VOL. 2, NO. 11, November 2012 ISSN 2222-9833
ARPN Journal of Systems and Software
©2009-2012 AJSS Journal. All rights reserved

http://www.scientific-journals.org

Lifecycle Management of Open-Source Software in the Public Sector A Model for Community-Based Application Evolution

[1]Jukka Kääriäinen, [1]Pasi Pussinen, [1]Tapio Matinmikko and [4]Tommi Oikarinen
[1,2,3] VTT – Technical Research Centre of Finland, Oulu, Finland
[4] Finnish Ministry of Finance, Public Sector ICT, Helsinki, Finland
Email :{ [1]jukka.kaariainen@vtt.fi, [2] pasi.pussinen@vtt.fi, [3] tapio.matinmikko@vtt.fi, [4] tommi.oikarinen@vm.fi}

ABSTRACT

Similarities in public sector organizations allow the reuse of SW products using open-source (OS) like characteristics. This enables cost savings as organizations can reuse and tailor SW products developed by other organizations free of charge. However, without coordination SW products will evolve in an uncontrolled manner. This article argues the importance of application lifecycle management during the evolution of SW products in the public sector's open-source communities. The state-of-the-art and state-of-the-practice are well established in open-source development and SW product lifecycle management but they need to be applied for public sector. This article defines models for the lifecycle management of open-source like SW products in the public sector. The models have been developed in cooperation with OS specialists, lifecycle-management specialists and a number of public sector organizations. The results and experiences of this research are interesting to the public sector in other countries that are considering open source as a way to increase the cost efficiency of the information systems development. This article is also targeted at IT research organizations and software companies that operate with public sector organizations.

Keywords: *Lifecycle management, open source, public sector, information system, software configuration management*

1. INTRODUCTION

Currently Open-Source Software (OSS) is increasingly used in the private and public sector in Finland (e.g. [1]). Similarities in public sector organizations allow the reuse of SW products using open-source like characteristics. This enables cost savings as organizations can reuse and tailor SW products developed by other organizations free of charge. For instance, Finnish municipalities have developed SW products for the public sector in cooperation with a special working group from the Ministry of Finance (Public Sector ICT). The intention was that SW products are licensed so that the source code remains open, are freely distributed and are together further developed by the public sector community so that SW products will serve Finnish public sector organizations in a cost-efficient way. However, without coordination these SW products will evolve in an uncontrolled manner and they may not evolve in the direction that best serves the needs of public sector organizations. To enable this coordination, the Ministry of Finance has started a research effort to define models for the management of open-source like SW products' evolution in the public sector (i.e. lifecycle management). The Ministry of Finance, VTT and a number of public sector organizations have collaborated in defining these models.

The purpose of this article is to describe the models and discuss in more detail the model that was identified as having the most potential for piloting SW product lifecycle management in the public sector in future. Furthermore, the article aims to encourage scientific and professional discussion about the community-based management of open-source software in public sector.

This paper is structured as follows. First, the background of open-source software and SW product lifecycle management will be presented. Next, the paper will present the research process describing how the models were developed. Then the models will be introduced and the most advanced model (Model 3) will be presented in more detail. Model 3 will then be discussed and, finally, conclusions from the research will be drawn.

2. BACKGROUND

The traditional prevailing practice for software acquisition in the public sector is simply purchasing software from different vendors. Each city and municipality does their own purchasing with either local or international software vendors. As the public sector is a large customer for software, with multiple organizations each with special requirements for software (i.e. legislative requirements), the vendors usually need to customize software specifically for public organizations. The price of the software and customizations is paid by the public sectors actors, who use public funds (i.e. tax payer's money). Once software has been customized for one customer, vendors usually sell the same software to other cities or public organizations in the country with full price. This leads to a situation, where public sector organizations buy the same software with public funds multiple times. This happens because vendors retain the source code and the developed customizations for themselves, therefore being able to sell the same thing over and over again. Recently, public sector in Finland has started efforts to license SW products so that the source code would remain open and re-distributable instead of remaining proprietary or closed. To further support the development of this kind of software, an open source community based approach could be applicable. However, to function more as a community that develops the software according to the member's needs (much like open source communities do), the community needs practices and guidelines on how to govern the software and it's lifecycle. From the point of public sector, there would be reason to develop governance models for the public sector to ensure efficient use of public funds and lifecycle of software made for public sector organizations.

This section presents the introduction for open-source software and SW product lifecycle management that are the background for the new operational mode discussed above.

2.1 Open-source software

Open-source software (OSS) has gained a lot of attention from academics, both from business studies, (e.g. [2, 3, 4, 5] and software engineering aspects (e.g. [6]). The term "open source" (OS) encompasses both the intellectual property strategy and the development methodology for open-source software (OSS) [7]. Community-driven OSS has also gained a market share in computer software, such as operating systems (with the Linux operating system), professional software (MySQL, Apache, Alfresco) and desktop software like web browsers (Firefox, Chromium), office suites (Libre Office) and graphics software (GIMP). In these market segments and others, OSS competes with proprietary software as a free alternative. Although OSS can be acquired, usually with a lower price or no price at all, the costs of using and adopting OSS are not necessarily lower than those of proprietary software.

OSS is developed within communities that consist of individuals (volunteers) and/or company members (or individuals from participating organizations). OSS development projects can be categorized into two sub-categories: community-founded projects, launched by individuals, and sponsored projects, launched by commercial companies, government agencies or non-profit-organizations [7]. Communities or repositories for public sector OS software are, for instance, at European level *Joinup* (*joinup.eu*) and in Finland *Yhteentoimivuus* (*yhteentoimivuus.fi*). *Joinup* is meant for sharing and reusing open-source software, semantic assets and other inter-operability solutions for public administrations. *Yhteentoimivuus* is a delivery channel for public sector interoperability assets administrated by the Finnish Ministry of Finance.

Information and support are spread among the people in the community and more importantly; innovations are shared within the community [8]. The community-based software production and innovation method has also provided business opportunities, as companies have successfully formed business models around OSS. This implies a shift from owning resources to coordinating resources [8]. Furthermore, Dahlander and Magnusson [8] argue that, "*to benefit from the external resources of knowledge, firms must identify where the knowledge resides and how it can be captured and subsequently used*". It is noticeable that acting in an OS community can be something other than just producing code; examples of different roles are organizing workshops and conferences, and participating in political and marketing campaigns that further the interests of the OSS movement [9]. Indeed, the OSS movement has enjoyed PR-activities organized by large ICT companies, like IBM [10].

The community-driven development method has also received some interest from organizations to be used as an internal development method [11], Wesselius provides a case where an internal open source (ISS) method is used in an organization, with some success. The business drivers of normal open-source software are however missing in this kind of activity, hence adopting an OSS community inside an organization is not a straight forward process.

2.2 Software product lifecycle management

The discipline that keeps the evolving software product under control is called Software Configuration Management (SCM). It is a well-known concept in software engineering and it has been widely discussed in literature and articles, for example in [12, 13, 14, 15]. Estublier et al. [16] present SCM as one of the software engineering domains in which research has impacted successfully. SCM enables that:

- Changes to the software are coordinated, enabling the software to evolve in a direction that benefits users.
- The quality of the software remains good due to the reduction of human errors that relate to missing versioning conventions, responsibilities and common processes.
- Up-to-date, adequate information about the evolving software product is available for the right people, at the right time and through the right communication channel.

The importance of SCM is emphasized in the development and maintenance of critical software where the malfunction of the software may lead, for instance, to personal injury, financial losses, security threats or environmental damage. Jonassen Hass [13] argues that it is difficult to provide unambiguous rules for CM for different classes of products, but the need for formality and automation increases with the level of criticality. For example, in a safety-critical product that may cause many people to be killed, the need for a definite and careful CM is high. Eskeli et al. [17] maintains that the customers of business critical software products need the confidence that the software they have acquired has a clear lifecycle, support and maintenance for years. The configuration management issues of open-source software have been studied, e.g. in [18, 19, 20, 21]. Asklund and Bendix [18] examine configuration management practices in open-source software projects and discuss best practices and how lessons learned from open-source software can be transferred to more traditional ways of developing software and vice versa. Erenkrantz [19] and Michlmayr et al. [20] have studied release management practices and problems countered in open-source projects. JHS 169 [21] gives recommendations for the use of open-source software in the public sector (published by JUHTA; the

Finnish public sector's IT-negotiation group). The recommendation discusses several issues concerning open-source software that need to be taken into account when public sector organizations plan to move towards the use of open-source software. This discussion includes the considerations related to configuration management issues.

In the past few years, the concept of Application Lifecycle Management (ALM) has emerged to indicate the coordination of activities and the management of artifacts (e.g. requirements, source code, test cases, and releases) during the lifecycle of software products. ALM as a concept is quite new [22] and has mostly been discussed in professional articles, e.g. [23], [24], and [25]. Murta et al. [26] and Schwaber [27] present SCM tools as the usual foundations of ALM infrastructures. Chappell [25] present ALM as the combination of three aspects: governance, development, and operations (Figure 1). Rossberg [28] found that people often equated ALM with operations and maintenance activities, without a development phase. However, the whole lifecycle contains activities related to the development of the first version of the software product, as well as its operation, and upgrades during the software product's lifecycle. According to Chappell [25] "*development occurs in the first part of an application's lifecycle, then happens periodically as the application is updated. Each iteration would contain some requirement definition, some design, some development, and some testing*". Therefore software products evolve over time towards the needs of the business as coordinated by a governance function. The governance function controls how a software product evolves based on bug-fixes and feature changes (i.e. software product management). The role of product management is to work as a coordinator between marketing needs and requests for R&D capabilities in order to develop the products within defined timelines, and budget and quality requirements [29]. Product managers gather product ideas and transform them into product features that solidify the ideas, and SW projects are responsible for realizing these features in accordance with the schedules. This process of transforming customer ideas into achievable features, and assigning them to practical development projects, is essential for effective product development and can also be applied in the context of open-source software.

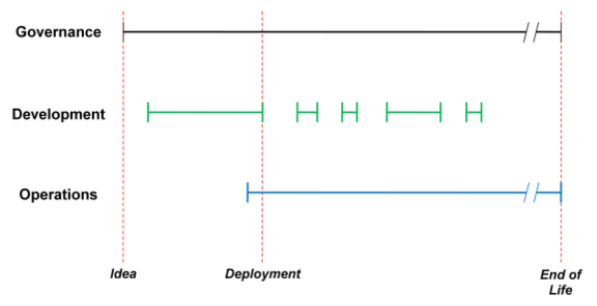

Figure 1: ALM aspects [25].

The concept of software evolution has been discussed, for instance, in [30, 31, 32, 33]. Rajlich and Bennett [30] extend the discussion of traditional software maintenance towards a software evolution model where maintenance is a series of distinct stages (Figure 2). Their versioned staged model presents a series of evolution versions. *Initial development* corresponds to the *Idea-Deployment*-phase in Chappell's model. Evolution corresponds to SW product updates that happen between the *Deployment-End-of-Life*-phase in Chappell's model. The versioned staged model leads to a situation where the software product has one major evolution path where earlier versions are supported only using *servicing* that represents bug-fixes and minor functional changes. Rajlich and Bennett [30] argue that the development of evolution versions requires strong competence (domain, structure of software) that has been accumulated during the development of the initial version since evolution changes tend to be comprehensive, affect the structure of the software. They continue that expertise is less important during servicing because big software changes are not allowed (e.g. limited to inputs and outputs, with the software considered a black box). Therefore, the competence of the staff that takes care of the implementation of evolution versions is more important and servicing can even be outsourced [30]. Capiluppi et al. [31] have studied the staged model from an open-source software evolution viewpoint. They conclude that even though OS development brings some special features to the model, the building blocks of the model are also capable of presenting the evolution of OS software.

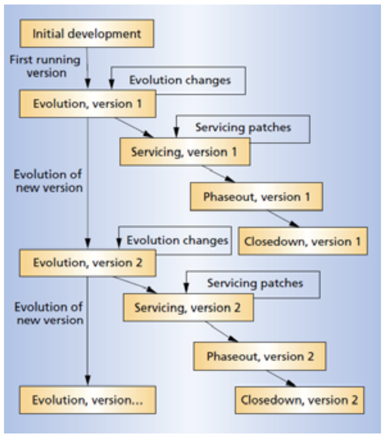

Figure 2: Versioned staged model [30].

3. RESEARCH PROCESS

The models presented in this paper are based on background study, discussions with OS- and lifecycle-management specialists (expert opinions) and workshops arranged in cooperation with several organizations that operate in the field of public sector IT systems. The background study was based on scientific literature, public sector recommendations, professional articles and white papers.

VOL. 2, NO. 11, November 2012 ISSN 2222-9833
ARPN Journal of Systems and Software
©2009-2012 AJSS Journal. All rights reserved

http://www.scientific-journals.org

This research consisted of four workshops that were arranged in Helsinki in the facilities of the Finnish Ministry of Finance. The model requirements and models were presented and feedback collected iteratively in these workshops. The workshop participants represent the following organizations:

- Association of Finnish Local and Regional Authorities (Suomen Kuntaliitto)
- City of Lahti
- Finnish Centre for Open Systems and Solutions, COSS
- Finnish Ministry of Finance (Public Sector ICT)
- Finnish Ministry of the Environment
- Finnish State Treasury, Government IT Shared Service Centre
- HH Partners
- Kuntien Tiera Oy (IT service center)
- VTT, Technical Research Centre of Finland

The results were reviewed and accepted by the representative of the Finnish Ministry of Finance. Furthermore, the models have been presented in several public sector IT seminars and discussed with several software companies operating in Finnish public sector IT projects. The following background assumptions for the models were agreed in public sector workshops:

- There are similarities in public sector organizations that allow the reuse of SW products.
- The licensing model can be defined so that it enables the SW product's distribution and reuse.
- There is commitment towards the cooperation and SW products' reuse among public sector organizations.
- SW product that is developed for one public sector organization can be tailored for the use of other public sector organization and, therefore, cost-savings are possible.

4. COMMUNITY-BASED APPLICATION LIFECYCLE MANAGEMENT MODEL

The next sub-section presents the overall description of the models that were developed in this project. The most extensive model, known as the "*community-based application lifecycle management model*" will be introduced in more detail.

4.1 Models for application lifecycle management in the public sector

The starting point for model creation was that some public sector organizations started to develop SW products (with Public Sector ICT) that will be licensed so that the source code would remain open and re-distributable instead of remaining proprietary or closed. However, this needs practices and guidelines on how to govern the software and it's lifecycle. This project proposed models that could be used for managing the SW products and their evolution. Figure 3 presents the big-picture of the models developed in this project. The figure shows that the latter models cumulatively increase management aspects over the former models. Therefore, the ALM features of former models are included in the latter model with more elaborate ALM features that gradually increase the formality of how the evolution of the SW product is controlled. The applicability of the models depends on management needs. Reusable SW products that are used in public sector to business-critical activities definitely need stricter ALM models than those that are just SW prototypes made to understand the feasibility of certain technologies or concepts. We do not claim that certain model is better than another, but the models should be applied based on the needs of the public sector (e.g. the type of SW product that will be subject for application lifecycle management).

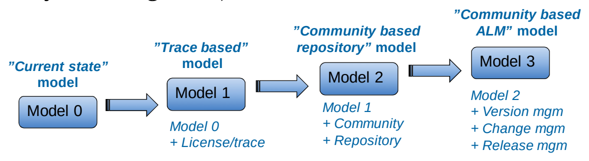

Figure 3: Models for application lifecycle management in the public sector.

- **Current state model (Model 0):** for the ad-hoc reuse of SW products among public sector organizations. No control or visibility about what SW products are available for reuse, who has reused the SW products or how the products evolve.
- **Trace based model (Model 1):** SW product will be provided to other public sector organizations if requested. Permission will be asked and information about the reuse will be provided for the financier of the first version. This controls who has reused the SW product and the financier can provide information about the SW product reuse for public sector organizations.
- **Community-based repository model (Model 2):** Financier and public sector organizations form the community. The community has a repository that can be used for storing, seeking, discussing and distributing reusable SW products. The community has documentation practices for reusable SW products. The reused SW product, with organization specific modifications, needs to be registered back to the repository for further reuse.
- **Community-based ALM model (Model 3):** The community has a repository where the baseline version of the SW product is stored. This leads to the linear versioning model where the baseline product evolves through changes that the community has collectively agreed to according to version, change and release management practices. This model requires a "*product manager*" who coordinates the SW product's evolution as mandated by the community.

4.2 The community-based application lifecycle management model for the public sector

In this sub-section, Model 3 (the community-based application lifecycle management model) will be discussed in more detail. The model is depicted and explained in Figure 4 and Table 1. In Model 3 public sector organizations and the financier form a community that has the highest authority for SW product-related decisions (much like the OSS communities). The product manager has a mandate to coordinate the SW product-related lifecycle management activities. The mandate is given by the community. The community manages SW product according to a "*SW product lifecycle management plan*" approved by the financier of the first version of the SW product.

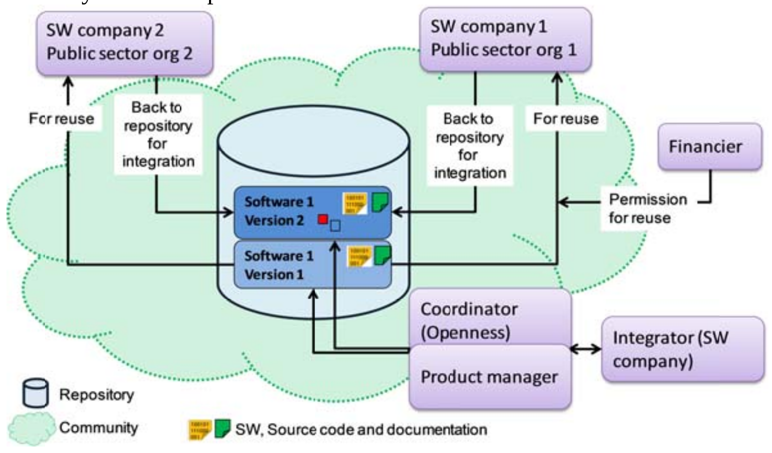

Figure 4: The community-based application lifecycle management model.

Table 1: Different aspects of Model 3.

Aspect	**Description**
Ownership	The community owns the SW product.
Lifecycle mgmt practices	The community manages the SW product according to the "*SW product lifecycle management plan*" approved by the financier of the first version of the SW product. The approval should happen before the invitation for tenders since the implementer of first version needs the ALM requirements for their tender (e.g. documentation requirements). The plan will define who will do what and when related to the lifecycle management activities. For example the SW product documentation requirements, versioning model, change management practices, release management practices and financing of lifecycle management activities, etc will be defined in this plan.
License	SW product versions are available for public sector organizations with the permission of the financier of first version. The SW product has a license agreement defined that prevents anyone from stealing the software from the community.
Roles	**Financier:** funds the first version of the SW product. Initiates lifecycle management responsibilities and activities. Operates in the community. **Public sector organization:** uses the SW product. Operates in the community. **SW company (public sector organization's SW supplier):** tailors, deploys and maintains the SW product for a public sector organization as ordered by the public sector organization. **Product manager:** coordinates the SW product-related lifecycle management activities so that the SW product evolves towards the direction that serves the needs of the public sector organizations that belong to the community. Has the mandate for the operation from the community. **Coordinator (Openness):** coordinates the operation in the community. Defines and checks the license information when the SW product version is acquired and received from the integrator. **Integrator (SW company):** develops a new SW product version (baseline version) according to an assignment from the community. At the beginning this will be the developer of the first version.
Financing	Community has a financing model defined for a SW product lifecycle management. For example: ***Coordinator (Openness):*** *Annual fee that the community members will pay (equal distribution).* ***New SW product version (baseline version):*** *New version will be financed by the community members (equal distribution).*

	Bug fix version: *New version will be financed by the community members if not under warranty (equal distribution).* ***Organization specific tailoring, deployment and maintenance:*** *Each organization takes care of its own expenses.* ***Working in the community (e.g. community meetings):*** *Each organization contributes its own expenses to the community and its meetings.*
Role of the community	The community consists of public sector organizations and financiers. Has the highest authority for all decisions concerning the SW product.
Repository	The community has a repository for storing, versioning, discussing and delivering a SW product. The community agrees what documentation will be delivered with the SW product version and what information (metadata) will be visible and accessible in the repository.
Reuse, delivery	Public sector organizations may reuse SW product versions with the permission of the financier. They are obliged to inform what kind of changes they have made to the software (SW code changes should be provided if requested). These changes will be integrated into the baseline version if seen a feasible by the community.
Version mgmt, change mgmt, release mgmt	The community has version, change and release management practices and responsibilities. All changes, versioning and releasing will be done according to agreed practices and responsibilities. For example: *The SW product has a time-based release strategy (one major release per year). Community members can propose changes and new functionality for each SW product version. The product manager prepares a change proposal for the next release (major version (major release)) and presents that to the community in the community meeting. The community makes the decision about the content of the next release. The product manager and coordinator prepare the order for the next release and communicate that to the integrator. The integrator delivers the SW product version package with the agreed documentation to the product manager and coordinator. The product manager and coordinator will audit the package and documentation. The new version will be presented to the community that makes the decision about the acceptance of the delivery.* *In a case of an emergency change, the product manager communicates with the community and triggers a bug-fix change (minor version (minor release)) if it cannot wait for the next major release.*

5. DISCUSSION

As already mentioned before, we do not claim that certain model is better than another, but the models should be applied based on the needs of the public sector (e.g. the type of SW product that will be subject for application lifecycle management). The selection of the model should be done as a part of the financing process. The financer (e.g. Public Sector ICT) of the first version of the SW product needs to think about what kind of lifecycle management the upcoming SW product might need. This is important since the selection of model affects to the tendering process (e.g. SW product's documentation requirements in Model 3 are stricter than in Model 1). However, this should not prevent the flexibility. For instance, SW product originally meant as an example implementation (using e.g. Model 2 for management) might later on become a recommended implementation in public sector and thus requires stricter management model (transition to the use of Model 3).

Public sector's workshop participants estimated that Model 3 is the most potential model for further research, since it enables coordinated lifecycle for SW products. It was also identified and agreed that there are upcoming public sector SW products that are suitable for piloting the Model 3. Next in this section, Model 3 will be compared with the models of Chappell [25] and Rajlich & Bennett [30] and the benefits and challenges of the model will be discussed.

When compared (Figure 5) with the versioned staged model [30] it can be noted that there are some similarities, even though, the concepts of evolution and servicing need to be interpreted into this context. The evolution version corresponds to the baseline versions that are stored into the repository. The development of the baseline version is coordinated by the product manager, coordinator (openness) and community. The development of a baseline version is realized by the integrator. In this context servicing versions correspond to organization specific versions that are tailored and maintained by SW companies as ordered by the public sector organizations. There are also similarities related to competence aspects. According to Rajlich and Bennett [30] the staff's expertise is crucial in the development of evolution versions. They state that during evolution it is important to "*keep the original team together*". On the other hand, with regard to servicing, the competence is less crucial and Rajlich & Bennett [30] argue that "*servicing can therefore be outsourced*". Model 3 has a similar approach. At the beginning it is feasible to select the developer of the first version as the integrator. Other SW companies that tailor and deploy the baseline version for their customers (i.e. public sector organizations) become the natural maintainers of customer-specific versions. On the other hand, unlike the versioned staged model, Model 3 proposes that later on the company working as the integrator could be replaced because other SW companies have also gained competence regarding the baseline version. This will prevent vendor lock-in in the integrator role but is challenging since knowledge transfer is not easy.

When mapping the aspects of ALM [25] with Model 3 from the baseline version point of view it can be argued that all ALM aspects can be found from Model 3 (Figure 6). From baseline version point of view, the community, product manager and coordinator (openness) take care of governance functions. The development of baseline versions will be done by the integrator as guided by the governance function. On the other hand, other SW companies that tailor, deploy and maintain open SW product versions for their customers (i.e. public sector organizations) will handle the operations function for their customers and also provide feedback about the operation of the SW product to the governance function. However, this comparison is problematic since there are two interlinked types of lifecycles: the baseline version's lifecycle (evolution) and the customer-specific versions' lifecycles.

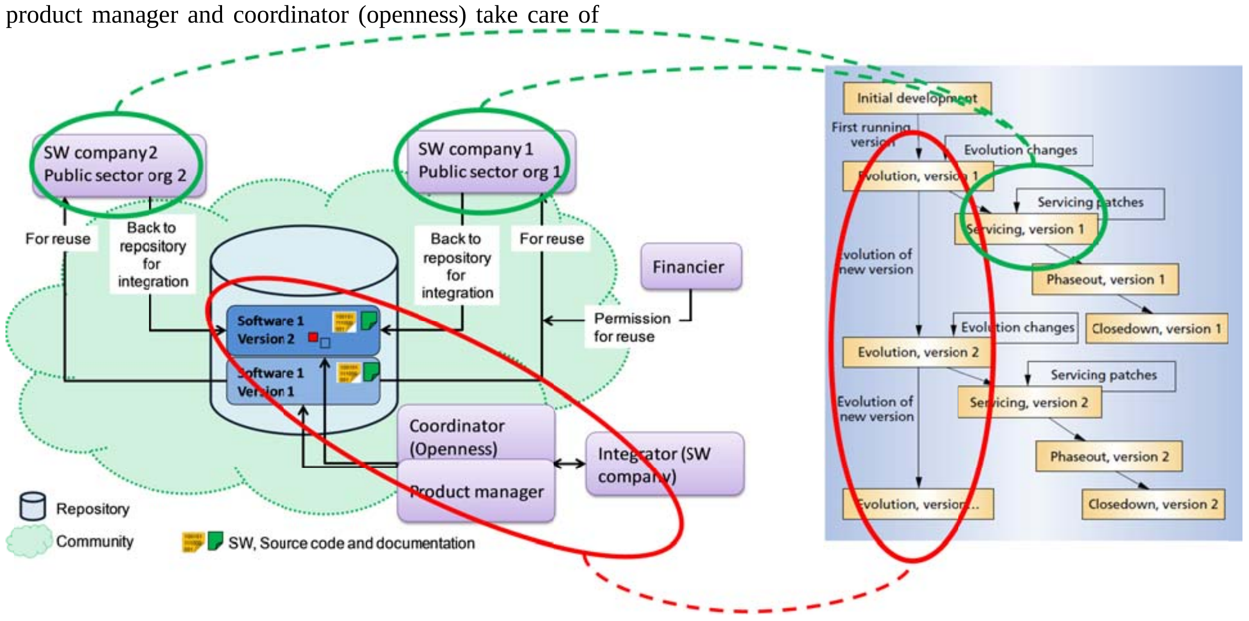

Figure 5: Comparing Model 3 and the model from [30].

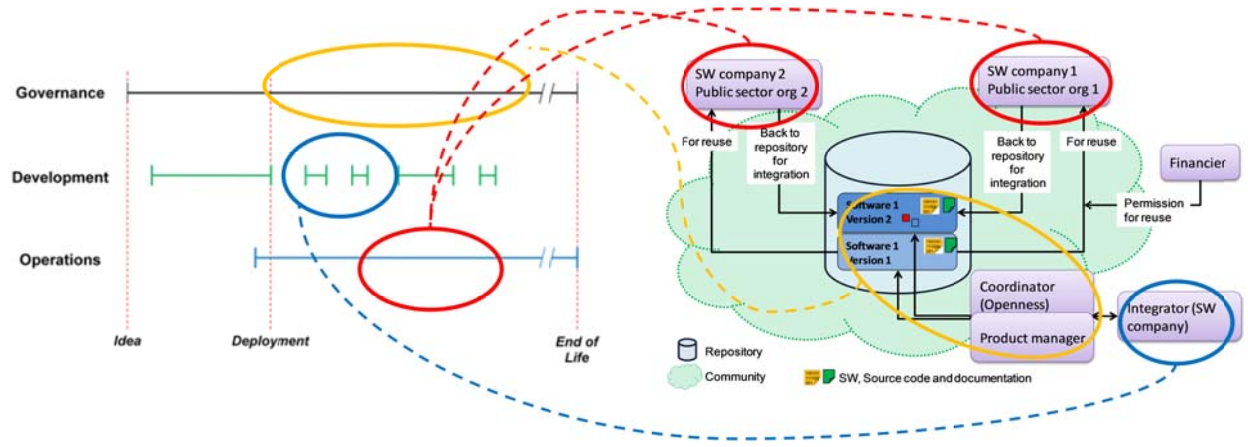

Figure 6: Comparing Model 3 and the aspects of ALM from [25].

The Model 3 has several potential benefits but also a number of challenges that need to be solved when deploying the model. The potential benefits of Model 3 are:

- **Flexibility:** each public sector organization can tailor the baseline version of the SW product with their own SW company (SW supplier) (cf. servicing in [30]). The SW supplier handles the deployment and maintenance of the customer-specific version.

- **Simplicity:** the SW product has one version path. The baseline version can be tailored according to the needs of the public sector organization but parallel baseline versions are not maintained. Cf. [21, 18].
- **Coordination:** baseline versions and their evolution/maintenance are coordinated by the product manager and community in cooperation with the integrator.
- **Reusability:** SW reuse is possible. SW product versions and their related documentation are available through a repository and they evolve in a coordinated way towards the needs of the public sector organizations.
- **Visibility:** the repository works as a portal for the public sector community. The portal can be used for storing, searching, delivering and discussing SW products.
- **Community:** the community forms a channel for public sector organizations to cooperate and through which they can affect the lifecycle of the SW product. It is possible to utilize the competence of community: SW product and deployment experiences, ideas/needs for new functionality. The community adapts the several potential benefits of OSS communities.
- **Harmonization:** Model 3 could harmonize the usage of SW products in the public sector. Different public sector organizations may reuse the same SW products yet organization-specific tailoring has been enabled.
- **Openness:** allows small, local SW companies to participate in public sector SW projects (especially the tailoring and maintenance of customer specific versions) cf. [21].

Potential challenges of Model 3 are:

- **Competence:** the maintenance and development of the baseline version requires strong competence (cf. [30]). Therefore, the developer of the first version could be in a dominating market position concerning the integrator role (vendor lock-in). Product manager's role is demanding.
- **Balance:** requires clear practices, responsibilities and repositories for lifecycle management but these need to be simple enough to avoid bureaucracy (cf. [18]). Furthermore, it requires effort and financing, for instance, the work of product manager, repository, etc. => costs vs. benefits?
- **Financing:** the financing model could be challenging since there are several roles/functions that need financing as well as several public sector organizations working in cooperation. How to finance the development of the SW product (how to finance the work of the integrator)? A similar problem from an inner-source-software (ISS) development point of view was addressed in [11].
- **Commitment:** public sector organizations need to operate actively in the community or the community will wither away. Difference of opinion inside the community. Is it possible to find competent person for each role?

6. CONCLUSIONS

This article describes the models for coordinating the evolution of open-source like SW products during their lifecycle. The models have been defined in cooperation with the Ministry of Finance, VTT and a number of Finnish public sector organizations. The basic idea is that SW products are licensed so that the source code always remains open, are freely distributed and are further developed by the public sector community so that SW products will best serve the needs of the Finnish public sector organizations throughout their lifecycle.

The research defined three models for the lifecycle management. The selection of the model has been discussed and the most complete model called the "*Community-based ALM*" model has been presented and discussed in detail. The study shows that even though the model has many potential benefits, there are several challenges that need to be tackled to enable the open community to work fluently.

Next the models will be concretized and piloted in real-life cases in the public sector. Concretization means that process guides, documentation templates and repositories will be defined for a model. Pilots will be used to validate the applicability of the models in real-life situations. The model 3 has been selected for piloting first. We would also like to encourage scientific and professional discussion about the community-based management of open-source software in the public sector.

REFERENCES

[1] Puhakka, M., Seppänen, M. and Oksanen, V. 2006. *A study of the deployment of open source software - Finnish experiences from public and private sector*. In: Maula, M., Hannula, M., Seppä, M., Tommila, J., (eds.). Frontiers of e-Business Research 2006, ICEB + eBRF 2006 - Conference Proceedings. 28.11.-2.12.2006, Tampere, Finland Tampere University of Technology and University of Tampere.

[2] Lerner, J. and Tirole, J. 2002. Some Simple Economics of Open Source. *Journal of Industrial Economics*, 50(2), 197-234.

[3] Bonaccorsi, A. Rossi Lamastra, C. and Giannangeli, S. 2004. Adaptive Entry Strategies under Dominant Standards - Hybrid Business Models in the Open Source Software Industry, *SSRN Electronic Journal*, pp. 1-23, 2004.

[4] Hecker, F. 1999. Setting Up Shop: The Business of Open-Source Software, *IEEE Software*, Volume 16 Issue 1, pp. 45-51.

[5] Favaro, J. and Pfleeger, S., L. 2011. Software as a Business, *IEEE Software*, July/August, 23-25.

[6] Riepula, M. 2011. Sharing Source Code with Clients: A Hybrid Business and Development Model. *IEEE Software*, July/August, 31-35.

[7] West, J. and O'Mahony, S. 2005. *Contrasting Community Building in Sponsored and Community Founded Open Source Projects*. Proceedings of the 38th Annual Hawaii International Conference on System Sciences.

[8] Dahlander, L. and Magnusson, M. 2008. How do Firms Make Use of Open Source Communities?, *Long Range Planning*, vol. 41, no. 6, pp. 629-649.

[9] Stam, W. 2009. When does community participation enhance the performance of open source software companies?, *Research Policy*, vol. 38, no. 8, pp. 1288-1299, Oct. 2009.

[10] IBM. 2011. *http://www-03.ibm.com/linux/linuxconNA2011/*. Retrieved on March 6th 2012.

[11] Wesselius, J. 2008. The bazaar inside the cathedral: business models for internal markets, *IEEE Software*, vol. 25, no. 3, pp. 60–66.

[12] Moreira, M. 2004. *Software configuration management implementation roadmap*, John Wiley & Sons, 244p.

[13] Jonassen Hass, A. 2003. *Configuration Management Principles and Practice*, Addison Wesley.

[14] Bersoff, E., Henderson, V. and Siegel, S. 1980. *Software Configuration Management - An Investment in Product Integrity*, Prentice Hall.

[15] IEEE Std-828. 2005. *IEEE Standard For Software Configuration Management Plans*. IEEE Std 828-2005 (Revision of IEEE Std 828-1998).

[16] Estublier, J., Leblang, D., van der Hoek, A., Conradi, R., Clemm, G., Tichy, W. and Wiborg-Weber, D. 2005. Impact of software engineering research on the practice of software configuration management, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 14, No. 4, October 2005, pp. 1–48.

[17] Eskeli, J., Heinonen, S., Matinmikko, T., Parviainen, P. and Pussinen, P. 2010. *Challenges and Alternative solutions for ERP's*, VTT, Oulu. 55 p. Research report : VTT-R-05936-10 , http://www.vtt.fi/inf/julkaisut/muut/2010/VTT-R-05936-10.pdf

[18] Asklund, U. and Bendix, L. 2002. *A study of configuration management in open source software projects*. IEE Proceedings of Software Engineering 149(1): pp. 40-46.

[19] Erenkrantz, J. 2003. *Release Management Within Open Source Projects, 3rd Workshop on Open Source Software Engineering*, ICSE'03, International Conference on Software Engineering, Portland, Oregon, May 3-11, 2003.

[20] Michlmayr, M., Hunt F. and Probert D. 2007. *Release Management in Free Software Projects: Practices and Problems*, OSS2007: Open Source Development, Adoption and Innovation (IFIP 2.13), Springer, pp295 – 300

[21] JHS 169. 2009. *Avoimen lähdekoodin ohjelmien käyttö julkisessa hallinnossa (the use of open-source software in public administration)*, Julkisen hallinnon tietohallinnon neuvottelukunta (JUHTA; Public sector's IT-negotiation group), Version 1.0, 23.02.2009. (In Finnish.)

[22] Weiß, G., Pomberger, G., Beer, W., Buchgeher, G., Dorninger, B., Pichler, J., Prähofer, H., Ramler, R., Stallinger, F. and Weinreich, R. 2009. *Software engineering – processes and tools*, In: Hagenberg Research, Eds.: Buchberger, B., Affenzeller, M., Ferscha, A., Haller, M., Jebelean, T., Klement, E.P., Paule, P., Pomberger, G., Schreiner, W., Stubenrauch, R., Wagner, R., Weiß, G. and Windsteiger, W., Springer-Verlag, Berlin, Heidelberg, pp. 157–235.

[23] Doyle, C. 2007. The importance of ALM for aerospace and defence (A&D), *Embedded System Engineering*, Vol. 15, No. 5, June, pp. 28 - 29.

[24] Doyle, C. and Lloyd, R. 2007. Application lifecycle management in embedded systems engineering, *Embedded System Engineering*, Vol. 15, No. 2, March, pp.24 – 25.

[25] Chappell, D. 2008. *What is application lifecycle management*, Chappell & Associates, White paper.

[26] Murta, L., Werner, C. and Estublier, J. 2010. *The Configuration Management Role in Collaborative Software Engineering*, In: Mistrík et al., Collaborative Software Engineering, Springer-Verlag, Berlin, Heidelberg, pp. 179 – 194.

[27] Schwaber, C. 2005. *The Expanding Purview Of Software Configuration Management*, Forrester Research Inc., White Paper, 22 July.

[28] Rossberg, J. 2008. *Pro Visual Studio Team System: Application Lifecycle Management*, Apress; 1 edition, 344 p.

[29] Grynberg, A. and Goldin, L. 2003. *Product Management in Telecom Industry – Using Requirements Management Process*. In: Proceedings of the IEEE International Conference on Software—Science, Technology & Engineering (SwSTE'03)

[30] Rajlich, V. and Bennett, K. 2000. A staged model for the software life cycle, *Computer*, July/2000.

[31] Capiluppi, A., González-Barahona, J., Herraiz, I. and Robles, G. 2007. *Adapting the "Staged Model for Software Evolution" to Free/Libre/Open Source*

Software, IWPSE’07 September 3-4, 2007, Dubrovnik, Croatia.

[32] Bennett, K. and Rajlich, V. 2000. *Software Maintenance and Evolution: a Roadmap*, Proceedings of the Conference on The Future of Software Engineering, Anthony Finkelstein (Ed.), ACM Press, pp. 73 - 87.

[33] IEEE Std 14764. 2006. *IEEE Standard For Software Engineering — Software Life Cycle Processes — Maintenance*. IEEE Std 14764-2006.