Patterns for Product Line Engineering

Christa Schwanninger, Michael Kircher, Siemens AG

Introduction

Software Product Line Engineering (PLE) has become an important way of building and reusing software. In PLE, the focus is shifted from building isolated products to building families of related products, while reuse is discussed not at an individual object level , e.g. libraries or components, but as a whole: organizational, process-wise, and also life cycle wise end-to-end from requirements to actually deployed variability at the customer. We consider only those platforms PLE efforts that are planned and prescribed to be reused in a dedicated product portfolio or to build solutions in a specific market segment, hence typical general purpose technology platforms, such as Eclipse, JBoss, or .NET, do not fall into this category.

While a lot of literature on PLE exists, there is still no consistent collection of easy to use and practical patterns. The patterns described in this paper build on the existing literature, like the book of Clements and Northrop [3] or Klaus Schmid's PhD Thesis [8]. With this initial collection of patterns we hope to motivate more patterns on PLE. The patterns describe how to centralize and objectify decision making, how to partition the domain into several sub-domains, how to set up your releases, and how to avoid dangerous complexity.

This paper is intended for any leadership staff concerned with the product portfolio content, organization, process, technology, architecture, or actual project execution of a PLE initiative. We assume some background in PLE. Refer to [1] or [2] for a more detailed introduction.

PLE has to be adapted to the specific organization at hand to be successful. Many factors influence how PLE is executed in a specific organization. The ones with the largest impact are: size of the organization, kind of business, number of product lines, maturity of the domain, and process discipline. While those factors influence PLE, they will in return get influenced by introducing PLE.

The patterns in this paper apply largely to mid- and large-scale organizations, as they typically have several product lines with several involved sub-organizations with varying business goals, which makes decision making a multi-dimensional optimization problem. Organizations with only a dozen involved staff do not have such issues; instead most of the challenge becomes a technical issue of how to deal with variability – which is not the focus of this paper.

The patterns deal with setting up the organization, the process and methodological best practices. This paper does not provide any analysis advice when to use a PLE effort or not, instead it assumes that a return on invest (ROI) analysis has been done in advance and that a PLE approach proved superior over classical one-off development.

For readers who are not familiar with PLE terminology, please read the glossary at the end of the pattern collection.

Scoping Patterns

Central Decision Making

Context

An organization is adopting product line engineering. Up to now it developed products or solutions in parallel, with copy&paste reuse, technological platform usage, or using a standardized infrastructure. Every product development group was responsible for deciding on the scope and release plan of its product, a task called product management. The organization wants to increase the benefit from reuse among products/solutions. The development of a set of reusable core assets is started. It has to be decided what these core assets should be: the scope of the reuse infrastructure. The reason for starting the product line is to maximize the return on investment (ROI) for building reusable assets for the whole organization. Figure 1 shows the desired state, one domain engineering activity that produces reusable assets and several application engineering activities that reuse these assets.

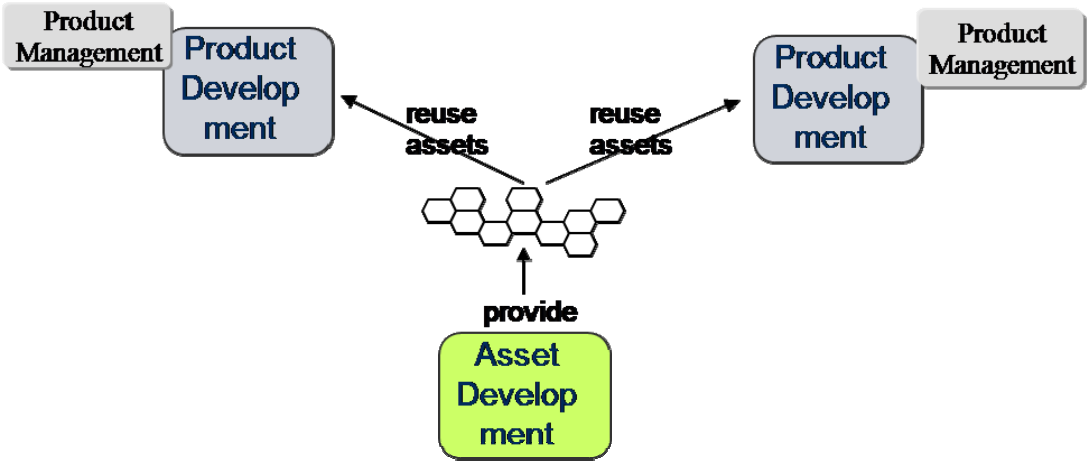

Figure 1: Relationship between domain and application engineering

Problem

Who decides what assets should be developed as reusable core assets and what assets should be kept product specific?

- Every product manager typically pushes his own products. Reuse is only interesting if it saves effort or otherwise benefits his products. Supporting his own products is what he is trained to do.
- Without moderation either the strongest in a group of product managers dictates what will be implemented as reusable core asset or the product managers agree on some compromise, e.g. everybody gets their most important features in. This selection is likely not maximizing the profit of the whole organization.
- Core asset development is confronted with all product managers and their corresponding priorities. For deciding about a product portfolio strategy deep market knowledge is required, which typically is not available in development. Development is inclined either to

give in on the pressure of the strongest or decide by themselves what has the highest priority, often according to purely technical criteria.

Solution

Centralize decision making regarding scope and priorities. The best way to do this is introduce a new role in the organization, the product line product manager. His goal is to optimize the product portfolio, the solution scope, which shall be supported by the shared core assets. For bigger organizations the product line product management can also be staffed with several individuals.

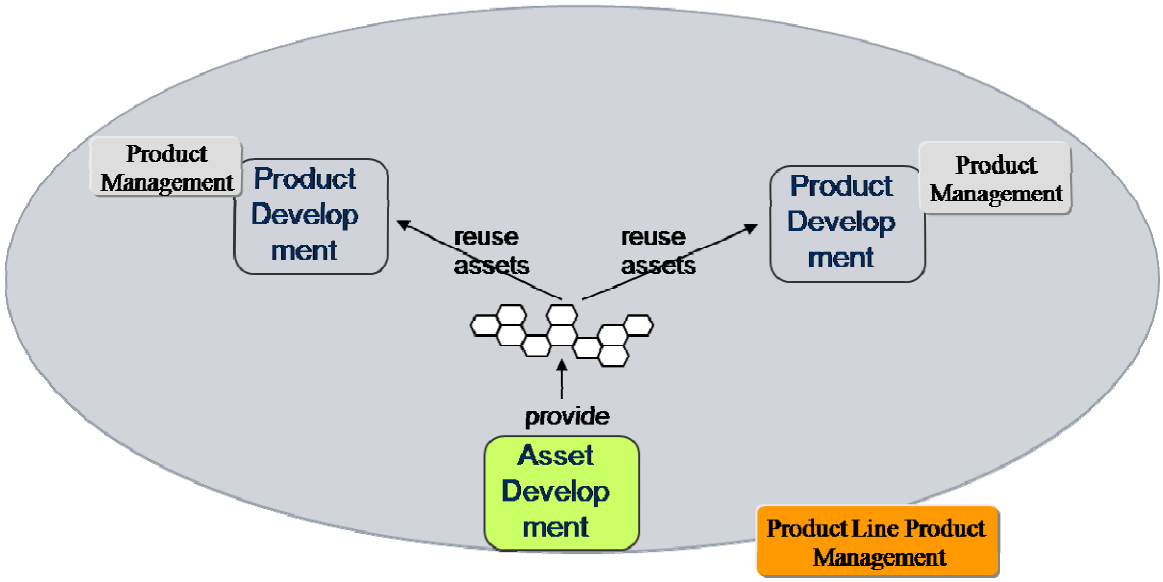

Figure 2: The Product Line Product Manager is responsible for the whole product line.

Core asset development should not decide about feature priorities. However, core asset development estimates the effort for features and analyzes dependencies among features, which is essential input for deciding on what should be implemented as core asset. Be aware that product management alone is not sufficient to find the overall optimized scope for a business, because constraints like quality, time, and cost might not be represented sufficiently; see the patterns *Balance Constraints* and *Collaboration between Problem and Solution Experts*.

The Product Line Product Manager is responsible for the overall product portfolio, therefore is neither part of any product development team, nor of the core asset development team. There might be conditions that render a dedicated, independent role not possible, e.g. if the organizations are separate profit centers with their own business targets and no operational responsibilities are at a common management level. The Product Line Product Manager would in this case be part of the core asset developing organization, which makes his position in the overall organization weaker. It requires additional process support to deal with this situation. This sub-pattern language is yet to be mined.

Rationale

For effective core asset development a single responsibility for prioritization and scoping is required. In order to prioritize features and scope the product line to maximize the economic benefit of the

whole organization, knowledge about the market(s), competition, and customers is important for that role. Development typically lacks the full understanding of customer and business value.

This pattern works best if the core assets actually implement domain specific – customer visible – core assets. Such assets typically heavily influence the actual product portfolio. In contrast, if the core assets are mainly technical infrastructure assets, the product portfolio is only marginally influenced by the core asset scope. In such cases the product line architect role instead of the product manager, may perform the scoping, because he can adequately judge the scope of the mainly infrastructure core assets. The Product Line Architect then fulfills the role of the Product Line Product Manager as well.

The Product Line Product Manager role resolves the conflict amongst product managers and between product management and core asset development. We saw organizations that suffered from this conflict. In the end, the product lines broke apart and the whole effort of doing PLE was seen as a failure.

The Role Product Line Product Manager has to know how to manage the conflicting interests between his role and the product managers. This conflict can be resolved with *Comprehensible Decision Making*.

References and Known Uses

This pattern is applied at a Siemens Healthcare group developing an imaging platform. The platform customers are different imaging product lines, which were heavily involved in defining the scope of the new platform from the beginning. However, there was no dedicated product manager for the platform. Hence, far too many features were assigned to the platform. There was no clear prioritization, and despite working hard, the platform customers were never satisfied with the scope of the platform. Two years later the organization installed a platform product management. The platform product manager is now responsible to define a common vocabulary and to drive scoping of core assets in collaboration with the product managers of the different products. He moderates scoping sessions and makes the scope and the rational for the scope transparent to product developments. The organization managed to regain trust from the product groups. Installing a product line product manager improved the situation considerably for both, domain and application engineering.

A separate group at Siemens postal automation started a first platform about 10 years ago to benefit from reuse among their customer projects. There was no platform product manager, even no explicit scoping process. The platform users were not happy with the platform, quickly started to clone and own parts of the platform and the platform team was turned into a project team for one specific customer after three years. Today, the organization has a very strong product management that knows about the value of reusable assets and actively supports scoping and marketing of reusable assets. All bidding invitations go through the product line management group, that is a team of product line product managers and product line architects. The scoping is done in several steps: deciding which existing core assets will be offered to the customer, which core assets should be adapted and which should be built anew in case the customer project actually is won. Based on this the offer is made to the customer. When the customer accepts, the product line management team includes the effort for building the assets into the overall product line development plan.

Comprehensible Decision Criteria

Context

The pattern *Product Line Product Manager* was applied. A conflict potential is remaining between individual product managers and the Product Line Product Manager responsible for the whole product line portfolio.

Problem

The product managers responsible for single product/solution are a group of peers who have all conflicting goals, which is promoting the specific products they are responsible for. Agreeing on a scope for the reusable base assets means resolving conflicts that arise from these different goals.

- Every product manager typically pushes only his own products. Reuse is only interesting if it saves effort or otherwise benefits his products. Supporting his own products is what he is trained to do.
- Core assets often get special funding. The cost is shared between all products/solutions. Often product managers want to unburden their product/solution specific budget and put as much development effort as possible to the core asset development. This way more budget is available for making own products better, cheaper or earlier on the market..
- When the selection of what shall be a core asset and what shall be product-specific seems not credible, development will not respect the scoping decisions. Development will decide on its own priorities.

Solution

Define and communicate clear criteria for evaluating the value of features. The criteria should be derived from the organization's mission, vision, and strategy, because only then the decisions are in line with the business targets. Evaluate the cost and benefit for every feature and for every product according to these common criteria. Decision making boils down to calculating these values and deciding according to objective numbers.

The best way to do this is to define criteria for the benefit and the cost of each feature. Benefit criteria could be how much does the market request this feature or how well does it support the specific portfolio strategy. The cost side need not be an estimation of the development cost for the feature, since this may be hard to be calculated early in the process, but a mixture of estimated complexity, novelty, and effect on the current product line architecture.

Start with simple benefit/cost evaluation formulas and improve them over time. The product managers and architects that deliver the data for the formula typically differ in their accuracy and reliability of the data they provide. Install metrics comparing estimations with the later facts, e.g. estimated sales figures and actual sales figures for a product. The results are used as correction factors.

Rationale

Most of the potential conflict can be mitigated when every stakeholder shares their business value data and the result is merely calculated. The product line product manager drives the process and controls how fairly features are rated and improves the evaluation criteria and formulas.

References and Known Uses

The platform organization for imaging systems within Siemens Healthcare developed a formula for prioritizing features and calculating the cost of every feature. It is filled with information from product management and development to determine benefit and cost of every feature. Measures for a feature's benefit are for example whether it is innovative or just a "me too" feature and how often it would be reused, measures for its cost are for example the complexity and how crosscutting it is in the solution space. The results are taken to decide whether a feature should be implemented in the platform at all and what priority it should have.

Klaus Schmid suggests a similar procedure in [2]. In this context it was tested with two product lines of MARKET MAKER Software AG and with one product line in Bosch.

Separate Sub-domains

Context

The organization has to decide what will be developed as core asset and what will remain product specific – a typical scoping decision. Actually, the organization will also have to decide on a product portfolio that allows maximizing the benefit from reuse.

Problem

Budget and time for setting up a core asset base is limited. If finding out which assets are worth to be developed as core asset takes too many resources, the product line might die before it even starts.

- Not all areas in a domain are equally stable and therefore suitable for reuse.
- Scoping is a pure planning activity. It is not productive and accounts as negative item in the return on investment calculation for a product line.
- Often a market evaluation is an important pre-condition for scoping. The scoping activity has to be performed in a time frame that together with implementing the scope is short enough to catch the market window that was looked at in the evaluation.
- The features in the core asset base have to be selected in a way such that it is possible to form a consistent reference architecture. They cannot result in completely independent pieces of functionality in solution space. It must be possible to architect and implement a coherent chunk of functionality.

Solution

Divide the application domain in sub-domains that can be handled separately. The easiest way to identify sub-domains is to look at existing systems/products. Their division in subsystems is a good first subdivision of the overall domain. First estimate the reuse potential for every sub-domain based on past experience. Then continue closer investigations with one or two promising sub-domains. Find out what can be reused for these sub-domains and implement these assets.

The sub-domains might be application domain specific like telephony, short message service and data services in a mobile phone, or technical, like memory or power management.

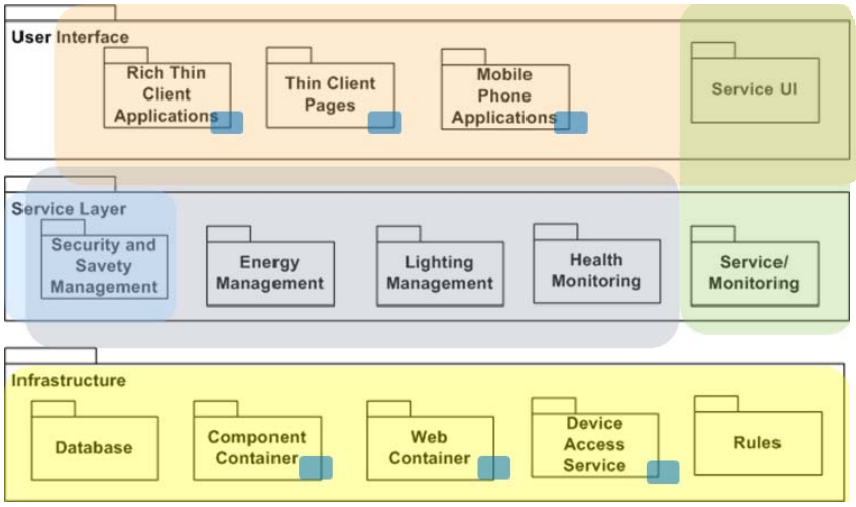

Figure 3: Consistent sub-domains may be vertical, horizontal, or scattered across the architecture.

Rationale

Implementing only core assets for one or two sub-domains gives early feedback to all stakeholders whether the organization and process changes for PLE were the right ones without bearing too high risk for the whole organization. With such potential quick wins the organization is motivated to extend the product line also to other sub-domains.

References and Known Uses

Car manufacturers typically divide their cars in sub-domains like chassis, engine, electrical system, or safety systems. This division is typically also mirrored in the organizational setup; similarly the configuration options for the customers are grouped in these sub-domains.

The features of the imaging platform in Siemens Healthcare are grouped into about 10 sub-domains that structure the problem as well as the solution space.

Klaus Schmid introduces a new approach for evaluating sub-domains, the domain potential assessment in [2].

Start with Product Feature Map

Context

An organization wants to benefit from reuse. The organization already implemented systems in the domain for a while. Existing products or solutions are available.

Problem

To benefit from reuse in current and/or future product portfolio the organization needs to find out which artifacts should be implemented for reuse.

- Taking already available assets and turn them into reusable assets by increasing the quality and adding variation points is a low risk option to get a set of core assets. However it is unclear which variants should be supported.
- Development knows what it takes to build certain functionality and what should be reusable. But still this is at most a projection from the past product portfolio.
- Domain analysis looks at sub-domains that presumably have reuse potential, identifies the commonalities and variants in each sub-domain and gives a feeling for reuse potential. However, if all variants possible in a sub-domain are considered, this will not support the business of the organization optimally.

Solution

Before the scoping of core assets starts, the products or solutions that shall be supported by these core assets have to be clear. Thus, the first step is to identify the products and market segments that shall be addressed by the product line. Next, the characteristics or features that are included in each of the products are listed to see the commonalities and variability between the products or market segments. This is the starting position for further commonality/variability analysis. The envisioned product portfolio restricts the list of possible features in each sub-domain. At the same time products get better shaped by deciding for each additional feature if it should or should not be in the product.

Rationale

If the product portfolio or market segments of interest are not the starting point for core asset scoping, the assets will typically be more generic than necessary. The advantage of product line engineering over simple reuse as propagated in the 1990s is that only the variability elementary necessary to support a specific business is built into the core assets and not more – see *Favor Platform Simplicity*. Core assets have to be build considering variability that is to be expected for future portfolios, but the variants should be clearly identifiable and evidently motivated.

References and Known Uses

Product/Feature maps are quite common in portfolio development. Car manufacturers like Audi and BMW present their portfolio in such lists.

Similarly, Siemens Industry Drive Technologies scope their product portfolios with the help of product/feature maps und use the customer visible part to present their product portfolio to customers, Figure 4 shows an example from http://www.automation.siemens.com/.

Explosion-protected Motors – Technical Data

Motors	Increased safety – "e"	Increased safety – "d"	Increased safety – "n"	Dust explosion protection	NEMA EX
Voltage and power ranges	0.12 - 165 kW	0.25 - 950 kW	0.09 - 1000 kW	0.06 - 1000 kW	1-300HP
	All commonly used voltages				230, 460 & 575 V with 60 Hz
Frame sizes and designs	63M - 315L	71M - 450	63M - 450	56M - 450	140 - 440
	All common construction types				
Rated speed and rated torque	750 - 3600 min^{-1} 0.3 - 10300 Nm	750 - 3600 min^{-1} 0.3 - 10300 Nm	750 - 3600 min^{-1} 0.3 - 10300 Nm	750 - 3600 min^{-1} 0.3 - 10300 Nm	900-3600 min^{-1} 1.5-1772 lb-ft
Application area	Ex-Zone 1 II 2G Ex e II T1-T3	Ex-Zone 1 II 2G Ex de IIC T1-T6	Ex-Zone 2 II 3G Ex nA II T3	Ex-Zone 21/22 Zone 21: II 2D Ex tD A21 IP65 T120°C Zone 22: II 3D Ex tD A22 IP55 T120°C	Class I, Group D, Class II, Groups F&G Division 1 hazardous areas

Figure 4: Example of Siemens Automation and Drives Portfolio

Klaus Schmid suggests this procedure in [2]. In this context it was tested with two product lines of MARKET MAKER Software AG and with one product line in Bosch.

In [3] Paul Clements and Linda Northrop suggest **"**Developing an attribute/product matrix", which is one pattern in their collection of PLE patterns. The patterns in [3] were mined from experience with consulting companies that introduced or improved a PLE approach.

Collaboration between Problem and Solution Experts

Context
An organization decides to build a core asset base. It tries to optimize the return on investment.

Problem
Neither product management, nor development alone is supposed to know all details around commonality and variability.

- Product management knows all facts around market, marketable features, value of features, and products. Further, it knows which features are common between products on a high level of abstraction. However, with more detailed analysis variability quickly increases, especially if non-functional requirements vary. Such variability can not be spotted on a high level of abstraction. Hence, the complexity and cost for features cannot be determined to the full extent by product management.

- Development knows the cost for developing assets, both for product-specific and core assets. Development has the 'solution space' knowledge for determining the actual variability and dependencies between variation points. However, development lacks the knowledge about the value of features for customers.

Solution
Let scoping be lead by product management, but make it an iterative and collaborative effort between product management and development. Product management initially sets up the product map with feature assignments to products. Development analyses the feature in depth by drafting designs and identifying technical dependencies between variants. It might for example be necessary to include certain features into the core asset base that would not have been selected, if it were only based on their business value for the customer, but because they are necessary to develop higher prioritized features efficiently. The additional dependencies and cost are fed back to product management that re-estimates the cost/benefit ratio for features and products. Both parties iterate over the product portfolio commonly and alternately.

Rationale
While product management is responsible for setting up a product portfolio that optimizes the business of an organization, development is responsible for minimizing the cost of a product portfolio, potentially optimizing reuse and to decide whether a feature set can be implemented in a consistent platform architecture at all. Neither can product management estimate efforts reliably, be it for product-specific or core assets, nor can development decide on priorities of products or features. The feedback from development often influences the product portfolio directly. Product management is made aware of existing knowledge or assets that can be marketed by changing the portfolio.

References and Known Uses
For deciding which features go into the imaging platform at Siemens Healthcare, data from product management is used to calculate the value of a feature and from architects to get the cost of features.

The Siemens postal automation group and part of the hot rolling mill automation have product line teams installed that consist of product management (or sales) and architects. Together they decide whether a feature is developed in the platform or only for one specific solution.

In [3] the risks of scoping include “scope includes the wrong products”. Further, the authors describe that the reason typically is the missing collaboration between product management and development.

General Patterns

Regular Platform Releases

Context

Only a limited number of knowledgeable people exist per core asset that can change or extend a specific core asset. Core assets face high demand of change and extension by the products that are reusing them. How do you mitigate the pressure?

Problem

Accepting the high pressure, specifically regarding time, can result in several shortsighted solutions.

- Several branches of the same code parts for different products in parallel. The consequence is high merge efforts between the branches and overhead of switching between the branches for individual developers.
- High flexibility of the code parts to accept any variant, e.g. overly 'strategized' design or creating a #ifdef hell.
- In urgent cases application developers are inclined to fall back to copy & paste reuse, or even re-write the functionality product specific.

Solution

Therefore, prioritize your total feature backlog considering when reusable components have to be available for which products and provide high quality releases regularly, e.g. every 3 months. If a platform user urgently needs any change or extension, the feature backlog can be reprioritized accordingly.

Rationale

Regular releases alleviate the high pressure that typically leads to multiple branches; hence branches can typically be avoided or at least minimized in their number. In order to be accepted by the platform users the quality of the regular releases must be sufficiently high. Fully automated regression test suites that are executed on every change, up to the extent of Continuous Integration, secure the quality while minimizing extra efforts.

However, providing regular releases challenges the organization to define a proper versioning strategy that regulates which releases are compatible, e.g. all minor releases, and which releases contain incompatible API changes, e.g. major releases.

Favor Platform Simplicity

Context

Source code of reusable assets gets complex very easy. Often it is the well-intended flexibility designed and implemented by developers that increases the complexity.

Problem

Platforms are intended for longevity, but the high complexity lets them die the too-complex-and-not-maintainable-any-more dead faster than expected. How do build the simplest possible platform?

- Platform development is often defined as providing the software with enough variability mechanisms, such as heavily applying the Strategy pattern and allowing for declarative programming via configuration files.
- Hard wiring only the very current requirements might lead to monolithic design, hence badly maintainable and evolvable software, too.

Solution

Therefore, establish the mindset of the beauty of simplicity. Favor simplicity instead of flexibility. Firstly, differentiate between the inherent complexity of a problem and the accidental complexity. At all times be aware of the inherent complexity of a problem.

For example, the demanded variability of an application domain is inherent complexity, while the solution approach, e.g. flexibility, dynamicity, or genericity are typically sources of accidental complexity. In many regards this also means that the use of the classical GoF design patterns has to be limited to the bare minimum. Basically it is “Do the simplest thing that could possibly work” [9], while not obstructing future potential extension.

Reducing the inherent complexity is only possible by reducing the required variability, which in turn means reducing the scope of the product line. Accidental complexity can be avoided by demanding that all flexibility in the platform has to be justified by variability in the chosen scope of the problem space.

Rationale

While the solution might sound trivial, it is not. With all the intrinsic motivation and creativity of a typical software developer, only lots of discipline and experience will allow him to actually focus on the essentials. Simplicity should guide itself on the actually demanded variability, the scope, of the product line, not the general interest in technology, preparing for all potential variations or the application of the most patterns. So in some sense this pattern is meant as reminder of adhering to best practice and pragmatism. In hardware development creating flexibility is much more expensive compared to software, hence overly-flexible design in hardware is less of an issue.

The role of the software architect, often also called technical lead, is crucial in this aspect. He guides the organization by establishing the right mindset. Continuous review of design decisions by development teams allows him give input and potentially correct misguided design decisions.

References and Known Uses

At the imaging platform development effort in Siemens Healthcare this pattern is applied to ensure the longevity of the platform as well as the development efficiency of the applications on top. The group introduced the tracking and tracing of dependencies between solution and problem space variability. Due to the regulatory requirements, which already require some extent of tracing, this step was not too difficult. The additional tracing causes some overhead in maintaining, but on the positive side forces developers to design for simplicity.

Balance Constraints

Context

Strategic and operational decisions need to be made. Every project follows the constraints of the project management triangle [4]. Mapped to the area of software development they are defined as:

- Scope – the functionality especially the unique selling points offered by a product
- Quality – the developmental as well as operational qualities, both determining the total-cost-of-ownership
- Time – the duration it takes to develop the product, hence the time-to-market
- Cost – the cost of personnel, mostly wages, and costs for the infrastructure, like PCs, servers, building. Typically the largest part of the cost is directly related to the development time and involved staff.

Problem

Individual roles in a software development organization typically span not all four categories, hence decisions can easily be concentrated on scope, quality, time, or cost only. The classical separation is as follows.

- Project management typically feels most responsible for the triple constraint time, cost, and developmental quality, such as regulatory compliance and availability of all documentation artifacts.
- Product management typically feels most responsible for scope, time, and customer visible quality, like performance and usability.
- Technical leadership (architects) typically feels most responsible for scope and developmental as well as operational quality.

Letting only one or two of the roles decide on a strategy leads to potentially unbalanced decisions.

Solution

With every decision make sure you have a balance between the four constraints through the involved roles. A typical steering committee should consist of representatives from all streams of leadership (project management, product management, technical leadership). Only this way you can make sure that sound decisions with proper risk assessment are being made.

Rationale

The partitioning between project management, product management, and technical leadership is the classical separation. Agile methodology approaches provide a slightly different partitioning, e.g. Scrum product owners are typically broader set up and care about time, scope, and quality. The Toyota approach [10] would even go so far to integrate all constraints into a single person: the chief engineer. Of course in this case there is no need anymore to balance the constraints, besides making sure you pick the ‘right’ chief engineer for your business success.

References and Known Uses

Many groups in Siemens, among them the postal automation group and the hot rolling mill automation group, intensively apply this pattern to their advantage. They are well aware of the different forces of a project and use the respective perspectives to the advantage of creating innovative and excellent products.

Acknowledgements

We thank our shepherd Jason Yip for his elaborate comments and his patience. Special thanks also to our Writer's Workshop group at EuroPLoP 2009: Rene Bredlau , Eduardo Fernandez, Claudius Link, Klaus Marquart, Dietmar Schütz, Markus Völter, and Alain-G. Vouffo Feudjio. In this paper we base on the foundations laid by authors like Klaus Schmid, Linda Northrop, and Paul Clements. Our learning continued out of their product line engineering experiences.

Terminology

Core asset – A set of reusable artifacts, i.e. requirements, domain models, architecture, patterns, concepts, documentation, test cases, budget plans, work plans, process, tools, workflows and code assets.

Platform – The sum of all core assets is typically called the product line platform. However, the term platform is sometimes also used for a base of technologies on which other technologies or processes are built [1], e.g. an operating system, middleware or component container. In this paper we refer to the first interpretation.

Mission, Vision, Strategy – Mission = why do I exist? Vision = where would I like to be? Strategy = how would I like to get there?

Total-cost-of-ownership (TCO) – The sum of all costs related to developing, maintaining, supporting, installing, servicing, and owning a certain product of the providing as well as receiving party.

Time-to-market (TTM) – The time it takes to deliver a product from initial conception to actual availability on the market to be bought by customers. The longer the delay the higher the chances for the competition bring their products into the market and endanger the potential market share for the own product.

Project management triangle – The triangle describes the constraining relationship between scope, time, and cost assuming the quality to be fixed. All are interrelated; you cannot change any of them without adopting the others [4].

Scoping – Scoping [1][3][5] is an activity to find the boundaries of a product line that can be divided into three sub-steps [6][7]: product line scoping, scoping the domain, and scoping the reusable assets (platform). The objective of product line scoping comprises defining the set of products and the main features to be included in the product line. Domain scoping represents the process of identifying appropriate boundaries for the sub-domains of the overall product line. Scoping the assets results in the decision which of the required functionality a development organization should implement as reusable base assets and which it should consider product specific. For simplicity we can stick to the

following definition: Scoping means setting the focus of reuse on the functionality that promises an optimal return on investment [8].

Application and Domain Engineering – Domain Engineering (DE) is the sub-process of PLE that is responsible for building and managing reusable base assets, while Application Engineering (AE) is about building products or solutions from these base assets. Variation in organization forms – PLE organizations may be split in a Domain Engineering part that is responsible for implementing the core assets and several Application Engineering parts that build products/solutions based the core assets and adding product specific features.
Alternatively, PLE organizations may as well be divided along sub-domains that do both, development of core assets and product/solution development. This setup is common in mature PLE organizations, mostly for solution development. Such a division is very efficient since no handover between the two disciplines is necessary. How to set up the best organization for a product line would be a pattern language on its own.

Proactive evolution – Core assets are identified up front and developed for reuse.

Reactive evolution – Creation of core assets based on existing, previously product-specific assets.

References

[1] Pohl, Böckle, van der Linden, Software Product Line Engineering, Springer, 2005

[2] Klaus Schmid, Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines, Dissertation, Fraunhofer IRB Verlag, 2003

[3] Clements, P. & Northrop, L. Software Product Lines: Practices and Patterns, Addison-Wesley, 2002

[4] http://en.wikipedia.org/wiki/Project_triangle

[5] K. Czarnecki and U. W. Eisenecker, Generative Programming. Methods, Tools, and Applications. Amsterdam: Addison-Wesley Longman, 2000.

[6] Jan Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison- Wesley Reading, 2000.

[7] Klaus Schmid, Steffen Thiel, Jan Bosch, Susanne Johnsson, Michel Jaring, Bernhard Thomé, Siegfried Trosch, Scoping, EASPS consortium wide deliverable CWD1.2.4, 2001.

[8] Klaus Schmid, A Comprehensive product Line Scoping Approach and Its Validation, In Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), ACM Press, 2002, pp. 593-603

[9] XP guidance "Do the simplest thing that could possibly work"

[10] http://www.poppendieck.com/leadership.htm