From SPLs to Open, Compositional Platforms

Jilles van Gurp & Christian Prehofer
Smart Space Lab
Nokia Research Center
Helsinki, Finland

Abstract. *In this position paper we reflect on how software development in large organizations such as ours is slowly changing from being top down managed, as is common in SPL organizations, towards something that increasingly resembles what is happening in large open source organizations. Additionally, we highlight what this means in terms of organization and tooling.*

Trends and Issues

Over the past decade of our involvement with Software Product Lines, we have seen the research field grow and prosper. By now, many companies have adopted SPL approaches for their core software development. For example, our own company, Nokia, features prominently on the SEIs Product Line hall of fame [SEI 2006]. Recently, we [Prehofer et al. 2007], and others [Ommering 2004] have published articles on the notion of compositional development that decentralizes the development of software platforms and products. The motivation for our work in this area is that we have observed that the following trends are affecting software development:

- **Widening platform scope and more diverse products.** As "victims" of their own success, successful product lines allow for the creation of an ever wider range of products. Necessarily, these products have increasingly less in common with each other. Particularly, they are likely to have substantial product specific requirements and require increasing amounts of variability in the platform provided features to deal with conflicting and overlapping requirements in the base platform. In other words, the percentage of functionality shared across all products relative to the total amount of functionality in the platform is decreasing. At the same time, the percentage of platform assets actually used in any particular product is also decreasing.
- **Platforms stretch over multiple organizations.** As platform and product development starts to span multiple organizational entities (companies, business units, open source projects, etc), more openness towards different and conflicting requirements, features, roadmaps and processes in different development entities is required. This concerns both open source software and commercial platforms that are developed and productized differently by third party companies.
- **Time to market and innovation speed.** While time to market has always been a critical issue, it is particularly an issue with the growing size and complexity of Software Product Lines. In general, large scale software projects tend to have longer development cycles. In the case of Software Product Lines that have to cater for more and more heterogeneous products, length of development cycles

Dagstuhl Seminar Proceedings 08142
Combining the Advantages of Product Lines and Open Source
http://drops.dagstuhl.de/opus/volltexte/2008/1540

> tends to increase as complexity of the work related to defining, realizing and testing new functionality grows increasingly complex. However, time to market of features does not only include the product line development cycle but also the time needed to do product derivation as well as the development cycles of any external software the Software Product Line integrates. Worst case is that a feature first needs to be integrated in one of these dependencies; then it needs to be integrated into the next major release of the Software Product Line before finally a software product with the new feature can be developed and put in the market.

We are seeing examples of this in Nokia as well. For example, Nokia has software development spread over several major phone platforms (S30, S40, S60 and Linux Maemo) and launches multiple products from each of those platforms every year. Interesting to note here is that Nokia has never really retired a mobile phone software platform and is actively using all of them. Roughly speaking, S40 evolution is in sync with the popularization of the notion of Software Product Lines since the mid nineties. It is indeed this product line that is featured on the before mentioned SEI SPL hall of fame [SEI 2006].

Development for products and platforms is spread over many Nokia locations all over the globe as well as a complex network of subcontractors, customers and supplying companies. Additionally, the use of open source software and the intensive collaboration Nokia has with many of the associated projects are adding more complexity here. Finally, time to market is of course very important in the mobile phone market. Products tend to be on the market for only short time (e.g. 6-12 months) and developing them from a stable software platform can take more than a year in some cases. This excludes time needed for major new releases of our software platform. Consequently, disruptive new features in the platform may take years to reach the market in the form of new phones.

The way large organizations such as Nokia manage and organize their software and platform development is constantly pushing the limits of what is possible with software engineering & architecting tools and methodology. Nokia is one of a handful of companies world wide that manage tens of millions of code across its product lines and products.

We see Software Product Lines as a way to develop software that has arguably been very successful in organizations like ours. However, we also note that increasingly development practice is deviating from practices that are prescribed by literature on Software Product Lines particularly with respect to centralized definition, control, ownership and management of software assets and products. Therefore, we argue that now the research community needs to adapt to this new reality as well.

The complexity and scale of the development organization increasingly make attempts to centrally manage it futile and counter productive. Conflicts of interest between stakeholders, bureaucracy, politics, etc are all affecting centralized platform and product decision making and can end up leading to unworkable compromises or delays in the software development process.

Additionally, it is simply becoming impossible to develop software without depending on at least some key open source projects. Increasingly the industry is also participating as an active contributor in the open source community. Arguably, most of the open

source community now consists of software developers sponsored in some way by for profit organizations. For example, Nokia is a very active participant in the mobile Linux community (the Maemo Linux platform) and ships products such as the N810 internet tablet where the majority of lines of code is actually coming from externally owned and run open source projects and even direct competitors (e.g. Intel and Motorola).

This changes the game of balancing product and platform requirements, needs and interests substantially from what is generally assumed in a classical SPL context where a single company develops both platform and products in house and where it is possibly to drive both product and platform development in a top down fashion. This simply does not work in a context where substantial amounts of critical software in a product are coming from external sources that are unwilling / unlikely to take orders from internal product managers or other types of executives external to their organization.

Effectively, this new reality necessitates a different approach to software development. Rather than driving a top down decomposition of products and features and managing development and software assets per this hierarchy, as is very much the consequence of implementing practices advertised in SPL literature, we propose to adopt a more compositional style of development.

Compositional Development

In our earlier work [Prehofer et al. 2007], we outlined an approach to adopt a more compositional approach to development. Rob van Ommering has argued along similar lines but still takes the traditional perspective of a (large) company managing a population of products [Ommering 2002][Ommering 2004]. However, what we propose here is to further decentralize development and organize similar to the open source community where many independent development teams of components, framework and product owners are working together. Each of those teams is acting to represent their own interests (and presumably those of whomever they work for). Their perspective on the external world is simply that of upstream and downstream dependencies. Downstream are the major users and customers that use the software the team produces. These stakeholders act as primary source of requirements and probably also funding. Upstream, teams operate that produce software required for using and developing the software. These teams in turn depend on their downstream dependencies and funding.

This decentralized perspective is very different from the centralized perspective and essentially allows each team to optimize for what is required from them downstream and what is available to them upstream. For example, requirements for each team come primarily from their downstream dependencies. Since there is no central controlling entity that dictates requirements, picking up these requirements and prioritizing them is very much the task of the teams themselves. Of course, they need to do so in cooperation with their downstream dependencies. Generally, especially when crossing organizational boundaries, requirements are not dictated but rather the development teams try to asses the needs of their most important customers.

Organization

As Conway's Law [Conway 1968] predicts, the architectural decomposition of software is reflected in organizations. In many open source communities, project team dependencies reflect the architecture decomposition of software into packages, frameworks, libraries, components, or other convenient units of software decomposition. Obviously, without at least some structure and management in place, the approach advocated here results in total anarchy, which is not a good organizational model to accomplish anything but chaos.

Again, we look at the open source world where organizations such as Ubuntu, Eclipse, Apache and Mozilla are driving development of thousands of projects. Each of these organizations has a surprisingly sophisticated organizational structure that comes with rules, best practices, decision making processes, etc. While there are no binding contracts enforcing these, participants in the community are required to play by the rules or risk being ignored.

In practice this means, participants voluntarily comply with practices and rules and take part in what is often called a meritocracy where important decisions are taken by those who have the merits to do so. Generally, this requires a track-record of making important contributions and having the trust of the community. For example, in the Eclipse foundation, which was founded by IBM, this means that individuals from some of their major competitors such as BEA and Red Hat actually lead some of the key projects under the eclipse umbrella. These individuals are essentially trusted by IBM to do the right things even though they work for a major competitor.

Organizations such as Eclipse exist to represent the common interests of the project teams they are composed of. For example the eclipse foundation, which is very much a corporate driven (and financed) institution, represents a broad consortium of stakeholders that covers pretty much the entire spectrum of Java (and increasingly also non-Java) enterprise, desktop and mobile/embedded software related development tooling. In the past two years, they have organized two major, simultaneous releases of the major projects. In their latest release, which goes by the name of Europa, they managed to synchronize the release process of around 20 of their top level projects which are collectively developed by thousands of developers coming from dozens of companies. Many of these companies are competitors. For example, BEA and IBM are directly competing in the enterprise market and major contributors to multiple eclipse projects.

What this proves is that the way the Eclipse Foundation organizes development is extremely effective and scalable because it involves dozens of organizations and hundreds/thousands of individuals producing, integrating and testing an enormous amount of new software in a very short time frame. Organizing like this brings in the necessary flexibility to seamlessly work with numerous internal and external teams and acknowledges the reality that even internally relations between teams can be complex and difficult to manage centrally.

Tooling

A consequence of decentralizing is that aligning the use of tools across development teams becomes essential. When collaborating, it helps if tools between teams are at least similar and preferably compatible/the same. SPL research has over the past few years focused on tooling for variability management, configuration management and requirements management. However, getting these tools adopted and using them effectively in a context of thousands of software development teams that are collaborating is quite a challenge; especially since many of these tools are either in house developed or only used in a handful of companies.

Tooling in the open source community tends to focus on the essentials. That being said, the OSS community has also produced many development tools that are now used on a massive scale. For example, Mozilla has had a pioneering role through their contribution of important tools such as Bugzilla and Bonsai (bug tracking and build monitoring). The whole point of the Eclipse foundation seems to be development tools. Additionally, they have a project called equinox that implements a very advanced framework that provides many interesting variability technologies and has put into mainstream use notions of using API versioning and provided and required interfaces on components. In short, there seems to be a gradual migration of SPL like tool features to mainstream tooling. Additionally, eclipse is of course a popular platform for developing such tooling in the research community.

Conclusions and Future work

In this position paper we tried to highlight a few of the key issues around the ongoing trend from integrational development towards a more open ecosystem where many stakeholders work on many pieces of software that are integrated into products by some of the stakeholders. We are currently working on an article about what it means to go from a software development practice to a compositional approach in terms of organizational models, practices and other aspects. In that article, we will list a number of practices that we associate with compositional development and evaluate these against practices in open source communities as well as selected SPL case studies from the research community.

Arguably, SPLs have vastly improved software development in many companies over the past decade or so. Therefore, the key issue for the next decade will be re-aligning with the identified trends towards larger software development ecosystem while preserving and expanding the benefits that SPL development have brought.

We do not see compositional development vs. SPL development as a black and white kind of thing but instead regard this as a wide spectrum of development practices that each may or may not be applied by individual companies. The more they apply them, the more compositional their development becomes. In any case, the right set of practices is of course highly dependent on context, domain, stakeholders, etc. However, we observe that in order to scale development and in order to work with hundreds or even thousands of globally and organizationally distributed software developers effectively, it is necessary to let go of centralized control. Compositional development in this open environment is vastly more complex, organic, and so we believe, more cost effective.

References

[Conway 1968] M. E. Conway, How do committees invent, Datamation, 14(4), pp. 28-31, 1968.

[Ommering 2002] R. van Ommering, Building product populations with software components, proceedings of Proceedings of the 24rd International Conference on Software Engineering (ICSE 2002), pp. 255-265, 2002.

[Ommering 2004] R. Van Ommering, Building Product Populations with Software Components, Ph. D thesis, University of Groningen, 2004.

[Prehofer et al. 2007] C. Prehofer, J. van Gurp, J. Bosch, Compositionality in Software Platforms, in A. De Lucia, F. Ferrucci, G. Tortora, M. Tucci eds., Emerging Methods, Technologies and Process Management in Software Engineering, Wiley, 2008.

[SEI 2006] Software Engineering Institute, Product Line Hall of Fame, http://www.sei.cmu.edu/productlines/plp_hof.html, 2006.