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

Slinger Jansen

Sjaak Brinkkemper

institute of information and
computing sciences,
utrecht university

technical report UU-CS-2005-041

www.cs.uu.nl

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

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. Finally, organisational and architectural changes are proposed to increase quality of service and product quality for software vendors.

1. INTRODUCTION

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, and 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 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 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 [2]. 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 [3]. 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 [4], 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 [5]. 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 [6], the latter being stressed again by the introduction of the Software Maintenance Maturity Model [7].

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) [8] 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 [9], 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.

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 case studies that have been performed at medium to large software vendors into their development and CCU processes are presented. 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, 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

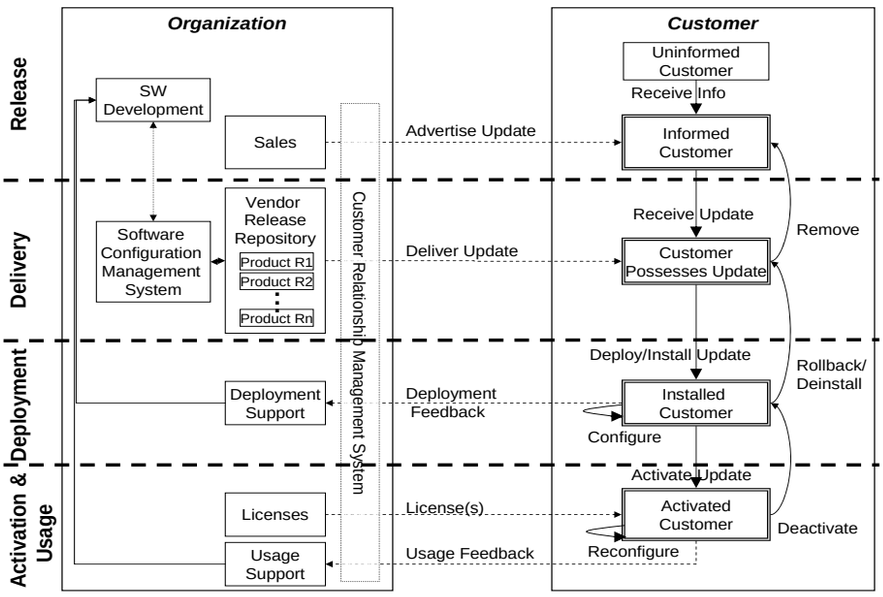

Figure 1: CCU Model

CCU support tools.

Section 2 describes the CCU model and its process areas, along with the key practices belonging to each process area. The approach taken in the case studies and the eight studies that were performed and evaluated using the CCU model are reported in Section 3. A description of the results per case study is also provided there. The key practices and combined results of the case studies are discussed in Section 4, where we also defend the claims made in the paper. Finally, the future work and our conclusions are presented.

2. PROCESS AREAS FOR CUSTOMER CONFIGURATION UPDATING

In this section the key process area of customer configuration updating is modelled. This model explicitly defines customer actions, enabling a software vendor to better manage and predict the key practices that need extra focus. Much akin to the CMM [8], the model uses the concepts of key practices, features, and process areas. Key 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 area 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 area, once it responds correctly to one of these customer triggered actions.

To describe the key practices for CCU, its process areas need to be established. These process areas are found using a previously established model for software updaters [3] 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 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 process areas are distinguished, being release, delivery, deployment, and the activation and usage process areas. Both the process models of the Software Dock [10] and SOFA [11] 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.

The presented model provides four process areas, being release, delivery, deployment, and activation and usage. The process areas are separated by dotted lines in figure 1 and are further described in the sections below.

2.1 Release Process Area

The release process area describes the release of a software product for a specific vendor and the interaction with its customers. The features within the release process area are

- Release process management
- Product knowledge management

With respect to release process management a primary key practice is a formalized release procedure that describes step by step how a release is created. Another key practice is the sharing of knowledge within the organisation about the next release, such that all employees whose jobs are in some way related to the new product release are aware of the functionality in the next release, the

release date, and the policy on sharing such information. Such awareness creates transparency within the software development organisation, improving the relationship between the sales and development departments. This is related to the key 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.

One key practice of the release process area 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 software 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. Also, customer communication in the release process area is most interesting to the sales department of a product software vendor. A sales department must have insight into the purchasing guidelines and processes of a customer organisation. 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 [7]. Microsoft, for instance, releases its security updates for all its products on the second Tuesday of the month[1] and they have communicated this with their customer base.

2.2 Delivery Process Area

The delivery process area concerns the delivery of software, licenses, and product knowledge to customers. The key practices belonging to the delivery process area are focussed on the following features:

- Delivery methods to customers
- Customer side delivery

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.

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

2.3 Deployment Process Area

The **deployment process area** contains key practices that enable a product to be deployed, removed, and updated. The key practices in the deployment process area are categorized into:

- Environment checking
- Local configuration management
- Deployment process automation

The key 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 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 key 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 [12], 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 key practice is that updates do not require downtime when performing an update, allowing the customer to use the product without interruptions.

Customer organisations often use different testing and acceptance stages according to the IT Infrastructure Library (ITIL) [13] 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 key practice is related to the externalisation of all user and configuration data, which enables a transparent configuration environment [14]. 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 Area

The **activation and usage process area** concerns the activation and working of a product at the customer site. The activation and usage process area focuses on the following features:

- License management
- Feedback management

License management 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 key practice belonging to license management is that licenses need to be stored in some coded fashion, to hinder piracy

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
CADComp	60	3	4.000	Delphi	PVCS
HISComp	100	2	40	Delphi	VSS
IBOSS	710	5to10	1.000.000+	Java	CVS
WSOSS	388	NA	1.000.000+	Java	SubVersion

of products. Finally, 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 must also explicitly manage its customer licenses. To begin with, 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.

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.

3. THE CASES AND THEIR KEY 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 key practices these software vendors have implemented. Each of the key practices has been evaluated using a list of criteria, which have been left out for the sake of brevity. An open circle shows that the vendor has implemented the facilities that provide a key practice, yet it has not implemented the key practice. These can be considered "quick wins" for the vendor.

3.1 Case Study Approach

To produce these results six descriptive case studies [15] were performed at Dutch software vendors. These case studies resulted into six case study reports[2]. During several months of doing the case studies, facts have been collected from several 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. CCU model coverage looks different for an open source product than for the other products presented. To begin with, 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).

The validity threats to our case studies are construct, internal, external, and reliability [15] 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[3], an organization that aims to share knowledge between research institutes and software vendors in The Netherlands, with over 100 members. The comparison shows that the six cases are a cross-section of the Dutch software industry. Finally, to defend reliability we would gather the same results if we redid the case studies, 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.

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

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

Table 2: Release key practices

Release Key Practice	Software product vendors							
	ERPC	CMSC	FMSC	HISC	CADC	OCSC	IBOSS	WSOSS
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;								

3.2 Hospital Information Management 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 international hospitals 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. Each customisation that is built for a customer is included in a separate customisation branch, which is merged with the trunk later on. Such variable functionality is activated using a coded license file. *HISComp* does not gather automatically any technical information on customer sites and the working of the product heavily depends on the customer's system manager. [16] *HISComp* releases patches and service packs containing multiple patches irregularly and main releases periodically.

3.3 On-line ERP Information Portal for Large Businesses

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 [5] the results of this case study were published due to the extraordinary integration this company has achieved within its product data management (PDM), customer relationship management, and software configuration management. The main lesson learnt was that a company can serve many customers as long as it focuses on making CCU effort as low as possible. *ERPComp* applies the KISS (Keep it Short and Simple) principle to such an extent that they have abolished version management. The use of a proprietary product data management 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. Furthermore *ERPComp* has developed its own product deployment and update tool, and reports the version number of the latest download by the customer to the CRM software, such that the support department can always see what version of the software the customer is currently using.

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* products. *CMSComp* attempts to find a personified solution for each customer organisation. *CMSComp* currently employs 65 people. *CMSComp* has been experiencing such rapid growth over the last years that they have had to limit growth to keep it manageable at 6%. To serve the growing amount of customers with this restriction, *CMSComp* has started a partner program, where partner companies can provide the same services as *CMSComp*, using the *CMSComp* product.

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 customers. The product is checked with an unencoded XML license file that is accessible to the customer. License files cannot be generated automatically. Due to the large amount of customisation that is implemented during the building of a site, the content management product has a transparent software architecture especially adjusted to enable such customisations. Due to the complexity of the software, deployment is a complex two hour process per web server. Due to the fact that *CMSComp* generally has access to their customers' web servers, remote deployment and updating are possible.

3.5 Providing a Counter Service On-Line

OCSComp is an application service provider that provides commercial organisations web statistics. They provide page count solutions to any type of customer, from small counters on personal websites, to large navigation tracking counters on e-commerce sites. *OCSComp* currently employs around 100 people, based on multiple European locations. *OCSComp* does not deliver software to customers, since customers visit the *OCSComp* portal to see the data that was gathered while people surfed their sites.

Table 3: Delivery key practices

Delivery Key Practice	Software product vendors							
	ERPC	CMSC	FMSC	HISC	CADC	OCSC	IBOSS	WSOSS
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;								

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 explains *OCSComp*'s presence 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. Recently, they have started testing a new version of their software, which has been completely reimplemented using J2EE technology.

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 development and CCU processes. They are very strong in product development and provide services to many large companies. They provide different types of deployment for their product, as to allow both *high network traffic, low deployment effort* and *low network traffic, high deployment effort* scenarios. *FMSComp*'s weakest area 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 CAD plug-in for Building Design

CADComp currently employs 60 employees. *CADComp* produces software plug-ins for AutoCAD that support building services and building management consultants in the Dutch industry, by creating drawing libraries, tools, and enhancements for two three dimensional drawing tools, being AutoCAD and IntelliCAD. *CADComp* and its 60 employees at present serve 4000 customers.

Due to the nature of their product, *CADComp* must deliver its products to customers by unpacking a common CAD 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. *CADComp* 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. This complexity is caused by a complex software architecture that allows *CADComp* to deliver its plug-in for different CAD applications.

3.8 Mozilla Firefox

Mozilla, from hereon referred to as *IBOSS*, currently owns *Firefox*, one of the most successful open source development projects currently available. The *IBOSS* internet browser, created by the Mozilla Foundation, provides a viable alternative to other browsers such as Opera and Internet Explorer.

On the other hand *IBOSS* has implemented some update key 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 *IBOSS* website). Also, *IBOSS* does not report any information back to the Mozilla Foundation, by use of feedback servers (such as *Apache*'s TraceBack) or another form of automatic post-installation feedback.

3.9 Apache's HTTP Server

Apache development, from hereon referred to as WSOSS, 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 [17]. 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 *WSOSS* does not have many of the key practices for the features of local configuration management and feedback management. Another reason for the absence of these key practices is that the *WSOSS* HTTP server is used for public websites, where automatic deployment and feedback could compromise security.

4. EVALUATION OF THE PROCESS AREAS AND FEATURES

This section discusses and describes the impact and effort required for making improvements in each process area. These results have been generalised for the eight cases and are summarised in Table 6. Each of the following paragraphs describes the problems and the availability of tools per feature.

The **release process management** feature describes the skills of a company to plan and manage their product and update releases. The maturity of a software vendor can often be established by looking at the key practices for this feature, because it is essential to all other vendor side process areas. Primarily, to have all key practices for this feature, the vendor should manage its software with a PDM system. By doing so, the vendor is forced to manage all secondary artifacts, such as manuals, boxes, and DVDs as explicit as the product itself. Both the open source cases do not have a re-

Table 4: Deployment key practices

Deployment Key Practice	Software product vendors							
	ERPC	CMSC	FMSC	HISC	CADC	OCSC	IBOSS	WSOSS
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	○	○	○	○		○		

Legend: ●: Currently implemented; -: Not applicable; empty: Not implemented
○ Requires some manual steps, yet would be easy to automate;

Table 5: Activation and usage key practices

Activation and Usage Key Practice	Software product vendors							
	ERPC	CMSC	FMSC	HISC	CADC	OCSC	IBOSS	WSOSS
Licenses are coded	●		○	○	○	○		
Licenses have an effect on software	●	●	○	○	●	○		
Licenses are managed explicitly by customer	●		○		●			
Possible to renew licenses automatically	●		○		○	●		
Licenses are generated using contracts	●	○	○					
Temporary licenses are distributed	●							
Awareness of customers configuration	○	○	○	○	○	●		
Feedback from a user is possible through the product						●	●	
Usage reports are created		●			○	●		○

Legend: ●: Currently implemented; -: Not applicable; empty: Not implemented
○ Requires some manual steps, yet would be easy to automate;

lease planning that is adjusted to customer requirements, due to less market pressure for early releases. The tools used to support the key practices in this process area are numerous, and contain tools that support software configuration management and many proprietary tools that support software artifact and product management.

The **product knowledge management** feature is strongly represented for all cases. The vendor must manage the relationships of its products to other products in both a human readable format, for the support, sales, and development department, and a computer readable format, to allow for automatic conflict detection and even automatic dependency resolution. Also, the availability of past product releases is required such that customers can download older versions. *IBOSS* does not provide such functionality, due to the fact that the source of their products is always available. The downside of this is that a customer can never download older versions of the software automatically to be used with a set of other applications, without having to build the source code. Another example is *ERPComp*, which only provides the latest version of the software and no other, such that users will always use the latest version. The question remains whether a vendor wishes to provide customers with more flexibility, or whether this simplification and therefore cost saving method does not scare off customers. Many tools are available for knowledge management and distribution, however, each organisation has its own channels for distribution.

The **delivery methods to customers** feature is dependent on many different factors, such as bandwidth, network policies, security, and infrastructure. Coverage of all key practices in this process area is rather weak, with the automatic pull key practice as an extreme. To improve in this process area, a software vendor must carefully review whether the software architecture and the vendor organisation itself do not restrict customer communication. Integration of the CRM system throughout the complete organisation is required to gain serious improve in this area. There are some tools available that already supply such integration, though a lot of customisation is required [5].

The **customer side distribution** feature is dependent on the format of deployment and installation packages, the product software architecture, and possible storage locations. The key practice to allow a customer to use any medium for deployment enables customer organisations to freely deploy software using its proprietary methods of deployment. An example encountered in the cases is customers requesting for Microsoft Installer packages (msi) because their internal deployment and distribution tools require msi packages. The key practice that allows a customer to download updates and deployment packages from any location can reduce network load on the vendor repository as well, since the customer does not allow each user system to go on-line individually and download the latest updates from the vendor. Tool support is found in package managers, such as rpm-update, Portage, and Microsoft's open source project Wix[4].

Environment checking, a feature of deployment, requires the deployment application or software product to first scan the system on which the update will take place, for the availability of required components and possible resource constraints such as disk space. If such constraints or missing components are encountered, these issues must be resolved automatically. There are not many tools

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

available (besides package managers) that can support these key practices, mostly due to the complexity and number of different deployment environments.

The **Local configuration management** feature is highly dependent on the operating system and deployment tools used by the customer. Some deployment tools have integrated the build process into the management of software packages [18], whereas other tools are primarily focussed on copying of files from one location to another, such as InstallShield[5]. Implementing the key practices in this process area require large development efforts and changes to the software architecture, such as the rollback key practice, which is generally not implemented because changes to the data model cannot be rolled back [19] [20] without a versioned database management system. Subsets of the key practices in this process area are often covered by the operating system, such as the deployment capabilities of Gentoo's Portage or the registry and the deinstall key practices for Microsoft Windows. Customisation key practices are hardly represented in the presented case studies, showing a large opening for product and service quality improvements. Customisation management requires heavy development effort and integration with the CRM system, to store customer configuration settings, such as network architecture, used database system, and operating system. This information is used to deliver the appropriate updates and fixes to specific customers and to perform market and requirements research. The backup of data key practice is usually provided through commercial database management tools, and therefore the results presented in Table 4 might be misleading. However, providing a mechanism to backup all external data and configuration information with the press of a button is a valuable key practice because customers are allowed to perform quicker and more reliable backups. Some of the key practices of **customisation management** are supported by development platforms such as J2EE that force developers to store configuration information in external XML files.

To improve the feature of **deployment process automation** the two previous features must be combined. Both automatic dependency resolution and local configuration management must be automated to perform automatic updates and deployments. Tools that support such automation are not widely available and an automatic update and deployment solution requires a specially adjusted software architecture.

License management consists of both license management on the customer side and the vendor side. Customer side license management is usually easy to implement, since many license management mechanisms, such as to renew the license, are already implemented under the hood. To provide the key practices within customer side license management development effort is required mostly. On the other hand, vendor side license management requires changes to the CRM system, such that it can store and distribute licenses, and requires organisational changes, such that license generation and renewals are done automatically by the sales department. Improving vendor side license management requires little effort, due to the fact that some type of CRM and license management is usually already present in an organisation. Customer side license management solutions exist, such as ManageSoft's[6] software management suite. Dedicated vendor side license management systems, such as Hasp[7], provide many key practices that are required in this process area.

Improving in the area of **sales and lead management** may require changes to the product, such that the product are 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. The reasons for this key practice are numerous. Often a customer will have an old version of the software running, requesting outdated and expensive support on old (and even buggy) functionality. Also, when customers are not aware of the newest functionality within a product, they might opt for a competitor who simply told the customer first about one market sensitive characteristic of their product. Pilot customers can pre-evaluate the software and have a say in the final set of requirements and in commercial cases use the product at a discount price. Pilot customers increase market awareness for the vendor and improve relationships with some of its primary customers. New functionality can also be made available to customers using temporary licenses, which allows customers to test new functionality before actually purchasing it. This process area is limited by trust and network infrastructure issues. It will require some changes to the CRM system to get messages to the right customer organisations. Some commercial tools are available in the form of PDM and CRM systems, but once again integration and customisation effort are large and structural communication between the sales and development departments about new product features is required.

The **feedback management** feature is valuable to a software vendor because it will introduce new requirements on the product, show what the most used functions of a product are, and where most errors occur. Though improvements in this process area requires a lot of effort, the products discussed in this paper already implemented different error logging mechanisms, sometimes even with a "send error report to vendor by e-mail"-button. No commercial tools were found to handle such feedback although some tools such as Mozilla's TraceBack and the components presented in [21] provide similar functionality. Network infrastructure, privacy, and security should be taken in consideration carefully when improving these areas. Effort to improve this feature is low, whereas the implementation of feedback is highly profitable. Such feedback reports can even be linked to customers, informing the vendor of its customers' configuration. This information is used by the support department to determine a customer's configuration, but also to inform the development department of "proven" configurations. A well-known example of feedback error reporting is the feedback function in Microsoft's Windows XP. However, other mechanisms are imaginable [22], such as usage reports (which can also be used for pay-per-usage scenarios) that can help improve the knowledge about which functionalities are most used by customers.

5. DISCUSSION

Now we put the key process area 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 eight presented cases, Tables 2, 3, 4, and 5 lead to the following observations:

- Software vendors focus insufficiently on customer side configuration management

[5]http://www.installshield.com
[6]http://www.managesoft.com/
[7]http://www.aladdin.com/

Table 6: Process area impact assessment

	Software Architecture	Development Effort	CRM Impact	Organisational Impact	Available tools	Change project size
Release						
Release process management	none	none	large	large	SCM systems, no PDM	medium
Product knowledge management	none	none	medium	medium	SCM systems, no PDM	medium
Delivery						
Delivery methods	some	none	some	some	none	Small
Customer side distribution	some	medium	none	none	package deployment tools	medium
Deployment						
Environment checking	medium	medium	some	none	some deployment tools	medium
Local configuration management	large	medium	none	some	some tools, operating systems	medium
Deployment process automation	large	large	some	some	none	large
Activation and Usage						
License management	some	medium	large	medium	some	medium
Feedback management	some	heavy	medium	some	some	medium

- 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 side, Table 4 shows that software vendors do not implement key 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 key practices stated for the deployment process, these problems can be avoided.

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. Another example where licensing is not such a big issue are the B2B (business-to-business) software vendors we researched. *CMSComp*, for example, provides unencoded XML license files to its customers and defends that choice by saying that trust in B2B markets is more important, since *CMSComp* will simply offer the functionality to them if the customer chooses to change the license file.

All cases do not sufficiently implement the key 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. For the two open source cases customer feedback seems to be underrepresented, whereas deployment and user feedback seem to be 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.

To process customer usage feedback, to store product compatibilities, and to handle the huge volume of requirements on a product, a software vendor must have a **high-level product knowledge infrastructure**. Such a product knowledge infrastructure is used to communicate product information throughout the development department, the organisation, and even its customers.

Interestingly enough, many key 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 [23]. Especially the explicit manner in which products and product configurations are managed by the PuLSE approach [24] sets an example for product software companies. The combination of the concepts of this research and PuLSE paves the way to an integrated software product data management system that manages all artifacts and information for a software product family.

In our search for tools that can provide the key practices presented in this paper and in [3], undiscovered product niches have been encountered. To begin with it seems that there are no product data management systems that explicitly manage licenses, software products, their 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 [10] generally do not support the key 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 [5] 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.

6. 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 key practices and position themselves in the market. This evaluation technique, however, requires a new classification of software product companies, which is used to further analyze the results from such research. 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. We are currently negotiating with several software vendors whether the concepts shown by Elbaum et al. [22] 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.

7. 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 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.

[3] 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.

[4] 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.

[5] 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.

[6] F. Niessink and H. van Vliet, "Software maintenance from a service perspective," in *Journal of Software Maintenance: Research and Practice*, vol. 12, no. 2, 2000, pp. 103–120.

[7] 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.

[8] 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.

[9] F. Niessink, V. Clerc, T. Tijdink, and H. van Vliet, "The it service capability maturity model," 2005.

[10] 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.

[11] 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.

[12] S. Jansen, "Alleviating the release and deployment effort of product software by explicitly managing component knowledge," in *Proceedings of the Workshop on Development and Deployment of Product Software*. US Education Service, 2005, pp. 21–30.

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

[14] 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.

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

[16] 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.

[17] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "A case study of open source software development: the apache server," in *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*. Limerick, Ireland: ACM Press, 2000, pp. 263–272.

[18] E. Dolstra, "Integrating software construction and software deployment," in *11th International Workshop on Software Configuration Management (SCM-11)*, ser. LNCS, B. Westfechtel and A. van der Hoek, Eds., vol. 2649. Portland, Oregon, USA: Springer-Verlag, 2003, pp. 102–117.

[19] S. Jansen, G. Ballintijn, and S. Brinkkemper, "Software release and deployment at exact: a case study report," in *technical report SEN-E0414*. CWI, 2004.

[20] S. Jansen, "Software Release and Deployment at Planon: a case study report," in *Technical Report SEN-E0504*. CWI, 2005.

[21] K. Renaud and R. Cooper, "An error reporting and feedback component for component-based transaction processing systems," in *Proceedings of the 1999 User Interfaces to Data Intensive Systems*. Washington, DC, USA: IEEE, 1999, p. 141.

[22] S. Elbaum and M. Diep, "Profiling deployed software: Assessing strategies and testing opportunities," in *IEEE Trans. Softw. Eng.*, vol. 31, no. 4. Piscataway, NJ, USA: IEEE, 2005, pp. 312–327.

[23] J. Bosch, "Software product lines: organizational alternatives," in *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*. IEEE, 2001, pp. 91–100.

[24] 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*. New York, NY, USA: ACM Press, 1999, pp. 122–131.

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 Tijs van der Storm for their many inspiring ideas that contributed to this paper. Finally, the authors wish to thank Gerco Ballintijn for performing the *HISComp* case study.