Applying Open Source Software Principles in Product Lines

Frank van der Linden
Philips Healthcare, PO Box 10000, 5680 DA Best, The Netherlands, +31 40 276 4577
frank.van.der.linden@philips.com

Abstract

Product line engineering improves the management of variability and reuse in embedded systems. It helps to exploit mass customisation leading to individualised products to the customer. It has lead to heavy development processes and many global planning for a complete range of products (a product line). Open source development does not have a strong impact in embedded systems companies, partially because the culture of open source developers that do not adhere to global planning and heavy processes. However, both developments need (often) distributed development, and here is the point where software product lines and open source of them can profit from achievements of each other.

This paper investigates several options of using open source 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 to options for open source development to benefit from variability management of software product lines and in this way improve the possibility to customise the resulting products.

keywords: software product lines, open source software, inner source, variation management, commodification,

Introduction

In the last decades, embedded systems providers have introduced product line engineering in order to improve the management of the diversity of their product 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 means to develop software applications using platforms and mass customisation. Mass-customisation enables the fast and efficient production of an individualised product to 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 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 open source development, since it appears to be a profitable way to obtain good software. Open source development has many different forms, but the prime aspect of it is that a distributed group of people work together to produce a piece of software. However, it is a different from doing distributed development than what is presently in use within most companies doing product lines. Open source communities work for their own motivations, often related to the drive to produce a piece of functionality and to share this with others.

There is not only a benefit of open source for product line engineering. Because of the diverse use of open source software, product line practices can be attractive for open source communities. However, many characteristics of product line engineering do not match to present day practices in open source development. For instance the managed processes are not always feasible or accepted. It is profitable to combine the benefits of both approaches it is necessary overcome these differences. Presently, there is limited interaction between the open source and product line development communities, although its interest is already stated in literature: “... 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 open source and product line development. It is based on results obtained in the ITEA project COSI. The following observations can be made: Software product lines needs usually large investments, long term planning, explicit variability management, and often this means large, thus distributed development. The IT industry uses open source 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 open source and product line practices, to the benefit for all involved.
In this paper we explain the aspects of software product line engineering (section ), and relate them to open source practices (section ). Finally we have a short section on the use of product line practices in open source development (section ).

**Product line engineering basics**
This section is based on two books of introducing software product lines , and applying these in embedded systems industry .

**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 (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 conducted based on completely different life-cycle models.

**Variability**
Software product line engineering aims at supporting a range of products supporting 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. This is depicted in Figure 2; variability is initially defined at the 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 influence 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 recognised to make a distinction between the variability model and a main system model. This is much easier to apply in complex settings, scale much better and support the decision process better. Several proposals exist for variability models. The most important groups are

- Variability models that are based on feature trees. This describes a hierarchy. At the top level a single variation point exist – 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). 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. Both kinds are supported with well designed variation points and variants. Variability in 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.

### Open source for product lines

The interest of open source within companies doing product lines originates from the recognition that large part of software is becoming a commodity, which makes it highly interesting for embedded systems developers to introduce open source software in their product, and thus in their product lines. This issue of commodification of software is discussed first. Next we explain the ways to leverage open source in product lines.

#### On Commodification of Software

As a consequence of the commodification of software, a large part of the software is not product specific any more. Software is often not built by a single company alone. Instead software is produced in a close cooperation spread over 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 (5 to 10 percent) of the software is differentiating. 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 COTS or open source software. In the case of product lines, the commodity part is usually part of the commonalities of the software. As such the domain engineering is more influenced by software and may need different ways to adjust it to a specific application.

... 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 costs (open source software).

---

1. Our experience from the European software industry is in line with Perens, who estimates that “perhaps 90% of the software in any business is non-differentiating.”

---

**Figure 3: efficient and effective software development**
Any software in systems, including product line software, is moving from top to bottom in Figure 3. Software starts as being differentiating 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 a move is the case of DVTk software\(^2\) 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 with respect to Figure 3 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 open source developments.

**Open source for product lines**

Because of the size of the development, often product line organisations are involved in distributed development, which works very efficiently within open source communities. Moreover, in doing reuse it is not wise to ignore the large amount of software that is available in the open source. Within the COSI project we experimented with case studies exploiting open source 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 .

2. Using open source development tools in the development of the product line. This is the easiest form of using leveraging open source in product lines. Aspects of this form will be discussed in section .

3. Using open source components in the products of the product line. This induces more involvement and planning. It is described in section .

4. Opening up products of the product line. Products, or a complete product lines that were originally developed inside the company are contributed to open source communities, see section .

5. Establishing a symbiotic relationship. Developing the product line using the resources of an open source community, see section .

**Inner source**

*Inner source* is a way to exploit the advantages of distributed development in the open source way, but in a wish to avoid problems with planning, ownership and control. Several companies have adopted to an inner source development model . In inner source development a set of teams, collaborate in a cooperative eco-system. Similar to open source development, inner source development applies an open, concurrent, model of collaboration. It implies distributed ownership and control of code, early and frequent releasing, 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 and setting priorities of 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.

\(^2\) DVTk – Dicom Validation toolkit, open source software supporting medical image exchange compliance, http://www.dvtk.org/
Inner source is an established way of development within Philips Healthcare. The company delivers a product line involving a wide range of medical imaging products to its customers (hospitals in general) in various modalities, ch. 15. 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 developing 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 application 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 are to have easy access to all information of the product line. As in open source development the policy is to release early and often to enable fast information flow from domain to application. Have 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 side effect that the platform is more widely in use than before, and it suits better the needs of the application groups. Inner source enabled new product launches since 2005. It led to a time to market reduction with at least 3 months.

However the inner source model does not help to introduce third party software in 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.

### Using open source tools

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

- **Stylebase for Eclipse** – open source tool supporting the reuse and sharing of architectural knowledge
- **Subversion** – Providing version management
- **Semantic MediaWiki** – a tool that support the collaborative development of documents within a collaborative development environment

### Using open source components

The use of open source software 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 soft-

---

2. [http://stylebase.tigris.org](http://stylebase.tigris.org)
3. [http://subversion.tigris.org](http://subversion.tigris.org)
ware in product lines always suffer from the fact 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 different architecture mechanisms and interfaces than 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 open source 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 own developers are involved in the development of open source, the company may have influence in which way the software evolves. This may even lead to incorporation of own architecture mechanisms and standards in the open source software.

Instead of waiting for new releases of COTS, and keep contact with the supplier, open source adapts in a more continuous way. Instead of the normal contact with the vendor of COTS, companies need to get involved in the open source community. The possibility to obtain early versions of new software that may still have some bugs help 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 commodity, but still important for the own product line. Decisions have to be taken on which parts (and which versions) of the open source 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 open source only in several “trial” applications, as is the case with normal use of COTS. Both domain and application engineering may be involved in open source development.

A concern of open source development is related to the issue of licensing. This concern will affect the way open source software 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 open source and software provided under a ‘traditional’ proprietary software license. However, a company often needs to donate some own software (glue, interface, daemon, …) to the open source community to demonstrate good will. Special care has to be made when software from several sources is incorporated in the product line. Licenses for different pieces of open source software 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
- Open source use in application development may indicate a move toward commodification, which in turn may mean that the software should become part of the commonality. Note that this is not always that case, as the open source relate to application specific code for only a few products.
- New versions of open source in domain software may make parts of some applications obsolete.
- Open source 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 a context of being involved in the open source development, the company may introduce its own variability management (tooling) into the community. This eases the configuration problem for the open source community as well. However, the community may not appreciate it and 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 open source community, to keep the tooling acceptable for their own use.

In case of conflicting architecture standards, an option is to use the practice for COTS. Create proprietary wrappers or glue code to connect the own mechanisms to those that are available for the open source components. This may need frequent updates, as both of own and external mechanisms evolve independently. This implies effort in the 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 open source software, together with a lack of good documentation may make this approach a laborious ef-
fort.

Another option is that the company adopts the architecture mechanism that is in use within the open source com-
munity. This has as disadvantage that both domain and application engineering needs to transfer, which my
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 open source community, to keep the tooling acceptable for their own use.

Opening up products of the product line

Many companies have 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 Figure 3, 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 accommodate the own software to the needs
of the customers, and this is a kind of knowledge that 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 with opening up software is the DVTk software⁷. This software sup-
sports the verification of compliance to 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 the company. Therefore Philips decided in 2005 make it open source. This enables
sharing the development and maintenance on a much widerscale. In particular the lead to faster development
and maintenance of the software, especially for those parts of DICOM that are generic and not only relevant to
the needs for Philips.

Symbiotic relationship

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

This can be very effective, in particular for tools or components that are already obtained elsewhere. Being in-
volved in the community enables the company to ensure that their requirements are considered, and that the soft-
ware proceeds in the way that is beneficial for the company. For instance Philips Healthcare is now part of the
Subversion⁸ community. The original goal was to improve the software on solving tree conflicts. This approach
was successful by its donation of a rename-ware merge tool, which was part of Subversion release 1.6. In the
mean time Philips is become an expert user of Subversion that really influences the future direction of the Subver-
sion project.

Product lines in open source development

Product-line practices

From the open source perspective, some research has investigated specific open source 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 “demonstrates how the Linux Kernel achieves some of the goals that the [Software Product
Line] guidelines also aim at”. Further, van Gurp analysed practices used in three large open source projects (Ecl-
ipse, Mozilla and Linux) with a view to suggest improvements to product line development, and concludes by sug-
gesting 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. have investigated specific development models for product lines with a view to analyse how a spe-
cific open source project has adopted such a model. They used a Framework for Software Product Line Practice⁹
developed by the Carnegie Mellon® Software Engineering Institute (SEI). In their analysis of the Eclipse 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

⁷ http://www.dvtk.org/
⁸ http://subversion.tigris.org
⁹ http://www.sei.cmu.edu/productlines/framework.html
develop core assets and those who use them.” In summary, although some open source products can be viewed as a product line (e.g. Eclipse), product line principles are not widespread in use within open source communities.

**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 open source communities, the architecture is often not well established, and compliance to the architecture is often difficult to check.

Open source communities have often other ways of dealing with variability as what is in use within a product line. In many cases, the open source 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 open source development. Variation in time and in space is 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.

However, open source 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.

**Two processes**

Within open source communities, there is no clear distinction between domain and application engineering. However, both practices are done within the communities. A core group of experience developers in the community is often involved in those activities that we can assign to domain engineering. Central software that is used in the complete software suite of the community is build by the core developers. Developers that 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 open source developments, and this is also one of the advantages of inner source development, see section.

The scope of the open source software 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 mechanism are allowed and which not, which also reduced the consistency of the different parts.

**Conclusions**

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

Technical problems may occur if the architecture of the open source is not compatible with those of the product line. The company may use it involvement to influence the open source community, or it may incorporate 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 open source development. In order to avoid some of the disadvantages of open source 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 open source components are a part of the product line.
References


Biography

Dr. Frank van der Linden works at Philips Healthcare CTO Office. He received his Ph.D. in Mathematics in 1984 at the University of Amsterdam. Since he is working for Philips. Initially his interest was in program semantics, formal methods and automatic theorem proving. Later his interest shifted to software development methods and architecture, mainly object-oriented and component-oriented methods. Since 1991 he got involved in a product line development within Philips. This stays his main interest up to now.

Frank was project leader of the three successive ITEA projects on product line engineering, and successively on distributed development, including open source 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 open source practices in embedded system industry. In this project he has organized several workshops on open source 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.
