Combining Systematic Reuse with Agile Development – Experience Report

Michael Kircher
Siemens AG, Healthcare Sector, SYNGO
Hartmannstr.16, D-91052 Erlangen
Germany
+49 9131 84-7392
michael.kircher@siemens.com

Peter Hofman
Siemens AG, Healthcare Sector, SYNGO
Hartmannstr.16, D-91052 Erlangen
Germany
+49 9131 84-4303
peter.hofman@siemens.com

ABSTRACT

This paper documents the experiences of Siemens Healthcare in mastering challenges when transitioning a large-scale dispersed platform development organization to Agile. Product Line Engineering aims at increasing productivity through reuse, but since strategic reuse requires up-front decisions, is also seen as heavy weight and process driven. Agile development on the other hand is perceived as lightweight, change friendly, but at the same time neglecting long term strategic planning. With this paper we want to report on our experience combining both approaches, PLE for strategic reuse and agile principles for achieving steady progress while still leveraging the long-term benefits. The key was to build the foundation on the common best practice of 'feature-orientation' present in flavors in both disciplines. Feature-orientation allowed merging both disciplines into a holistic approach that blends the benefits of product line engineering with those of Agility – resulting in improved product delivery, as well as employee and customer satisfaction.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *Life cycle, Productivity, Software Process Models.*

General Terms

Management, Economics, Experimentation, Human Factors.

Keywords

Agile, Lean, Hierarchical Platform

1. INTRODUCTION

Siemens Healthcare is one of the leading providers of biomedical technology, offering a complete spectrum of diagnostic technologies from in-vitro diagnostics to medical imaging and information technology. Over 45,000 employees worldwide develop trend-setting innovations focusing on supporting their customers' clinical and administrative workflows.

All products containing medical imaging functionality share a considerable number of commonalities. Even if the concerned products belong to completely different product lines, e.g. medical scanners or advanced visualization systems, they essentially have to fulfill the same requirements in their imaging functionality and may differ only in their configuration.

Supporting a common platform for these various products of different product lines can generate benefits from the financial, time-to-market and usability points of view.

However, experience shows that such big platforms are often bottlenecks in product lines; development is inefficient because feedback comes late when a component finally is used in a product; features do not meet the customers' expectations because platform developers requirements differ from product development and are far away from the end customer, which requires rework and threatens all the expected benefits we wanted to achieve with the platform.

We address the challenges with a combination of agile development and product line engineering in our platform development organization.

2. Challenges

2.1 Challenges in Scoping

Siemens Healthcare supports several product lines of several business units that build healthcare imaging products using a common platform. Image management and image visualization has a lot of reuse potential; the requirements of a number of sub-domains do not differ much from one product to another.

When we changed our processes to more structured reuse some years ago, a major challenge was to come up with the scope for each platform release that had to support several products of several product lines with conflicting interests. For one product line a feature from the platform is an essential enabler, for another product line it is at best a "nice to have" feature.

Deciding whether a stakeholder request really should be developed by the platform is a key challenge, given the amount of requests that arrive each day. There is a good chance that the resulting scope supports every product line a little bit, but not enough to really benefit each one, or that the resulting feature set cannot be implemented within a consistent reference architecture at all.

With endless lists of requirements mixed together from different stakeholder groups and without deep understanding of each others business goals, scoping sessions often become endless negotiations, where every application engineering group tries to get as many requirements into the platform as possible, since this unburdens resources for product specific development. To get the support of other groups, requirements are easily marked as

SPLC '12, September 02 - 07 2012, Salvador, Brazil

commonalities, even if all but one groups only needed the requirement for their own products in later releases or these requirements were not essential for their products.

For agile development a defined product backlog is essential, especially for a platform that has to support many products. So, how could the different stakeholder groups – the different product lines – best agree on a common scope for the platform?

2.2 Challenges in Organizational Decomposition

Over the past decade multiple organization forms had been tried. The latest approach was to decompose the organization in departments that reflect the subsystem structure of the architecture. With this it was easy to focus on the subsystem and optimize their scope, design, as well as development. The teams could act consistently across the different roles of the sub-organization, as every role participating in the value stream of the sub-system was close. The world of the subsystem was fine – but not the world of the system.

After several development iterations it was noticed that the overall system consistency suffered. Also, as the subsystems were aligned in parts according architectural layers and not according to features or feature groups, end-customer features required deliveries from multiple departments, multiple teams and with this created severe inefficiencies because of the communication overhead over department fences. We needed to find a way how to make the development teams as much independent of each other as possible to work efficiently, but at the same time produce end-customer features consistent both, from usability, as well as from design perspective, within the feature and across the architecture.

2.3 Challenges in Process Efficiency

We followed an iterative process that prescribed a layered integration. The idea was to finish architectural layers bottom-up one after another. Every lower layer should provide a sound foundation for the next higher layer, until once every 6 weeks the system would be complete, to be fully integrated and tested.

As medical devices have to fulfill certain pre-requisites with regards to patient safety, development organizations have to adhere to regulations. In essence those expect that you have made very conscious and documented choices how to come from design input – requirements – to a solution design. The choices have to be documented because the processes and evolution of the product has to be reliable and repeatable –product creation should not be just a coincidence.

Because of those regulations a certain level of process descriptions and development documentation is necessary. However, if regulations become too heavy weight, e.g. if an organizations does more than absolutely necessary, it can become inefficient easily and kill creativity and motivation by drowning the organization in document writing, signing, reading, reviewing, revising, while not actually creating value – or creating little value after lots of delays involving activities around functionality that potentially never gets shipped to a customer: the organization produces waste.

While there was some uncertainty whether agile processes and practices would scale to a large-scale, regulated platform development, the need for a change was obvious – at latest when people were confronted with the value stream analysis results – as they are typically done in lean analysis.

To address the three challenges of Scoping, Organizational Decomposition, and Process Efficiency we applied Product Line Engineering (PLE) and Agile/Lean Development best practices jointly.

The feature-oriented [6], feature-driven [2] approach with the feature model structuring the overall domain and scope of our software product lines became the foundation on which further PLE and Agile benefits could be leveraged.

3. Feature-Orientation

In this paper we use the term “Feature Orientation” to refer to the sum of understanding about features, their structuring, and development as defined by

1) Feature Oriented Software Development [4]
2) Feature-Driven Development [2] as well as
3) Feature Modeling [3]

The term feature is widely used in the context of product line engineering but also agile development. Products can be thought of by being a composition of a set of features in order to fulfill customer requirements. In [6] a feature is defined as an “prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system” while according to [4] a feature is “an increment in program development or functionality”. To our opinion both definitions do not contradict but much more extend each other, especially when blended.

Feature Oriented Software Development is a paradigm connected to Software Product Line Engineering. One of its key principles is the composition of products from individual features. Features are observed holistically including their impact on the solution space, e.g. all aspects and artifacts are considered in feature-based program synthesis.

Feature-Driven Development on the other side is more connected to the agile methods where product value creation is “driven from a client-valued functionality (feature) perspective”.

Feature modeling was introduced in [6] as part of the domain analysis and domain-modeling phase to systematically describe the common and variable features shared among the products of a product line. The analysis of common and variable features is a crucial part of the product line scoping process. In feature modeling, a feature model represents the features of a family of systems in the domain and relationships between them [6]. A feature model structures and decomposes the entire platform or product functionality in a hierarchical tree.

The advantage of blending the three methodologies together is that not only the elicitation criteria become crisper, but also all lifecycle aspects from analysis, decomposition, documentation, structuring, planning, implementing, “accepting”, etc. clearly defined and guided by a set of existing best practices.

4. Agile Development

Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, teamwork, self-organization and accountability; a set of engineering best practices that allows for rapid delivery of

high-quality software; and a business approach that aligns development with customer needs and business goals.

There are many specific agile development methods. Our organization employs a adaptation of the Scrum process [9]. Scrum is a process framework based on iterative & incremental and time-boxed development, a disciplined process guidance, strong customer involvement and constant process improvement. The purpose of these practices is to minimize the risk in a development project, especially when requirements are not fully known up front or may change during development. Therefore, after rough planning for the overall development effort, prioritization and planning of product features is a recurring issue and can be adapted in every iteration. Likewise, architecture is subject to changes and re-factoring as new features are implemented.

However, platform or product line development is different from product development when deciding on the scope of the work to be done and when determining the architecture. In product line development several products depend on the reusable parts developed by the platform organization. Typically, platform and products are developed in parallel, not sequentially. Therefore, it is necessary to invest more in both scoping (balancing requests and priorities from various platform users) and the architecture design beyond what is common practice in agile development. This focus on up front activities generally leads to more rigid, more waterfall, and more documentation centric development approaches.

How to conquer and limit the rigidness will be explained in the next sections.

5. Feature-Orientation as Key Enabler

The following graphic in Figure 1 displays how Agile and PLE practices – applied carefully and consciously – can extend each other's benefit. Note that the topics are used to structure the further proceeding of this paper.

Feature-orientation is the glue that firstly enables many of the value stream optimizations according to lean and agile principles, and secondly allows for the transparency and overview to base rational and systematic reasoning, business-driven and with a clear focus on the customer.

Figure 1: Feature-orientation as foundation for PLE and Agile

The main messages of the paper are:

1. Agile methodologies and Product Line Engineering can be effectively combined.
2. Organizational decomposition into value-stream oriented Scrum teams benefits greatly from feature modeling.
3. Feature modeling supports well the continuous scoping to changing business needs.

The following subsections explain the main ingredients of the approach.

5.1 Backlog-driven Development

The key of Scrum is the existence of a backlog, which consists of items with the same level of granularity that can be prioritized and groomed before being implemented. The creation of such a backlog for a platform is not easy with multiple stakeholders present.

Our feature model contains existing functionality, as well as planned or wished functionality in its consistent representation. The feature model reflects the overall domain of imaging applications that is relevant for our products, and helps all the stakeholders to get the same understanding of the domain and better see the variability within the domain. It focuses the discussion and gives a good overview on areas that are already very well supported by the platform and on areas that are not covered, maybe because the requirements are advanced or because the variability is too high to implement the features in the platform.

Features that are to be developed in a future release are linked to a platform backlog; features scheduled for next release are additionally linked to the release backlog of that release.

To focus a release with coherent sets of features, every release gets a handful of mottos (or goals) that guarantees that some products get real value, while other products have to wait for another release. Mottos are goals that are to be achieved – such as: "Enable radiologists to use the product for clinical routine in the area XYZ." The advantage of this is that there are products that really make a difference as opposed to many little improvements in many products that do not differentiate the whole product line from competitor products. The order of features in the platform backlog is also called 'roadmap' and consists of mottos assigned to releases.

After adding the attributes that are required to plan and control the project and the different releases of the platform, we are prepared for the project management challenges with the additional benefit of having a direct relationship from each backlog item to the business needs, because the backlog items are linked to features and they are linked to mottos. .

Progress monitoring in so-called burn-down charts allows for quite transparent tracking of a project. As with every iteration the development teams velocity get more validation, hence certainty. The velocity applied to the high-level groomed remaining backlog items can be used to predict the point in time when the platform scope is actually completed. Especially in large-scale development organizations with fairly fixed budgets, hence fixed timelines, time-boxed development is best practice. And if things do not turn out as expected, knowing the divergence early enough

– e.g. 9 months ahead of planned release – is a necessary luxury. Two options exist, delay the platform and product delivery, or reduce scope together with dependent products. Both are a misery product wise.

But not only feature completion can be tracked, also the time until it is sold and used by end-customers, basically the time until it is returning its invest. It can be quite an eye opener reflecting on this transparency, how much dead capital is buried in modern software products, resp. their development [5]. The dead capital can be visualized by tracking implementation increments in relation to their first customer usage.

5.2 Family Model

A feature model describes the problem space. In our development we link the problem space description to the solution space for traceability and impact analysis and for supporting the various types of variability when a product is derived from the product line. Solution space models are called family models; a term adopted from the company Pure Systems. The main family model consists of a hierarchical architecture model consisting of subsystems and their components, which allow an n:m mapping between the features and the components. When doing this mapping in an early phase of the project it helps us to …

- … know whether additional components or subsystems have to be created to cover the new functionality
- … know the complexity of a feature, for instance by knowing how many subsystems are affected by the new feature
- … estimate the expected effort to implement a feature
- … identify areas that require a redesign or re-factoring (e.g. if a mapping from one feature to many components is done, it indicates that a re-design could be required)

One main advantage of using the family model is the possibility to optimize efforts when performing a local regression test. Instead of having to perform a complete test-run of the system again after each modification we perform only an impact test based on the impact of the executed source code changes. The impact is determined by using the information of the family model and how it is linked to the feature model. For each change in the code, the affected features are identified through the trace links and only their tests are re-executed.

5.3 Business-driven Platform Evolution

Strategy discussions around product line evolution can now be supported with crisp mottos instead of long and tedious Excel sheets, because a motto would be the root of a clearly identifiable sub-tree in the overall feature tree. As the mapping from the motto to the features is unique, a detailed lookup can be done any time.

The connection to the family model allowed for early and more complete impact estimations, enabling higher quality of effort estimations, as they are the bases of business decisions.

5.4 Variability Management

So far we have not discussed the situations when individual products require different functionality from the platform. This might be functionality that can be added or removed for a specific product incarnation or has variation points that can be set or customized for a product. Historically, the approach was to just ignore the fact that functionality can vary and support every product alike, meaning a maximalist platform approach was used, which lead to lots of variability all over the platform. With the transparency the feature model gave us, it is now the goal to maintain a minimalist hierarchical platform.

The feature model with our commonality/variability analysis methods helps us realizing our declared goal of a minimalistic platform. Product specifics are not added to the general pool of reusable components, but are put to either the product or – if reuse can be discovered at a different level – higher-level reuse pool. Figure 2 shows our hierarchical platform that helps minimizing the variability on every layer.

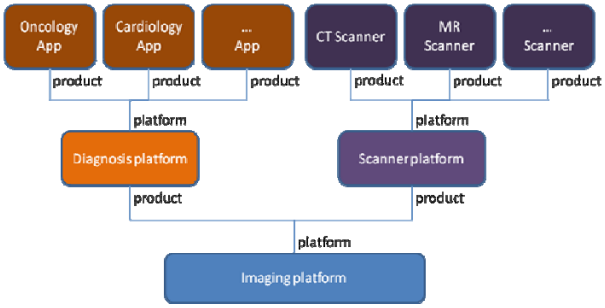

Figure 2: Hierarchical Platform

5.5 Organization of documentation

Developing medical products means: being very transparent, persistent and verbose on the decision and design input processes. Where persistent means that written documentation needs to be maintained. In agile requirements engineering the documents would be requirement specifications and user stories. Both document types highly benefit from the clear and stakeholder value driven separation of the problem space achieved through the feature-oriented partitioning, because author responsibility as well as decomposition from market requirements to more fine granular development requirements is easier and more natural.

Roles related to requirements engineering, such as risk managers, usability engineers, and technical documentation appreciates clear decomposition as well, easing their job. We were able to reduce the communication overhead and increase product consistency with our feature-oriented approach.

5.6 Self-organizing Feature-oriented Scrum Teams

A best practice of lean thinking – and with a bit of practice quickly observable – is the demand that the dependencies between Scrum teams in the value creation chain need to be minimized. Having dependencies and/or cross-cutting constraints – so called steel threads in agile methodology – bind and limit teams unnecessarily to each other.

A sound decomposition – not to say a partitioning – of features in the hierarchy of the feature model and selecting coherent sub-trees, which we call 'feature area', assigned to individual Scrum teams is the solution. This way a Scrum team can truly deliver end-customer functionality while during the feature development step be as much independent as possible. The quality of independence is very much dependent on the quality of the feature partitioning in the feature model. Steel threads might still exist in the solution space, which can be mitigated by concepts of 'Component Guardians' as the like.

Structures with a single project lead and the rest of the organization following his command in detail do not scale because of the high need for collaboration and ad hoc problem solving. Hence such activities are best guided by self-organizing teams, which are responsible but also flexible enough to quickly react and avoid typical bottlenecks.

5.7 Organization Optimized around Value Stream

Scrum teams working on isolated feature areas allows already for a great deal of efficiency. Wasteful clarification, high-level integration and very often escalations are avoided. It becomes now important that not only feature-wise but also competency-wise the Scrum team can deliver independently. The Scrum team should have all process as well as design and technical know-how to act independently.

Typical crosscutting concerns like interactions and dependencies between hardware & software and operational qualities are best governed by non-Scrum 'helper' teams supporting the actual value streams.

One of the measured improvements was that individual hours of overtime could be reduced by 60% comparing the software release before and after the agile transition.

Another improvement was that the product release after the transition also adhered to its time-box and minimal viable product commitment – securing timely product delivery. This is not only valued by internal stakeholders like product developers, but also by customers – confirming Siemens as reliable partner.

5.8 Solution Simplicity

The main motivators why solutions get simpler is derived from Variability Management and Scoping. Being able to avoid complexity by not having to support specifics and exotic variants alleviates the development majorly.

Experience shows that complex technical solutions very often lead to complex user interactions. So simplifying solutions benefits also customers. If the usability follows simple paradigms the product can also be quickly understood and adapted by customers.

5.9 Early Product Feedback

Platform development finishes one feature after the other, instead of working in subsystems with arbitrary dependencies and late integration. This continuous delivery of platform artifacts to product developers in application engineering allows for early feedback from those development groups that are typically closer to the end-customer. Misunderstandings can be identified faster, e.g. through continuous integration, operational quality tests and smoke testing, and clarified earlier with less effort, compared to fixing issues towards the end of a release then classic system integration is done.

6. Summary

Applying product line engineering for more systematic reuse leads to additional process steps and documentation. The additional process steps and additional documentation artifacts can have a negative impact on the efficiency of the development team. While the long-term benefits of systematic reuse are proven [8] little experience has been shared until now how to keep development productivity also in the short term.

From the experience analyzed in this paper it can be concluded that PLE and Agile do not contradict each other, but complement each other when applied consequently on a foundation of feature-orientation. The application of lean principles not only alleviates the process and documentation 'burden' of medically regulated development, but also the 'burden' of systematic reuse. The extra-effort of documentation, necessary because of multiple reasons, ranging from regulatory tracing, communication artifacts due to the large-scale project size, to analysis and governance documentation can be chopped in increments and to a considerable extent be elaborated in iterations, instead of dedicated up-front phases. The disadvantage of up-front phases is that effort is spent on work that will not in the full extend or even not at all be relevant for later project phases – hence create waste. This waste typically tends to delay time-to-market, causes higher R&D efforts, and frustration of R&D staff.

The DONE criteria connected with the completion of every feature developed by a Scrum team became the main vehicle to ensure consequent and diligent follow-up on the previous increments to further complete and complement the artifacts.

7. References

[1] S. Apel and C. Kästner, An overview of feature-oriented software development, Journal of Object Technology, 2009

[2] Feature-Driven Development, http://en.wikipedia.org/wiki/Feature-driven_development

[3] Feature Model, http://en.wikipedia.org/wiki/Feature_model

[4] Feature Oriented Software Development, http://en.wikipedia.org/wiki/Feature-oriented_programming

[5] A. Heck and M. Kircher, Agile Transition of a big medical software product development, OOP Munich, 2012 http://www.slideshare.net/AgileAndrea/agile-transition-of-a-big-medical-software-product-development

[6] Kang, K., et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990

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

[8] J. S. Poulin, Measuring Software Reuse: Principles, Practices, and Economic Models, Addison-Wesley, 1996

[9] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2001

[10] pure::variants, Variant management tool, http://www.pure-systems.com/Variantenmanagement, 2006

[11] C. Larman, B. Vodde, Scaling Lean and Agile Development, Addison-Wesley, 2008