Customization of Open Source Software in Companies

Steffen Keßler and Paul Alpar

Institut für Wirtschaftsinformatik, Philipps-Universität Marburg,
Universitätsstraße 24, 35032 Marburg, Germany
{steffen.kessler,alpar}@wiwi.uni-marburg.de
http://www.uni-marburg.de/fb02/is

Abstract. Most papers related to Open Source Software (OSS) discuss the development of OSS, licensing issues, and motivations of developers. Research in the area of customization of OSS is rare, however. The process after the deployment of an OSS within a company remains unknown. There is a danger that it is often unstructured and error-prone since OSS develops in a more complex way than proprietary software. Based on our literature study, modifications of open source code do occur also in organizations outside of the software industry. Customization of applications is more common than customization of infrastructure software in these organizations. Therefore, we examine the process of deployment and adaptation of an OSS application software over several update iterations in great detail. This examination shows that this process has similarities with the process of deployment of proprietary software but it also exhibits important differences. Based on this case study, we also suggest a process model for customization of OSS applications in user organizations.

Keywords: Open Source Software, adoption, customization, adaptation.

1 Introduction

Research on OSS development is often focused on the distributed development process of OSS. In the complex set-up of an OSS project, it is obviously important to know how the project is organized, how this organization evolves over time, who does what and why. For organizations which want to adopt a specific OSS not only the quality of the current version of the software is important but also the organization of the OSS project. Therefore, quality assessment methods have been suggested to support the selection among several OSS (e.g., Navica's as well as CapGemini's "Open Source Maturity Model" and the "Business Readiness Rating") [22, 6, 1]. The subsequent process of adaptations, which often already begins with the deployment of the OSS within the individual organization, has not attracted a comparable share of interest.

Basically, the deployment of OSS can be compared to the deployment of commercial of-the-shelf software (COTS). Unlike custom software, which is tailor-made according to the requirements gathered within the requirements analysis, OSS

C. Boldyreff et al. (Eds.): OSS 2009, IFIP AICT 299, pp. 129–142, 2009.

and COTS will need to be customized to fit the requirements of the adopting organization. Both types of software may include options for configuration and parameterization, but in case of changes that are beyond these two options, customization of the software code is necessary. In proprietary software, which code is usually solely available to the vendor, customizations are limited to the utilization of Application Programming Interfaces (APIs) provided by the vendor. In contrast, the availability of the source code of OSS allows anyone inside or outside a company to modify every aspect of the OSS. Unfortunately, there is not much research on the customization of OSS so companies do not have any good guidelines to follow. Expected efforts are not known, project costs are difficult to estimate, and project management becomes an art rather than a repeatable process.

This paper discusses the process of deployment of OSS followed by alternating and entangled internal adaptations and adoptions of new releases. The goal is to develop a proposal for a process model for the task of customization of OSS within a user organization. The organization can be a non-profit (governmental) organization or a company but for the sake of simplicity it will be referred to as "company" in the rest of the paper. Our examination of (small-size) surveys and individual case studies regarding the use of OSS confirms the interest of companies in customization of OSS. Adaptations of non-infrastructure OSS appear to have a higher probability of occurrence; as a result, we chose a non-infrastructure OSS for our research as shown in Figure 1.

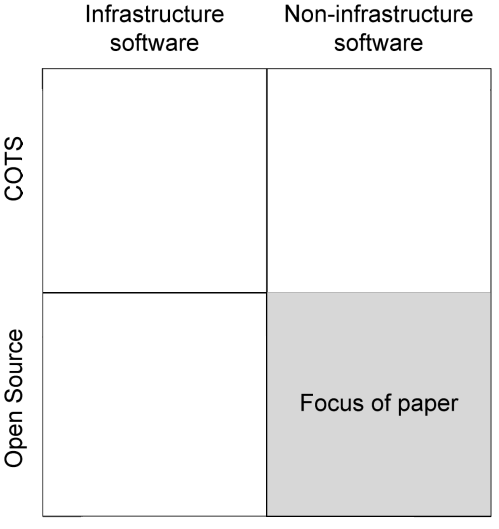

Fig. 1. Software classification

Data about the evolution of OSS within companies that do not play an active part in the OSS community is publicly not available. To collect such data individually from various companies is also very difficult due to the confidentiality of the projects and the time needed to observe the evolution process. Therefore, we chose to follow the paradigm of action research [3]. In action research, the researcher becomes part of the project team and can study the object of research in all necessary detail. The

project we studied belongs to the area of web engineering; projects in this area are a good starting point for research on OSS customization as they are often in a so-called perpetual beta state, i.e., there is no intention to reach a steady state. The projects are always in flux. Many products in this area are available as OSS; these projects often follow the "release early, release often" approach which has been made popular by OSS [10, pp. 19f.]. Releases often include bug fixes as well as new features; as a result, the continuous updates require a disciplined development, deployment, and support process [11].

In the next section, we provide some evidence that OSS customization within companies does occur. Our research approach and a short description of the case under analysis are given in section three. In section four, we develop a process model of the evolution of an OSS application within a company based on the generic system evolution process found in the literature and the results of our detailed case analysis of an actual OSS deployment and adaptation process in a company. Differences to the deployment and use of proprietary software are pointed out. Section five concludes the paper.

2 Relevance of Customization of OSS in Companies

According to studies on OSS adoption, the main motivation for the adoption of OSS is costs [e.g., 7, 27]. Nonetheless, in several small-size surveys as well as in some case studies companies revealed their interest in taking advantage of the adaptability of the open source code. As studies suggest, OSS adopters are reluctant to modify the source code of OSS in the infrastructure area [30, 5], e.g., source code of the operating systems Linux or Free/Open BSD. Wichmann [30] reports a comparably higher interest in the adaptation of non-infrastructure OSS. This caution may be caused by the importance of infrastructure systems as well as by the necessity for high programming skills in the infrastructure area. Personnel with such skills usually prefer to work for a software company rather than in a company that "just" uses software. These people may also prefer to contribute code to the (infrastructure) OSS project directly and participate in the peer reviews of OSS code in this way. Reasons for companies not to adapt the OSS source code are diverse, and may simply be based on the lack of need or a lack of resources as pointed out in a study by Madanmohan and De' [19].

We analyzed several OSS deployment projects which are documented as case studies, three in the academic area [18, 27, 9], one in the health care area [7] and one in the electronic business area [4]. All of them cover non-infrastructure OSS projects and mention cost as the main reason for OSS adoption. Although one case study claims that the availability of the open source code was not of interest, all case studies show that adaptations actually took place. Unfortunately, none of the case studies gave insights into the process of OSS customization.

The extent of changes may vary; OSS can also be used as the basis for software reuse, ranging from the reuse of a few lines of code to the reuse of a complete system. An unstructured reuse of code may, however, lead to differing evolution paths or a "fossilized code base" [29]. This emphasizes the importance of research on OSS customization. To decrease negative effects of a high number of changes in OSS

projects, an OS component with low code volatility may be preferred for software reuse [19]. Adaptations can be assigned to one of the three categories of maintenance: corrective, adaptive and perfective maintenance. In other words, they will not only address feature requests, but can also be needed to meet the companies' quality requirements during all stages of use. The choice for OSS is often based on costs. This cost consciousness is further supported by the ability to select the provider who gives the best offer for OSS adaptations. The organization can, for example, conduct a reverse auction to decrease the price. As controlling and benchmarking of software projects are still two of the current problems in IT controlling [17], increased knowledge of adaptation processes will improve the controllability of OSS adoption projects.

The relevance of customization of OSS for individual organizations may be decreased by restrictions imposed by the license of the particular OSS. Several OSS licences bind users to provide modified code to the public. If the organization is interested in gaining strategic advantages through individual modifications, it may consider only OSS which will allow adaptations to remain proprietary. Even though this behavior is contrary to the intentions of many OSS developers [24, pp. 67ff.], it is allowed by several OSS licenses, e.g., the BSD license type.

3 Case Analysis: Tools and Approach

3.1 Customization Case

To study all the details of the process of OSS customization, we use records and data gathered during the deployment and use of a special-purpose web-based OSS Content Management System (CMS) within an internet start-up company. The system is called Pligg [www.pligg.com]. The Pligg CMS project has been started in 2005. It provides a CMS suitable for provision of news items by end users which can be commented and voted for by other users. The news items are sorted by votes so that the most popular news appear on the "front page" while others appear on deeper waiting queues. Users receive points for their activity and the votes their news submissions receive. The figure calculated on this basis is often referred to as karma; it reflects partly the reputation of the user within that community. A number of further functions are available. The best-known site of this kind is probably Digg [www.digg.com] which is not built with Pligg and offers less functionality. The Pligg CMS is still quite popular with over 36000 registered users in its support forums. It has been used for numerous applications all over the world [e.g., http://www.dealigg.com, http://www.ecofuse.com, http://www.bestofindya.com/news], even by large companies, e.g., Intel [http://software.intel.com/sites/coolsw/]. The CMS instance we studied is available at the website http://www.colivia.de.

3.2 Documentation of Changes in OSS Projects

Many OSS projects use revision control systems (e.g., CVS, SVN) to keep track of code changes. While they are also used by developers of proprietary software they are especially suitable as a tool for OSS projects because they support distributed programming, the usual approach of OSS development [13]. Revision control systems

support projects with a large number of developers who are allowed to submit code to the version control system. Several other tools can be used in addition to or instead of version control systems; in some OSS projects, users and developers utilize bug reporting tools (e.g., Bugzilla) to submit source code changes, including updates, fixes, and sometimes even new features, rather than only bugs [16, 21]. This practice may result from the reluctance to use separate systems for the documentation of enhancements and bugs or from a different view on bugs. Such a view is clearly expressed by Raymond [24, p. 42]: "When your development mode is rapidly interactive, development and enhancement may become special cases of debugging – fixing 'bugs of omission' in the original capabilities or concept of the software". As a tool for discussion and indirectly documentation of code changes, communication tools like mailing lists and discussion forums are also used in OSS projects. All these tools are used while the code changes are developed or, at least, when they are submitted for use.

Another set of tools tries to determine code changes ex-post, i.e., by comparing the new code version with the old one. This can occur just after the new code was submitted but also many months after the submission. The ex-post change extraction approaches can be classified into lexical, syntactic, and semantic differencing techniques.

Lexical (text-based) change extraction approaches can be used for all kinds of code and projects. They do not distinguish between different types of changes, e.g., between changes of program comments and changes of the program code itself. Current approaches utilize information from the aforementioned version control systems; they store changes based on a text analysis without providing further information. Fluri et al. point to the lack of information regarding granularity, type, and significance level of changes [8]. As a result, advanced methods for change extraction have been developed. Syntactic and semantic differencing techniques [e.g., 20, 12] try to provide more information regarding changes. Several change extraction approaches depend on the programming language or they are limited to class-based programs in general. A number of approaches exist, for example, for class-based change extraction in Java projects [25].

3.3 Approach

Although advanced change distilling approaches exist, we chose a text-based approach as the most adequate method to derive changes because the CMS described in 3.1 is programmed in PHP and it is not completely class-based. We used a code differencing tool called CSDiff [www.componentsoftware.com/products/CSDiff/] to visualize changes in the source code ex-post. During our research, there were nine official releases of which eight were adopted for the internal release. The information in this case study is derived from the external code development by the OSS community as well as from the internal code adaptations.

To extract data from the project without risk of losing data, the initial official release was taken as the basis. Then, a code snapshot of the internal code base was created for the purpose of ex-post analysis every time before an update was initialized by a new official release. In addition to the ex-post documentation, the adaptations between the initial official release and the initial internal release were documented

regarding modified or added functionality. From the beginning both the internal code base including all adaptations done before the next new official update and the official update were archived. Feature and change requests were recorded separately.

During our examination, a "change" denoted a modification between two code files, in our case PHP code files. Changes were counted on a file basis; code changes leading to incompatibilities between the official and internal release were examined by the project team manually. Several code comparisons were needed to identify changes performed both in the external OSS community and internally. The first type of changes was derived by comparing code of each new official release by the OSS community with the old official release. Internal code changes were recorded by determining first differences between the old official release package and the internal code archive before the adoption of the new official release. Second, CSDiff was performed to determine changes between the new and old internal code releases. Since no adoption of complete new releases takes place, files with changes in the official release as well as in the internal code base needed to be updated manually.

Based on literature and the data collected during the study, we induce a model of the change process in an OSS customization project. We identified several instances which are different from the process of adaptations in proprietary software. Some of them will be presented in the next section.

4 Process of OSS Deployment and Use

4.1 Process of Deployment and Use of OSS

The unplanned use and modification of OSS can lead to problems. Saleck [26, p. 172] points to the following problems: unmanageable versions that have not been audited, simultaneous use of different software versions within the same company, and unnecessary repetition of tasks. OSS projects, therefore, need defined processes for software evolution. The software evolution process requires an appropriate configuration management, which includes version and change management [28, pp. 738f.]. OSS evolution is different from software evolution in COTS projects so the process model needs to reflect the additional possibilities of code modification within companies.

Adaptations of OSS in companies include the following cases:

- Adaptations based on changes of the configuration of the OSS product are similar to the deployment process in case of proprietary software; they will not be discussed in this paper.
- Adaptations based on the use of APIs add functionality using external modules. The adaptation process does not have to differ from the process used for APIs of proprietary software; this option does not necessarily benefit from the availability of the source code. Changes to internally developed modules will only be necessary in case of requirement changes and in case of changes to the API documentation. This type of adaptation relies mainly on the documentation of the API. The adoption of new external releases will, therefore, require an analysis of the API documentation as well as testing that can reveal errors in case of undocumented changes.

- Adaptations based on the modification of the actual source code of the OSS product are the type of adaptation that is different from proprietary software. The process model of OSS evolution within a company should merge the process of internal modifications with the process of the introduction of external changes.

Source code changes in a company lead to an internal code base (subsequently also referred to as "internal code tree") different from the code within the official release. OSS adaptation processes should help to minimize inefficiencies; the case of a fork, i.e., a split of an OSS project into two competing projects should be avoided because they take away resources from each other as described by Raymond [24, pp. 72f.]. Even though adaptations in a company may produce an internal code base which is called a pseudo fork, we assume that the internal code base will be kept as close as possible to the official code base to participate in the advantages of OSS evolution.

This paper focuses on adaptations of the source code in companies as we see it as the main difference in the software evolution of COTS and OSS. We use the system evolution process (Figure 2) proposed by Arthur [2] and amended by Sommerville [28, p. 540] as a basis for extensions specific to the OSS evolution process.

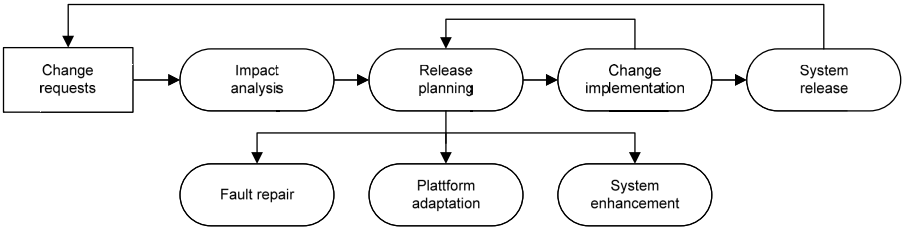

Fig. 2. The system evolution process [28, p. 540]

OSS evolution starts within a company with the deployment of the software. Immediately required changes are identified, their impact is analyzed, the change is planned and carried out resulting in a new system; change requests after deployment initialize the same steps [28, p. 540f.]. This generic process applies both to COTS and OSS.

Now popular approaches to IT service management such as the IT Infrastructure Library (ITIL) [http://www.itil-officialsite.com] and current versions of established system development models such as the V-Model, a model widely used in the German government sector, also address software adaptations as part of the software evolution. The generic process described above corresponds in parts to tasks in ITILv3 that are named there service transition [23, p. 151ff.]. The V-Model-XT includes information on reference roles, reference work products and reference activities [14]. However, in both cases the emphasis is on management rather than execution aspects.

Based on our action research of the Pligg CMS customization, we propose a distinction of two process chains in the evolution of OSS. As shown in Figure 3, both process chains use the generic system evolution process from Figure 2 as the basis, but they include different process steps to establish a new system release. This is

caused by differentiating internal change requests from new official releases as a special kind of change request. The right process chain is initiated by change requests within the company; the left process chain is necessitated by new official releases or the use of other external code, e.g., a bug fix available in the OSS support forums. Of course, a company may choose to ignore these but, if it wants to continue to benefit from the developer community's efforts in further improving the OSS, it will have to go through this process sooner or later.

The main difference between COTS evolution and OSS evolution in companies is caused by the above mentioned internal code tree used to document changes of the OSS source code. Even if all internal modifications are submitted to the public OSS project, the maintenance of the internal code tree cannot be omitted as modifications may not be (immediately) integrated into the next release. The internal code tree is required to keep track of changes; it includes internal bug fixes as well as feature enhancements or completely new code.

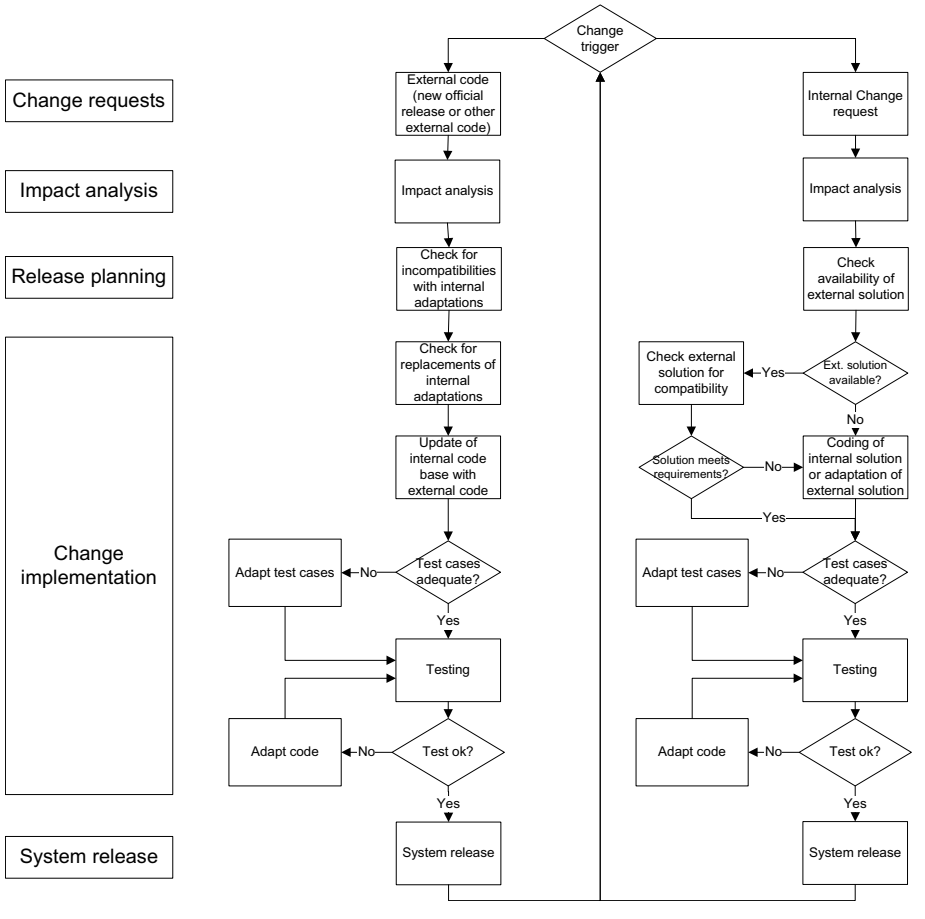

Fig. 3. Process model for OSS customization

External code which becomes available between two official releases in SVN, support forums or another form may be used for modifications of the internal code tree. In this case, the external code has to be treated as internal code until it is integrated into the official release and the official release is integrated into the internal code base.

To avoid negative impacts of pseudo forking, the internal OSS branch requires a complete documentation of all internal changes. In case of an official release, internal adaptations need to be reviewed and if possible substituted with official code to keep the deviation of the internal code base from the official code base as small as possible. To keep the code base in a similar coding style, internal modifications should follow the implicit or explicit coding styles used within the OSS project.

As the combination of internal and external code may cause conflicts between the code bases, testing is explicitly mentioned as an activity in the process. Testing in OSS projects may not be sufficient or may not cover all possible configurations of the OSS [21].

4.2 Internal Code Development

During the observation period of the discussed project, no feedback regarding internal adaptations was provided to the OSS community; accordingly, code which made internal adaptations obsolete was not the result of code submissions by the start-up to the OSS project. The official forum provided by the CMS project was used as an additional resource for external changes (bug fixes and features), which allowed us to study how much information from the forums was included in the official releases. The OSS community does not provide an additional mailing list in this project, which is a means for discussion and information in many other OSS projects.

Changes between updates were introduced in the forums or directly in the SVN code. SVN is used by the CMS development team, but they do not provide stable SVN versions in addition to the official releases made available on the project home page and moved the SVN location several times, which impedes an analysis of the SVN code. Therefore, we used the diff software CSDiff to extract the changes between two consecutive official releases.

Our data relies only on the official releases posted on the OSS project's website. With the data gathered during the iterations, we were able to analyze the impact of new official releases and the effect of changes based on internal requests on the internal code base. We included all changes (new features, bug fixes, etc.) and extended our analysis to record external changes that required changes to the internal code base. Figures for all iterations can be found in Table 1.

From the CMS deployment, every official update except one was incorporated in the internal code base. The number of changed files in official and internal versions did not "drift apart" during use; this is due to the fact that all of the official code was used as a basis for internal modifications. Even though no feedback was provided to the OSS community, several changes in the internal code tree got obsolete over time. Also, several incompatibilities between the internal and official code base were tracked down using text differencing between the codes of the releases.

The extent of reused code from an OSS project may be used as a guideline for the appropriate update frequency. Using large amounts of code increases the complexity

of tracking a bug in case of an incompatibility. Therefore, with a very high percentage of reused code, as in our case, the frequency of updates should be high. If the internal project only reuses small parts of code from the OSS, which could be the case when only individual functions or objects from the OSS project are used, the frequency of updates can be lower. Reusing OSS code in other software may cause dependencies that will necessitate very frequent updates and, therefore, may increase the overall risks of the project. With an increasing amount of (re)used code from different sources, the chance for a complete code audit decreases while the possible number of bugs increases; therefore, a high frequency of updates will help to take advantage of bug fixes delivered by the OSS community. This will lead to a higher quality of the internal code without the need for engagement of additional internal resources for bug fixing.

The starting point in the observed project was an initial release which was introduced as the first internal code base and then modified before the initial deployment. After the deployment, user feedback was used to further enhance the internal code base. If a bug was found or a feature request was made, external sources were checked first, in this case the support forums and the SVN repository. An internal fix or feature was developed only if no suitable code from the OSS community was found. This new code was kept in the internal release at least until a new official release became available. In case of a new official release, internal modifications were checked against the new external code changes.

Table 1. Changes in the internal and official releases during eight iterations

Release switch from ⟶ to	New Files (Ext.)	Mod Files (Ext.)	Del. Files (Ext.)	New Files (Int.)	Mod. Files (Int.)	Del. Files (Int.)	Official vs. Internal
0.90 ⟶ 0.95	100	99	100	106	99	101	3 incompatibilities 9 obsolete changes
0.95 ⟶ 0.96	1	35	0	1	35	0	1 obsolete change
0.96 ⟶ 0.97	0	18	0	3	23	0	3 obsolete changes
0.97 ⟶ 0.98	1	40	1	1	43	1	1 obsolete change
0.98 ⟶ 0.981	4	9	0	4	9	0	-
0.981 ⟶ 0.982	0	11	0	0	11	0	-
0.982 ⟶ 0.99	36	61	9	38	65	11	6 incompatibilities
0.99 ⟶ 0.995	10	34	1	10	34	1	-

4.3 Code Conflicts

Throughout the update iterations of the CMS we studied, several conflicts caused by incompatibilities of internal changes with changes in a new official release occurred. These conflicts had to be resolved by modifications to the internal code if the official release was to be adopted. Any type of maintenance, corrective, adaptive or perfective, could cause incompatibilities.

Other tasks may be required after an external release has been adopted in addition to code changes; this may include new or modified test cases as well as changes of the documentation.

We identified several specific situations that may occur when a company uses external code to update internal releases. First, internal modifications may need to be kept despite the adoption of a new official release because the reason for the internal modification still persists. Second, internal modifications may have to be altered to avoid incompatibilities. Third, internal modifications may become completely obsolete as they have become part of the official release. Finally, test cases or internal documentation may have to be modified after an update.

Two types of changes will be discussed here in detail: known bug fixes that are not yet part of the official release and disruptive code changes.

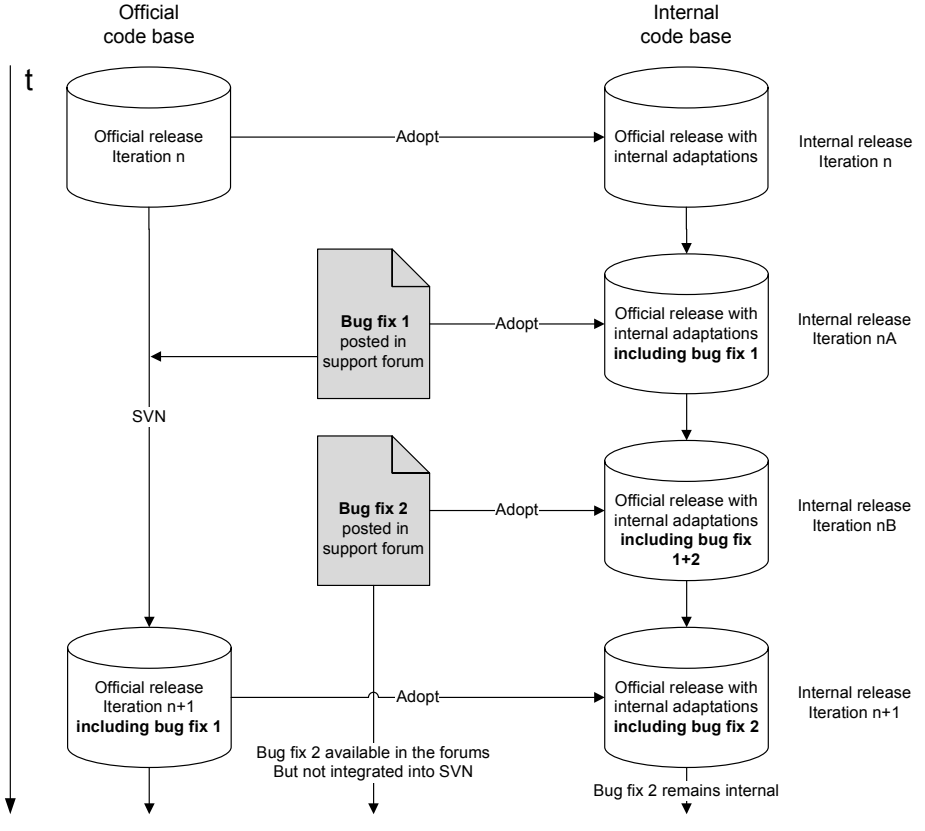

Fig. 4. Known bug fixes not yet part of the official release

Figure 4 shows the case of a modification, specifically a bug fix, available through the support forums which is not integrated into several new official releases.

This situation occurred several times. In one case the available bug fix did not enter the latest release during the observation period (version 0.99) although it was published in the support forum several releases before the latest one. Such bug fixes need to be handled as internal code until they become part of the official release although their origin is external; this includes a check for compatibility during the internalization of the following official releases. Also, they may be the cause for incompatibilities between the internal and official release during the next update. This also shows that a distinction between internal and external code is not sufficient in the internal documentation. It also must be documented if there is external code that is not part of the official code.

We witnessed during the observation period also a case of a potential drastic change. The core developers revealed the plan to completely recode the OSS. This situation is depicted in Figure 5.

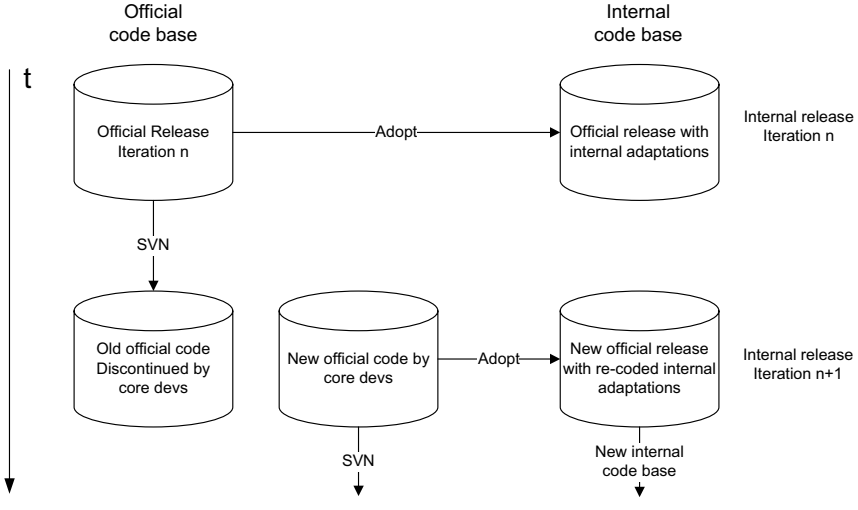

Fig. 5. Disruptive code change

This change was planned by the CMS core team in order to switch to a new licensing model. They announced that the next new version would be a complete rewrite with a new code base. Users will have to decide whether to use the new version or continue to use the old version and eventually lose support. Such a disruptive code change may in part be compared to a new version of proprietary software which is incompatible with extensions by third parties. If customizations of COTS were undertaken based on APIs, this code may still work with a new version even after a major code change; adaptations of the source code, however, will very likely be lost after such a disruptive change. Users of COTS will probably not like to remain without vendor support for a long time. Users of OSS will be able to continue with the old version as they have the source code. This only bears the danger that the support by the OSS community decreases but paid-for third party support remains an option that would not be there without the source code.

5 Conclusions and Outlook

Most discussions of OSS use in companies concentrate on costs based on the fact that OSS is usually free to use. The other important characteristic of OSS, the possibility to modify the source code in order to custom tailor it to the organisation's needs, is seldom discussed. While research on the maintenance process in OSS projects is ongoing [15], only a few papers describe OSS customization in companies. Therefore, we studied the process through action research. The resulting process model can be used by companies as a reference for their OSS adoption and use. It can also be considered a more specific guideline for change and release management within a comprehensive IT service management framework when OSS is used.

Organisations that only adopt OSS for cost reasons without any intention to change the source code can still use the process model. For them, only the left branch of the process model in Figure 3 is relevant. We can conclude based on our study that when a company customizes OSS, an internal change and release management is essential to avoid chaos. The internal change management has to distinguish between three types of code: internal code, external code that is not part of the official release and official release code.

The results from our research need to be validated with data from other projects; unfortunately, data from internal adaptations of OSS are rare. Even though we achieved simple traceability with textual differencing tools, approaches for fine-grained source code change extraction as presented in [25] could lead to a faster and more detailed analysis of OSS evolution if adequate data is available.

References

1. Aberdour, M.: Achieving Quality in Open Source Software. IEEE Software 24(1), 58–64 (2007)
2. Arthur, L.J.: Software Evolution. John Wiley & Sons, New York (1988)
3. Avison, D., Lau, F., Myers, M., Nielsen, P.A.: Action Research. Communications of the ACM 42(1), 94–97 (1999)
4. D'Agostino, D.: Case study: Backcountry.com. CIO insight 05.2006:44-54 (2006)
5. Dedrick, J., West, J.: An Exploratory Study into Open Source Platform Adoption. In: Proc. 37th Annual HICSS, Hawaii, USA, Track 8 (2004)
6. Duijnhouwer, F.-W., Widdows, C.: Open Source Maturity Model. Cap Gemini Expert Letter (August 2003), http://kb.cospa-project.org/retrieve/1097/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.31.pdf (accessed February 29, 2008)
7. Fitzgerald, B., Kenny, T.: Open Source Software can Improve the Health of the Bank Balance - The Beaumont Hospital Experience (2003), http://pascal.case.unibz.it/retrieve/2758/fitzgeraldkenny.pdf (accessed October 27, 2007)
8. Fluri, B., Würsch, M., Pinzger, M., Gall, H.C.: Change Distilling; Tree Differencing for Fine-Grained Source Code Change Extraction. IEEE Transactions of Software Engineering 33(11), 725–743 (2007)
9. Fox, L., Plummer, S.: Opening the Lines of Communications with Open Source Software. In: Proc. 34th annual ACM SIGUCCS conference on User services, Edmonton, Alberta, Canada, pp. 114–117 (2006)

10. Golden, B.: Succeeding with Open Source. Addison-Wesley, Boston (2005)
11. Hoyer, V., Schroth, C., Stanoevska-Slabeva, K., Janner, T.: Web 2.0-Entwicklung – ewige Beta-Version. HMD - Praxis der Wirtschaftsinformatik 255, 78–87 (2007)
12. Jackson, D., Ladd, D.A.: Semantic Diff: A Tool for Summarizing the Effects of Modifications. In: Proc. International Conference on Software Maintenance, Victoria, BC, Canada, pp. 243–252 (1994)
13. Jansen, S., Brinkkemper, S.: Definition and Validation of the Key Process Areas of Release, Delivery and Deployment for Product Software Vendors: turning the ugly duckling into a swan. Technical report UU-CS-2005-041, Institute of Information and Computing Sciences, Utrecht University, Netherlands (2005)
14. Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (n.d.). V-Model-XT, Berlin, Germany, http://v-modell.iabg.de/dmdocuments/V-Modell-XT-Complete-1.2.1.1-english.pdf (accessed October 25, 2008)
15. Koponen, T., Hotti, V.: Open Source Software Maintenance Process Framework. In: Proc. Fifth Workshop on Open Source Software Engineering, St. Louis, MO, USA, pp. 30–34 (2005)
16. Koru, A.G., Tian, J.: Defect Handling in Medium and Large Open Source Projects. IEEE Software 21(4), 54–61 (2004)
17. Kütz, M.: Grundelemente des IT-Controllings. HMD - Praxis der Wirtschaftsinformatik 254, 6–15 (2007)
18. Lovett, J.: Open Source-A Practical Solution. In: Proc. 35th annual ACM SIGUCCS conference on User services, Orlando, Florida, USA, pp. 221–223 (2007)
19. Madanmohan, T.R., De', R.: Open Source Reuse in Commercial Firms. IEEE Software 21(6), 62–69 (2004)
20. Maletic, J.I., Collard, M.L.: Supporting Source Code Difference Analysis. In: Proc. 20th IEEE International Conference on Software Maintenance, Chicago, Illinois, USA, pp. 210–219 (2004)
21. Michlmayr, M., Hunt, F., Probert, D.: Quality Practices and Problems in Free Software Projects. In: Proc. First International Conference on Open Source Systems, Genova, Italy, pp. 24–28 (2005)
22. Navica (n.d.) Open Source Maturity Model (OSMM), http://www.navicasoft.com/pages/osmm.htm (accessed March 10, 2008)
23. Olbrich, A.: ITIL kompakt und verständlich, 4th edn. Vieweg+Teubner, Wiesbaden (2008)
24. Raymond, E.S.: The cathedral & the bazaar. Revised edn. O'Reilly, CA (2001)
25. Sager, T., Bernstein, A., Pinzger, M., Kiefer, C.: Detecting Similar Java Classes Using Tree Algorithms. In: Proc. 2006 International Workshop on Mining Software Repositories, Shanghai, China, pp. 65–71 (2006)
26. Saleck, T.: Chefsache Open Source, 1st edn. Friedr. Vieweg & Sohn Verlag, Wiesbaden (2005)
27. Sinnet, C.J., Bar, T.: OSU Helpdesk: A Cost-Effective Helpdesk Solution for Everyone. In: Proc. 32nd annual ACM SIGUCCS conference on User Services, Baltimore, Maryland, USA, pp. 209–216 (2004)
28. Sommerville, I.: Software Engineering, 8th edn. Pearson Studium, München (2007)
29. Spinellis, D., Szyperski, C.: How Is Open Source Affecting Software Development? IEEE Software 21(1), 28–33 (2004)
30. Wichmann, T.: Free/Libre Open Source Software: Survey and Study. Berlecon Research GmbH, Berlin, Germany (2002), http://www.berlecon.de/studien/downloads/200207FLOSS_Use.pdf (accessed October 15, 2007)