Understanding the dynamics of product software development using the concept of co-evolution

I. Heitlager, S. Jansen, R. Helms and S. Brinkkemper

Abstract— **The context in which software is developed determines its evolution. Specifically for software developed as a product by startups, the uncertainty of market, platform and team not only determine the evolution of the product, but also of the process. During the lifecycle of the product the organization changes and different product improvement and process improvement patterns can be observed. With this in mind the Abernathy and Utterback dynamic innovation model proves to be an applicable view for the evolution of product software and its process. In this paper we introduce this theoretical model as a basis for further research in the co-evolution of product software development. Based on an anecdotal case study of a Software-as-a-Service product an example of the co-evolution of process and product in the software product industry is provided.**

Index Terms— **Software evolution, software process evolution, product software, co-evolution**

I. Introduction

THE product software industry is a rapidly growing sector [1]. This industry builds software as a standardized product which is sold to many customers [2]. Newcomers are attracted by the outlook of selling a standard product to many. Even after the burst of the dot-com bubble, many new product software companies start. Unfortunately many have ceased to exist, while trying to follow the footsteps of the Redmond giant.

The entrepreneurial nature of these startup or green field companies is highly attractive. However little seems to be known about the dynamics of those companies operating in the product software industry. A common image is a linear approach; one starts with an idea, selects a development method, elaborates on the idea, creates the product and finally implements the product in the market. As if all product introductions would in fact follow the same path.

These startups are characterized by their high dynamics, meaning rapid changes, and great uncertainty. The uncertainty exists on three levels: market, platform and team level. On the other hand they have to meet economical objectives, meaning they have to carefully balance investments with returns. This restricts their moves and choices.

It is too optimistic to assume that market uncertainty is resolved with the first introduction of the product to the market. Therefore product innovations will continue to take place to meet market demand. This leads to the evolution of the software product. As the company starts software development and support processes cope with the major changes in the early stages of the product. However these processes are of a different order than processes required if the company has to cope with scale.

The linear viewpoint is too restricted and in reality product software companies show much more diverse dynamics. While they try to reduce uncertainty on all three levels on one hand and have to commit to economical objectives on the other hand, both product and process co-evolve as the company is maturing.

The community is now challenged to formulate a unified theory on the evolution of software and software process [3] [4]. At the same time, the industry is also much in need [2] as some feel they have to 'rewrite the books' of software engineering [5][6][7]. To understand the dynamics and co-evolution of these product software startup companies we would like to seek theoretical support in the dynamic innovation model of Abernathy and Utterback, a well established innovation model for manufacturing industries [8][9][10] .

We show an example of the dynamic nature of these startup environments as we describe the evolution of the product, process and organization in its initial stages in a short anecdotal case study. We discuss a Software-as-a-Service (SaaS), formerly known as Application Service Provider (ASP), product operating in the Supply Chain Management

I. Heitlager is with VivaCadena b.v., Keizersgracht 169-1, 1016 DP, Amsterdam, The Netherlands since the start of the company in 2000 as director technology (ilja.heitlager@vivacadena.com) and affiliated as external researcher at the Institute of Information and Computing Sciences of the University of Utrecht, The Netherlands.

S. Jansen and R. Helms are with the Institute of Information and Computing Sciences of the University of Utrecht, The Netherlands.

S. Brinkkemper is full professor at the Institute of Information and Computing Sciences of the University of Utrecht, The Netherlands (s.brinkkemper@cs.uu.nl)

Industry. We reflect on the choices made and generalize on these findings using the dynamic innovation model of Abernathy and Utterback.

In the next section of this paper we elaborate on the difference between the development of software in a project based environment and product software developed in startup company. In section III we discuss the anecdotal case study, after which an explanation of the Abernathy and Utterback theoretical model follows in section IV. We conclude in section V.

II. The characteristics of a product software environment

The construction of new piece of software in a start up company for a niche market is an exciting endeavor. The niche market is unknown and therefore the revenues and requirements are uncertain, hard to predict. The team is new and last, but not least, the platform to develop against is still to be selected. Starting from these uncertainties the business is seeking exponential growth by 'opening up the market'.

Looking for growth is a natural response of a business to prevent itself from early termination. A *project* however has quite the opposite objective. By its very nature a project has to end itself. There is a fundamental difference between software delivered as a project and software developed as a product. It is essential to understand these differences because they will lead to different dynamics.

Since the goal of a project is to end itself, it looks for certainty by being rigid in preferably all of the quadrupled: time, resources, features and quality. This is shown in the controversy between Disciplined and Agile methods of software construction on following a plan versus the embracement of change [13]. Security is formulated, at least from the contract point of view. These contracts are preferably offered to well trained and experienced teams.

Market uncertainty however is not offering the luxury of contract security. These kinds of environments attract a more entrepreneurial crowd, broadly orientated, with a sense for market needs and prepared to pick up and perform whatever is required. Furthermore, since new ground is covered many issues are addressed for the first time. And these are not necessarily handled in the most optimal and repeatable way.

Any system looking for exponential growth, such as a business, is requiring flexibility and adaptability before optimal efficient execution. Growth can only be assured, if the system adapts to new requirements. As such for green field business environments the agile principles, like valuing interactions over process and working software over documentation, seem to be more appropriate [13] [14]. We observed however that these principles simply don't last. A business will change as it matures. In its initial organization is more *fluid*; agile and entrepreneurial. As product and market mature, many major changes are expected. However as the market demand increases, the organization has to adopt and prepare for scale. The product has to service many different customers. Moreover, the organization has to keep up with the large volume of requests and demands. This situation is requiring more structured and efficient processes. This is much more in line with disciplined rigid methodologies.

In these dynamic environments the decision making cannot consider the product itself, but has to consider the production means, meaning the development and support processes, also. In such environments an integral view on the product development as whole is required, taking all business and engineering aspects into account [12]. It is essential to take these into account in order to have a better understanding of to the dynamics and evolution of both product and its production processes. This conclusion is also drawn by Lehman with the formulation of the 8th law, evolution processes must be considered multi-level, multi-loop and multi-agent feedback systems, as with the 'FEAST hypothesis' that to achieve major process improvement global dynamics must be taken into account [3]. This will be explained in the next section.

III. An Anecdotal case study

We would like to exemplify the global dynamics and co-evolution of product and process by the following anecdotal case study. In this case study we will give a brief historical overview of major events regarding the evolution of the product, development and support processes and organization.

The object of study is VivaCadena. This Dutch company was founded in 2000. Its mission is to become a world leader as strategic service provider for enterprises, which see strategic value in their distributions chains. Its market is retail companies, wholesale companies and manufacturing companies with respect to finished goods distribution. The first author of this paper started as employee number 3 when the company was founded.

The company constructed some prototypes initially, which were tested in the market with customers. After one year the first real customer arrived and the first version of the software, with the same name as the company, was constructed. The choice was made for a Software-as-a-Service based application. VivaCadena offers its functionality as an add-on product on existing Enterprise Resource Planning (ERP) applications by exchanging data over the web. The choice for add-on software was made, as since from the start the constraint was set not to rebuild existing ERP systems. Instead the core guiding principle was to offer additional intelligence and leverage on the existing IT infrastructures.

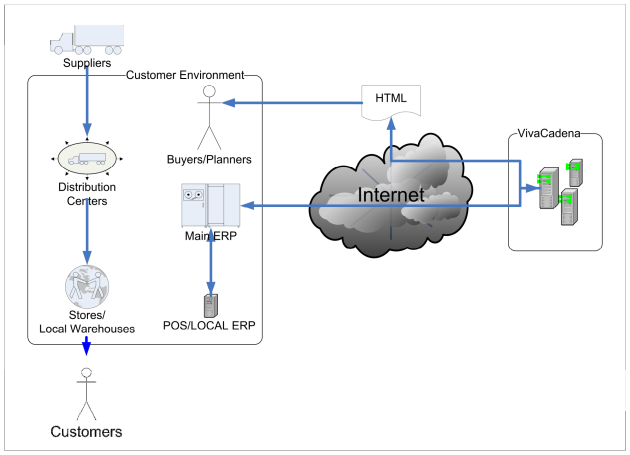

Figure 1, VivaCadena Enterprise Architecture

In short VivaCadena is collecting transactional information about the supply chain once a day from central systems (Figure 1). It will send back procurement parameters once a day. The solution offers an improved planning methodology based on the notion of responsive supply chains.

The customer enterprise has already invested into the IT execution environment to capture sales and automatically place orders. Physically it is exchanging transactional and master data daily from the reference environment. After processing at the VivaCadena servers new planning parameters are sent back and users have access to exception reports over the internet. The complete discussion of the application itself however is outside the scope of this paper.

The investment mentality was to start with a small team (5 persons) to be flexible and to keep start up costs low. Although not yet formulated in 2000, the team operated according to the agile principles [14] of valuing working software over documentation and relying on close interaction instead of process. In this entrepreneurial environment the organization relied on broadly oriented individuals without written job descriptions. The first version of the product was built with generic off-the-shelf database, interfacing and reporting technology. The product, while already being delivered and operational, underwent many major changes. New insights were incorporated easily without too much emphasis on process. The application, being an add-on product, was mapped straight to the customer data models without much consideration for data exchange interfacing standards.

The marketing strategy was to prove the concept. Therefore a wide range of different customers was targeted. As a consequence the early selection of the customer platform could not be made.

To put it differently: The product would still be in its formulating stage, if it was defined to connect to all possible platforms. This would surely lead to failure as it would run out of resources before being finalized. Although we do not deny the truth that applying changes during requirements formulation is cheaper than after implementation, the company had to deliver a product. Given the market uncertainty it was assumed to be better to apply changes than to have a product which is not attractive to the market or not ready at all. Only once the market is more stable and requirements are better known it is probably the right time to improve the change process by better requirement management. Getting to know the market is also a matter of restricting all possible options and focus on the current opportunities.

The first three implementations of the product therefore lacked a standard architecture and each individual implementation looked like it was built up from the ground, mixing generic and customized code in one monolithic application. As such, bugs had to be fixed and tested for each individual installation. During that first period focus was more on delivery than on structure.

The application was a clear example of Lehman's second law; that complexity increases unless work is done to maintain it [3][3][2]. The number of implementations was still manageable by the small team, however it was clearly understood that this approach was not scalable.

A fundamental difference between this particular application and more traditional shrink-wrapped software is that the VivaCadena application can be marked as *consult-ware,* a popular term used for software that requires adaptation to the specific environment for its implementation. It has a similar problem feature set, but in the roll-out it still relies on individual implementation, as it has to be fit to the specific supply chain and platform configuration.

As mentioned before the monolithic nature and direct fit to customer environment was perceived as non-scalable. And after VivaCadena implemented the first three customers, the company felt it had gained enough knowledge. As such requirements could be reformulated for the development of a new architecture. The monolithic nature made the system hard to maintain, extend and test. To make the product scalable and attractive to the many platforms, a clear distinction between custom and generic code was required. With this reformulation a *dominant design* emerged, leading to version 2.0. A choice was made to deliver a generic product with standard data loading and processing primitives based on an event driven architecture. The architecture contains a standard extendable set of business logic primitives to process the daily load. An xml based domain specific language was developed to customize for each individual installation and cope with the various platforms and environments.

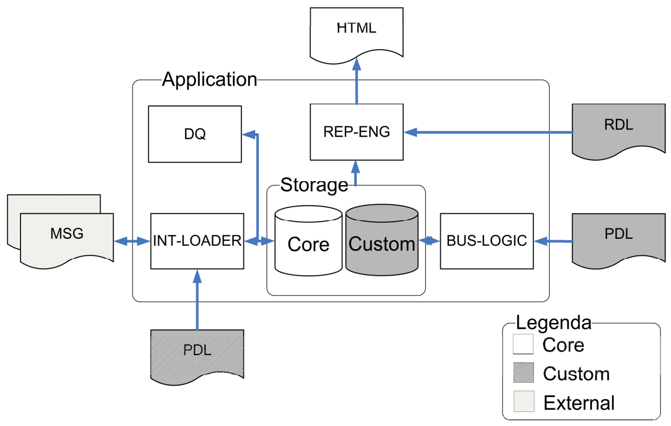

Figure 2, VivaCadena High Level Architecture

The application (Figure 2) has, on a high level, the following modules: A core database (**Storage: Core**) which is under strict versioning, a flexible and configurable interface layer (**INT-LOADER**) to load, transform, filter and enrich customer data (**MSG**). The loader is based on an event-driven architecture. A configurable set of business logic (**BUS-LOGIC**) primitives to calculate the planning and supportive parameters. It is configurable to the specific implementation, using a Process Definition Language (**PDL**). For the delivery of exception reports it contains an off-the-shelf reporting engine (**REG-ENG**), configurable using a Report Definition Language (**RDL**).

The important notion is the clear distinction between core development of data model and processing primitives and the customization layer which composes features using a secondary xml based domain specific language. The secondary language allowed for easy configuration of the processing primitives while restricting the implementation team leaving the core feature extensions to the core development team. As stated core elements are under strict version control.

A data quality (**DQ**) module is a supportive module for root cause analysis after incidents. Since this is an add-on product many incidents are caused by interfacing issues not necessary internal to the system. Many times incidents are caused by data issues from the customer. A lesson learned from the first few implementations.

Once the architecture was set, architectural erosion was controlled. Adaptation required for specific platforms were handled in the customization layer using the Process Definition Language. Furthermore the flexible architecture allowed better extendibility since extensions to the core layer now only take place in predefined areas. The extensions in VivaCadena are classified in one of the following categories:

1. Support: all features and extensions required to support the operation.
2. Technical: all features and extensions required to cope with platform issues.
3. Conceptual: all features and extensions considered functional requirements delivering the actual functionality.

It was found that already after 8 installations the majority of conceptual improvements stabilized as a clear example of the fifth law of Lehman on the Conservation of Familiarity [2].

Initially, the team was solely occupied with the delivery of the product. With the arrival of more customers things changed however. Having more customers required more resources and therefore new processes. Quality had to be ensured and changes in operational systems had to be propagated fast and in a controlled and repeatable way, without interruption. The initial team, knowing the application by heart, was able to propagate changes without much effort or error. The growing team saw less experienced people. The new team members did not have the chance to grow with the application, like the original team members. Therefore they lack a thorough and deep understanding of the application.

The product evolved from a custom build specific monolithic application to a flexible architecture with a core and custom layer. Using this approach the application is protected against architectural erosion from the many different platforms encountered in the different customer environments. It also allowed better reuse and made way for improvements of support processes.

The lack of experience as a consequence of growth is buffered by more mature risk averting procedures. And the standard core architecture allowed so. A whole new set of process improvements where implemented, like the development of unit tests for individual parts of the software. Also automated deployment tools and packagers were developed to ensure changes were propagated in a controlled manner. A Development Test Accept and Production (DTAP) environment was installed, to pretest every change before rolling into production. This is a well known industry practice to use multiple similar environments to separate core development from testing and production. It allows to ‘pretest’ changes before they are accepted into production.

Product innovations still continued. New requests where formulated by customers, which could not be resolved in the initial core layer. The majority of the effort however was on process improvements.

Basically more process was implemented by adding more artifacts like tests and checklists. The notion of adding more process later in a project is known, but unelaborated, by the advocates of the Unified Process [16]. As such a whole constellation of products was created for testing, configuration, monitoring and deployment. As an insight we state that as the business grows not only the product itself evolves, but also its production means and support artifacts.

It was observed that while the initial team was still relying on their craftsmanship to, for example, configure the product directly into the database, the growth required more specific tools to test and check. It required the mechanization of menial and repeatable support tasks. For example, the original team knew which parameters to check before releasing a new implementation. The new team members did not always check all parameters, although prescribed in documents. By molding these tests into code the configuration parameters check was standardized. Using this approach discipline is enforced by mechanization.

Although the team was initially a great supporter of the Agile camp they had to shy away from a few principles due to increase of scale and introduction of new people. For example the need for documentation became eminent for training and supporting the new team members. Relying on one-on-one knowledge transfer is too time consuming. Also solely relying on tacit knowledge in people is not considered a business continuity principle. At the start, the product was too volatile to have the documentation cope with the changes. As soon as the dominant design appeared documentation was brought back to standard.

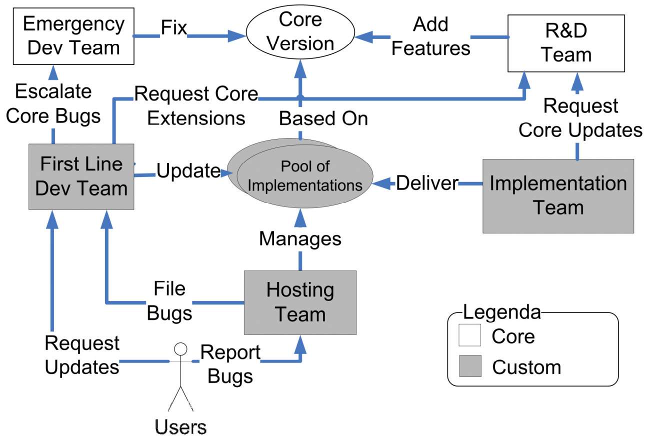

Figure 3, VivaCadena 2.0 organization chart

With the growth, the company reorganized two years ago and set up multiple teams (Figure 3). The core versions are being further developed by the original R&D team which still operates according to Agile principles. New implementations are done by an implementation team, which operates under a firm waterfall like project approach. Such was now possible since the company had now learned the pitfalls of performing the implementations. Simple changes are maintained by a First Line development team on an ad-hoc basis. This team is operating in a pure reactive mode. Most of the simple changes can be performed at customization level and the First Line team operates as a filter to the core R&D team. The First Line development team also operates as a root cause analysis team for new and unknown incidents and bugs. With root cause analysis solutions and process improvements are formulated either on tool, operational or implementation level. Only those that are caused at core application level are escalated to the Emergency development team, basically one of the R&D team members 'on call', for further analysis and resolution.

What is observed is that the company organization structure and the application architecture evolve together and that due to the nature of the company both see a clear separation in core and customization. It is assumed that with the further growth of the company more specialization will occur.

Currently the company has a dozen implementations over several continents with about 40 employees under direct contract and subcontract. It is preparing itself for a new wave of customers. As such it is standardizing its procedures and measurement systems to further improve quality. As a technique it is separating roles and responsibilities and has implemented quality measurement systems like ISO. It is now improving the implementation process to make it more standard and predictable.

When still being small, the company was relying on a few generalists valuing people and interactions. Currently it has to rely, for its implementations and further development, on process and tool. The company is living according the Deming rule, "it is not the person, it is the process".

IV. ABERNATHY AND UTTERBACK MODEL REVISITED

While seeking for a better understanding of the observed phenomena, we found a resemblance with the dynamic model of innovation of Abernathy and Utterback for manufacturing companies [8][9][10]. The essence of the model is that the innovation patterns in product and process of a *productive unit* change as it matures.

The model is based on historical studies in the manufacturing industry in different areas such as photo cameras, electronics industry, the development of the DC 3 airplane and other technologies. It states that the innovations in product, process and company structure change drastically as the company evolves from a small technology driven enterprise to a high volume producer (see Figure 4). The *productive unit* can be as simple as a startup company or a single department of a larger company to the industry as a whole.

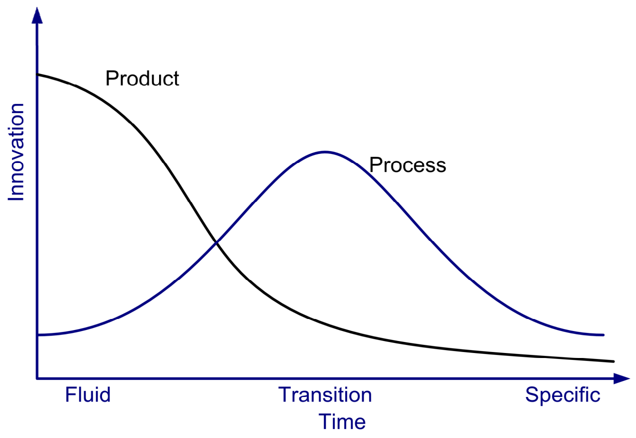

Figure 4, The Abernathy and Utterback dynamic model

The model states that there are distinctive patterns. At the start of a company the pattern is *fluid* as the atmosphere is entrepreneurial and organization structure is flat. The company is relying on broadly oriented and highly skilled labor. The product characteristics are underdetermined and the product sees many major changes many times with many custom designs. As such production processes are inefficient. This clearly marks the first year of the company.

The innovation patterns change as soon as market demand increases. This phase is called *specific* and emphasis is on cost reduction. The innovations are mostly on quality and pressures to reduce costs. The product characteristics are stabilized with undifferentiated standard products. Changes still occur but are of incremental nature and the costs of changes are high. The organizational structure is rigid and structured. As the company is maturing the service management and implementation process are showing signs of rigidity. This is the structure VivaCadena is now moving towards.

As the productive unit moves from *fluid* to *specific* it has to go through a *transition* phase. In this phase a *dominant design* emerges, which is marked by a reduction of product innovations. The organization sees many project and tasks groups mostly occupied with process innovations to prepare for scale. This is also found in the second stage of VivaCadena.

For a full explanation of the model we refer to [8][9][10]. The mapping of the domain of manufacturing to software development does require some clarification. In our mapping we translate product innovation with any of the functional requirement activities. We have mapped process improvement to non-functional requirement implementation and process improvement of the development and service management processes. The concept of dominant design is translated to the specific architecture which makes all future progress possible. In the case study the advent of the VivaCadena 2.0 architecture.

V. Conclusion

It has often been asked by researchers how to set up an evolutionary process of software development. The reasons for evolution can only be understood by looking at a higher abstraction level. Evolution caused by the system or context in which software is developed.

Our research is focused on the specifics of product software. This industry is characterized by great uncertainty in market and platform. Resolving market uncertainty requires a flexible organization. This offers many opportunities for startup companies as opposed to more rigid and mature organizations. However these startup companies add a third level of uncertainty due to the team composition.

Although the body of knowledge on methods and techniques is large, the acceptance and usage seems to be low [19]. VivaCadena was no different initially. We find an explanation in the fact that market uncertainty demands matching requirements first before process execution excellence becomes a dominant theme. Market uncertainty can however only be taken away with actual working software, and as a result this software has to evolve and will see many changes.

The discussion between the Agilists and Disciplined disciples in the software engineering methodology discussion is often about which method is more appropriate. A much heard statement is that agile methods only work for small teams [13]. As market uncertainty can only be taken away with actual working software an Agile approach seems more appropriate. However as market demand is increases the organization has to change to cope with the high volume. This provides a need for more disciplined and process improving/cost reduction approaches. As such the discussion should not be held about the mutual exclusion of the methodologies, but rather on their coexistence *in time.*

New customers will bring new requirements. Therefore changes to the software will occur. However the Abernathy and Utterback model predict that these changes are of a more incremental nature. Therefore during the complete lifecycle of the software product both product and process evolve, but with different patterns. With the increase of market demand the transition from *fluid* to *specific* has to take place.

The transition speed from *fluid* to *specific* is determined by the market itself and not so much by a good development process alone. We would like to state that a good evolution theory should incorporate the notion of a dominant design, being a firm architecture on which the product and process can evolve.

As soon as the architecture is defined, scale up can take place. If not, Lehman's second law will be active as soon as you hit the market. The importance of early investments in architecture for flexibility described in the studies of MacCormack [17][18].

We seek to formulate an evolutionary step-up model for the product software development process and its organization in co-evolution with software itself. As a start we have only generalized on basis of an anecdotal case study and therefore our research needs more empirical evidence. The evolutionary model will support in the decision making for innovation managers and product software engineers regarding how en when to invest in product or process. Our initial findings have shown that for product software development, like for manufactured products, the dynamics, and therefore the order of co-evolution of product and process, are predetermined.

Acknowledgment

The authors would like to thank Yohyon van Zantwijk founder and owner of VivaCadena b.v. for the opportunity to write this paper. The authors would also like to thank the anonymous reviewers for there remarks.

References

[1] OECD, *The software sector: Growth, structure and policy issues*, OECD Reports DSTI/ICCP/IE(2000)8/REV2, 2001.
[2] L. Xu, S. Brinkkemper: "Concepts of Product Software: Paving the Road for Urgently Needed Research", CAiSE Workshops (2) pp. 523-528, 2005.
[3] M.M. Lehman, "Rules and Tools for Software Evolution Planning and Management", *Annals of Software Engineering*, vol. 11, no. 1, pp. 15-44, 2001.
[4] M.M. Lehman, *SETh proposal*, EPSRC, 2001.
[5] Web 2.0 conference, see http://www.web2con.com/.
[6] T. O'Reilly, Web 2.0 Manifest, O'Reilly, 2005, Available: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.
[7] 37 Signals, *Getting Real*, 2006, Available: http://www.37signals.com/
[8] J.M. Utterback and W.J. Abernathy, "A dynamic model of process and product innovation", *Omega*, 1975.
[9] W.J. Abernathy and J.M. Utterback , "Patterns of Industrial Innovation", *Technology Review*, 1978.
[10] J.M. Utterback, *Mastering the Dynamics of Innovation*, Boston, MA: HBS Press, 1994.
[11] M.M. Lehman, D.E. Perry, W.M. Turski, "Why is it so hard to find Feedback Control in Software Processes?", *Proceedings of the 19th Australasian Computer Science Conference*, Melbourne, Australia, Jan.-Feb. 1996.
[12] K. Rautiainen, et all, "4CC: a framework for managing software product development", *Engineering Management Journal*, vol. 4, no. 2, June 2002.
[13] B. Boehm and R. Turner, *Balancing Agility and Discipline, A guide to the perplexed,* Reading, MA: Addison Wesley, 2005.
[14] Agile Manifesto, 2001, Available: http://www.agilemanifesto.org/.
[15] K.H. Bennett and V.T. Rajlich, "Software Maintenance and Evolution: a Roadmap" in *The Future of Software*, C. ACM, pp. 73-87, 2000.
[16] W.E. Royce, *Software Project Management: A Unified Framework.* Reading, MA: Addison-Wesley, 1998.
[17] A. MacCormack, "Product-Development Practices that Work: How Internet Companies Build Software", *MIT Sloan Management Review*, vol. 42, no. 2, pp. 75--84, 2001.
[18] A. MacCormack and R. Verganti, "Managing the Sources of Uncertainty: Matching Process and Context in Software Development", *Journal of Product Innovation Management*, vol. 20, no. 3, pp. 217-232, 2003.
[19] M. Cusumano, A. MacCormack, C.F. Kemerer, W. Crandall, "A Global Survey of Software Development Practices", MIT Report 178, June, 2003, Available: http://ebusiness.mit.edu/.