate and deliver licenses automatically.

Tables 1, 2, and 3 show the key practices each software vendor has implemented. Each of the key practices has been evaluated using a list of criteria, which have been left out for the sake of brevity. Also for the sake of brevity the specifics of each of the software vendors that were evaluated for this study were left out. The software vendors we evaluated ranged in company size between 65 and 1500 employees and had between 20 and 1 million customers. At the time of writing the usage and activation key practices have not yet been fully evaluated and therefore are not included in this paper.

3. DISCUSSION

The first question that needs to be answered is whether a software vendor's success relies on its customer relationships. In the commercial cases encountered and presented in this paper 50%-70% of their yearly revenue was coming from service contracts from existing customers which in our view shows that customer retention and the maintenance of relationships is essential to survive in the current industry. In the case of open source products, where many of the users of the product are also developers, testers, and quality assurance team members, the same premise on customer relationship management holds. Since CCU is a customer focused process, the improvement of these processes will lead to better customer re-

Table 3: Deployment key practices

Deployment Key Practice	Software product vendors							
	ERPC	CMSC	FMSC	HISC	CADC	OCSC	Firefox	Apache
Configuration completeness checking of external components	●	○	○		○	●	○	●
Automatic resolution of dependency issues							○	
(Semi-)automatic local update process	●			○	○	●	●	
Rollback from an update is possible						●		●
Rollback from an install is possible					○	●		
Updates require no downtime		○			○	○		
Test, acceptance, production environments	○			○		●	○	○
Updates can cope with local customisations	○	●	○	○	○	-	○	
External configuration to allow trace		○				○	○	●
Integrity checks of SW artifacts at customer	●	●				-	●	●
Detection and exploration of customer environment	○					-	○	○
Data backups are done through the product	○	○	○	○		○		
Deployment feedback is sent to vendor	○							
Legend: ●: Currently implemented; -: Not applicable; empty: Not implemented ○ Requires some manual steps, yet would be easy to automate;								

lationships and possibly a higher customer retention rate. By applying the CCU model onto the eight presented cases, Tables 1, 2, and 3 provide us with a number of observations.

Even though some of the cases reported that up to 15% of their deployments failed at the customer side, Table 3 shows that software vendors do not implement all practices of the deployment process. The most commonly reported causes for deployment problems are faulty configurations, incompatible updates, and customisations. By implementing the key practices stated for the deployment process, these problems can partly be avoided [13].

Also, license management, which includes contract registration and automated license creation, is not sufficiently represented in the cases. This area leaves open an opportunity for an integrated contract and license management tool that plugs into any CRM system. For obvious reasons license management is not such a large issue in open source software, although some sense of consciousness throughout the industry about open source licenses would improve customer organisations' awareness of their acquired (open source) products. Often redistribution rules are not respected, simply because customer organisations are not aware of them. Another example where licensing is not such a big issue are the B2B (business-to-business) software vendors we researched.

If anything can be learnt from this research, it is that software vendors must integrate their CRM, PDM, and SCM [4] systems to automate the processes related to CCU. Such automation provides more efficient methods to perform repetitive tasks such as license creation, license renewal, product updating, error reporting, usage reporting, product release, and manual configuration tasks, such as backups. The second main lesson is that usage of feedback reports supplies software vendors with the largest test bed imaginable, and therefore deserves more attention. The presented CCU model can be used as a guideline for software vendors or for the development of a software manufacturing and software product data management system.

Next to the case studies we will be performing in the future, we are planning to build a benchmark site where software vendors can evaluate their own key practices and position themselves in the market. As a continuation on one of the cases we have been offered to implement a subset of the presented key practices within that organisation. We will investigate the implementation of these key practices and use it to validate the results of this research.

4. REFERENCES

[1] L. Xu and S. Brinkkemper, "Concepts of product software: Paving the road for urgently needed research," in *First International Workshop on Philosophical Foundations of Information Systems Engineering*. LNCS, Springer-Verlag, 2005.

[2] S. Jansen, S. Brinkkemper, and G. Ballintijn, "A process framework and typology for software product updaters," in *Ninth European Conference on Software Maintenance and Reengineering*. IEEE, 2005, pp. 265–274.

[3] A. Mockus, P. Zhang, and P. L. Li, "Predictors of customer perceived software quality," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM Press, 2005, pp. 225–233.

[4] S. Jansen, S. Brinkkemper, G. Ballintijn, and A. van Nieuwland, "Integrated development and maintenance of software products to support efficient updating of customer configurations: A case study in mass market erp software," in *Proceedings of the 21st International Conference on Software Maintenance*. IEEE, 2005.

[5] A. April, J. H. Hayes, A. Abran, and R. R. Dumke, "Software maintenance maturity model (smmm): the software maintenance process model." in *Journal of Software Maintenance*, vol. 17, no. 3, 2005, pp. 197–223.

[6] "CMMI SE/SW - Capability Maturity Model Integration," in *version 1.1 Pittsburgh*. Software Engineering Institute, Carnegie Mellon University, USA, 2002.

[7] "(iso/iec 12207) standard for information technology—software lifecycle processes." New York, NY: ISO – International Standard Organization, 1998, 85 S.

[8] R. K. Yin, "Case study research - design and methods." SAGE Publications, 3rd ed., 2003.

[9] A. Carzaniga, A. Fuggetta, R. Hall, A. van der Hoek, D. Heimbigner, and A. Wolf, "A characterization framework for software deployment technologies," in *Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado*, 1998.

[10] F. Plsil, D. Blek, and R. Janecek, "Sofa/dcup: Architecture for component trading and dynamic updating," in *Proceedings of the International Conference on Configurable Distributed Systems*. Washington, DC, USA: IEEE, 1998, p. 43.

[11] Central Computer and Telecommunications Agency, "Itil service support." Stationery Office Books, 2003.

[12] E. Dolstra, G. Florijn, M. de Jonge, and E. Visser, "Capturing timeline variability with transparent configuration environments," in *IEEE Workshop on Software Variability Management (SVM'03)*, J. Bosch and P. Knauber, Eds. Portland, Oregon: IEEE, 2003.

[13] S. Jansen and S. Brinkkemper, "Modelling deployment using feature descriptions and state models for component-based software product families," in *3rd International Working Conference on Component Deployment (CD 2005)*, ser. LNCS. Springer–Verlag, 2005.