Definition and Validation of the Key process of Release, Delivery and Deployment for Product Software Vendors: turning the ugly duckling into a swan

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

Sjaak Brinkkemper
Information and Computing Sciences Institute
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 practices of a software vendor in these processes. Nine 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. Finally, organisational and architectural changes are proposed to increase quality of service and product quality for software vendors.

1 Customer-Vendor Interaction

With the advent of increased amounts of bandwidth the communication between software vendors and their customers can greatly be improved by introducing automatic error feedback reporting, usage feedback reporting, electronic customer feedback, as well as license, patch, and update distribution. Whereas in the past customers and vendors could only communicate by mail and phone, the World Wide 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 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. 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 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 [11]. 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 different products [12]. 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 [15], 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 [13]. 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 [16], the latter being stressed again by the introduction of the Software Maintenance Maturity Model [4].

Even though the previous sources call for more attention to CCU, it is underemphasized in literature. The SWEBOK[1], 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) [2, 1] also does not provide adequate descriptions for CCU, which is explained by the fact that the

[1] http://www.swebok.org

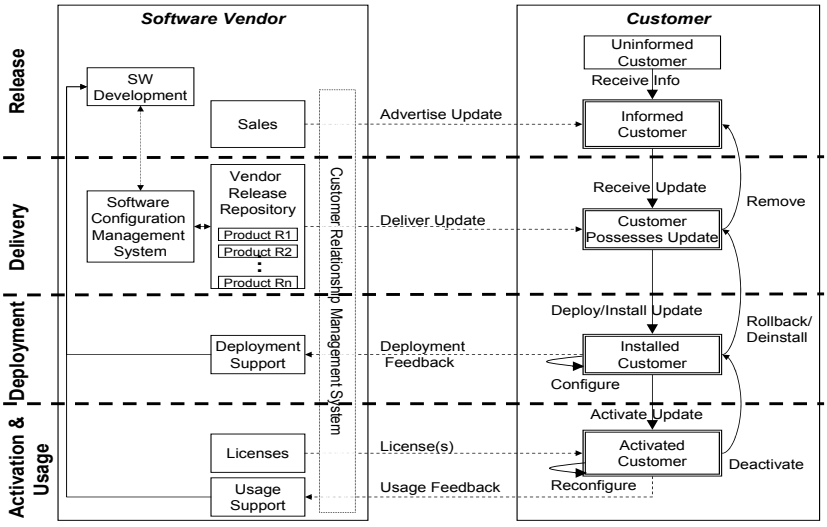

Figure 1. CCU Model

CMM does not focus on product software specifically. Attempts have been made in the release candidate of the IT Service CMM [16], 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.

Users of product software underestimate the underlying process that enables their applications to be updated successfully. This paper does not specifically address the issue of deployment, which within reasonable borders often is quite successful, especially for smaller products. This paper does, however, answer software vendor's needs for adequate process descriptions for CCU and allows them to explore the opportunities for process improvement and integration. The contribution of this paper thus is twofold. First, it provides process descriptions by presenting a model that describes and identifies CCU in Section 2. Secondly, nine case studies that have been performed at medium to large software vendors into their development and CCU processes are presented in Section 3. These case studies provide practical knowledge and specific process descriptions which are, similar to the presented model, focussed on customer interaction. The cases are evaluated using the model in Section 4, which reveals that several 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. Finally, our conclusions are presented in Section 5.

2 Customer Configuration Updating

In this section the key process of customer configuration updating is modelled. This model explicitly defines customer actions, enabling a software vendor to better manage and predict the practices that need extra focus. Much akin to the CMM [18], the model uses the concepts of practices, features, and process. practices are practices of a software vendor that enable features. Features are defined as a property of a process that improves product quality and quality of service. Each process identifies a cluster of related features that, when performed collectively, achieve a set of goals considered important for enhancing process capability. A software vendor possesses a feature within a process, once it responds correctly to one of these customer triggered actions.

To describe the practices for CCU, its process need to be established. These process are found using a previously established model for software updaters [12] that focuses on the customer. Due to the fact that software maintenance and deployment focuses solely on the customer, the model is extended with the organisational interactions that are required to fully support a customer's actions after an update is released. The CCU model, as depicted 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 distinguished, being release, delivery, deployment, and the activation and usage process. The process are separated by dotted lines in figure 1 and are further described in the sec-

tions below. Both the process models of the Software Dock [7] and SOFA [19] are contained in the presented model.

Processes in the model are triggered by customer actions. These actions are becoming aware of, downloading, deploying, reconfiguring, activating, and deactivating the release. When a vendor 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 hopefully 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.

2.1 Release process

Software release management encompasses the identification, packaging, and delivery of the elements of a product, for example, executable program, documentation, release notes, and configuration data (SWEBOK). With respect to release management a primary practice is that the vendor has a formalized release scenario that describes step by step the creation of a release, a release planning that describes the planned releases with time durations and/or dates, and a defined policy on how this last document is shared within the organization. Such awareness creates transparency within the software development organisation, improving the relationship between the sales and development departments. This is related to the practice that the sales, development, and support departments must all be aware of the product's relationships with other components, such that no late surprises at a customer site are possible. For example, if a product comes in simplified Chinese, it might not be compatible with a large number of commercial database management systems, even though the original release of the application, which was in English, did work with those components. To shorten release times the product can be managed using a product data management (PDM) system. By doing so, the vendor is forced to manage all secondary artefacts, such as manuals, boxes, and DVDs as explicit as the product itself.

Another practice of the release process with respect to product knowledge management is that all versions of the software that have been released by an organisation, must be stored in a release repository that mirrors the releases in the software configuration management system. This enables customers using older versions to reinstall and update their product at any time. The same way releases must be managed explicitly, the vendor must manage explicitly all internally used development and CCU support tools. Finally, the vendor must manage all external components that are included and packaged with the product.

The vendor must make a conscious effort to keep its customers updated on the latest news and product releases using any channel of communication, such that customers are not lagging behind in either product releases or product release information. Sales and lead management includes the use of pilot customers that pre-evaluate and test the software before an official release (at a discount price). One relevant aspect that determines product quality is strategic planning of product releases and updates. Customer organisations utilize product software in such an intensive manner that an update is a costly matter, due to down time, system instability, and the number of systems that require the update. A software vendor must establish the best time when an update is published and what the possible consequences are of deploying the update [4]. Microsoft, for instance, releases its security updates for all its products on the second Tuesday of the month[2] and they have communicated this with their customer base.

2.2 Delivery process

The delivery process concerns the delivery of software, licenses, and product knowledge to customers. To begin with software vendors must enable customer organisations to perform deployment using whichever medium a customer chooses, such as DVD, CD, a local area network, or the Internet. Secondly, customers must be able to remotely deploy applications and updates onto a user system without physically having to touch it. Thirdly, the product must supply a mechanism for automatic pull of updates, such that the customer can check for updates and download them automatically on a regular basis. The customer must be able to abstract from the download site of the vendor, allowing the customer to use an internal download server. If possible, the product must send back a deployment report after a customer has deployed the product, to inform the vendor whether the deployment was successful or not.

2.3 Deployment process

The **deployment process** contains practices that enable a product to be installed, removed, and updated from a customer's system. The practices related to local configuration management are prone to many issues, such as missing (external) components, incomplete downloads, erroneous deployments, and overwritten customisations. To improve the deployment a deployment tool must inspect the local configuration, to see whether external components are missing and whether the local system provides enough resources, such as disk space. Also, downloaded packages must be

[2] http://www.microsoft.com/athome/security/default.mspx

checked for integrity and completeness. In the cases of missing components and files that do not pass their integrity checks, some automatic resolution must be implemented. Finally, it must be possible to rollback from an update or deployment to return to the previous configuration.

Customisations are widely applied for specific business domains and for specific customers. In many cases these customisations account for a large portion of their total revenue, which proves that explicit customisation management is vital to many software vendors. A practice for a software product with many different customisations at different customers thus is that the main product is updated without overwriting local customisations. Once these issues have been tackled [11], the software vendor can make these processes as quick and easy for the customer as possible by implementing (semi-)automatic deployment, update, and rollback procedures. Another practice is that updates do not require downtime when performing an update, allowing the customer to use the product without interruptions. Tool support is found in package managers, such as rpm-update, Portage, NIX [9], Microsoft's open source project Wix[3], and installers, such as InstallShield[4] and the free InnoSetup[5] tool.

Customer organisations use different testing and acceptance stages, such as proposed by the IT Infrastructure Library (ITIL) [8] and ISO 20000[6], before actually implementing software in the entire organisation. This requires that deployments are done quickly, and that configuration settings and data files are moved separately from the software. This practice is related to the externalisation of all user and configuration data, which enables a transparent configuration environment [10]. Within such an environment all configuration and user data is accessed externally from the product, which allows for relationships to be established between configuration data between products, thus enabling sharing of user configuration data such as e-mail account settings between e-mail clients, font sizes between applications, or even appearance settings between operating systems. Such externalisation allows for the product to perform product data backups as well, enabling quicker and more reliable backup retrieval actions.

2.4 Activation and Usage process

The **activation and usage process** concerns the start-up, initialisation, authentication and running of a product at the customer site on the customer system. One practice of usage and activation is customer-side license management, which enables a customer organisation to manage licenses explicitly, and activate the product with a different license on each start-up, allowing customers to use test and development versions, and to provide different functionality to different user profiles. Another practice belonging to license management is that licenses need to be stored in some coded fashion, to hinder piracy of products. Also, to have maximum commercial flexibility, the licenses should control large parts of the software, such that any functionality is activated or deactivated using the licenses. The vendor too, must explicitly manage its customer licenses. A vendor must be able to automatically renew a license for a customer, such that the vendor can renew or prolong a license without much effort. To achieve this, it must be possible to generate licenses from contracts automatically. Customer side license management solutions exist, such as ManageSoft's[7] software management suite. Dedicated vendor side license management systems, such as Hasp[8], provide many practices that are required in this process, yet do not provide any features for integration with a vendor's CRM system.

Feedback management allows a vendor to gather large amounts of data about its customers and its product as it acts in the field. Feedback can come from either automatic sources or manual customer triggered sources. Feedback is used, in the automatic case, to provide knowledge to the vendor about product usage and knowledge about the customer's configuration. Finally, the user should be able to report errors and questions to the software vendor through the software product. This allows users to state questions and report bugs about specific screens and unclear functions in the product. Feedback management requires changes to the product, such that the product is used to communicate with the user, by form of a daily pop-up, or a message to the sales department if a user attempts to use an unpurchased feature a number of times. Currently there are a number of scientific projects that focus on error feedback, such as the Skoll project [14], the GAMMA project [17] and a commercial product called FogBugz[9], that provides a system that can be used to insert probes into the code of any product.

3 The Cases and their practices

In this section the anonymised cases are described. Some generic information is provided on each software vendor and the reasons why the case was included in this research are stated. A description is also given on how the case studies were performed. Table 1 provides some statistics on each organisation that is part of our research set. Tables 2, 3, 4, and 5 show the practices these software vendors have

[3] http://sourceforge.net/projects/wix/

[4] http://www.installshield.com

[5] http://www.jrsoftware.org/isinfo.php

[6] ISO/IEC 20000 is the first international standard for IT Service Management. It is based on and is intended to supersede the earlier British Standard, BS 15000.

[7] http://www.managesoft.com/

[8] http://www.aladdin.com/

[9] http://www.fogcreek.com/FogBugz/

Table 1. Some Statistics on each Organisation

Software Vendor	Employees	CCU employees	Customers	Technology	CMS
ERPComp	1500	15	160.000	ASP+ Delphi	Proprietary
CMSComp	65	5	140	Java	SubVersion
FMSComp	160	3	900	Delphi + Java	VSS + CVS
OCSComp	115	2	20	C++	CVS
PLUComp	60	3	4.000	Delphi	PVCS
HISComp	100	2	40	Delphi	VSS
Mozilla	710	5to10	1.000.000+	Java	CVS
Apache	388	NA	1.000.000+	Java	SubVersion
Eclipse	150	NA	1.000.000+	Java	CVS

implemented. Each of the practices has been evaluated using a list of criteria, which have been left out for the sake of brevity, but will soon be published as an on-line survey.

The evaluation model used to establish each organisation's CCU practice adequacy, the Spice[10] framework for evaluation models is used. Spice is 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. Spice allows for the creation of evaluation criteria for practices within defined processes. A vendor is then said to be fully adequate for a practice once it fits all criteria. If a vendor implements only some of the practices it is said to be partly adequate, if it implements most of the practices it is said to be largely adequate, and if it implements none of the criteria it is said to be not adequate at all.

3.1 Case Study Approach

To produce the results in this paper six descriptive case studies [21] were performed at Dutch software vendors. These case studies resulted into six case study reports[11]. During several months of doing the case studies, facts have been collected from interviews, document study, software study, and direct observations at the vendor. Three open source organisations were included to evaluate their key CCU practices. For these three cases on-line material was used 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.

The validity threats to our case studies are construct, internal, external, and reliability [21] 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 in follow-up interviews and observing the processes as they were going on a complete view has been 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. Finally, to defend reliability we would gather the same results if we redid the case studies due to the established protocol, with one major proviso, which is that many of the case study reports, published after the case study, lead to improvements in each of the software vendors' organisations. To ensure reliability, the case study reports were reviewed by key informants. The open source cases were selected from the three categories proposed by the OECD for packaged software [3]. From the application software category Firefox was selected, from system infrastructure software the Apache http server project, and finally Eclipse for the development tools category.

The validity of the conclusions of this research is threatened by the situationality of the practices that are specified for CCU. Many of the practices only apply for specific products, specific customers, and specific product architectures. For example, when an application such as Eclipse is upgraded, only small amounts of customer data need to be separately updated, whereas for some of the B2B cases the

[10] ISO/IEC 15504 (Software Process Assessment) is an International Standard under development in ISO/IEC JTC1/SC7/WG10, whereWG10 is the working group for development of standards and guidelines covering methods, practices, and application of process assessment in software product procurement, development, delivery, operation, maintenance, and related service support.

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

Table 2. Release practices

Release practice	Software product vendors								
	ERPC	CMSC	FMSC	HISC	PLUC	OCSC	Firefox	Apache	Eclipse
Pilot customers	3						2	2	2
Release planning/roadmap is published internally	3	1	1	2	1	1	3	3	3
Internal dependencies managed	3	2	2	1	1	2	2	2	3
External dependencies managed	2	2	2	1	1	2	2	1	3
Tools used to support CCU are managed explicitly	3	2	3	1	3	1	3	3	3
There is a formalized release scenario	3	2	3	2	3	2	2	2	2
Releases are stored in a repository	1	2	3	3	2	3	3	3	3
Release planning is adjusted to customer req	2		2	2					
Legend Empty: Not adequate 1: Partly adequate; 2: Largely adequate; 3: Fully adequate									

datamodel undergoes structural changes between revisions. Also, the ITIL practices are only relevant for software used within organizations. Therefore a vendor's CCU cannot be judged to be of low quality when it only implements limited CCU practices. On the other hand a vendor's CCU can be said to be immature, which only means that the vendor is not applying CCU practices to their full potential. This is a limitation for growth in customer numbers and generally means that many of the CCU activities are done manually and thus time consuming.

3.2 Hospital Information System

HISComp business activities are the production and sale of medical information systems, the customization of their products for customers, and the reselling of all required third-party hardware and software. *HISComp* currently has a customer base of approximately 40 hospitals in Europe and currently employs approximately 100 employees. *HISComp* is a typical software developer with a traditional and straight-forward way of distributing software via CDs. Patches are released on a website and the customer's system manager is responsible for deploying the patch, using a detailed list of instructions [5].

3.3 On-line ERP Information Portal

ERPComp is a manufacturer of software for accounting and enterprise resource planning (ERP) that has established an customer base of over 160,000 customers, mainly in the small to medium enterprise sector. Through autonomous growth and acquisitions the number of employees has grown to 2,025 in 2004. The International Development department employs 365 developers on different international locations. *ERPProd*, *ERPComp*'s product is a front office application that provides organizations with financial information, multi-site reporting, and supports relationship and knowledge management. Employees, customers and company partners are provided with real-time on-line access to information across an entire organization. In an earlier paper [13] the results of this case study were published due to the extraordinary integration this company has achieved within its PDM, customer relationship management, and software configuration management.

3.4 Content Management System

CMSComp is a web technology company that focuses on content management, online application development and integration of backend systems into web portals. The services of *CMSComp* include consulting, development, implementation, integration and support of interactive web applications. These services are supported by *CMSComp* product. *CMSComp* attempts to find a personified solution for each customer organisation. *CMSComp* currently employs 85 people. *CMSComp* has only recently started focussing on their product, instead of the services the company used to provide. The content management and display product is generally deployed on a web server, where it will remain unchanged, until updated manually by *CMSComp*.

3.5 Providing a Service On-Line

OCSComp is an application service provider for large commercial organisations. *OCSComp* currently employs around 100 people, based on multiple European locations. *OCSComp* does not deliver software to customers becuase customers visit the *OCSComp* portal to see data that was gathered for *OCSComp*'s customers. The ASP case adds some interesting data to our research. To begin with *OCSComp* is much more capable at local configuration management and deployment processes, due to the fact that their servers are freely accessible by the organisation itself. This

Table 3. Delivery practices

Delivery practice	Software product vendors								
	ERPC	CMSC	FMSC	HISC	PLUC	OCSC	Firefox	Apache	Eclipse
Vendor uses all channels for customer comm.	1	1	1	1	2	3	1	1	1
Automatic pull/Push	1				1	3	1		1
Delivery using any medium (Internet, DVD, etc)	3		1	1	1		3	3	3
Download site abstraction	3	1	2	3	1		3	3	3
A download report is sent back to the SW vendor	1						1		
Legend Empty: Not adequate 1: Partly adequate; 2: Largely adequate; 3: Fully adequate									

explains *OCSComp*'s adequacy in local configuration management, and product data and SCM features, and can therefore not be compared to other product software companies in this area. Due to the fact that customers log into *OCSComp*'s website on at least a weekly basis, *OCSComp* uses this channel to communicate the product information and new functionality to its customers. Finally, licensing has not been connected to CRM and requires an employee to copy the information from a contract into the license management system.

3.6 Facility Management System

FMSComp is an international software vendor that produces facility management and real estate management software for organisations. *FMSComp*'s products are marketed through four international *FMSComp* subsidiaries and eight international partners. At present *FMSComp* employs 160 full time employees. *FMSComp* is an extremely good tool builder and has built many tools that are not managed explicitly, sometimes resulting in loss of knowledge about the source code or even the source code itself. These tools, however, have improved their CCU. *FMSComp*'s weakest process is licensing, even though they have a semi-automatic license generation process. The software has an in-built function to create a feedback report that is used to inform *FMSComp* of problems in their software. However, this report must be e-mailed to *FMSComp* manually by the customer.

3.7 Plug-in for a Large Application

PLUComp currently employs 60 employees. *PLUComp* produces software plug-ins for a large application that support building services and building management consultants in the Dutch industry. *PLUComp* and its 60 employees at present serve 4000 customers. Due to the nature of their product, *PLUComp* must deliver its products to customers by unpacking a common application and repacking it with their plug-in, using InstallShield for the deployment process. They use both software and hardware licensing mechanisms. Due to the size of their final deployment package they use CDs for distribution. *PLUComp* makes no assumptions about the customer's network connection and therefore does not do any user or deployment feedback. Backups of user configuration data and files are complex, due to the fact that such knowledge is stored in many different formats, databases, and files, spread out over the complete deployment.

3.8 Mozilla Firefox

Mozilla currently owns *Firefox*, one of the most successful open source development projects available. The *Mozilla* internet browser, created by the Mozilla Foundation, provides a viable alternative to other browsers such as Opera and Internet Explorer. *Mozilla* has implemented some update practices in their product, such as an automatic update function that is used to update the local product installation. Mozilla does not, however, keep strong ties with each customer due to its large number (75 million downloads, according to the *Mozilla* website). *Mozilla* reports error reports back to the Mozilla Foundation, by use of feedback servers, using *Mozilla*'s Talkback.

3.9 Apache's HTTP Server

Apache development began in February 1995 as a combined effort to coordinate existing fixes to the NCSA http program, to become a well known and successful open source product. At present it is the most used HTTP server software for servers on the world wide web. The product is used mostly by web server maintainers with some technical knowledge, and therefore *Apache* does not have many of the practices for the features of local configuration management and feedback management. Another reason for the absence of these practices is that the *Apache* HTTP server is used for public websites, where automatic deployment and feedback could compromise security.

3.10 The Eclipse Project

The *Eclipse* project is an open source initiative that nowadays consists of four subprojects, the *Eclipse* platform,

Table 4. Deployment practices

Deployment practice	Software product vendors								
	ERPC	CMSC	FMSC	HISC	PLUC	OCSC	Firefox	Apache	Eclipse
Configuration checking for external components	3		1		1	3	1	2	1
Automatic resolution of dependency issues					2	3	2		
(Semi-)automatic local update process	2			1	1	2	2		2
Rollback from an update is possible						3		3	
Rollback from an install is possible		1	1				2	3	1
Updates require no downtime							1		1
Test, acceptance, production environments	2	1	1	2	1			1	
Updates can cope with local customisations	2		2	1			2	2	3
External configuration to allow trace [10]	1	1	1	1	3	1		2	3
Integrity checks of SW artifacts at customer	1						1		
Exploration of customer environment	2	1						1	1
Data backups are done through the product	2		1	1	1	1			
Legend Empty: Not adequate 1: Partly adequate; 2: Largely adequate; 3: Fully adequate									

the *Eclipse* OSGi, the *Eclipse* Java Development Tools, and the *Eclipse* plug-in development environment. These four projects are used to create the *Eclipse* SDK, a widely used developer kit for Java applications (and any other platform, if the plug-ins have been created). Due to the plug-in architecture, dependencies can be explicitly managed, and updates for plug-ins can be performed automatically and at runtime, without overwriting customisations. *Eclipse* is not unique in the fact that it does not include any licensing or feedback features, much like *Apache*.

4 Discussion

Now we put the key process of CCU up for 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 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 relationships and possibly a higher customer retention rate. By applying the CCU model onto the nine presented cases, Tables 2, 3, 4, and 5 lead to the following observations:

- Software vendors focus insufficiently on customer side configuration management
- Licensing and contract integration is rare
- Software vendors do not focus on deployment and usage feedback
- Software vendors neglect explicit product knowledge management

With these observations and customer retention, product quality and quality of service in mind, a number of conclusions can be drawn. Even though some of the cases reported that up to 15% of their deployments failed at the customer's site, Table 4 shows that software vendors do not implement practices in the area of **customer side configuration management**. The most commonly reported causes for deployment problems are faulty configurations, incompatible updates, and customisations. By implementing the practices stated for the deployment process, these problems can be avoided [11].

Also, vendor side **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.

All cases do not sufficiently implement the practices of **usage and deployment feedback**. Such feedback, however, is used to gather essential product knowledge, such as product incompatibilities, common user errors, and usage statistics of product functionality. This knowledge is translated into requirements for future products and product fixes.

To process customer usage feedback, to store product compatibilities, and to handle the huge volume of require-

Table 5. Activation and usage practices

Activation and Usage practice	Software product vendors								
	ERPC	CMSC	FMSC	HISC	PLUC	OCSC	Firefox	Apache	Eclipse
Licenses are coded	2	1	2		3				
Licenses activate modules	3	2	3	2	3	3			
Licenses are managed explicitly by customer	3		2		2				
Possible to renew licenses automatically	2		1		2	2			
Licenses are generated using contracts	3		2		2	2			
Temporary licenses are distributed	2	1	1		1	1			
Awareness of customers configuration	1	1	1	1	1	1	1		
Feedback can be given by user	1	1	1	1	1	3	1	1	1
Usage reports are created and sent to the vendor		1							
Error feedback is sent to vendor		1				3	3		
Feedback is used and analysed	2	1				1	2		
Legend Empty: Not adequate 1: Partly adequate; 2: Largely adequate; 3: Fully adequate									

ments on a product, a software vendor must have a **software product knowledge infrastructure** [20]. Such a product knowledge infrastructure is used to communicate product information throughout the development department, the organisation, and its customers. As part of the software product knowledge infrastructure a PDM system must be used to manage the software product.

The open source cases contrast with the commercial cases in a number of interesting ways. CCU model coverage looks different for an open source product than for the other products presented, even though many of the CCU issues experienced are similar. Licensing is an underrepresented aspect of open source products for obvious reasons and bugs tend to be reported using other channels than the product itself (Bugzilla, for instance). Thus for the three open source cases customer feedback seems to be underrepresented, whereas deployment and user feedback are integral parts of the open source development process. Open source software products, however, can improve their development process by implementing automatic usage and error feedback as well, using such applications as *Mozilla*'s open source TalkBack project. Due to the fact that the development trunk of open source software can be downloaded at all times, open source projects also do not structurally need to assign pilot customers.

One of the main lessons learnt from this research is that a company can serve many customers as long as it focuses on making CCU effort as low as possible. The use of a proprietary PDM system for software products allows *ERPComp* to reason and store information about their software and share knowledge about product items throughout the company, such as compatibility information. The integration of their SCM and CRM systems allows customers to log into the *ERPComp* customer portal and download software the customer has purchased, including a license file for that customer. This license file is managed on both the customer and vendor side and must periodically be renewed by the customer.

Interestingly enough, many practices in the areas of customisation management, internal product relationship management, and product data and software configuration management are an integral part of software product line development. Especially the explicit manner in which products and product configurations are managed by the PuLSE approach [6] sets an example for product software companies. The combination of the concepts of this research and PuLSE paves the way to an integrated software PDM system that manages all artefacts and information for a software product family.

In our search for tools that can provide the practices presented in this paper and in [12], uncovered product niches have been encountered. To begin with it seems that there are no PDM systems that explicitly manage licenses, software products, fixes, and their patches, in such a way that customers can log in and download them. Secondly, feedback sending and feedback analysis applications seem to be in short supply. Finally, operating systems and deployment tools [7] do not fully support the practices for local software configuration management.

If anything can be learned from this research, it is that software vendors must integrate their CRM, PDM, and SCM [13] 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 PDM system.

5 Future Work

The presented material allows for a larger evaluation of the customer configuration updating process. 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 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 practices within that organisation. We will investigate the implementation of these practices and use it to validate the results of this research. We are currently negotiating with several software vendors whether feedback mechanisms can be implemented within their products, to evaluate the usefulness of such functionalities and to get more practical experience with field data gathered from customers.

References

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

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

[3] Information technology outlook 2002. "Organisation for Economic Co-operation and Development", 2002.

[4] 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*, volume 17, pages 197–223, 2005.

[5] G. Ballintijn. A case study of the release management of a health-care information system. In *proceedings of the IEEE International Conference on Software Maintenance, ICSM2005, Industrial Applications track*, 2005.

[6] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. Pulse: a methodology to develop software product lines. In *SSR '99: Proceedings of the 1999 symposium on Software reusability*, pages 122–131, New York, NY, USA, 1999. ACM Press.

[7] 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.

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

[9] E. Dolstra. Integrating software construction and software deployment. In B. Westfechtel and A. van der Hoek, editors, *11th International Workshop on Software Configuration Management (SCM-11)*, volume 2649 of *LNCS*, pages 102–117, Portland, Oregon, USA, 2003. Springer-Verlag.

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

[11] 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)*, LNCS. Springer–Verlag, 2005.

[12] 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*, pages 265–274. IEEE, 2005.

[13] 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.

[14] A. Memon, A. Porter, C. Yilmaz, A. Nagarajan, D. Schmidt, and B. Natarajan. Skoll: Distributed continuous quality assurance. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 459–468. IEEE Computer Society, 2004.

[15] 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*, pages 225–233, New York, NY, USA, 2005. ACM Press.

[16] F. Niessink and H. van Vliet. Software maintenance from a service perspective. In *Journal of Software Maintenance: Research and Practice*, volume 12, pages 103–120, 2000.

[17] A. Orso, D. Liang, M. J. Harrold, and R. Lipton. Gamma system: Continuous evolution of software after deployment. In *Proc. of the ACM International Symposium on Software Testing and Analysis (ISSTA'02)*, pages 65–69, 2002.

[18] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. The capability maturity model: Guidelines for improving the software process. In *SEI Series in Software Engineering*. Addison-Wesley Publishing Company, 1995.

[19] 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*, page 43. IEEE, 1998.

[20] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma. A reference framework for software product management. Institute of Computing and Information Sciences, Utrecht University, Technical report UU-CS-2006-014. Submitted for publication.

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

Acknowledgements

We are very grateful to the representatives of the six software vendors that allowed us to study them so closely. Furthermore, the authors thank Vedran Bilanovic, Hans van Vliet, and Merel Van Geest for their many inspiring ideas that contributed to this paper. Finally, the authors wish to thank Gerco Ballintijn for performing the *HISComp* case study.