Improving the Customer Configuration Update Process by Explicitly Managing Software Knowledge

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

ABSTRACT

The implementation and continuous support of a software product at a customer with evolving requirements is a complex task for a product software vendor. There are many customers for the vendor to serve, all of whom might require their own version or variant of the application. Furthermore, the software application itself will consist of many (software) components that depend on each other to function correctly. On top of that, these components will evolve over time to meet the changing needs of customers. To alleviate this problem we propose to alleviate the software release and deployment effort and reduce risks associated with it. This will be achieved by explicitly managing typical knowledge about the software product, such as configuration and dependency information, thereby allowing software vendors to improve the customer configuration updating process. The proposed solution of knowledge management at both the customer and vendor site, is validated through industrial case studies.

Categories and Subject Descriptors

D.2.7 [**Software**]: Software Engineering—*Distribution, Maintenance, and Enhancement*

General Terms

Management, Reliability, Design

1. INTRODUCTION

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 (CCU) 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 usage and 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 no 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 carefully managed. After all, reputations take years to create and can be ruined in one day. 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 maintenance should be seen as a customer service, thereby improving customer interaction, the latter being emphasized by the introduction of the Software Maintenance Maturity Model [6].

Even though the research presented above focuses on the improvement of the CCU process, case studies have shown [7, 8, 5] that many issues remain unsolved. Large parts of the CCU process are still performed manually, such as quickfix distribution and deployment, license file creation, and error feedback reporting. Next to this, informal surveys have shown that up to 15 percent of deployments of products are unsuccessful, due to missing components and configuration errors by the deployers. System administrators for large networked environments too, experience many problems with respect to deployment, caused by heterogeneous environments, faulty configuration update tools, lack of knowledge about system and software constraints, and security issues.

The area of research has already matured over the years and many groups have attempted to solve the problems described. To begin with different tools have been developed and are being developed to improve release and deployment of software. Perhaps the most famous research tool is the Software Dock [9], a deployment toolset that assists software developers in distributing their software automatically, through the use of deployable software descriptions [10], and communicates to customers the availability of new software packages. Some of the weaknesses of the Software Dock are the lack of a redistribution layer, the lack of functionality for (pre)configuration and instantiation, the limited number of supported package formats, and the limited support for deploying multiple similar versions of one component. A newer tool called Nix [11] has solved many dependency problems in the deployment of a system working on a Linux platform. The tool's weaknesses include the requirement for complete redeployment of a system (also known as "stop the world"), the fact that the tool is built for one platform only, and that it does not include any customer-vendor interaction. Finally, the component framework Sofa [12] has been developed with a number of tools. Sofa solves many deployment is-

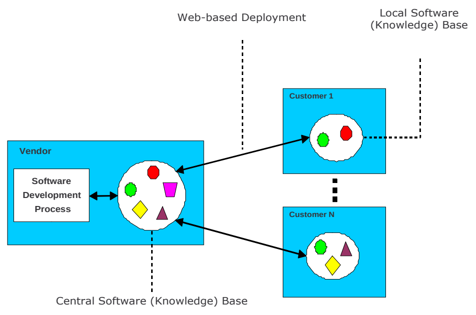
Web-based Deployment
Local Software (Knowledge) Base
Customer 1
Vendor
Software Development Process
Customer N
Central Software (Knowledge) Base

Figure 1: Software Knowledge Bases for Software Delivery

sues, however, it has weak support for configuration evolution and the framework assumes a limited four state model being *source*, *built*, *deployed*, and *running*.

Next to the scientific tools there is a large amount of different commercial and open source release, delivery, and deployment tools. Some common commercial examples are InstallShield and PowerUpdate, and some open source examples are Portage, rpm-update, and APT-Get. None of these, however, provide the same scope of functionality as the scientific tools. More importantly, the scientific, the commercial, and the open source tools do not support the complete CCU process for any software product [3].

The research is related to the work done on feature description languages [13, 14], knowledge creation during software development [15], and software knowledge bases [16]. Also, the research is closely related to the implementation of transparent configuration environments, which attempts to make all the options that are available at different binding times transparent to the developer, deployer, and customer. These solutions, especially the ones focusing on transparent configuration environments, are highly useful for the product software industry, yet have not been implemented sufficiently in current practice. Finally, to explore the state-of-the-practice of software deployment, different case studies have been reviewed, such as a number of case studies in the Finnish software industry [17]. Also, our own case studies have added to the problem overview substantially [7, 8, 5].

The hypothesis on which this research is based is that these problems, which concern both product software vendors and customers, can be solved by explicit management of software knowledge throughout the complete product software lifecycle. The knowledge stored in these databases, such as configuration restrictions, software updates, and usage data, can be shared between product software vendors and customers to improve the software development, maintenance, and CCU processes. Two issues that need to be solved with respect to this communication are that customers might not be willing to share such information and that the communication is intercepted by malignant thrid parties. The first issue is part of the research and can be handled contractually between customers and vendors. The second issue is outside the scope of this research.

We continue this research abstract with a proposed solution in the following section. In section 3 the research methods are presented. The abstract ends with a discussion and a conclusion in section 4.

2. PROPOSED SOLUTION

One can easily see that component dependencies, evolution, and variability lead to a combinatorial explosion of possible component configurations and that a manual (or even semi-automatic) administration of the actual configurations used by a customer is labor-intensive and error-prone. The desired insights and quality control are thus not achieved. In order to improve upon this situation one needs the following (see Figure 1):

- A **Software Knowledge Base (SKB)** information system that contains exhaustive information about all software artifacts and their constraints. In addition, it has a number of built-in types of artifacts and knowledge about these artifacts (e.g., a component can only be delivered if it passes all tests). In this way, the detailed configuration of a user can be computed on the basis of a few key parameters. This is a fully automatic process with guaranteed quality. The software knowledge information system should also be able to answer what if questions of the form “What happens if we upgrade for customer C component X from version 6 to version 7 and add a new component Y?” In this way planning and quality control of the upgrading or replacement of components can be supported.
- A **Web-based Delivery Process** to deploy, upgrade and replace software components. Based on the information generated by the software knowledge information system, this process controls the remote configuration of software components.

The economic motivation for the project is to further strengthen the manufacturing and deployment of product software in the Netherlands. The scientific goal is to develop methods and techniques for the fully automatic consistency checking and web-based deployment, upgrading and integration of product software.

To establish the required functionality for such tooling, the current solutions have first been evaluated using common evaluation frameworks where available, such as Conradi and Westfechtel’s version model [18] for software configuration management tools, and created new ones when necessary. This lead to the creation of a framework of evaluation for software updaters [3].

Also, to establish the need for such tools, a model of the CCU has been created to evaluate the capabilities of software vendors with respect to the CCU process [19]. The CCU model, as seen in Figure 2, 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,

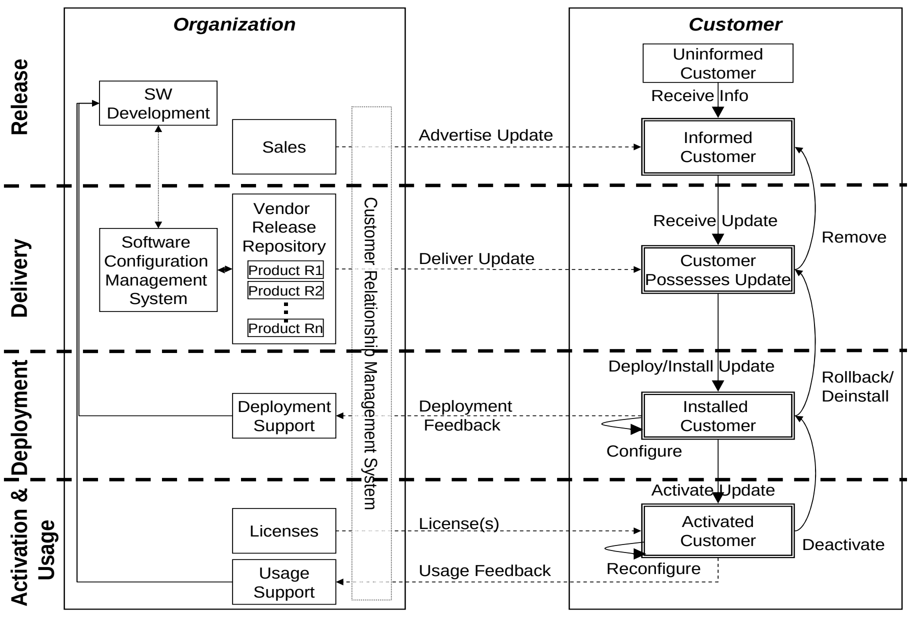

Figure 2: CCU Model

delivery, deployment, and the activation and usage process areas. Both the model on which the Software Dock [20] is based and the SOFA [21] update model are contained in the CCU 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.

3. RESEARCH METHODS

The research is built up out of consecutive steps. The research is started with literature study and tool evaluation. Secondly, a number of descriptive case studies are undertaken at product software companies to identify whether the problem statement is accurate and still applicable. Thirdly, a solution is proposed and designed. The solution design is evaluated to explore whether the solution can actually be used in a practical setting. After this is successful the solution is implemented in a lab environment. When the experiments are successful, a number of case studies are undertaken to implement the solution at software vendors. The results of this action research are used to evaluate and validate the research.

The literature and tool studies have resulted in a framework for evaluating product update tools [3]. The framework has already been used to establish the features that current generic deployment tools are lacking. The descriptive case studies, of which some have been published [5, 8], have been used to create the CCU model, which can be used to evaluate the capabilities with respect to the release, delivery, and deployment processes of product software vendors [19]. The CCU model is also expected to be useful for evaluating the tools and solutions that are implemented. A number of knowledge management and distribution tools are currently under development for the research project. One such tool is a tool that enables modelling of the deployment process for feature oriented component based software products [2] before the deployment takes place. This enables the user or system manager to evaluate the effects and restrictions before deploying a software product by asking this *proof of concept* tool “what-if” questions.

Due to the fact that the research is largely of a qualitative rather than a quantitative nature, it is closely guarded for validity threats. For example, the validity threats to our descriptive case studies are construct, internal, external, and reliability threats. With respect to construct validity, the same protocol was applied to each descriptive 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 descriptive 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[1], an organization that aims to share knowledge between research institutes and software vendors in the Netherlands, with over 100 members. The comparison shows that our descriptive 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

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

that many of the case study reports, published after the case study, lead to improvements in the CCU process of the software vendors' organisations.

The research is largely based on the case studies and communication with the Dutch software industry. Most of the communication with industry happens through the Platform for Product Software. The platform serves the research as a resource for case studies, but also as a forum where Academia meets the industry to exchange knowledge. Another resource for case studies and work in the area of product contexts, is the Netherware project[2] where students (towards the end of their studies) are encouraged to develop their individual product and start a software company. Netherware contributes to the research by providing different products in different contexts that can be evaluated freely and openly. Currently the research is in the solution design and implementation phase.

4. CONCLUSION

Even though the research is well underway, a number of issues still need to be tackled. One, and probably the main issue, is that it is complex to prove that the knowledge management tools have actually improved the quality of service or whether this is simply caused by other generic product improvements. The solution to this problem probably lies in other quantifiers, such as release effort, update effort, and deployment automation.

This research abstract provides a helicopter view of the work that is being performed with respect to the improvement of the CCU process by explicitly managing software knowledge. The research is aimed at alleviating release, delivery, and deployment efforts for product software vendors, system administrators, and product software users. This is achieved by providing evaluation frameworks for tools and organizations, by building tools that explicitly manage software knowledge, and by evaluating these tools in industrial settings.

5. REFERENCES

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

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

[8] 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.

[9] R. S. Hall, D. Heimbigner, and A. L. Wolf, "A cooperative approach to support software deployment using the software dock," in *International Conference on Software Engineering*, 1999, pp. 174–183.

[10] R. Hall, D. Heimbigner, and A. Wolf, "Specifying the deployable software description format in xml," 1999.

[11] E. Dolstra, E. Visser, and M. de Jonge, "Imposing a memory management discipline on software deployment," in *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*. IEEE, 2004, pp. 583–592.

[12] P. Hnetynka, "Component model for unified deployment of distributed component-based software," 2004.

[13] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," SEI, CMU, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-21, 1990.

[14] T. van der Storm, "Variability and component composition," in *Software Reuse: Methods, Techniques and Tools: 8th International Conference (ICSR-8)*, ser. LNCS. Springer, 2004, pp. 86–100.

[15] P. Klint and C. Verhoef, "Enabling the creation of knowledge about software assets," vol. 41, no. 2-3, pp. 141–158, 2002.

[16] B. Meyer, "The software knowledge base," in *Proceedings of the 8th international conference on Software engineering*. IEEE, 1985, pp. 158–165.

[17] J. Tiihonen, T. Soininen, T. Mannisto, and R. Sulonen, "State of the practice in product configuration – a survey of 10 cases in the finnish industry," in *Knowledge Intensive CAD, First Edition*. Chapman et Hall.

[18] R. Conradi and B. Westfechtel, "Version models for software configuration management," vol. 30, no. 2, p. 232, 1998.

[19] S. Jansen and S. Brinkkemper, "Definition and validation of the key process areas of release, delivery and deployment of product software vendors: turning the ugly duckling into a swan," in *Technical Report, UU-CS-2005-041*. Utrecht University, 2005.

[20] 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.

[21] 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.

[2]http://www.netherware.nl