A Tentative Technique for the Study and Planning of Co-Evolution in Product Software Startups

Ilja Heitlager, Remko Helms, Sjaak Brinkkemper
Department of Information and Computing Sciences
Utrecht University, The Netherlands
i.heitlager@sig.nl, r.helms@cs.uu.nl, s.brinkkemper@cs.uu.nl

Abstract

Startups play an important role in the production of product software. We state that these companies follow a specific dynamic in which both product and software process evolve. Products do not simply grow linearly by adding features, but will also have to improve on non-functional qualities of the software, like portability or reliability. The software process evolves by adding new and improving existing process models. In this paper we introduce a technique to visualize the prioritization of evolution steps in both the software product and the software process. With this technique to co-evolution can be studied and this knowledge serve as input for the planning of future startups.

1. Introduction

Product software (PS) is produced by a wide variety of companies. A dominant part is taken up by startups [2]. These companies show varying results, with reports of 60% to 90% of these companies not surviving after 5 years [2][3]. Out of those that survive only few prove to be *promising* startups, adapt and become big [4][5].

Success for these companies depends on many factors. The reasons for their becoming might also be different. However all startups share a common pattern: a few individuals starting with scarce resources and good intentions to grow their company into a mature organization. The high mortality rates might lead someone to think that a successful company is just sheer random luck. However we follow Bhidé that the steps taken by these software companies are actually conscious acts [4].

Starting PS companies face uncertainty at three levels: *market*, *platform* and *team* [6]. This will be explained in the next section. When gaining momentum in the market, these companies bootstrap themselves out of this uncertainty. We state that with the growth of the company, both the product and the process models will evolve. This is what we mean with co-evolution and in previous work we have suggested that PS companies, like manufacturing companies and industries, follow a specific dynamic [7].

In this paper we propose a technique to both study and plan this co-evolution of product and process. Since all steps in the evolution of startup SP companies are conscious steps, we want to study and understand evolutionary patterns for different companies. The proposed technique will visualize the co-evolution and show what process area or product area receives attention next. The technique is based on the ISO 12207 for software lifecycle processes [8] and the ISO 9126 standard on the quality of software [9].

This paper is structured as follows: In the next section we provide the motivation behind this paper. In section 3 we continue to elaborate on the dynamics of startup product software companies. The planning technique will be described in section 4. Some examples are discussed in section 5. Following in section 6 we discuss related research. Conclusions and further research are discussed section 7.

2. Motivation

The main assumption in our research is that starting PS companies bootstrap themselves to mature companies by evolving both their product and their development process as a conscious act. A better understanding of this co-evolution will support future companies. The central research question is:

> *"How to describe the co-evolution of product software companies and can this be applied as a planning tool for future product software startups."*

New product software development within startups differs from project and contract based software development. First of all contract or project based software development is closed development since the project is expected to stop sometime. Product software development is open ended. As long as customers are interested the software will evolve due to new requirements. If not steps will be taken to make to product attractive again which will also lead to changes of the product.

Secondly, unlike project based development, new product software development face three levels of uncertainty at the start: *market*, *platform* and *team*. The first level of uncertainty, market uncertainty, denotes the fact that there will only be a market once the product exists and that a new product will have to be constructed without a clear understanding of the market requirements. This will automatically lead to evolution of the software.

The second level of uncertainty is the target platform on which to build the software. Due to business reasons the software might need to be made compatible with other

0-7695-3002-8/07 $25.00 © 2007 IEEE
DOI 10.1109/SE.2007.14

platforms. While hungry for the first customer, it might even lead to radical changes. Like for example NPR experienced when it rewrote its software from C/Unix to the COBOL/Mainframe platform just to get the first customer [4].

The last and third level of uncertainty is team skills. Most companies start with scarce resources and therefore small teams, which have to start working from scratch and have ample opportunity to train. Contract based development is often done by big companies which have more facilities to train their personal. They have more changes to get spread experience and to amortize investments in process optimization. Since promising startups offer little security these companies will probably attract a possibly talented but inexperienced crowd [4].

All these factors combined result in a ad-hoc informal organization with a need to adapt opportunistically to market demand [4]. We currently leave aside whether starting companies are blessed or cursed with these conditions. However as soon as the product starts selling, the company has to accommodate. This will result in more people and therefore more process. Identical to regular product manufacturing companies a product software company has to make the transition from an ad-hoc informal organization to a more rigid optimized and process controlled organization, which will be independent of individuals.

3. The dynamics of maturing product businesses

Small companies are not expected to remain small forever. As the product matures these companies gain more business. More business leads to more work, a growing organization, more people. And in the case of software development, this means more requirements, therefore evolution of the software. Depending on the nature of the software the growth will have an impact on the software, its architecture and the organization.

Any popular website like eBay or Amazon will see a tremendous amount of usage which requires a high performance architecture. The add-on supply chain management software package described in [6] will require a flexible architecture for easy customization of the software. Other PS companies have to operate efficiently to process the deluge of change requests.

To better understand the dynamics of PS companies, we have made the comparison with product manufacturing based on the Abernathy and Utterback model on the dynamics of innovation [6][18]. The insight gained from the Abernathy and Utterback model is that as time passes by a product does not grow or improve linearly, as shown in Figure 1.

The model explains that during the evolution the product and the organization together will see phase changes. The rate of growth on the product is understood to decrease, but more important the character changes once a larger group of

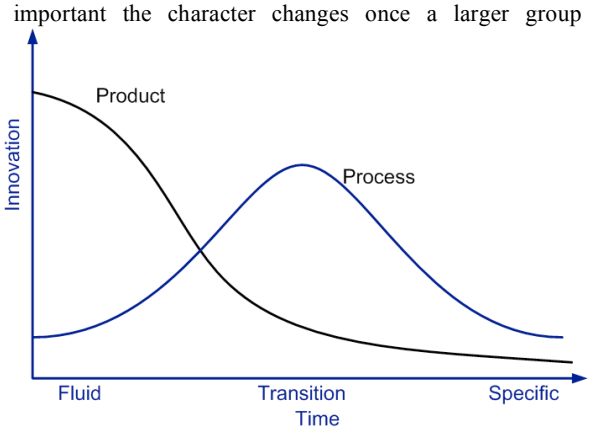

Figure 1. The universal model of innovation of Abernathy and Utterback.

customers start buying the product. This transition will force process improvements. This change requires refocusing. Different processes, which are fit for volume, are needed as the company matures to a high volume producer.

The Abernathy and Utterback model discriminates between three phases, as is summarized in Table 1. Each phase is characterized by a different rate of innovation or improvements to either product or process and in each phase the character of the tools and research will also differ.

In the *Fluid* state the processes are flexible, ad-hoc and inefficient. Changes can be applied quite easily. Products are build with generic tools. Market is still immature but different customers will see different implementations. Research, if any, will be unspecified and not necessarily contribute directly to the bottom line.

In the *Transition* phase the company has to prepare for a higher volume. At lease one product design emerged to accommodate the volume. Research will now take place on specific topics and process start to become more rigid.

For example in the *Fluid* phase the organization can do with simple XP based user stories to plan the development of the

Table 1, characteristics of the three Abernathy and Utterback phases, taken from [18].

	Fluid	Transition	Specific
Product	Various different implementations	At lease one product design to accommodate volume	Standardized products, mostly undifferentiated
Process	Flexible and inefficient	More rigid, changes in big steps	Efficient, rigid, cost of change is high
Tools	General purpose, skills required	Some parts automated	Special purpose, mostly automatic

software [10]. In the transition phase the requirement description will become more formal and documented,

In the *Specific* phase the processes are most rigid and focused on cost reduction. The product is standardized and tools will accommodate the process and research will be focused on process optimizations. The software change processes will now be fully automated and managed including metrics to prioritize what to incorporate in the next release.

4. Visualizing and planning co-evolution

To manage the evolution of product software one must balance the investments in product and process. At the right moment the transition has to be made to prepare for scale. What is required is a tool to express the planning of the evolution of the product and processes. The planning matrix, an example is shown in Figure 2, is based on the maturity assessment technique of DYA [22]. The matrix used here is a temporal decomposition of activities on different process and product areas. By doing so it allows to express *what* to apply *when*. To provide a common terminology the technique is based on the software standards ISO 12207 for a common reference on software processes and the ISO 9126 on software quality.

The matrix provides a simple map to reason about what will come next in the near future. Unlike the assessment techniques

	FLUID				TRANS				SPECIFIC			
	1	2	3	4	5	6	7	8	9	10	11	12
software requirements analysis		A			B				C			
software architectural design					A			B		C		
software detailed design			A		B					C		
software coding and testing	A			B			C					
software integration		A								C		
software installation			A								C	
quality assurance process	A			B				C				C
maintenance process			A			B				C		
problem resolution process				A			B				C	
documentation process		A			B					C		
configuration management		A		B				C				
Functionality	A		B		C		D		E			
Reliability			A		B			C				
Usability								A		B		C
Efficiency					A			B			C	
Maintainability		A			B			C				
Portability			A				B				C	

Figure 2. Example of the co-evolution planning matrix.

however, it does not specify capability levels for each individual process area following the same steps of specifying, managing, measure and optimize.

Each row in the matrix is either an ISO 12207 process area above or an ISO 9126 product quality characteristic below the line. Here we have selected the most relevant development processes from ISO 12207. A wider or smaller selection can be applied as well, for example when more mature product management processes are required. Each column is a priority step, for this example we have separated them into three groups of 4 steps each. The priorities provide an order and they should not be confused with actual time scales like days or years.

The letter denotes an improvement step. Each consecutive letter is an improvement within the process area or product area, either small or large. Each step denotes by a letter represents a group of actions taken.

Basically, the matrix combines the planning of the evolution of the process and the product in one single overview, it abstracts from how the processes are improved. It assumes that both will improve in bursts and only defines the areas and orders them in a time schema. It indicates which process or product area to address next. The actual method fragments to apply or to improve is outside of this schema, but should be selected from a Product Software Knowledge Infrastructure which is envisioned by our group [16].

5. Examples

In this section we discuss a few examples out of many possible patterns we expect to find once the matrix is applied to describe evolutions within companies. In the examples we explain some possible evolution steps by providing fragments of the matrix. The essence of the matrix is to describe the order of both product and process improvements. In these examples we describe the actual steps taken informally in text.

5.1. Evolving requirements management

Requirements management is a process area that is expected to grow with the organization as we discussed in [6]. In this example we describe a typical startup, which has a few developers that only have to justify themselves to each other. The features are added rather ad-hoc without a formal evaluation. After more customers are installed, both the preparation of the change and the planning of changes will be formalized since it now involves the customer requests as well. This is reflected in the matrix as:

	...				
	1	2	3	4	
requirements management	A		B		..
software detailed design			A		..
software coding and testing	A				..
management process	A	B			..
Functionality	A				..

With the start developers start coding focusing on functionality. They will setup a basic way of working, with some basic attention given to requirements analysis by writing and selecting user stories. In the second step some more attention is given to the scheduling by centralizing all open tasks in one issue tracker. In the third step requirements management is improved with a more formal report and an additional detailed design report.

5.2. Client software portability

Product software increasingly web-based. For this class of software it is required to be compatible with a wide range of

browser in order to serve the complete customer base. However serving a wide range of browsers also means to extend the test facilities and test case to make sure the software stays compatible with al these browser.

In other words fulfilling the non-functional requirement of being compatible with a wide range of browsers requires and extension of both the software and the testing process. Such a decision requires resources and the decision to allocate these resources might be postponed until finalization of the first part of the functionality or awaiting customer request.

Such a decision in reflected in the matrix as

	...				
	1	2	3	4	..
software coding and testing	A	B			..
Functionality	A				..
Portability		A			..

This system has a basic portability and in step two an update to both the testing process and portability of the software is performed, because of requirement of browser compatibility.

5.3. Server software portability

Portability is not only an issue on client side but can also be an issue on the server side. For example hardly any information system can do without database software. To build such a system at least one database management system has to be selected. Making the software compatible with multiple DBMS is however not a technical requirement but a marketing decision.

	...				
	1	2	3	4	..
software coding and testing	A		B		..
documentation process		A	B		..
Functionality	A				..
Portability			A		..

In order to address a wider market the software should be made compatible with multiple DBMS. This requires extension of the software to accommodate multiple DBMS. furthermore the testing processes need to be extended and installation documentation will have to be improved. Such a decision is reflected in the matrix as

At the first step some attention is given to functionality and a basic coding and testing process. The system runs on a simple DBMS. After a first version documentation is written for installation purposes. In the third step the decision has been made to upgrade the software to address multiple DBMS. Next to that the testing process requires update to test for various DBMS and installation manuals are updated to discuss installation for the various systems.

5.4. Operational reliability

This example is a typical example that has to do with scale. For a S-a-a-S product software company the hosting of the system is done by a hosting team related to that company. It will support many installations for different customers. The workload for the operational team increases as the number of customers increase with for example version updates. To reduce the workload more effective update procedures will have to be developed. Automation is applied here to repeat the menial tasks also reducing the changes of error. With a few customers a company can still rely on manual labor. With many customers these procedures are candidate for improvement.

	...				
	1	2	3	4	
software installation	A			B	..
quality assurance process			A	B	..
configuration management		A		B	..
Functionality	A				..
Reliability			A	B	..

In the first step the system has some basic installation procedures. In the second step attention is given to configuration management by installing a version control system. Reliability is getting attention in the third step as soon as a staged installation environment is installed by adding more identical hardware configurations. New versions are pre-tested on a test and acceptance environment. Now the company can automate the installation process, which is part of configuration management. This is a new increment in reliability of the software.

5.5. Adjust for multiple installations

Product software that has to be implemented at many sites is most flexible if the original source code is available and altered for each individual installation. However the changes for architectural erosion increase with each individual installation. This will make configuration management harder since each individual installation has to be treated individually. This is a typical situation that every product software company with customizable software has to deal with.

A possible approach to reduce complexities is to split the architecture and to provide a second layer with a domain specific language to specify customer specific adaptations. This approach has been applied in the case described in [7]. The rationale behind it is that an identical core is maintained between each installation, which will reduce the testing effort and control code erosion between installations.

	...				
	1	2	3	4	
software requirements analysis		A	B		..
software architectural design			A		..
software coding and testing	A		B		..
documentation process		A	B		..
configuration management		A	B		..
Functionality	A				..
Maintainability	A		B		..

In the first step software is developed ad-hoc in a basic development process. Second requirement is to make sure the software has configuration management to keep track of each individual installation and some requirements management to

understand what each installation contains. In the third step the software is made fit for multiple configurable installation by altering the architecture and adding a domain specific language.

6. Related research

The mainstream body of research in software engineering methodologies is about contract based product software, which is, as previously explained, opposite to product software development. For an overview, see for example [10].

We share the view with research team behind 4CC that a holistic approach is required in which software development and business requirements need to be taken into account [13]. Although this research does seem to start from a one-shot process approach, meaning it does not provide details on how to evolve the process with the organization.

The wide body on agile methods, like extreme programming does emphasis the adaptability due to changing environments, but offers ample support in planning different process areas and prioritizing non-functional requirements [11]. Research with the big software companies like the research on Microsoft and the browsers wars offer great insight in managing dynamic and highly competitive environments, but do not offer hardly any further insight on boot-strapping small companies [14][15].

The current techniques for measuring process maturity like CMMI [21] are overwhelming and mainly focus on the predictability by making the process quantifiable. In this paper we propose a lightweight technique for the study and planning of the evolution. The base assumption is that starting environments require lightweight processes and that the processes mature together with the product in incremental steps.

Within our own group on method engineering research is done on incrementally improving processes. This is called incremental method evolution in method engineering. So far most of this research concerns the more mature organizations and more specifically the (product) requirement related process areas [16].

7. Conclusions and further research

A great volume of software is produced within startup product software environments. These companies start very ad-hoc, trying to overcome the uncertainties of market, team and platform. The biggest struggle for these companies is to survive with only scarce resources.

Due to their size small companies have ample opportunity to learn. This paper tries to transcend that and offers a view and a tool to plan and manage the dynamics of such companies. The planning tool has been matched with standardized terminology on software process and product characteristics to give it a universal character.

With this tool, a technique is available to study, analyze and compare the evolution of real companies.

The matrix is formulated out of the conviction that for starting PS companies the process and product evolve hand in hand. Each PS company will experience a phase change when a larger group of customers will consume the product and make use of the services of the company.

The co-evolution planning matrix provides a means to reason about the order of improvements on a higher abstraction level. What is left is the size of the improvement and the actual method fragment addition to apply within the step. We expect to find good and bad practices in the evolution of product software companies, while we continue our research with case studies to validate the theoretical assumptions on the phase changes and the co-evolution of product and process.

References

[1] L. Xu, S. Brinkkemper: "Concepts of Product Software: Paving the Road for Urgently Needed Research", CAiSE Workshops (2), 2005, pp. 523-528.

[2] E. B. Roberts, *Entrepreneurs in High Technology: Lessons from MIT and beyond*, Oxford University Press, New York, NY, 1991.

[3] M.R. Chapman, *In Search of Stupidity: Over Twenty Years of High Tech Marketing Disasters, Second Edition*, Apress, Berkeley, CA, 2006.

[4] A.V. Bhidé, *The Origin and Evolution of New Businesses*, Oxford University Press US, New York, NY, 2003.

[5] J. Livingston, *Founders at Work: Stories of Startups' Early Days*, Apress, Berkeley, CA, 2007.

[6] 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, May 2003, pp. 217-232.

[7] I. Heitlager, S. Jansen, R. Helms and S. Brinkkemper, "Understanding the dynamics of product software development using the concept of co-evolution", *Second International IEEE Workshop on Software Evolvability (at ICSM 2006),* 2006.

[8] International Organization for Standardization. "ISO 12207:1995: Information technology – Software life cycle process", Geneva: ISO.

[9] International Organization for Standardization. "ISO 9126-1:2001: Software engineering -- Product quality -- Part 1: Quality model", Geneva: ISO.

[10] R.S. Pressman, *Sofware Engineering: A Practitionar's Approach,* Boston, Boston, MA, 2005.

[11] K. Beck, *Extreme Programming Explained: Embrace Change,* Addison-Wesley, Boston, MA, 2000.

[12] A. MacCormack, "Product-Development Practices that Work: How Internet Companies Build Software", *MIT Sloan Management Review*, vol. 42, no. 2, 2001, pp. 75-84.

[13] K. Rautiainen, et all, "4CC: a framework for managing software product development", *Engineering Management Journal*, vol. 4, no. 2, Jun. 2002, pp. 27-32.

[14] M.A. Cusumano and R.W. Selby, *Microsoft Secrets*, Free Press, New York, NY, 1995.

[15] M.A. Cusumano and D.B. Yoffie, "Software development on Internet Time," *Computer*, vol. 32, no. 10, Oct. 1999, pp. 60-69.

[16] I. van de Weerd, J. Versendaal and S. Brinkkemper, "A Product Software Knowledge Infrastructure for Situational Capab Maturation: Vision and Case Studies in Product Management", Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Luxembourg, 2006.

[17] M.E. Fayad, M.Laitinen and R.P. Ward, “Thinking objectively: software engineering in the small”, *Communications of the ACM*, vol. 43, no. 3, Mar. 2000, pp. 115-118.

[18] J.M. Utterback, *Mastering the Dynamics of Innovation,* HBS Press, Boston, MA, 1994.

[19] M.M. Lehman, “Rules and Tools for Software Evolution Planning and Management”, *Annals of Software Engineering*, vol. 11, no. 1, 2001, pp. 15-44.

[20] M.Laitinen, M.E. Fayad and R.P. Ward, “Thinking Objectively: The problem with scalability”, *Communications of the ACM*, vol. 43, no. 9, Sep. 2000. , pp. 105-107

[21] CMMI Product Team, Software Engineering Institute Capability Maturity Model Integration (CMMI), Version .1, CMU/SEI-2002-TR-012, 2002.

[22] M. van den Berg and M. van Steenbergen, Building an Enterprise Architecture practice: Tool, tips, best practices, ready to use insights, Springer-Verlag New York, Secausus, NJ, 2006.