Agile and Software Product Line Methods: Are They So Different?

Kun Tian
Department of Computer Science
The University of Texas at Dallas
Mail Station ECSS 3.1
2601 North Floyd Road
Richardson, Texas, USA 75083
skearth@gmail.com

Kendra Cooper
Department of Computer Science
The University of Texas at Dallas
Mail Station ECSS 3.1
2601 North Floyd Road
Richardson, Texas, USA 75083
kcooper@utdallas.edu

Abstract

The need to rapidly deliver high quality software that meets the changing needs of stakeholders has driven the definition of methods, including Agile and Software Product Line approaches. Although many of the goals of such approaches are similar, they appear to differ significantly in principles and practices. The purpose of this work is to compare Agile and Software Product line approaches from engineering, software quality assurance, and project management perspectives. The results of the study can be used to determine the feasibility of tailoring a software product line approach with Agile practices, resulting in a lighter-weight approach that provides mass customization, reduced time-to-market, improved customer satisfaction, etc.

1. Introduction

The need to rapidly produce quality software and respond to changes in a flexible and quick manner has driven the definition of new techniques, tools, and notations. These approaches share some common goals: increasing the productivity of the development teams, reducing products' time-to-market, reducing development costs and improving customer satisfaction. Two such methods are agile methods (AMs), including Scrum [5], Extreme Programming [6], DSDM [7], and FDD [8], and software product line methods (SPLMs), including PLUSS [9], COVAMOF [10] and FOOM [11].

AMs are software processes that share the same values: individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to changes over following a plan. The Agile Manifesto inspired 12 principles for Agile process [2]. Among these principles, the highest priority is to satisfy the customer through early and continuous delivery of software. Agile methods use short iterations (sprints) that are typically two to four weeks long. Satisfying the customer also involves recognizing the need to change requirements, even late in development, to support the customer's competitive advantage. The customers are highly involved, as they receive frequent deliverables of working software and work together with the technical people daily throughout the project. Working software is the primary measure of progress on the project, as opposed to modeling artifacts, etc. The software is built by motivated individuals, who have an environment and the support they need to get the job done. The self-organizing teams strive for technical excellence (i.e., best requirements, best architecture, etc.) and simplicity (i.e., maximizing the amount of work not done, such as extensive documentation for planning, requirements, architecture, etc.). The project proceeds at a pace that is sustainable over the long run and includes regular reflections on how to become more effective at implementing necessary changes.

Figure 1 presents an overview of an Agile engineering process. AMs minimize requirement engineering and design modeling practices so that the team can begin working on code as soon as possible. The team applies incremental development and short iterations. Artifacts in agile process are managed collectively by team members. The entire process relies on tacit knowledge for development effort coordination. The source code is placed under configuration management in a repository (e.g., using CVS)

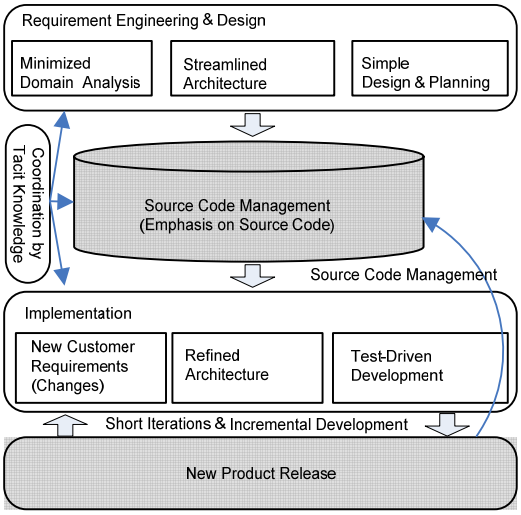

Figure 1 Overview of Agile Methods

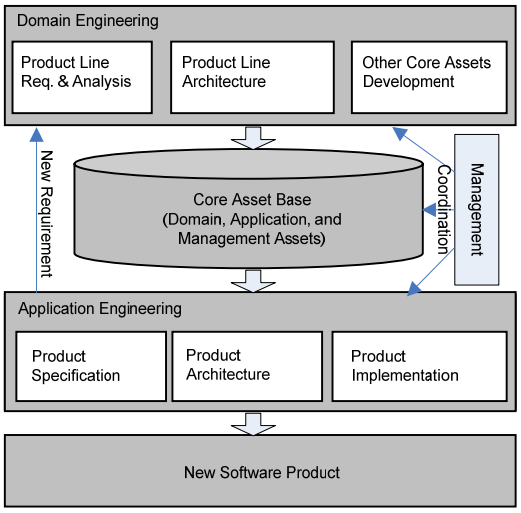

Figure 2 Overview of Product Line Methods

Software product line methods (SPLMs) are practices-based, or plan-driven, software development approaches in which a set of software-intensive systems that share a common, managed set of features are produced from a set of re-usable core assets in a prescribed way [3][4]. A core asset is a software artifact that is re-used in the production of customized products in a software product line (SPL). The assets include the requirements, architecture, components, modeling and analysis, plans, etc. A SPL product can be quickly assembled from core assets, and hence it achieves manufacturing efficiency. SPLMs support mass customization, which is "producing goods and services to meet individual customer's needs with near mass production efficiency" [15][15]. Mass customization in SPLMs is transparent: customers can obtain a unique product by having their special requirements implemented; their common requirements are assessed before production begins [14].

A SPLM has three main activities to run a software development organization in an efficient and consistent manner: core assets development, application development and management (refer to Figure 2). Core asset development is based on Domain Engineering, which is fundamental to leveraging the commonalities between software products and services developed. Application development is the process of building new and customized software product from a base of core assets artifacts. Management acts as a strategic objectives provider which coordinates the interactions of core assets and application development.

Many defined processes, such as the well-known rational unified process (RUP) [20], have been designed so that the users can tailor them. For example, Agile Modeling [17] is a tailored version of RUP, whose scope is to describe how to model and document in an effective and agile manner to enhance other, more complete software processes. The CMU SEI software product line approach also tailors some RUP practices for its own use, for example in architecture definition and architecture-based development. The question under investigation in this research is: “Is it feasible to tailor SPLM and/or to integrate Agile practices into it to obtain a lighter-weight, yet still plan-driven approach that combines the advantages of both SPLMs and AMs?” To answer this question, a comparison of the two approaches is needed.

The purpose of this paper is to present a comparison of SPLM and AM methods. The comparison criteria span engineering, software quality assurance and project management concerns, which are presented and discussed in Sections 2, 3 and 4 respectively. This paper uses more established SPLMs and AMs in the comparison. This comparison is expected to provide a useful foundation to the community, which can be extended to include additional results available in the literature and additional comparison criteria. Conclusions are presented in Section 5.

2. Engineering Activities

A comparison of engineering activities in SPLMs and AMs is presented here that focuses on requirements engineering, software architecture, and

re-use. Additional criteria, such as a comparison of the implementation activities have also been considered, but are not presented here due to space constraints. A summary of the comparison is presented in Table 1.

2.1 Software Requirements Engineering

Both AMs and SPLMs assume that changes to the requirements during product development are going to occur. However, they have different strategies to deal with them. Agile developers' strategy is focused on incremental development and close interactions with customers, such as small increments and dynamic requirement prioritization in XP [6]. According to the world view of Agile methodologists, organizations are complex adaptive systems in which requirements are emergent rather than pre-specifiable [12]. Consequently, Agile developers seldom perform comprehensive software requirement elicitation, specification, analysis, validation, or management activities. AMs work very well when future requirements are largely unpredictable.

The strategy in SPLMs is to look for solutions to meet customers needs by customizing the core assets [3][4][14]. As SPLM analysts stress the paramount importance of process definition and documentation, they try to predict changes in the beginning of the process (by market analysis, product line scoping and studying business case, etc.) and maintain the variability models as core assets. Requirements in SPLMs are largely done by defining the product line scope [14]. Successful scoping must target the right products for the domain, determined by factors such as knowledge of similar systems and future market demands, and it must have the appropriate scope, because too large or small scope will impair the capability of SPLMs to achieve variability and/or economies [3]. A key element in conducting good software requirements in SPLMs is to carry out thorough analysis, perform careful predications and make smart selections. However, possible cases can occur in which the customer's desired product is out of a SPL's scope. If that happens, the management will let the customer know the cost associated with amending and revising the core assets to meet their needs. Then the customer will decide whether to accept the cost or modify their requirements to obtain benefits such as better maintainability and faster delivery. In SPLMs, customer collaboration is managed through customer interface management practice, where a number of customer representatives (marketers, a product manager, domain experts, a user's group coordinator, etc) that each is explicitly assigned roles and responsibilities act as a bridge that connect customers (product users) and developers.

An interesting question here is that, if the knowledge, based on which SPL developers perform scoping, is not sufficient enough for a thorough analysis, can we apply AMs to elicit further knowledge
for SPL developers to produce a good scope? AMs may help SPL developers produce a prototype swiftly and modify it quickly according to any change they make as they gradually clarify the targeted scope, and hence they can refer to the prototype to gain deepened knowledge of the products. So integrated software practices (both requirements and developments) from AMs may enhance SPLMs when some of the information about the scope is predicable and some of it is not.

2.2 Software Architecture

One major principle in AMs emphasizes simplicity: to maximize the amount of work not done [1]. Based on this principle, some AMs like XP call for removing or not considering architectural features that are not of immediate interest for the current iteration.

Comparably, a key to supporting mass production in a SPL is the definition and maintenance of product line architecture, which satisfies the general needs of the product line and the individual needs of products by explicitly admitting a set of variation points required to support the spectrum of products within the scope. A variation point in a product line architecture identifies one or more locations at which the variations will occur to create customized applications. And the customized applications may vary from each other in terms of their behavior, quality attributes, platform, network, physical configuration, middleware, scale factors and a multitude of other ways [14].

If an SPLM is tailored to be more agile, then there is the risk that valuable architecture supports for the other products in the family of product line can be discarded or damaged, as AMs have not been designed with the product line problem in mind. However, some AMs like DSDM [7] do more architectural preplanning to ease this problem. DSDM is an AM which recognizes that 80% of the software's value comes from 20% of its full requirements [7]. Once the critical 20% of the requirements are identified, it seems reasonable to start building the software. There are seven phases in DSDM: pre-project, feasibility- study, business-study, functional modal iteration, design and build iteration,

Table 1 Comparison Summary – Engineering

	Agile Methods	Software Product Line Approaches
Software Requirements	Emphasis on quick response to requirement changes with short iterations and small increments for the application. Direct customer collaboration; customers must participate in the whole software project lifecycle.	Application requirements and domain requirements are both engineered. Indirect customer collaboration using well-trained customer representatives.
Software Architecture	Minimal emphasis on the application architecture features beyond the immediate iteration.	Application architecture and domain architecture are both engineered.
Implementation	Iterative and incremental implementation; minimum effort on plans and models, using "just enough" approach.	Plan-driven implementation with an emphasis on application engineering process (application designs, architecture, etc.) developed using core assets. Feedback from the product-specific requirements is used to update core assets; substantial effort on plans and models.
Reuse	Optimistic use of COTS; applies streamlined domain engineering activity, without emphasis on development of reusable core assets.	Special emphasis on maturity and reliability of COTS; foundation of the method is re-using core assets defined in domain engineering activity.

implementation and post-project. The first three and last phases (pre-project;
feasibility and business studies; post-project) are concerned with the entire scope of the project whereas the second three (functional model and design and build iterations; implementation) form the iterative and incremental phases of development.

2.4 Re-use

2.4.1 COTS. Both Agile and SPL developers can make opportunistic use of COTS components in an application development. However, if a SPL solution uses COTS for its core assets, then the components need to be very carefully assessed to ensure they are flexible enough to satisfy the long term variability needs for a SPL. SPL developers should generalize their solutions using COTS to include a general purpose integration mechanism that spans the product line [14].

2.4.2 Domain Engineering. Domain engineering originates from research in software reuse. Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. Domain engineering involves of creating a set of reusable assets for building systems in a particular problem domain. Then the reusable assets are assembled into customer-specific software systems in the corresponding application engineering process. "*Domain engineering and application engineering are complementary, interacting, parallel processes that comprise a model-based, reuse-oriented software production system.*" [18].

The purpose of domain engineering is to provide competence to application engineering for a set of related or similar systems. Domain engineering covers all the activities for building software core assets, which include domains identification, domain analysis (capturing the variations within a domain), domain design (producing adaptable design), and domain implementation (defining the mechanisms for translating requirements into systems from reusable components). The products of these activities are domain models, design models, domain-specific languages, etc.[18] Domain knowledge is needed to understand the commonality and variability of the expected products identified in a SPL scope.

An application engineering process is primarily concerned with developing a single system from the software assets created by a domain engineering process. The major activities of an application engineering process include using domain model to identify the specific customer requirements for a system, using domain design models to specify a product configuration, using domain components or application generators to produce application codes, developing a partitioning strategy and coordination model to guide application development. The products of these activities are application requirements, application designs, application architecture, etc.[18]

2.5 Discussion

When considering what and how to tailor a SPLM to introduce more agility into the engineering activities, both the domain engineering and application engineering activities can be considered at multiple levels of abstraction (requirements, architecture, etc.). To engineer the requirements with more agility, the interaction with the customer could be combined with end-user programming made possible by domain specific languages. The definition and maintenance of the software architecture could be made more agile by drawing upon the DSDM approach. DSDM is agile while still plan-driven, and emphasizes the development of the software architecture for the success of project. In addition, the introduction of agility can benefit projects where some of the information about the product line scope is unavailable. Agility in using COTS components has been considered in the component community [19].

3. Software Quality Assurance Activities

A comparison of software quality assurance activities is presented here that focuses on configuration management, testing, and traceability. Additional criteria, such as a comparison of the review/inspection activity has also been considered, but are not presented here due to space constraints. A summary of the comparison in presented in Table 2.

3.1 Configuration Management

Software Configuration Management (CM) is concerned with establishing and maintaining the integrity of software products throughout the project's software life cycle. It refers to a discipline for evaluating, coordinating, tracing and implementing changes in artifacts that are used to construct and maintain software systems.

CM for a SPL organization is commonly viewed as a multidimensional version of the CM problem of a single software development process [14]. Each version of each product is maintained; the whole product line is managed with a single, unified CM process. Numerous artifacts are managed, including the requirements, architecture, components, test plans and cases, domain models, management schedules, budgets, and plans, and also production infrastructure like domain-specific languages, tools, teams and environment, etc. Each core asset has an associated process that specifies how it will be used to develop applications. These processes are core assets as well, which may be folded into the production plan for the product line.

Agile Configuration Management (ACM) is a recent research trend in CM [17]. It is a minimized and streamlined CM approach to do core CM task, which is designed to deal with changing CM requirements easily. The agility in ACM relies on the combination of good CM process, tools and automation. And the automation of version control enables software developers, testers and end users to work efficiently.

Product line CM needs to control the configuration of core assets. It must be aware of the fact that core assets may be developed by one team and used by others in parallel. Comparably in AMs, artifacts are managed by the development team as a whole: everybody has access to and can modify them. And the team relies on tacit knowledge to keep them updated about the current status and version of the artifacts.

3.2 Testing

Testing in SPLMs is a decentralized activity: responsibilities of testing are distributed across different parts of a SPL organization [3]. Developers must test core assets and a single product, and they must manage the interactions between them. Because test cases are also core assets that will be reused throughout a SPL, they need to be carefully planned [14].

In AMs, testing is carried out frequently; extensive tests that are out of immediate needs are prohibited to

Table 2 Comparison Summary - Software Quality Assurance

	Agile Methods	Software Product Line Approaches
Configuration Management	Minimized and streamlined CM approaches to do core CM tasks.	Extended CM approaches primarily for managing core assets.
Asset (Artifacts) Management	All are managed by the team as a whole; everybody could have access to them. It relies on tacit knowledge in the team to be conscious of the status of the artifacts.	Artifacts are identified as core assets, which are formally managed and kept in the core asset base. Each core asset has an attached process to demonstrate how it will be used in actual product development.
Testing	Unplanned activities; tests are carried out frequently and primarily during implementation phase to ensure that the code works.	Decentralized and planned activities; tests are reusable.

Table 3 Comparison Summary – Project Management

	Agile Methods	Software Product Line Approaches
Technology and Tools	Emphasis on simple and efficient tools. Technology impact only affects current project.	Emphasis on robust and powerful tools that support parallel works. Technology impact expands as more products of a SPL are turned out.
Project Characteristics	Not suitable for large, distributed, security or safety critical projects.	Not suitable for small, single product oriented projects.
Development Team Organization	Agile developers are good at using tacit knowledge; they share the responsibilities to the outside environment as a whole.	SPL developers follow plans and processes well; each of them has a set of specified roles and responsibilities.

promote agility. Also, because testing in AMs is focused on a single software system, no planning is needed to achieve strategic reuse of the tests for multiple applications. Agile developers try to keep their code as free of errors as possible by carrying out tests early. This could be a good practice in SPLMs for small-scale tests: SPL developer could tailor their testing plans to include only generic and globally-significant tests rather than the minor ones such as unit tests.

3.3 Traceability

Product line CM must be able to trace each artifact in the core asset base, which is a substantial and complex effort. This demands more powerful and robust CM tools than those that performed sufficiently well in a single-system development.

In AMs, artifacts are managed collectively by team members. Traceability management without explicit knowledge may be problematic; however, it is still possible to trace artifacts if the team communicates well in tacit knowledge, which could help keep its members updated of artifacts' states and history.

3.4 Discussion

When considering what and how to tailor a SPLM to introduce more agility into the software quality assurance activities, each can be considered (configuration management, testing and traceability, etc.). For example, in SPLMs, consider the process description for each core asset that explicitly demonstrates how it will be used or customized for assembling actual products. A heavier-weighted description of this process could include activity diagram models with detailed descriptions and perhaps examples. A more agile approach could be considered that still includes a concise process description that is simpler to review and maintain.

Alternatively, software quality assurance activities related to the domain engineering activities could remain more comprehensive and practices-based, such as testing and traceability, while the application engineering could be tailored to be more agile.

4. Project Management Activities

A comparison of project management activities is presented here that focuses on technology and tool selection, project characteristics and development team organization. Additional criteria, such as a comparison of risk management, team communication models and the impact of outsourcing have also been considered, but are not presented here due to space constraints. A summary of the comparison in presented in Table 3.

4.1 Technology and Tool Selection

For AMs, technology impacts are shorter term, as they only affect the current project. At the beginning of an Agile project, the currently available technologies are identified and assessed for their applicability. However, due to the nature of SPL (developing a set of similar products in a prescribed way), management of a SPL organization needs to choose a technology that will have long term benefits. That is because technology impacts in a SPL increase as more products are delivered time. Consequently, technologies are evaluated for their both present and potentially future applicability: they need to support multi-version, cooperatively developed, long-lived, robust and changeable systems [14].

The Agile Manifesto values individuals and interactions over tools and processes. Agile methodologists prefer the selection of simple and easy to manage tools over powerful but unwieldy ones. For SPLMs, the focus on tools is on applying familiar tools to practices in a product line context [3][14]. This can be interpreted to mean that familiar tools are comparably easy to manage than new ones. This saves time and money spent on training developers to use them and reduces development risks. This is partly in accordance with AMs' emphasis on tools, because both indicate tools need to be used efficiently. The

difference here is Agile methodologists advocate using simple tools, which may not be powerful enough to support concurrently creating, maintaining and using multiple versions of core assets and products in SPL.

4.2 Project Characteristics

As agile methods have been primarily concerned with single product development, they do not support mass customization, a key feature of SPL approaches. In addition to this key difference in the project characteristics, the size and type of the project are also comparison points.

For small- and medium-sized, non-distributed projects, AMs can perform well because of reduced bureaucracy and efficient communication among a small number of co-located developers. But for large projects that a number of developers are involved in, perhaps in geographically distributed locations, maintaining the tightly coordinated teamwork without explicit knowledge is very difficult. Contrarily, plan-driven methods like SPLMs are better suited for this kind of project [13].

For software that emphasizes security and safety (e.g., embedded life-critical software), AMs are not well suited, as the focus on early and continuous delivery may lead to inadequate planning and under-treatment of the non-functional requirements that are critical to the software [13]. Plan-driven approaches, such as SPLMs, are better suited, as they promote transparent, traceable and predictable development.

4.3 Development Team Organization

Several critical people factors for AMs have been identified: amicability, talent, skill and communication. Given these factors, some have developed a view that AMs only work with premium people. Consequently, people may think that an Agile project may require more exceptional developers than a plan-driven project does, at least there should be a higher proportion of experienced developers in Agile projects teams than in the plan-driven projects teams. However, this is not necessarily the case [1].

A difference between AMs and SPLMs lies in allocations of developers. SPL organizational management will be more likely to move their best people to core asset development and assign product development tasks to developers with average skills. In Agile projects, all team members work together on the product development [1].

Another difference between AMs and SPL is their different strategies of assigning responsibilities to people. One of AMs' basic principles is that people are the most important ingredient of success and the best architectures, requirements and designs emerge from self-organizing teams. Responsibilities are not handed to individual team members from the outside but are communicated to the team as a whole, and the team determines the best way to fulfill them. No single team member is responsible for the architecture, or the requirements, or the tests, etc. [1] The team shares those responsibilities and each team member has influence over them. However, in SPL organization, every individual is allocated a well defined role with specified responsibilities.

4.4 Discussion

When considering what and how to tailor a SPLM to introduce more agility into the project management activities, each can be considered (technology and tool selection, development team organization), while taking into account the type of the project (size, distribution, single product or product line, safety or security critical, etc.).

Both AMs and SPLMs indicate tools need to be used efficiently. Agile methodologists advocate using simple tools, which are sometimes not powerful enough to support concurrent and parallel features desired in a SPL organization. Technology impacts are transient in AMs because they only affect their current projects. SPL organizations need to assess and select tools and technology more carefully, due to the increased risks associated with their long term use.

Established AMs are not particularly well suited for large, distributed, security or safety critical and product line oriented projects. The tightly coordinated communication of a larger number of developers, perhaps geographically distributed, is more difficult to achieve with the Agile principles. The focus on Agile principle of "early and continuous delivery" may lead to under-treatment of the non-functional requirements that are critical to delivering safe and secure software.

5. Conclusions

A comparison of Agile Methods and Software Product Line approaches has been presented in this work with respect to engineering, software quality assurance and project management criteria.

Based on the comparison, the established AMs and SPLMs share a common, high level goal: deliver quality software quickly. They also highly value the

ability of their methods and processes to accommodate changes, especially those of requirements. However, they have different strategies and practices for software development; our study complements the results presented in [13], which compares agile and plan driven approaches (not specifically product line approaches). Based on the comparison, it seems possible to bring the methods closer together by tailoring a SPLM with agility, to define an Agile Software Production Line Method (ASPLM).

A significant advantage of an ASPLM is that it could be used on a wider variety of projects, such as safety or security critical systems, larger-scale, or with distributed development teams. For example, an ASPLM could be applied to financial, medical, or secure communication systems. In addition, the ASPLM might be aimed at developing a set of similar systems from common core assets, which are like those in SPL but are simpler, flexible and adaptive because they could represent the most significant commonalities in a domain, rather than a comprehensive set. ASPLM leaves room for further development work to meet customers changing requirements, rather than pure customization of core assets.

Future work involves defining and validating an ASPLM that includes agile principles (for example as proposed in DSDM). We have selected a financial product line application, which provides a variety of credit card account reconciliation techniques. Due to the security critical nature of this application, an established AM is not a suitable choice. All three main activities of the SPL approaches need to be investigated for possible agility: domain engineering, product development and management. The introduction of agility does not need to be an "all or nothing" approach. Rather, a strategic introduction of agility to the main SPLM processes and the associated trade-offs and risks need to be considered. Defining a set of guidelines to assist with tailoring the ASPL will also be investigated.

6. References

[1] A. Cockburn, *Agile Software Development*, Addison-Wesley, Highsmith Series, 2001.

[2] R.C. Martin, Agile Software Development: Principles, Patterns and Practices, Prentice-Hall, 2002.

[3] P. Clements, and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley Professional, 2001.

[4] K. Pohl, G. Böckle, and F.J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer Verlag, 2005.

[5] Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum*, Prentice-Hall, 2001

[6] Kent Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, 1999.

[7] DSDM Consortium, http://www.dsdm.org/

[8] Steven R. Palmer, A Practical Guide to Feature Driven-Development, Prentice-Hall, 2003.

[9] Magnus Eriksson, Jürgen Börstler, Kjell Borg, The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations, Lecture Notes in Computer Science, Volume 3714, Oct 2005, Pages 33 - 44.

[10] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, Jan Bosch, COVAMOF: A Framework for Modeling Variability in Software Product Families, Lecture Notes in Computer Science, Volume 3154, Jan 2004, Pages 197 – 213.

[11] Samuel A. Ajila and Patrick J. Tierney, The FOOM Method - Modeling Software Product Line in Industrial Settings, In The 2002 International Conference on Software Engineering Research and Practice (SERP 2002), June 24-27, 2002, Las Vegas, Nevada, USA.

[12] Jim Highsmith and Alistair Cockburn, *Agile Software Development: The Business of Innovation*, *Computer*, Sept.2001, pp.120-122

[13] Barry Boehm, Get Ready for Agile Methods, with Care, Computer, Jan.2002, pp.64-69

[14] Carnegie Mellon Software Engineering Institute, *A Framework for Software Product Line Practice*, Version 4.2, http://www.sei.cmu.edu/productlines/framework.html

[15] B. Joseph Pine and Stan Davis, *Mass Customization: The New Frontier in Business Competition*, Harvard Business School Press, April, 1999

[16] Scott W. Ambler, Agile Modeling: Effective Practices for Extreme Programming and the Unified Process, Wiley, March.2002

[17] Joe Farah, *CM: THE NEXT GENERATION – Agile Configuartion Management*, Configuration Management Journal, Vol.5 No.4, April 2006

[18] Carnegie Mellon Software Engineering Institute, Domain Engineering Website, http://www.sei.cmu.edu/domain-engineering/domain_engineering.html

[19] Navarrete, F., Botella, P., and Franch, X., "How Agile COTS Selection Methods are (and can be)?}, 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 30 August - 3 September, 2005, Porto, Portugal, pp. 160-167.

[20] Philippe Kruchten, *Rational Unified Process, The: An Introduction*, 3rd Edition, Addison-Wesley, 2004