UPGRADE is the European Journal for the Informatics Professional, published bimonthly at <http://www.upgrade-cepis.org/>

Publisher
UPGRADE is published on behalf of CEPIS (Council of European Professional Informatics Societies, <http://www.cepis.org/>) by Novática <http://www.ati.es/novatica/>, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática,* <http://www.ati.es/>)

UPGRADE monographs are also published in Spanish (full version printed; summary, abstracts and some articles online) by Novática

UPGRADE was created in October 2000 by CEPIS and was first published by Novática and INFORMATIK/INFORMATIQUE, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <http://www.svifsi.ch/>)

UPGRADE is the anchor point for UPENET (UPGRADE European NETwork), the network of CEPIS member societies' publications, that currently includes the following ones:
- Informatica, journal from the Slovenian CEPIS society SDI
- Informatik-Spektrum, journal published by Springer Verlag on behalf of the CEPIS societies GI, Germany, and SI, Switzerland
- ITNOW, magazine published by Oxford University Press on behalf of the British CEPIS society BCS
- Mondo Digitale, digital journal from the Italian CEPIS society AICA
- Novática, journal from the Spanish CEPIS society ATI
- OCG Journal, journal from the Austrian CEPIS society OCG
- Pliroforiki, journal from the Cyprus CEPIS society CCS
- Tölvumál, journal from the Icelandic CEPIS society ISIP

ISSN 1684-5285

Monograph of next issue (August 2009)

"20 years of CEPIS: Informatics in Europe today and tomorrow"

(The full schedule of **UP**GRADE is available at our website)

CEPIS UPGRADE

The European Journal for the Informatics Professional
http://www.upgrade-cepis.org

Vol. X, issue No. 3, June 2009

Monograph: Libre Software for Enterprises

(published jointly with Novática*)

Guest Editors: *Jesús-M. González-Barahona, Teófilo Romero-Otero, and Björn Lundell*

* This monograph will be also published in Spanish (full version printed; summary, abstracts, and some articles online) by **Novática**, journal of the Spanish CEPIS society ATI (*Asociación de Técnicos de Informática*) at <http://www.ati.es/novatica/>.

Applying Open Source Software Principles in Product Lines

Frank van der Linden

Product line engineering improves the management of variability and reuse in embedded systems. It helps to exploit mass customisation leading to individualised products for the customer. It has led to substantial development processes and many global planning for a complete range of products (a product line). Open source software (OSS) development does not have a strong impact in embedded systems companies, partially because the culture of open source developers does not adhere to global planning and formal processes. However, both developments (often) need distributed development, and here is the point where software product lines and their open sources can profit from the achievements of each other. This paper investigates several options of using OSS and its methodology in product line development to reduce the problems of distributed development and to increase the quality. A small part of the paper discusses the options for OSS development to benefit from variability management of software product lines and in this way increase the possibility of customising the resulting products.

Keywords: Commoditisation, Inner Source, OSS, Software Product Lines, Variation Management.

1 Introduction

In recent decades, embedded systems providers have introduced product line engineering in order to improve the management of the diversity of their products and to reduce development effort. The move towards software product line engineering is usually strongly based on large-scale reuse, leading to cost and time-to-market reduction and quality improvement and maintenance cost reduction.

Software product line engineering is a way of developing software applications using platforms and mass customisation. Mass customisation enables the fast and efficient production of an individualised product for the customer. The artefacts used in different products have to be sufficiently adaptable to fit into the different systems produced in the product line. This means the management of variability in all artefacts of the product line. In particular a product line often defines a platform of architecture, components and supporting tools to provide efficient standardised mechanisms to manage variability.

Other trends in software engineering do not always relate to software product lines, whereas their application in an organisation doing product lines may be beneficial. Vice-versa, other approaches towards software engineering may benefit from the introduction of product line principles. In particular, this holds for OSS development, since it appears to be a profitable way to obtain good software. OSS development has many different forms, but the main aspect of it is that a distributed group of people work together to produce a piece of software. However, doing distributed development is different to what is presently in use within most companies doing product lines. OSS communities' work is based upon their own motivations, often related to the urge to produce a piece of functionality and to share this with others.

There is a benefit of OSS for product line engineering. Because of the diverse use of OSS, product line practices can be attractive for OSS communities. However, many characteristics of product line engineering do not match present day practices in OSS development. For instance the managed processes are not always feasible or accepted. It is profitable to combine the benefits of both approaches which are necessary to overcome these differences. Presently, there is limited interaction between the OSS and product line development communities, although its interest is already stated in literature [1]. Studying the identified problems in more detail and evaluating the proposed alternative approaches more carefully is a goal of future work in this area.

This paper investigates the relationship between OSS and product line development. It is based on results obtained in the ITEA project COSI [2]. The following observations can be made: Software product lines usually need large investments, long term planning, explicit variability

Author

Frank van der Linden works at Philips Healthcare CTO Office since 1984, when he received his Ph.D. in Mathematics at the University of Amsterdam (The Netherlands). Since 1991 he has been involved in product line development within Philips. Frank was project leader of the three successive ITEA projects on product line engineering, and successively on distributed development, including OSS development. As part of these projects he was member of the organizing committee of a series of workshops on conferences in product lines (PFE & SPLC). Between 2005 and 2008 he was leader of the ITEA project COSI, on distributed development and OSS practices in embedded system industry. In this project he has organized several workshops on OSS and product lines. He is editor of many proceedings of SWAPF (Software Architectures for product Families), PFE (Product Family Engineering) workshops and SPLC (Software product line conference). He is co-author of several books on Software product lines. <frank.van.der.linden@philips.com>.

management, and often this means large, thus *distributed* development. The Information Technologies (IT) industry uses OSS in a profitable way to obtain good software. It helps to reduce the costs through effort sharing. It helps to reduce the lead time through agile development. And it is an intrinsically *distributed* way of developing systems. This parallel concern for distributed development is the main inspiration for trying to combine OSS and product line practices, to the benefit of all involved.

In this paper we explain the aspects of software product line engineering (Section 2), and relate them to OSS practices (Section 3). Finally we have a short section on the use of product line practices in OSS development (Section 4).

2 Product Line Engineering Basics

This section is based on two books of introducing software product lines [3], and applying these in embedded systems industry [4].

2.1 Two Development Processes

Product line engineering emphasises the separation between the two concerns of building the platform robustly and being able to build customer-specific applications in a short time. This leads to two interrelated development processes (see Figure 1):

- Establishing the platform (*Domain engineering*), including the definition of the commonality and the variability of the product line.
- Deriving applications (*Application engineering*), including the binding of the variability in applications.

The separation into two processes also indicates a separation of concerns with respect to variability. Domain engineering is responsible for ensuring that the available variability is appropriate for producing the applications. This includes common mechanisms for deriving a specific application. Application engineering focuses on the development of the individual systems on top of the platform. It reuses the platform and binds the variability for the applications. These two processes are meant to be loosely coupled and synchronised by platform releases. As a consequence they can be based on completely different life-cycle models.

2.2 Variability

Software product line engineering aims at supporting a range of products which in turn support different individual customers. Instead of understanding each individual system all by itself, software product line engineering looks at the product line as a whole and the variation among the individual systems. Equally important it is to know what is common to all products. Throughout software product line engineering this commonality and variability must be managed. Variability is defined during domain engineering. It is exploited during application engineering by binding the appropriate variants. Management of variability involves the control of the definition and use of commonality variability within the product line. This includes the binding time of the variability.

Variability is introduced during many phases of the domain engineering process. At each level, variability from the previous level is refined and additional variability is introduced, which is not a result of refinement (see Figure 2). Variability is initially defined within stakeholder needs, and is refined in the following process phases. This is called internal variability. At each phase specific needs lead to the introduction of new *internal* variability.

Setting up the product line infrastructure is not a goal in itself. The ultimate aim is its exploitation during application engineering. This is also called the *instantiation* of the variability.

The following notions, which denote self-contained entities in all kinds of development artefacts, are important in the management of variability:

- *Variation Point:* the variation point describes where in the final systems differences exist.
- *Variant:* the different possibilities that exist to satisfy a variation point are called variants.

In most cases variation points do not change independently. Selection of a certain variant for a given variation point influences the possible choices for other variation points. In order to enable variability management, a variability model is needed to ensure the consistent selection of variants for variation points. In early proposals for variability modelling the model was often integrated in the underlying notation, using e.g. inheritance. However, it is generally important to make a distinction between the variability model and a main system model. This is much easier to apply in complex settings, scales much better [5] and supports the decision process better. Several proposals exist for variability models. The most important groups are:

- Variability models that are based on feature trees [6]. This describes a hierarchy. At the top level a single variation point exists representing where the variants represent the choices with the highest priority. Each variant is treated as a variation point for the next level in the hierarchy.
- The orthogonal variability model (OVM) [3]. Here variants and variation points are distinct elements. Specific relationships determine which variation points and variants should/can go together in the same system.

We distinguish between variability in time and variability in space [7]. Both kinds are supported with well designed variation points and variants. Variability occurs over time with the evolution of the product line and represents the existence of different versions of an artefact that are valid at different times. This is related to versioning of system elements. Variability in space deals with the existence of an artefact in different shapes at the same time. It leads to different configurations that are all valid systems at the same time. This is related to product-line practices.

3 OSS for Product Lines

The interest of OSS within companies doing product lines originates from the recognition that a large part of software is becoming a commodity, which makes it highly interesting for embedded systems developers to introduce

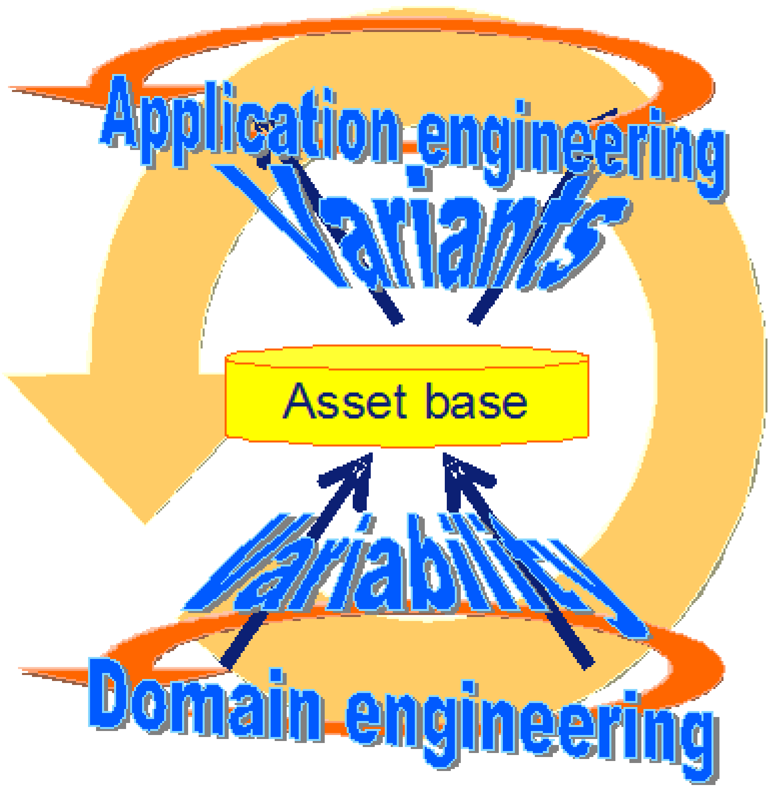

Figure 1: Product Line Engineering Process.

OSS in their product, and thus in their product lines. This issue of commoditization of software is discussed first. Then we explain ways to implement OSS in product lines.

3.1 On Commoditization of Software

As a consequence of the commoditization of software [8], a large part of the software is no longer product specific. Software is often not built by a single company alone. Instead software is produced in close cooperation spread throughout a company and beyond the company's borders. Third-party software is integrated in the commodity parts of software.

For most products, and product lines, only a small part[1] (5 to 10 percent) of the software is differentiated. This small part provides the added value over the competitors. The remainder is more or less common to the domain, or even across different domains; i.e. it is more or less a commodity. Effective and efficient software development only focuses on producing the differentiating parts. The commodity software is preferably to be acquired elsewhere, involving distributed development and external software such as commercial off the shelf (COTS) or OSS. In the case of

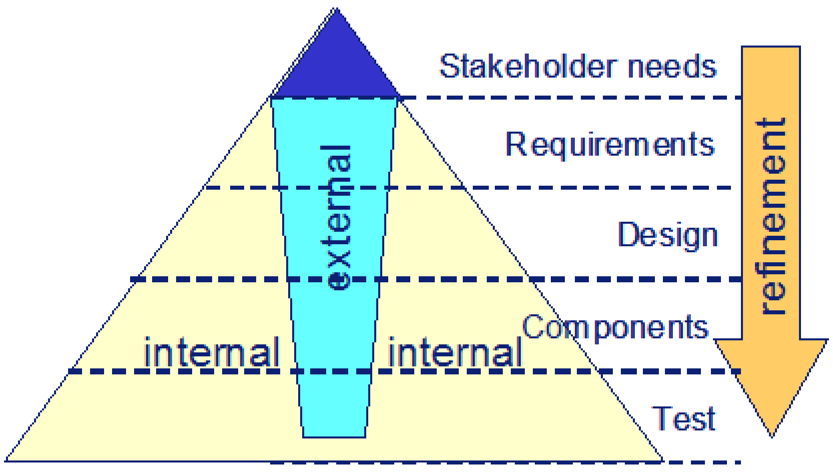

Figure 2: Variability Pyramid.

product lines, the commodity part is usually part of the commonalities of the software. As such the domain engineering is more influenced by software may need different ways to adjust it to a specific application.

Figure 3 shows the landscape of technology versus business decisions on making or acquiring software. There are two corner areas to be avoided in producing technology. The upper-right hand corner should be avoided at any cost, because it would mean passing the own added value to competitors. The lower-left hand corner has to be avoided in order to save on development costs, since commodity technology can be obtained cheaper by buying it instead of making it. Healthy software development is characterised by the middle area, from top-left to bottom-right. Differentiating software is developed within the organisation (top-left corner). Commodity software is bought at the market or even available at no cost (OSS).

Any software in systems, including product line software, is moving from top to bottom in the diagram. Software starts as being differentiated for some party. At a certain moment, the software does not provide enough added value to the products and it can be considered as basic to the business, at later stages the software even moves towards commodity.

Healthy software development is characterised by moving from left to right, and at the same time, from in-house to open collaborations. In order to avoid the top-right hand and the lower left-hand corners, these moves have to be made at the right pace. An example of such movement is the case of DVTk software[2] [8] of Philips Healthcare which has moved in several steps from proprietary and differentiating software to open source and commodity software during 1994-2002.

Each individual company needs to analyse their software in order to know when to change their way of collaboration. This especially holds for companies producing product lines. Because of the long lifetime of a single product line, continuously there are parts of it that move to commodity, whereas the product line has to stay alive and healthy. This requires additional effort during planning, definition of common and variable assets, and architecture mechanisms. An additional complication is related to the fact that companies have limited control over software created in OSS developments.

3.2 OSS for Product Lines

Because of the size of the development, product line organisations are often involved in distributed development, which works very efficiently within OSS communities. Moreover, in reuse procedures it is unwise to ignore the large amount of OSS that is available. Within the COSI project [2] we experimented with case studies exploiting OSS in 5 fundamental different ways:

1) Adopting the development practices within product line development. This is called inner source, and will be explained in Section 3.2.1.

2) Using OSS development tools in the development of the product line. This is the easiest form of using OSS in product lines. Aspects of this form will be discussed in Section 3.2.2.

3) Using OSS components in the products of the product line. This induces more involvement and planning. It is described in Section 3.2.3.

4) Opening up products of the product line. Products, or a complete product lines that were originally developed inside the company are contributed to OSS communities, see Section 3.2.4.

5) Establishing a symbiotic relationship. Developing the product line using the resources of an OSS community, see Section 3.2.5.

3.2.1 Inner Source

Inner source is a way to exploit the advantages of distributed development in the open source way, but with the need to avoid problems with planning, ownership and control. Several companies have adopted an inner source development model [9]. In inner source development a set of teams collaborate in a cooperative eco-system. Similar to OSS development, inner source development applies an open, concurrent, model of collaboration. It implies distributed ownership and control of code, early and frequent releases, and many continuous feedback channels. It makes use of organisation mechanisms already in place, e.g. for escalation of conflicts or setting up roadmaps. Inner source enables flexibility in starting, stopping, and changing of collaborations, in timing of and setting priorities for development teams across organisational (and geographical) boundaries. Companies can use inner source development as an intermediate step towards the integration of open source in their products.

Inner source is an established way of development within Philips Healthcare [10]. The company delivers a product line involving wide range of medical imaging products to its customers (hospitals in general) in various modalities [4]. The domain engineering group provides a platform consisting of a collection of reusable and configurable components, based on a common architecture. The domain engineering group faced the problem that they may become a bottleneck to the many application development groups. The domain releases are planned in a process with many stakeholders, so the resulting release schedule cannot always satisfy the planning of certain application engineering groups. Market dynamics can force product groups to change their planning, but the domain planning process lacks flexibility to adapt to these changes.

The inner source development is based on elements of open source development, supported by a Web-based collaborative development environment. It decouples applica-

[1] TPPT Our experience from the European software industry is in line with the view of Perens, who estimates that "*Perhaps 90% of the software in any business is non-differentiating*".

[2] Dicom Validation Toolkit, OSS supporting medical image exchange compliance. <http://www.dvtk.org/>.

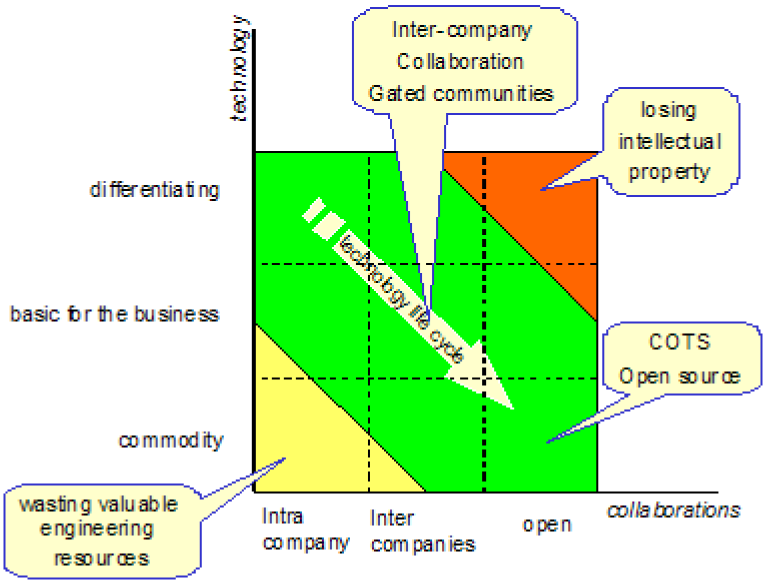

Figure 3: Efficient and Effective Software Development.

tion engineering from the domain engineering, as each application engineering group can decide themselves whether they:

■ *Use as is:* they wait for the next release of the platform.

■ *Contribute through patches:* take an earlier version of a component and optionally patch it.

■ *Work together as virtual team:* take responsibility for the development of some, for those group crucial, domain components.

The main inner source development principles have easy access to all information of the product line. As in OSS development the policy is to release early and often to enable fast information flow from domain to application. There is a distributed ownership and control of all domain assets. The domain team owns and develops domain components. An application developer is allowed to change components, but is responsible for the change. The change may be offered back to the domain team. The platform team can accept the patch into the platform, taking over ownership.

As a consequence of this, the involvement of the application teams in the domain is improved. This has as a side-effect that the platform is more widely in use than before, and is better suited to the needs of the application groups.

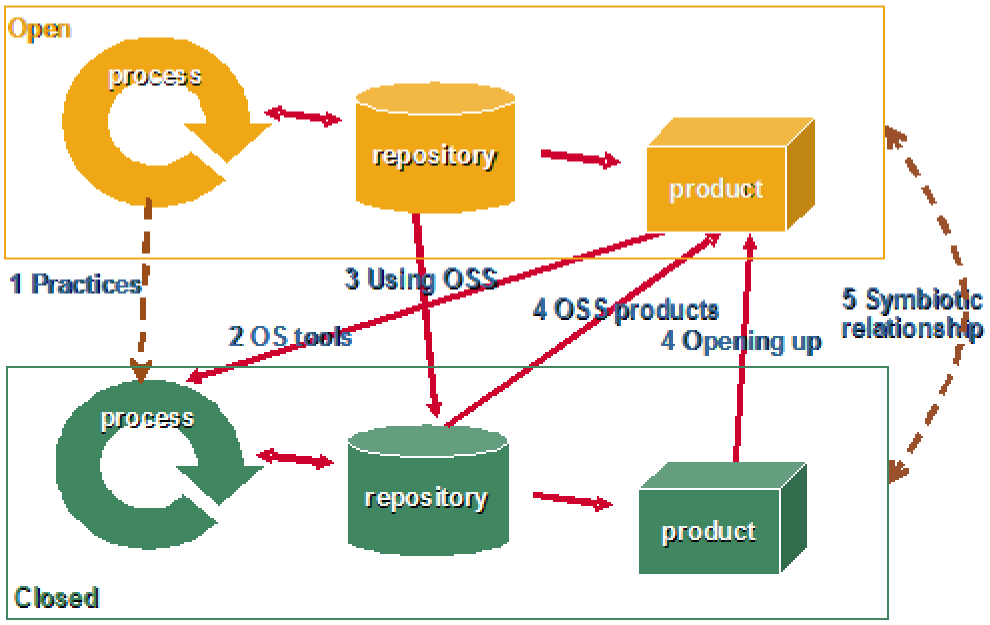

Figure 4: Leveraging Open Source Opportunities.

Inner source has enabled new product launches since 2005. It led to a time-to-market reduction of at least 3 months.

However the inner source model does not help to introduce third party software into the software product line. Collaboration is limited to the company. The control over the software is then clear and distributed development and maintenance is improved over the original situation. Inner source provides methods of collaboration and software development on top of the traditional methods. This is important, because it allows normal operations to proceed as usual, thus retaining the strong points of the traditional management structures, while creating room for new and more flexible ways of collaboration.

3.2.2 Using OSS Tools

This is the easiest form of benefiting from OSS in product lines. As the OSS does not appear in the final product, it is easy for the user to comply with the rules of the OSS community. Many tools are already available open sourced, and as long as they fit in the development model of the company they can be used. A complicating factor is that not many product line specific OSS tools exist. A list of useful product line tools can be found at the software product lines tool vendors' page[3]. Most of these tools are not open sourced. Some useful tools that are used within COSI for product line development and are OSS are:

- Stylebase for Eclipse: open source tool supporting the reuse and sharing of architectural knowledge[4].
- Subversion: Providing version management[5].
- Semantic MediaWiKi: a tool that support the collaborative development of documents within a collaborative development environment[6].

3.2.3 Using OSS Components

The use of OSS in product lines is in principle not much different from using other third party software (COTS) in the product line. However there are several points of attention. Planning of third party software in product lines always suffers from the fact that that it happens at a pace that is not controlled by the company: new releases are distributed that improve upon earlier ones in ways that are not controlled by the company. These improvements will influence the product line, since parts of the new third party features are used within domain engineering, influencing all products. Therefore the new release needs to be incorporated, and proprietary parts of the product line have to be adapted. Often the new release is based on architecture mechanisms and interfaces which are different to what was practice within the product line. In order to deal with the situation, the organisation needs to know in advance what is going to happen, and in the case of COTS software this often means good contacts with the supplier. In an OSS development, knowledge of future releases is best obtained through the involvement in the community itself. At least being connected to mailing lists already informs about what is going to happen. In addition, if some of the developers are involved in the development of OSS, the company may have influence in which way the software evolves. This may even lead to incorporation of its own architecture mechanisms and standards in the OSS.

Instead of waiting for new releases of COTS, and keeping contact with the supplier, OSS adapts in a more continuous way. Instead of the normal contact with the vendor of COTS, companies need to get involved in the OSS community. The possibility of obtaining early versions of new software which may still have some bugs helps to test it early for compliance with the product line, and fast incorporation within the family as soon as it is stable. Issuing bug reports and corrections helps the community to keep high-quality software, which is a commodity, but still important for the own product line. Decisions have to be taken on which parts (and which versions) of the OSS will become part of the platform, and what will be only used in some applications. In fact, it may be a strategy to incorporate latest versions of the OSS only in several "trial" applications, as is the case with normal use of COTS. Both domain and application engineering may be involved in OSS development.

A concern of OSS development is related to the issue of licensing. This concern will affect the way OSS can be used within a product line organisation. Several open source licenses ask to put the source of the additional software in the open community as well. There are technical options to deal with this through the provision of loose coupling of OSS and software provided under a "traditional" proprietary software license. However, a company often needs to donate some own software (glue, interfaces, daemons…) to the OSS community to demonstrate good will [11]. Special care has to be made when software from several sources is incorporated in the product line. Licenses for different pieces of OSS may be in conflict with each other. It should be communicated and managed within the organisation what is happening for several reasons:

- Ignoring licenses in one department may lead to the necessity of opening differentiating software of other departments.
- OSS use in application development may indicate a move toward commoditization, which in turn may mean that the software should become part of the commonality. Note that this is not always that case, as the OSS relate to application specific code for only a few products.
- New versions of OSS in domain software may make parts of some applications obsolete.
- OSS use may lead to adapted internal standards in the product line. The only way to deal with this is the agreement of all parties involved in the product line.

In the context of being involved in OSS development, the company may introduce its own variability manage-

[3] <http://www.softwareproductlines.com/resources/vendors.html>.
[4] <http://stylebase.tigris.org>.
[5] <http://subversion.tigris.org>.
[6] <http://semantic-mediawiki.org/wiki/Semantic_MediaWiki>.

ment (tooling) into the community. This eases the configuration problem for the OSS community as well. However, the community may not appreciate it and may discard the variability management. In that case the company may set up, or join, another community that does adopt the tooling, after which it announces that it uses this open tooling. This may secure the open maintenance of the own mechanisms and the acceptance by the community. However, at the same time complete control over these mechanisms is lost. The company needs to stay involved, and thus it needs to invest in the OSS community, to keep the tooling acceptable for their own use.

In the case of conflicting architecture standards, an option is to use the practices adopted for COTS. Create proprietary wrappers or glue code to connect the own mechanisms to those that are available for the OSS components. This may need frequent updates, as both own and external mechanisms evolve independently. This implies effort in domain development and possibly also in application development for certain application-specific wrapper code in the final products. This solution concentrates on technical integration. It works well as long as the solution does not interfere with reusability, clarity and manageability of integration architecture. The frequent changes of OSS, together with a lack of good documentation may make this approach quite laborious [11].

Another option is that the company adopts the architecture mechanism which is in use within the OSS community. This has a disadvantage in that both domain and application engineering needs to be transferred, which may mean a lot of effort. The company has to be sure that the mechanism is good enough for their purposes, and that they can still manage their own variability. As with the other approaches the lack of architecture description and the frequent changes may make this a difficult option. Also in this case the company needs to stay involved, and thus it needs to invest in the OSS community, to keep the tooling acceptable for their own use.

3.2.4 Opening up Products of the Product Line

Many companies have had good experience in opening up commodity software. Although at first glance opening software means giving away your intellectual property, this is normally not the case. Software that is at the lower part of diagram, i.e. commodity software, does not contain expensive intellectual property. This can be shared without much danger. An important asset is the know-how to adapt the own software to the needs of the customers, and this is a kind of knowledge which should not given away. Clients conceive that it is easier and cheaper to hire the company to install the software, than to do the entire configuration themselves.

For Philips Healthcare the first experience of opening up software was DVTk software [8]. This software supports the verification of compliance with the medical image exchange standard DICOM. Since clients will connect equipment from different vendors it is important that all companies comply with the standard. However, the DVTk is just commodity software which belongs to the company. Therefore Philips decided in 2005 make it open source. This enables sharing the development and maintenance on a much wider scale. In particular the lead time to faster development and maintenance of the software, especially for those parts of DICOM which are generic and not only relevant to the needs for Philips.

3.2.5 Symbiotic Relationship

Finally, own pieces of commodity software may be donated to the community to ensure its support by the community. This leads to less maintenance effort for the own company. Maintenance is shared with competitors and people in other domains, who all need the software. Improvements and testing of the software is done in the community, and the company itself is supportive in a collaborative way to ensure that the correct issues are addressed.

This can be very effective, in particular for tools or components which have already been obtained elsewhere. Being involved in the community enables the company to ensure that their requirements are considered, and that the software proceeds in a way which is beneficial for the company. For instance Philips Healthcare is now part of the Subversion community. The original goal was to improve the software for solving tree conflicts. This approach was successful through its donation of a rename-ware merge tool, which was part of Subversion release 1.6. In the meantime Philips has become an expert user of Subversion and thereby influences the future direction of the Subversion project.

4 Product Lines in OSS Development

4.1 Product Line Practices

From the OSS perspective, some research has investigated specific OSS projects with a view to analyse what and how certain product line principles and practices have been adopted. For example, an analysis of the Linux kernel [12] "*demonstrates how the Linux Kernel achieves some of the goals that the [Software Product Line] guidelines also aim at*". Further, van Gurp [13] analysed practices used in three large OSS projects (Eclipse, Mozilla and Linux) with a view to suggesting improvements to product line development, and concludes by suggesting to Product line owners that "*there is this set of practices that may be found in many open source projects that is known to work well at least in that context*".

Chastek et al. [14] have investigated specific development models for product lines with a view to analyse how a specific OSS project has adopted such a model. They used a Framework for Software Product Line Practice[7] developed by the Carnegie Mellon® Software Engineering In-

[7] http://www.sei.cmu.edu/productlines/framework.html

stitute (SEI). In their analysis of the Eclipse project and its community, they show that "*Eclipse has managed to strike an effective balance between planned directions and individual contributions and also shares the product line challenge of effective communication between those who develop core assets and those who use them*". In summary, although some OSS products can be viewed as a product line (e.g. Eclipse), product line principles are not widespread in use within OSS communities.

4.2 Architecture and Variability

An important asset in product lines is the architecture of the platform, which defines standards for the whole product line. All systems have to comply with this architecture to ensure effective use of the platform, and the reach the goals of product-lines to reduce development effort. In particular, in many OSS communities, the architecture is often not well established, and compliance with the architecture is often difficult to check [15].

OSS communities often have other ways of dealing with variability as is in use within a product line. In many cases, the OSS was never meant to be used in product lines, and problems related to complex configuration processes are abundant within the open source world. Variation points and variants are (almost) absent in OSS development. Variations in time and in space are not clearly separated. Variation is modelled in traditional ways through programming language and compiler directives. This is not very effective for product-lines (see Section 2.2).

However, OSS communities have developed effective ways to deal with internal configuration and product building. These mechanisms can be improved, described more clearly, enabling the introduction of effective variability management.

4.3 Two Processes

Within OSS communities, there is no clear distinction between domain and application engineering. However, both practices are performed within the communities. A core group of experienced developers in the community is often involved in those activities which we can assign to domain engineering. Central software which is used in the complete software suite of the community is built by the core developers. Developers who are less experienced or less active are often working on specific applications. However, as in product-line engineering, core functionality may emerge from these people as well. Promoting an asset from an application to the domain is relatively easy in OSS developments, and this is also one of the advantages of inner source development (see Section 3.2.1).

The scope of the OSS is often not well established. This eases the flexibility of it, and it enlarges the applicability of the software. However, the lack of a good architecture and a variability model makes it difficult to apply the software. It leads to differences in understanding of which mechanisms are allowed and which are not, resulting in reduced consistency of the different parts.

5 Conclusions

This paper investigated the use of OSS in product lines. The OSS development model is an inherently distributed and attractive way of building software since it has been shown that good quality software can be produced using this model. Companies can use the OSS communities for producing and maintaining commodity software. This frees company resources for producing differentiated software. However, such introduction of OSS in a product line is not without problems, and it still costs effort to stay involved within the OSS community. As the control and ownership of OSS is (in most cases) not within the company, it has to be involved and invest effort to reap the advantages of the OSS. The company has to consider its planning process for the product line, how to be involved and how to track the evolution of the OSS, when and how to introduce new versions. It has to take into account that legal problems will occur if application engineering is not aware of OSS in domain assets. Similar problems occur if application engineering connects OSS too tightly to domain assets that are considered to be differentiating for the company.

Technical problems may occur if the architecture of the OSS is not compatible with those of the product line. The company may use it involvement to influence the OSS community, or it may incorporate software code wrappers for its own products.

Product line organisations often have a large development organisation that is distributed, merely because of the size of the development. Distributed development gives a lot of coordinating problems, some of which are addressed by OSS development. In order to avoid some of the disadvantages of OSS use, an inner source development model may be used. This solves the distributed development problem, but it does not lead to the sharing of effort that occurs when OSS components are a part of the product line.

References

[1] Jan Bosch. "The Challenges of Broadening the scope of Software Product Families". Communications of the ACM, December 2006, pp. 41-44.

[2] COSI – Co-development using inner & open source in software intensive systems. ITEA project 2005-2008. <http://itea-cosi.org/> and <http://www.friprog.no/Laer-mer/Prosjekter/COSI-Library-of-Assets>.

[3] K. Pohl, G. Böckle, and F. van der Linden. "Software Product Line Engineering: Foundations, Principles, and Techniques". Springer, 2005.

[4] Frank van der Linden, Klaus Schmid, Eelco Rommes. "Software Product Lines in Action". Springer Verlag, 2007.

[5] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesh, and A. Vilbig. "A Meta-Model for Representing Variability in Product Family Development". In: Proceedings of the 5th International Workshop on Product Family Engineering (PFE-5), Siena (Italy), 2003, pp. 66-80.

[6] K. Kang, S. Cohen, J.A. Hess, W.E. Novak, and S.A. Peterson. "Feature-Oriented Domain Analysis (FODA) Feasiblity Study". Technical Report, Software Engineering Institute, Carnegie-Mellon University (USA), 1990.

[7] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink, and K. Pohl. "Variability Issues in Software Product Lines". In: Proceedings of the 4th International Workshop in Product Family Engineering (PFE-4), Bilbao (Spain), October 3-5, 2001, Springer, Berlin Heidelberg New York, LNCS 2290, 2002, pp. 13-21.

[8] Frank van der Linden, Björn Lundell, and Pentti Marttiin, "Commoditization of Industrial Software: A Case for Open Source". To appear in IEEE Software July-August 2009.

[9] A.A. Jilderda. "Inner Source Software Engineering at MIP fostering a meritocracy of peers". Research Report, Philips Research, The Netherlands, 2004.

[10] Jacco Wesselius. "The Bazaar inside the Cathedral: Business Models for Internal Markets". IEEE Software Vol. 25, No. 3, May/June 2008 pp. 60-66.

[11] Janne Merilinna and Mari Matinlassi. "Product Family Approach for Integration of In-house Software and open source Components". In F. van der Linden and B. Lundell (Eds.) Proceedings on the Second International Workshop on OSSPL07 Open Source Software and Product Lines 2007 (co-located with The Third International Conference on Open Source Systems – OSS 2007), June 14 2007, Limerick (Ireland). <http://www.itea-cosi.org/modules/wikimod/index.php?page=OssPl07 paper #2>.

[12] J. Sincero, H. Schirmeier, W. Schröder-Preikschat and O. Spinczyk. "Is The Linux Kernel a Software Product Line?". In F. van der Linden and B. Lundell (Eds.) Proceedings on the Third International Workshop on Open Source Software and Product Lines: OSSPL07 Asia (co-located with The 11th International Software Product Line Conference – SPLC 2007), September 10, 2007, Kyoto (Japan). <http://itea-cosi.org/modules/wikimod/index.php?page=OssPlas07>.

[13] J. van Gurp. "OSS Product Family Engineering". In First International Workshop on Open Source Software and Product Lines (co-located with The 10th International Software Product Line Conference – SPLC 2006), Maryland (USA), 2006. <http://www.sei.cmu.edu/splc2006/Gurp_paper.pdf>.

[14] G.J. Chastek , J.D. McGregor, and L.M. Northrop. "Observations from Viewing Eclipse as a Product Line", in F. van der Linden and B. Lundell (Eds.) OSSPL07 Asia (co-located with The 11th International Software Product Line Conference – SPLC 2007), September 10, 2007, Kyoto (Japan). <http://itea-cosi.org/modules/wikimod/index.php?page=OssPlas07>.

[15] Imed Hammouda and Tommi Mikkonen. "Open source Contributions as Platform Specialization Units". In F. van der Linden and B. Lundell (Eds.) Proceedings on the Second International Workshop on OSSPL07 Open Source Software and Product Lines 2007 (co-located with The Third International Conference on Open Source Systems – OSS 2007), June 14 2007, Limerick (Ireland). <http://www.itea-cosi.org/modules/wikimod/index.php?page=OssPl07 paper #3>.