Product Line Engineering in Large-Scale Lean and Agile Software Product Development Environments

Towards a Hybrid Approach to Decentral Control and Managed Reuse

Benjamin Blau
CPO Cross Development IT Portfolio
SAP AG
Walldorf, Germany
Benjamin.blau@sap.com

Tobias Hildenbrand
CPO Cross Development LEAN Center
SAP AG
Walldorf, Germany
Tobias.hildenbrand@sap.com

***Abstract*—Despite the fact that lean and agile software development has become mainstream recently, especially for larger-scale organizations building complex products, the methodology leaves many architectural questions unanswered. For instance, agile methods such as Extreme Programming propose late architectural decisions and frequent refactoring, while others suggest an "architectural runway" as infrastructure for a certain set of upcoming features. Software product lines consist of a set of software products that share a common, managed set of features. These product lines are developed from reusable core assets incorporating variations in order to derive customer-specific product variants. Hence, this research explores interoperability and complementarity of (a) lean and agile approaches in combination with (b) software product line engineering. With this position and discussion paper, we aim at both, complementing common issues in large scale lean and agile development and providing methodological guidance to make product line engineering more efficient. Our findings are based on observations and experience from a large-scale software company with several thousand developers working on various solution combinations of highly interdependent products.**

Keywords—lean product development, agile software engineering, software product lines

I. INTRODUCTION & BACKGROUND

Lean product development (LPD) and agile software engineering (ASE) have become a mainstream methodology recently. However, especially for larger-scale organizations building complex products, the methodology leaves many architectural questions unanswered. For instance, methods like XP propose late architectural decisions and frequent refactoring, while other suggest a so-called "architectural runway" as infrastructure for a certain set of upcoming features [14]. Software product lines (SPL), on the other hand, consist of a set of software products that share a common, managed set of features. Product lines are developed from reusable core assets incorporating variations in order to derive customer-specific product variants. In both, LPD/ASE and SPL engineering, non-functional requirements, such as availability, reliability, and dependability of software solutions, also need to be taken in into account.

Hence, this research explores interoperability and complementarity of lean and agile approaches in combination with software product line engineering (PLE) as a basis for further design science research [8]. With this work, we aim at both, addressing common issues in large scale lean and agile development and provide methodological guidance to make PLE more efficient. Our findings are based on observations and experience from a market leading large-scale software company with several thousand developers working on various solution combinations of highly interdependent products. This company has been implementing lean and agile principles for more than 3 years now and LPD/ASE has become the standard model for product creation [21].

In the following, we provide some essential background on large-scale lean and agile software developments as well as on software product line engineering. Our approach to combining the two while lean and agile principles are already in place, addresses three major dichotomies and areas of possible conflicts in LPD/ASE and PLE: (1) feature teams vs. component teams, (2) team empowerment and late decision-making vs. compliance with reference architecture and predefined policies, as well as (3) budget steering (features vs. platform quality and renovation). With respect to dichotomy (1), for instance, we develop and discuss a hybrid approach to benefit from both, ASE and PLE.

When looking at the software industry, competition as of today requires significant productivity gains along with innovation despite a very high degree of complexity in both, products and processes. In software development, customer orientation and early feedback is critical due to the fact that a complete upfront specification is not possible as mostly functional requirements evolve over time and changes are inevitable [7][20]. Especially when developing large-scale and complex software, such as business applications, spanning multiple locations and organizational units, complexity in terms of inter-related, even conflicting, system parameters and dependencies grow disproportionately [10] [15][19]. To mitigate these effects, elements of lean thinking, such as synchronization and transparency, have been transferred to the software world as well [18] and agile methods like Scrum seem to be promising for implementing lean principles in software development [9][21]. Moreover,

concrete agile techniques such as test-driven development (TDD), pair programming, and continuous integration, e.g. defined in Extreme Programming (XP), guide the implementation of lean and agile software development [6].

Despite faster feedback, the complexity remains the same and they do not scale easily in large enterprise environments. While agile and lean thinking share fundamental concepts and principles, lean principles are proven to be applicable on an enterprise and industry level [20][22] (see overview in Figure 1).

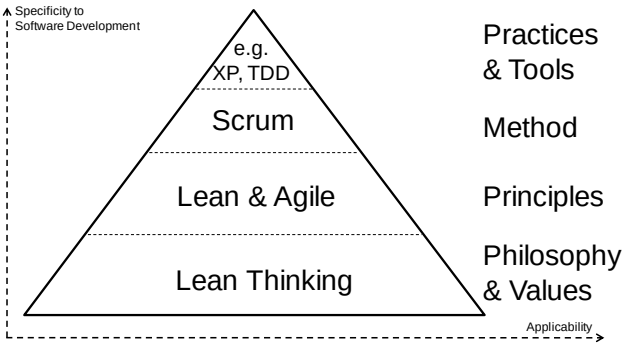

Figure 1. How Lean/Agile Principles Build on Lean Thinking

On the other hand, "a Software Product Line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [1]. PLE targets to leverage economies of scope by identification and reuse of commonalities and the proactive management of bound variability [5][11][20].

II. Approach to Large-Scale ASE & PLE

Agile software development practices and lean principles, which focus on process optimization as well as increases efficiency and productivity, have gained great momentum in large-scale software companies. Focusing on product consistency and systematic reuse, product line engineering on the other hand is a promising approach in traditional product development industries. The combination of both approaches bears a huge potential as most principles are complementing each other. However, there is only little research on introducing PLE in large-scale lean and agile software product development environments [3][16][17]. Despite complementing aspects of both approaches, there are certain aspects that seem contradictory in the first place. We identified the following three apparently dichotomous topics that need further investigation in order to achieve a seamless implementation of PLE in lean and agile environments: (A) feature teams vs. component teams, (B) team empowerment and late decision-making vs. compliance with reference architecture and predefined policies, as well as (C) budget steering (features vs. platform quality and renovation).

A. Feature Teams vs. Component Teams

Lean and scrum principles propose feature teams that implement a customer requirement reflected in a certain feature across the entire stack of the complex product. That means these teams work vertically across multiple components of the overall product in order to realize a specified feature end-to-end. Such an approach requires cross-functional teams with a broad set of skills that span the entire product in order to be self-sufficient [12][13][15].

In product line engineering, on the other hand, common features that are implemented by multiple products within the product line and are encapsulated in core assets developed in a separated organizational unit - i.e. the domain - with a clear responsibility through a dedicated domain owner. By nature, such domains are component-oriented as they comprise all implementations of a certain common feature of a product line in core assets and components. Dedicated component teams managed by the domain owner develop assets for reuse within a separated domain lifecycle that are reused in certain products within the product line (in some cases also across product lines).

When introducing product line engineering in a large-scale lean software development company, both organizational approaches regarding team structure need to be combined in a hybrid fashion with clear responsibility and overall coordination. Figure 2 depicts such a hybrid setup with feature and component teams and their common working model in the overall development process.

According to lean and agile principles, e.g. in Scrum, the overall product backlog entails all prioritized backlog items according to required features that are processed top-down. There are two distinguished types of backlog items, (i) those representing a common feature that is required by many products and developed within a certain domain (according to the PLE principles), and (ii) those that represent variable features individually required by a single product.

In a component-oriented manner, the component teams of the corresponding domain develop the common features in core assets that embody the main building blocks of the final product. These core assets are enhanced by variation points that allow for a bounded customization in order to meet individual customer requirements reflected by variable features. That means, these variation points can be shaped into concrete variants that together form a possible instantiation of the core asset [1]. As the domains work in a separated lifecycle, their individual roadmap needs to be synchronized with the prioritization of the overall product backlog. As there are obvious dependencies between the common features realized in core assets and the variable features affecting multiple components across the product stack, this task is crucial to the success of the hybrid model.

Backlog items that represent individual customer requirements reflected in variable features which are by nature only relevant for a single product are processed by feature teams as defined in the lean methodology. These teams work end-to-end within the entire product stack in order to realize these features. In detail, there are mainly two ways to implement a variable feature. As described in product line engineering, variable feature can be realized by instantiating variation points of core assets. That means, feature teams customize the predefined variation points of one or multiple core assets in order to create an individual

instantiation of the overall product (analogue to the configuration of a car that consists of a common platform, i.e. the basic model, with configurable elements e.g. a navigation system on top). The second way of implementing a variable feature is through so called glue-code. If the requested feature cannot be realized by existing core assets and their variations, additional programming effort needs to take place in order to adapt the overall product according to customer requirements.

The coordination of feature and component teams and the synchronization of both lifecycles are crucial to the success of such a hybrid model. According to product line engineering principles, the reference architecture defines the products within a product line, domains, core assets, the policies on must and should use and how the instantiation of variation points needs to be conducted. All teams must be compliant with the reference architecture to assure product consistency and systematic reuse. In summary, common features that are required in multiple products are realized by dedicated component teams in separated domains in a decoupled lifecycle. Variable feature that are individually required by single product are realized by feature teams that work across the whole product stack and implement the requirements end-to-end. More precisely, they instantiate variation points of the core assets developed by the component teams or if necessary enrich the product with glue-code. The lifecycles of products and domains need to be synchronized in a way that the dependencies between backlog items (processed by feature and component teams) are taking into account (e.g. core assets need to be implemented in order to instantiate variation points by feature teams).

B. Team Empowerment and Late Decision-Making vs. Compliance with Reference Architecture and Predefined Policies

One of the core lean principles is the empowerment of teams, i.e. handing over responsibility to the ones that directly generate value for the company, e.g. in terms of product quality, costs of development, and delivery timeline. The intention is to reduce overhead and loosen central governance in order to empower individuals and teams to decide flexibly and as late as possible within certain boundaries to increase speed and adaptability to fast changing conditions in today's markets. This principle is based on the assumption that processes are waste if they do not fully support value creation. Hence, if decisions can be made decentralized without going through heavy-weight governance processes, this increases speed and reduces cost of delay in a lean manner [22].

As PLE targets to implement systematic reuse across the whole company in order to increase product consistency and reduce cost, compliance with a lowest common denominator is inevitable, i.e. the reference architecture [2]. The reference architecture defines the product lines, their products and must and should use policies (predefined "decisions" in contrary to late decision-making in lean environments) for reusable assets as well as how their variation points are instantiated. More generally spoken, the reference architecture is a centralized structure in order to steer systematic reuse. Non-functional requirements, such as availability, reliability, and dependability of software solutions, also need to be taken in into account.

At first sight, the mentioned two principles (empowerment and compliance with reference architecture) might seem contradictory. However, feasibly combined, their combination fosters a powerful behavioral environment that enables a company to achieve both, speed and flexibility as well as consistent product structures. More precisely, the reference architecture can be seen as the lowest common denominator, i.e. the set of bounding conditions for teams and individuals [17]. Within these boundaries, teams can act in a fully empowered manner without procedural overhead that is not supporting their daily business.

C. Budget Steering: Features vs. Platform Quality and Renovation

The separation of domain and product engineering results in a loosely coupled setup which in general may foster local optimization within each area. In addition, the long-term and technical orientation of the domains and the customer-oriented (and thus more flexible) setup of the product units according to lean principles may result in different (and possibly conflicting) objectives. To balance these objectives and align them in a way that the overall business strategy can be successfully executed, the distribution of budget as a method to implement business incentives is a crucial and highly sensitive activity when combining lean and PLE principles. Budget steering is essential to balance reuse on the one hand and to foster customer orientation on the other hand. In general, there are multiple funding models possible when combining lean and PLE principles (e.g. a sub-contractor model, a tax model, and a separated funding approach). However, leveraging the dynamics and efficiency of an internal market situation, the sub-contractor model seems that most promising as it assures customer orientation while implementing incentives for cost efficiency and reuse. The sub-contractor model is explained in detail in the remainder of this section.

The market and product strategy embodies the strategic guidance for the overall portfolio process and the corresponding budget allocation. The prioritization of the marketed product lines and products is reflected by the budget allocation. In a sub-contractor model, the budget is allocated to the marketed products or to groups of marketed products addressing distinguished market segments. Hence, budget may be planned not at the product-level but at a market segment level, clustering marketed products that are relevant for a particular market segment. Based on this initial partitioning of budget and the mapping of the marketed products view onto the engineered product view, the budget can be further allocated to the internally engineered product lines and products that are required to assemble the corresponding marketed products. The assets that are required to assemble a certain engineered product are developed by the corresponding domains according to the PLE principles. The asset development is funded by the organization of the engineered products that work customer-

oriented and feature-driven according to lean principles. Such a model is characterized by a sub-contractor relationship between the product and the domain engineering organization, analogous to an open market of customers and providers. To assure the timeliness and of delivery and the agreed-upon quality of the requested assets, a contract, i.e. a service level agreement including indicators, objectives, and penalties, needs to be negotiated and committed to by both parties.

Although the customer-provider-relationship fosters customer- and market-orientation, domain engineering needs a fair share of dedicated budget in order to conduct code renovation, quality improvements, and other activities that are not directly requested by customers and hence are not funded by the product engineering organization. These domain-related fair budget shares can be either planned at the overall portfolio level or at the market segment level. In ASE, the common notion of an "architectural runway" recently evolved, i.e. a guaranteed technological infrastructure for a certain set of upcoming features that is also accounted for in agile planning and estimation [14].

III. Conclusion and Research Roadmap

Our research and considerations so far have shown that software product line engineering has the potential to complement already proven lean principles and agile software engineering techniques. In order to achieve this, a set of apparent dichotomies needs to be resolved. Our discussion encourages the position that a hybrid approach, e.g. for efficiently combining both, feature and component teams, building one product based on product lines with domain owners (see figure 2) is possible. We argue that integrating lean and agile principles and managed reuse in this manner is beneficial for large-scale software product development in terms of process efficiency and effectives, i.e. being able to build the right products in time.

Our research roadmap can be divided into two main streams that are interrelated: (1) we further elaborate on resolving the dichotomies of LPD/ASE and PLE and complement our hybrid approach as well as the underlying working model and (2) we start a pilot implementation for evaluating our hybrid approach with feature and component teams including domain owners. This pilot will be conducted for a complex product called "Business Process Management" (BPM). BPM consists of x feature teams and y component teams. The high-level backlog contains z backlog items in the form of user stories which in turn pertain to n underlying components. The pilot implementation will take place within a timeframe of m months. Further pilots will also include a budget steering model addressing new feature development versus investments in platform quality and renovation activities such as architectural refactoring and user interface design.

References

[1] Bosch 2002 - Bosch, J. and Florijn, G. and Greefhorst, D. and Kuusela, J. and Obbink, J. and Pohl, K., Variability issues in software product lines, Journal of Software Product-Family Engineering, 2002

[2] Bosch 2010 - Coordination Between Global Agile Teams: From Process to Architecture - Book chapter

[3] Bosch 2011 - Introducing Agile Customer-Centered Development in a Legacy Software Product Line - Software Practice and Experience

[4] Carbon 2006 - Integrating Product Line Engineering and Agile Methods: Flexible Design Up-front vs. Incremental Design

[5] Clemens 2003 - P. Clemens and L. Northrop, Software Product Lines, Addison Wesley, 2003

[6] Dyba, T., Dingsoyr, T. 2008. Empirical studies of agile software development: A systematic review. Information and Software Technology, Volume 50, Issues 9-10, pp. 833-859

[7] Geisser, Michael and Hildenbrand, Tobias (2006): "A Method for Collaborative Requirements Elicitation and Decision-Supported Requirements Analysis". In: Ochoa, Sergio F. and Roman, Gruia-Catalin (eds): IFIP International Federation for Information Processing, Volume 219, Advanced Software Engineering: Expanding the Frontiers of Software Technology, pp 108-122, Springer, Boston (chapter in book)

[8] Hevner, A., March, S., Park, J., and Ram, S. (2004). Design science information systems research. In: MIS Quarterly, 28(1):75–105.

[9] Hildenbrand, Tobias and Geisser, Michael and Kude, Thomas and Bruch, Denis and Acker, Thomas (2008): "Agile Methodologies for Distributed Collaborative Development of Enterprise Applications". In: Proceedings of the Second Workshop on Engineering Complex Distributed Systems (ECDS 2008) in conjunction with International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS 2008), pp 540-545, IEEE Computer Society, Barcelona, Spain

[10] Hildenbrand, T. (2008). Improving Traceability in Distributed Collaborative Software Development—A Design-Science Approach.Phd thesis, University of Mannheim/Frankfurt, Germany.

[11] Kruchten 2006 - The past, present, and future for software architecture - IEEE Software

[12] Larman, C., Vodde, B. 2009. Scaling lean & agile development: thinking and organizational tools for large-scale Scrum. Addison-Wesley

[13] Larman, C., Vodde, B. 2010. Practices for Scaling Lean and Agile Development. Addison-Wesley

[14] Leffingwell 2011 - Agile Software Requirements (pp. 383) - Addison Wesley

[15] Lee, G., Xia, W. 2010. Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data. MIS Quarterly, 34(1), pp.87-114

[16] Mohan 2010 - Integrating Software Product Line Engineering and Agile Development - IEEE Software, Volume: 27, Issue:3

[17] Nord 2006 - Software Architecture-Centric Methods and Agile Development - IEEE Software

[18] Poppendieck, M. and T. 2005. Introduction to lean software development. In Lecture Notes in Computer Science, Volume 3556/2005, pp. 1318-1321

[19] Redmiles, David and van der Hoek, André and Al-Ani, Ban and Hildenbrand, Tobias and Quirk, Stephen and Sarma, Anita and Silveira Silva Filho, Roberto and de Souza, Cleidson and Trainer, Erik (2007): "Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects". In: WIRTSCHAFTSINFORMATIK, Volume 49 (Sonderheft), pp S28-S38

[20] Reinertsen, D. G. 2009. The Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas Publishing.

[21] Schnitter, J. and Mackert O. 2010, Introducing Agile Software Development at SAP AG – Change Procedures and Observations in a Global Software Company. Proceedings 5th International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece, 2010, pp. 132-138

[22] Womack, J. P., Jones, D.T., Roos, D. 2007. The Machine that Changed the World. Free Press

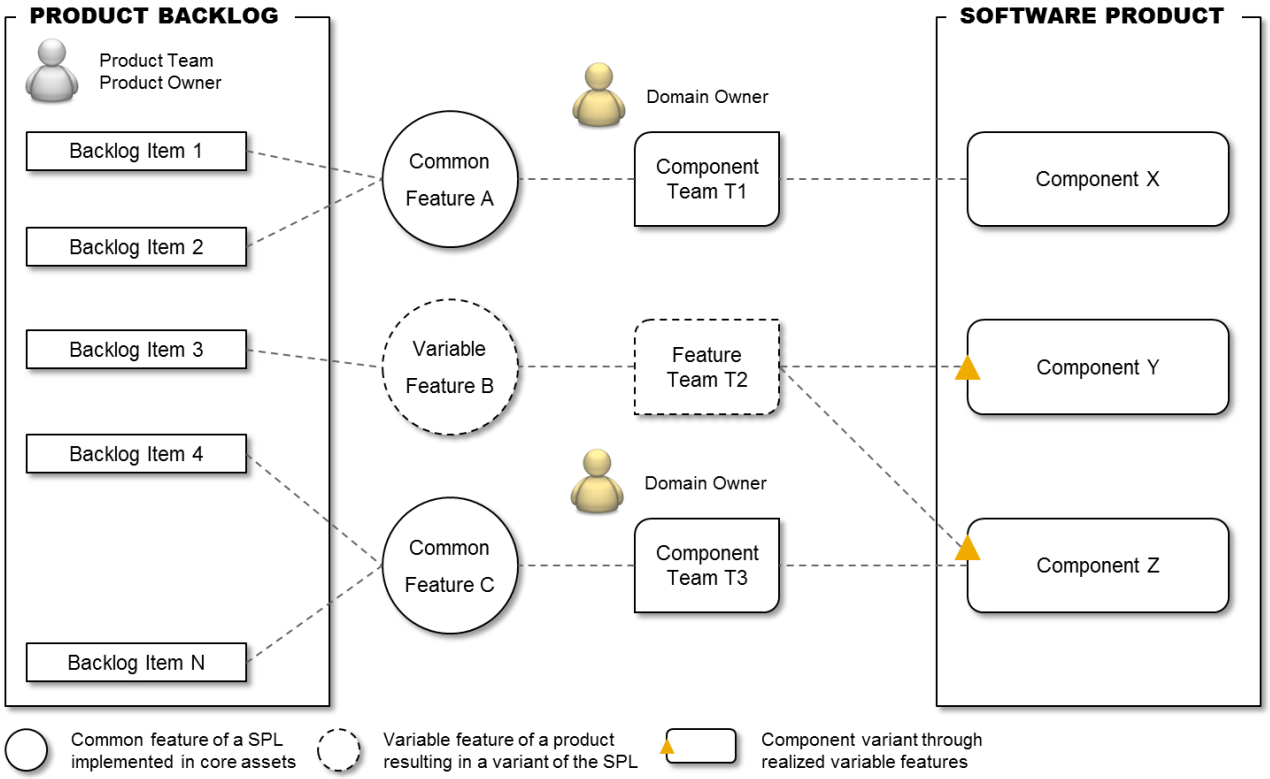

Figure 2. Hybrid organizational model with feature and component teams and their common working model.